

NODE.JS WORKSHOP

HARI 2: API NODE.JS

Adityo Pratomo (Framework)
Mozilla, 25 September 2016



REVIEW

- Dasar Node.js
- JS
 - function
 - anonymous function
 - callback
 - function sebagai object
 - object
 - array

MATERI HARI INI

- API Node.js
 - Module
 - HTTP
 - fs
- Membuat Server dengan Node.js

MODULE

- Sistem modular di dalam Node
- Blok penyusun sebuah aplikasi Node
- Seperti class di dalam C++
- Memungkinkan mengelompokkan sebuah script yang melakukan fungsi tertentu untuk digunakan di dalam script lain

MODULE

- Module bisa dibuat sendiri
 - ditempatkan di dalam direktori yang sama dengan script Node utama kita
- Alternatif lain, kita bisa menggunakan module yang tersedia di [npmjs.com](https://www.npmjs.com)
 - Install dengan `npm install nama-package`
 - Cek package yang tersedia via `npm search` atau situs NPM

MENGGUNAKAN MODULE

- Untuk digunakan, modul cukup diinisiasi di awal
- Module yang dibuat sendiri:
 - `var server = require (“./server”);`
- Module global (bawaan node / diinstal via npm):
 - `var fs = require (“fs”);`

MEMBUAT SERVER DENGAN NODE

- Kita akan langsung mempraktekkan membuat server menggunakan Node
- Dalam perjalanan, akan dibahas juga beberapa aspek Node
- Pembuatan server akan dilakukan dalam beberapa tahap

MEMBUAT SERVER DENGAN NODE

- Pada akhirnya, kita akan membuat sebuah server yang memungkinkan kita menulis di sebuah form dan hasilnya akan dikirim kembali ke kita
- Biasanya, ini bisa dilakukan dengan menggunakan kombinasi PHP+Apache/nginx
- Kita hanya menggunakan Node untuk menunjukkan bagaimana membuat server+aplikasi hanya dengannya
- Kita juga menggunakan module buatan sendiri untuk membuat arsitektur aplikasi lebih rapi

SERVER PART 1: HELLO HTTP

- Menjalankan server di localhost:8888
- Hanya memunculkan pesan “Hello World” dalam plain text di browser
- Menggunakan module global bernama HTTP
- Dijalankan dengan node `server.js`

```
//server.js
```

```
var http = require("http");
http.createServer(function(request, response) {
    response.writeHead(200, {
        "Content-Type": "text/plain"
    });
    response.write("Hello World");
    response.end();
}).listen(8888);
```

SERVER PART 1: HELLO HTTP

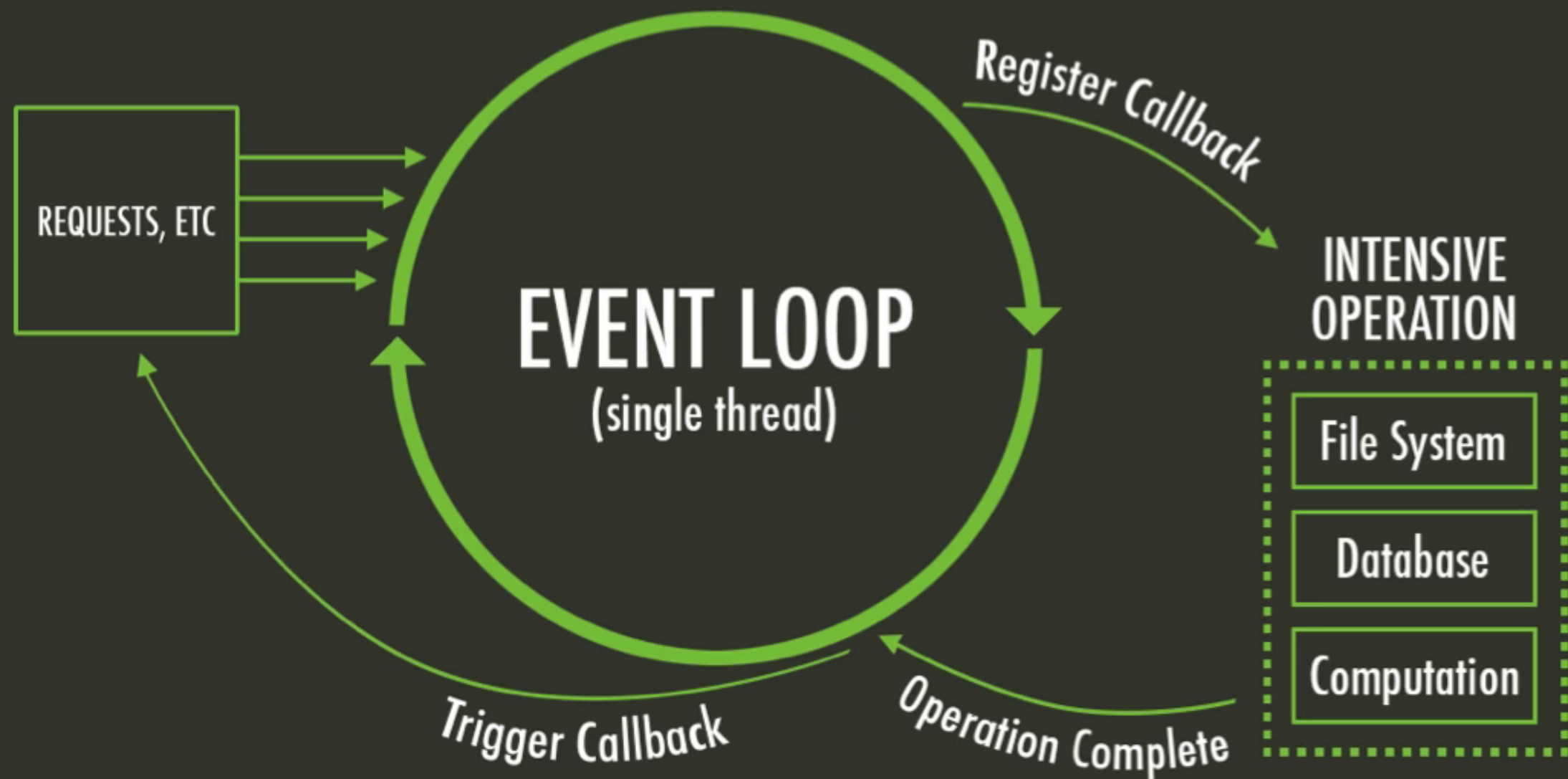
- Agar lebih rapih, kita tulis dalam bentuk berikut
- Menggunakan callback function
- Server berjalan terus, mengaktifkan event loop

```
var http = require("http");

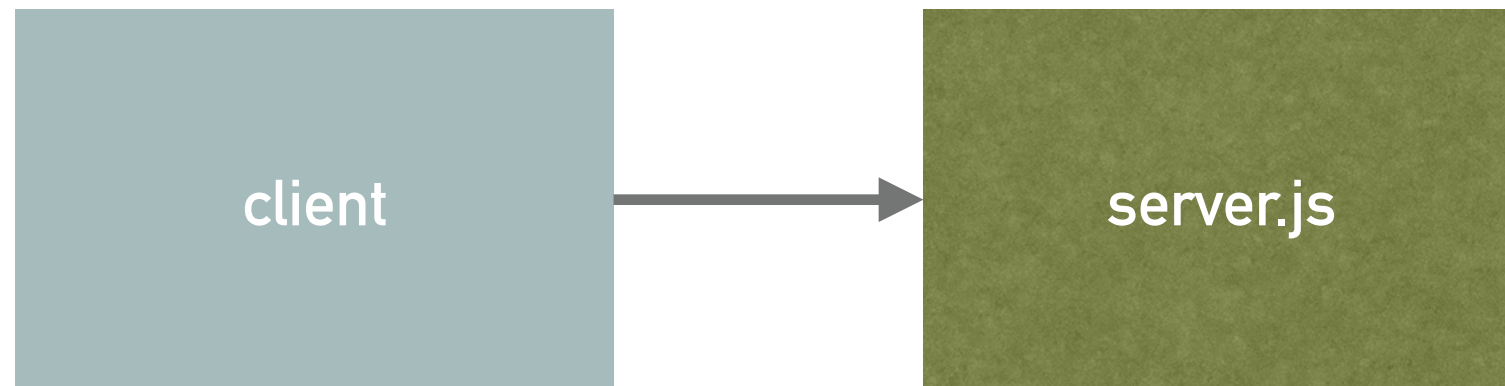
function onRequest(request, response) {
  response.writeHead(200, {
    "Content-Type": "text/plain"
  });
  response.write("Hello World");
  response.end();
}
http.createServer(onRequest).listen(8888);

console.log("Server sedang berjalan di localhost:8888");
```

SERVER PART 1: HELLO HTTP



SERVER PART 1: HELLO HTTP



SERVER PART 2: SERVER SEBAGAI MODULE

- Script server yang sudah dibuat, akan kita gunakan sebagai module
- Fungsinya, hanya sebagai server yang mengembalikan respond setiap mendapatkan request
- Script server ini kemudian menjadi module bagi aplikasi utama kita, yang akan kita beri nama index.js
- server.js dan index.js akan disimpan dalam direktori yang sama

SERVER PART 2: SERVER SEBAGAI MODULE

```
var http = require("http");

function start() {
  function onRequest(request, response) {
    response.writeHead(200, {
      "Content-Type": "text/plain"
    });
    response.write("Hello World");
    response.end();
  }
  http.createServer(onRequest).listen(8888);
  console.log("Server sedang berjalan di localhost:8888");
}

exports.start = start;
```

SERVER PART 2: SERVER SEBAGAI MODULE

- Untuk bisa digunakan sebagai module, fungsi utama dari server kita simpan sebagai start()
- start() lalu diekspor dengan menggunakan exports dan selanjutnya akan dikenal sebagai start

```
var http = require("http");

function start() {
  function onRequest(request, response) {
    ...
  }
  http.createServer(onRequest).listen(8888);
  console.log("Server sedang berjalan di localhost:8888");
}

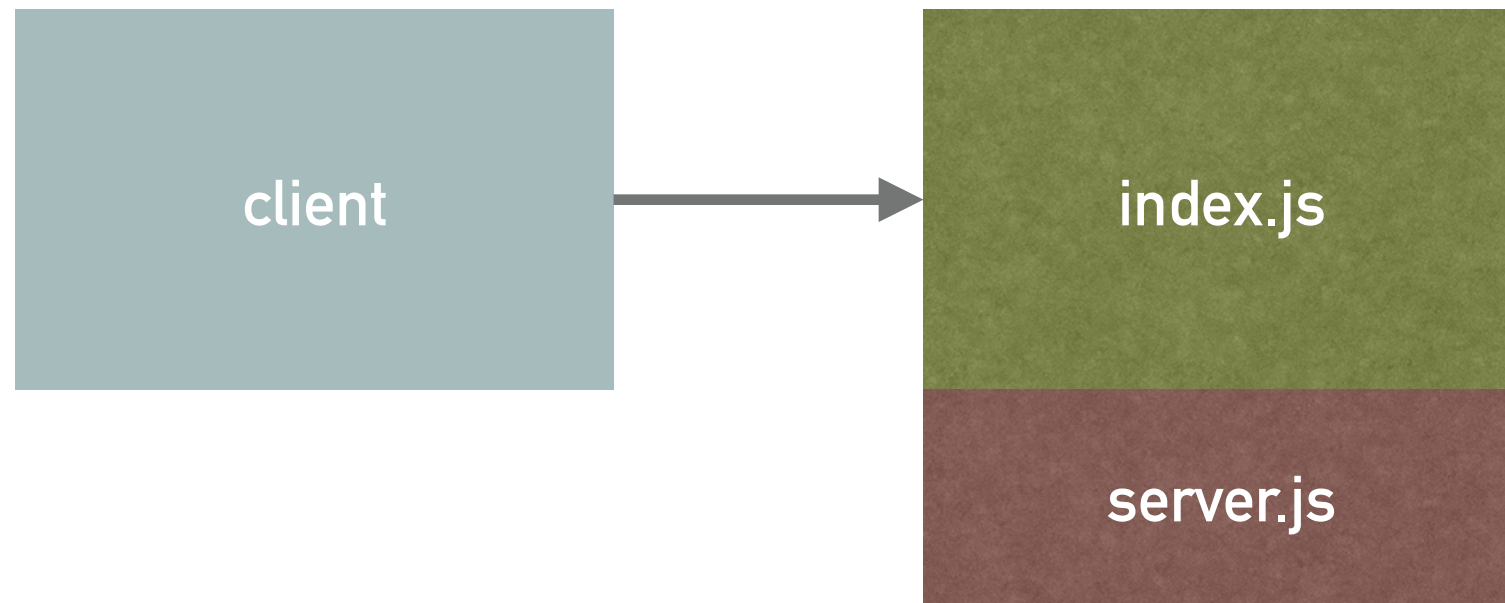
exports.start = start;
```

SERVER PART 2: SERVER SEBAGAI MODULE

- Di dalam index.js, server yang dibuat cukup digunakan dengan `server.start()`;
- Server selanjutnya dijalankan dengan `node index.js`

```
var server = require("./server");  
server.start();
```


SERVER PART 2: SERVER SEBAGAI MODULE



SERVER PART 3: ME-ROUTE REQUEST

- Sebagai server, tentu saja kita perlu melakukan routing
- Setiap request yang masuk, akan diarahkan ke route tertentu
- Route ini kemudian akan menunjukkan path yang dituju oleh request
- Router akan kita simpan dalam module bernama router yang disimpan di file `router.js`

SERVER PART 3: ME-ROUTE REQUEST

- Router ini akan menunjukkan path yang dituju oleh URL request
- Untuk melakukannya, maka fungsi route menggunakan argumen bernama pathname

```
//router.js
```

```
function route(pathname) {  
    console.log("Meroute request ke " + pathname);  
}  
exports.route = route;
```

SERVER PART 3: ME-ROUTE REQUEST

```
var http = require("http");
var url = require("url");

function start(route) {
  function onRequest(request, response) {
    var pathname = url.parse(request.url).pathname;
    console.log("Mendapat request ke " + pathname);

    route(pathname);

    response.writeHead(200, {
      "Content-Type": "text/plain"
    });
    response.write("Hello World");
    response.end();
  }
  http.createServer(onRequest).listen(8888);
  console.log("Server sedang berjalan di localhost:8888");
}
exports.start = start;
```

SERVER PART 3: ME-ROUTE REQUEST

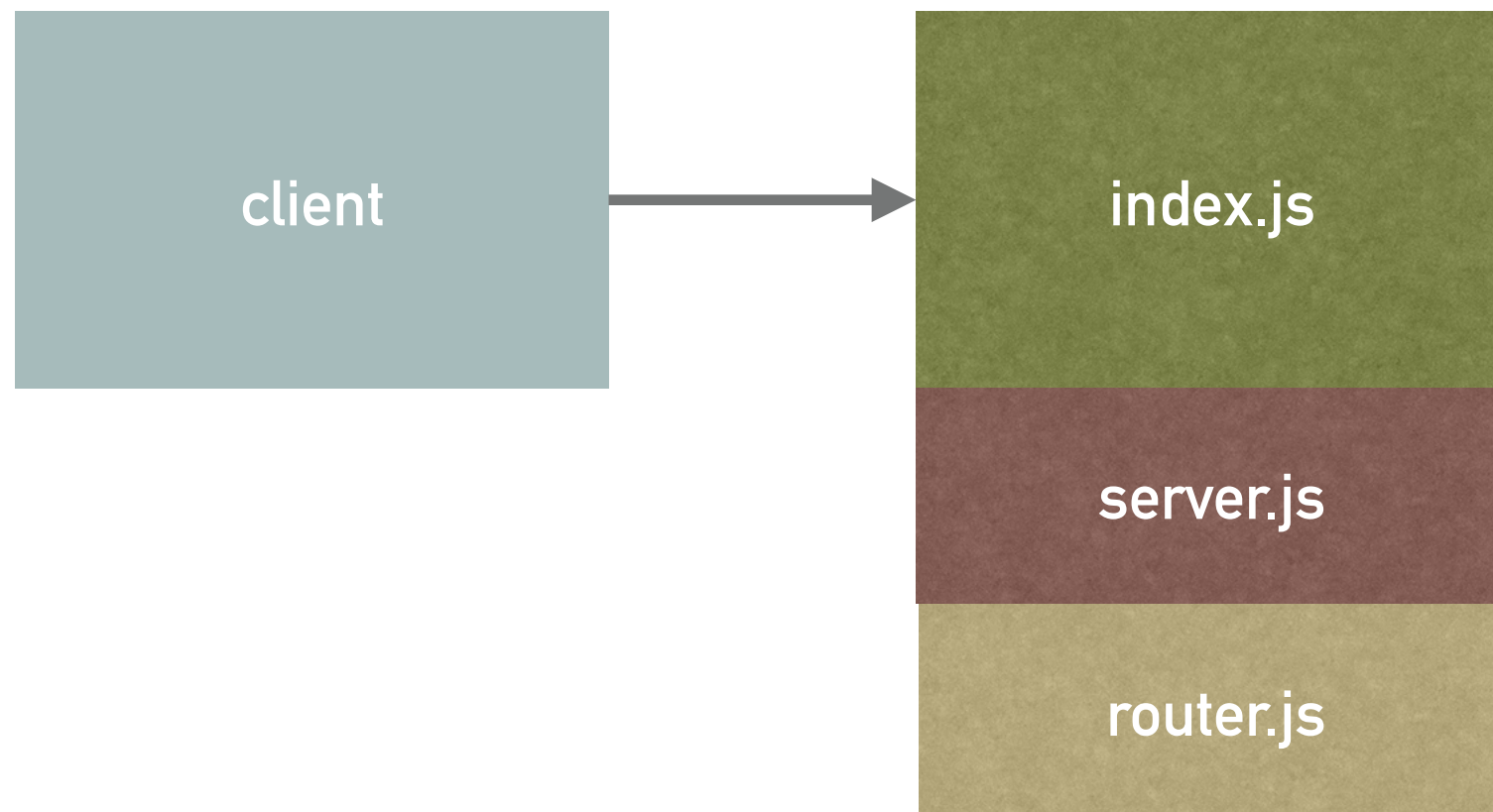
- Dalam server.js, kita gunakan juga module global url
- Module ini akan melakukan parsing terhadap url dan disimpan sebagai pathname
- Variabel pathname ini kemudian digunakan sebagai argumen untuk route()

SERVER PART 3: ME-ROUTE REQUEST

- Terakhir, dalam index.js, kita panggil module router
- Object route milik router dijadikan argumen bagi fungsi start milik server

```
var server = require("./server");  
var router = require("./router");  
  
server.start(router.route);
```

SERVER PART 3: ME-ROUTE REQUEST

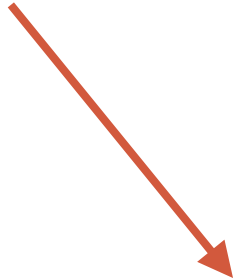


SERVER PART 3: ME-ROUTE REQUEST

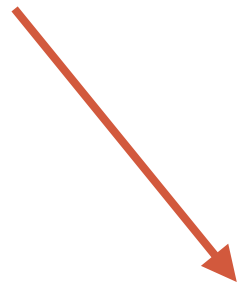
.....

```
//index.js  
var server = require("./server");  
var router = require("./router");
```

```
server.start(router.route);
```



```
//server.js  
function start(route) {  
    function onRequest(request, response) {  
        var pathname = url.parse(request.url).pathname;  
        ...  
        route(pathname);  
        ...  
    }  
    ...  
}
```



```
//router.js  
function route(pathname) {  
    ...  
}
```


SERVER PART 4: MENG-HANDLE REQUEST

- Sebelumnya, kita hanya meneruskan route saja
- Kali ini kita akan menghandle setiap request, sehingga request tertentu akan otomatis melakukan fungsi yang berkaitan
- Ini dilakukan oleh module yang kita beri nama requestHandlers
- Memungkinkan arsitektur yang bersih
 - router hanya melakukan routing
 - requestHandlers menerima route dan melakukan fungsi berkaitan

SERVER PART 4: MENG-HANDLE REQUEST

- Kita membuat 2 fungsi, yang akan dipanggil oleh route yang berkaitan

```
// requestHandlers.js

function start() {
  console.log("Memanggil request handler 'start'.");
}

function upload() {
  console.log("Memanggil Request handler 'upload'.");
}

exports.start = start;
exports.upload = upload;
```

SERVER PART 4: MENG-HANDLE REQUEST

- Dalam router.js, request handler dipanggil dengan format `handle[pathname]()`;
- Dicek juga apakah request handler yang dipanggil tersedia

```
//router.js
```

```
function route(handle, pathname) {  
  console.log("Meroute request ke " + pathname);  
  if (typeof handle[pathname] === 'function') {  
    handle[pathname]();  
  } else {  
    console.log("Tidak ada request handler ke " + pathname);  
  }  
}  
exports.route = route;
```

SERVER PART 4: MENG-HANDLE REQUEST

```
//server.js
```

```
var http = require("http");  
var url = require("url");
```

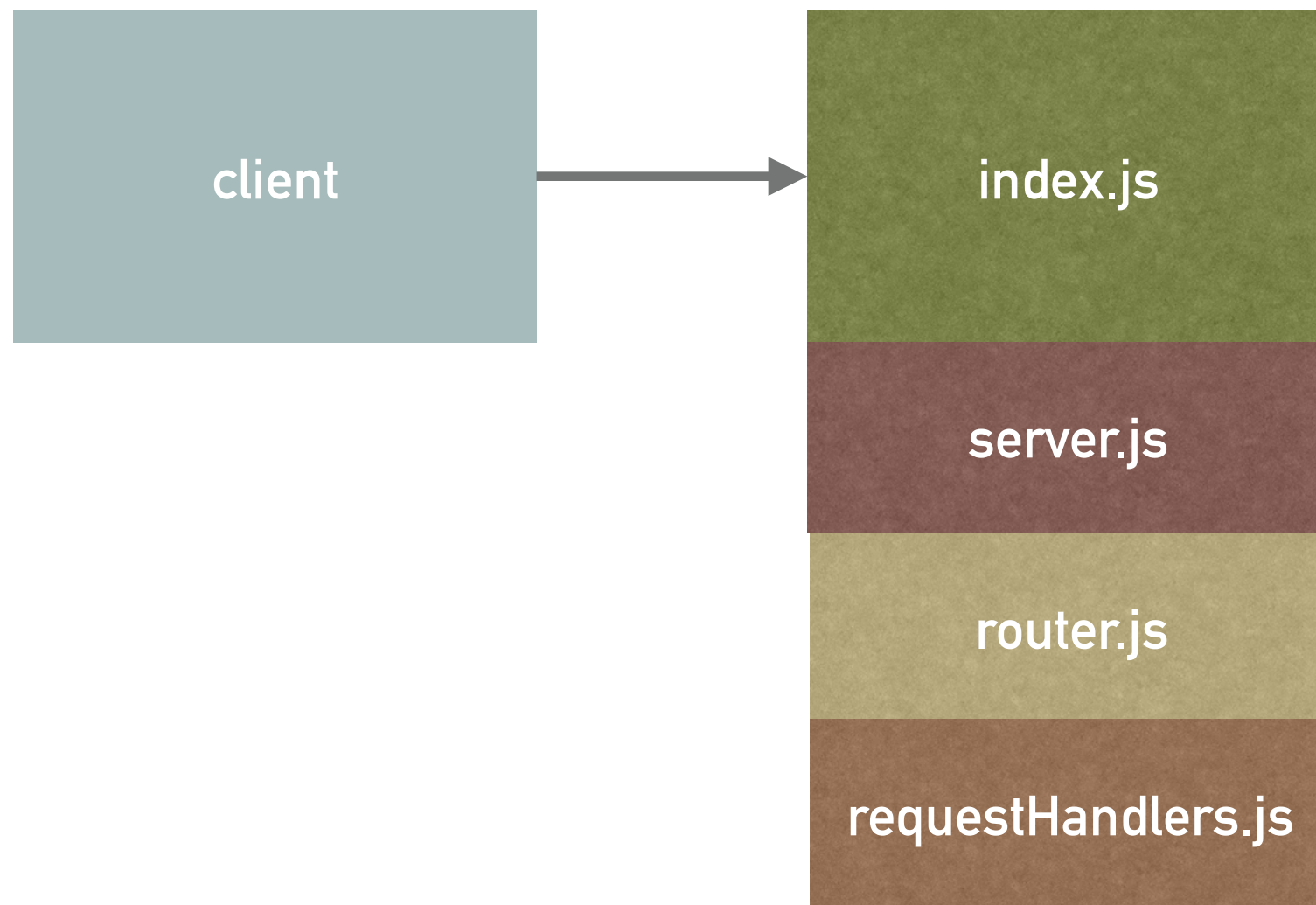
```
function start(route, handle) {  
    function onRequest(request, response) {  
        var pathname = url.parse(request.url).pathname;  
        console.log("Mendapat request ke " + pathname);  
        route(handle, pathname);  
  
        response.writeHead(200, {  
            "Content-Type": "text/plain"  
        });  
        response.write("Hello World");  
        response.end();  
    }  
    http.createServer(onRequest).listen(8888);  
    console.log("Server sedang berjalan di localhost:8888");  
}  
exports.start = start;
```

SERVER PART 4: MENG-HANDLE REQUEST

- Dalam index.js, kita inisiasi objek handle
- Objek ini akan dioper ke dalam fungsi-fungsi lain
- Ingat bahwa fungsi adalah object dan juga bisa digunakan sebagai argumen fungsi lain

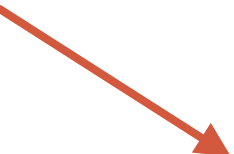
```
var server = require("./server");  
var router = require("./router");  
var requestHandlers = require("./requestHandlers");  
  
var handle = {};  
handle["/"] = requestHandlers.start;  
handle["/start"] = requestHandlers.start;  
handle["/upload"] = requestHandlers.upload;  
  
server.start(router.route, handle);
```

SERVER PART 4: MENG-HANDLE REQUEST




SERVER PART 4: MENG-HANDLE REQUEST

```
//index.js  
handle = {};  
...  
server.start(router.route, handle);
```



```
//server.js  
function start(route, handle) {  
    function onRequest(request, response) {  
        route(handle, pathname);  
    }  
}
```



```
//router.js  
function route(handle, pathname) {  
    if (typeof handle[pathname] === 'function')  
        handle[pathname]();  
}
```



```
function start() {  
    ...  
}
```

misal: memanggil start()

SERVER PART 5: MELAKUKAN POST REQUEST

- Untuk melengkapi server, kita lakukan operasi POST sederhana
- Kita akan menampilkan sebuah form, user bisa menulis di situ, dan setelah disubmit, hasilnya akan langsung ditampilkan
- Kita menunjukkan bagaimana event loop bekerja dalam Node
- Untuk menyederhanakan, view logic dilakukan dalam requestHandlers
 - Ia bertugas menampilkan elemen HTML yang sesuai
- Tidak ada perubahan dalam index.js

SERVER PART 5: MELAKUKAN POST REQUEST

- Dalam requestHandlers.js, function start() dan upload() akan diekspansi
- start() akan menampilkan elemen HTML berupa form dan tombol submit
- upload() akan menampilkan isi dari form tersebut
- fungsi upload() akan ditrigger oleh tombol submit
- module requestHandlers menggunakan modul global querystring, yang akan melakukan parsing terhadap isi form, sehingga yang ditampilkan berupa plain text

SERVER PART 5: MELAKUKAN POST REQUEST

```
var querystring = require("querystring");

function start(response, postData) {
  console.log("Memanggil Request handler 'start'.");

  var body = '<html>' +
    '<head>' +
    '<meta http-equiv="Content-Type" content="text/html; ' +
    'charset=UTF-8" />' +
    '</head>' +
    '<body>' +
    '<form action="/upload" method="post">' +
    '<textarea name="text" rows="20" cols="60"></textarea>' +
    '<input type="submit" value="Submit text" />' +
    '</form>' +
    '</body>' +
    '</html>';

  response.writeHead(200, { "Content-Type": "text/html" });
  response.write(body);
  response.end();
}
```

SERVER PART 5: MELAKUKAN POST REQUEST

```
function upload(response, postData) {  
  console.log("Memanggil Request handler 'upload'.");  
  response.writeHead(200, {  
    "Content-Type": "text/plain"  
  });  
  response.write("You've sent: " +  
    querystring.parse(postData).text);  
  response.end();  
}
```

SERVER PART 5: MELAKUKAN POST REQUEST

- Dalam router, argumen postData ditambahkan ke route()
- Pesan 404 ditampilkan dalam route, karena ia yang mengerti apakah sebuah route tersedia atau tidak

```
function route(handle, pathname, response, postData) {
  console.log("Meroute request ke " + pathname);
  if (typeof handle[pathname] === 'function') {
    handle[pathname](response, postData);
  } else {
    console.log("Tidak ada request handler ke " + pathname);
    response.writeHead(404, {
      "Content-Type": "text/plain"
    });
    response.write("404 Not found");
    response.end();
  }
}
exports.route = route;
```

SERVER PART 5: MELAKUKAN POST REQUEST

- Dalam server, kita mengimplementasikan `addListener()`
- Sebuah event input binding, yang akan mendengarkan event loop dan mentrigger sebuah event setiap ada perubahan pada input yang di-binding
 - Dalam hal ini, ia mengawasi perubahan pada data yang berada di tombol submit
 - Jika tombol ditekan, maka ia akan menjalankan callback function
 - Setelah selesai, ia memanggil fungsi `route()`

SERVER PART 5: MELAKUKAN POST REQUEST

```
var http = require("http");
var url = require("url");

function start(route, handle) {
  function onRequest(request, response) {
    var postData = "";
    var pathname = url.parse(request.url).pathname;
    console.log("Mendapat request ke " + pathname);
    request.setEncoding("utf8");
    request.addListener("data", function(postDataChunk) {
      postData += postDataChunk;
      console.log("Mendapat POST data chunk '" + postDataChunk +
        "'");
    });
    request.addListener("end", function() {
      route(handle, pathname, response, postData);
    });
  }
  http.createServer(onRequest).listen(8888);
  console.log("Server sedang berjalan di localhost:8888");
}
exports.start = start;
```

SERVER PART 6: MENAMPILKAN FILE HTML

- Memanggil HTML satu per satu seperti di requestHandlers saat ini, nampak menyulitkan dan tidak intuitif
- Sebagai alternatif, kita bisa menggunakan module fs untuk membaca dan menampilkan file yang ada dalam OS
- Tidak ada perubahan di file lain, hanya di requestHandlers.js
- Sebelumnya, kita buat dulu file index.html yang akan ditampilkan

SERVER PART 6: MENAMPILKAN FILE HTML

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
  </head>
  <body>
    <form action="/upload" method="post">
      <textarea name="text" rows="20" cols="60"></textarea>
      <input type="submit" value="Submit text" />
    </form>
  </body>
</html>
```


SERVER PART 6: MENAMPILKAN FILE HTML

- Dalam requestHandlers, kita tambahkan modul fs
- createReadStream() akan membaca isi dari file index.html
- pipe(response) akan mengirimkan isi file tersebut sebagai object response

```
var querystring = require("querystring");
var fs = require("fs");

function start(response, postData) {
  console.log("Memanggil Request handler 'start'.");

  response.writeHead(200, {
    "Content-Type": "text/html"
  });
  fs.createReadStream("./index.html").pipe(response);
}
```

LATIHAN

- Kembangkan aplikasi kita untuk menampilkan ibukota propinsi di Indonesia
- Jika diketikkan nama propinsi, maka server akan merespon dengan pesan “Ibukota Propinsi x adalah y”
- Buat modul baru yang akan memetakan propinsi dengan ibukota-nya
- Silakan lakukan berpasangan dengan peserta di sebelahnya

SUMMARY

- Sistem module dalam Node
- Node sebagai server menggunakan module HTTP
- Membuat arsitektur sederhana aplikasi Node

UNTUK DICoba DI RUMAH

- Buat agar kita bisa mengirimkan email via form di aplikasi kita
- Gunakan package nodemailer (<https://www.npmjs.com/package/nodemailer>)
- Kalau sudah berhasil jalan, verifikasi program dengan mengirim email ke didit@froyo.co.id (please, no spam :D)