Node.JS
Библиотеки и фреймворки

к.т.н. доцент каф. ВТ
Медведев М.С.

Node.JS

Представления и движок представлений Handlebars

Как правило, приложения Express для определения визуального интерфейса использует не стандартные файлы html, а специальные сущности - *представления*, из которых затем создаются html-файлы.

Преимуществом *представлений* является то, что мы можем определять в них некоторые шаблоны, вместо которых затем вставляется какое-то динамическое содержимое с помощью кода javascript.

Node.JS

Управляет представлениями специальный компонент - движок представлений (*view engine*), который также называют движок шаблонов (*template engine*).

Вообще движков представлений в Express довольно много: *Pug, Jade, Dust, Nunjucks, EJS, Handlebars* и другие.

Node.JS

Управляет представлениями специальный компонент - движок представлений (*view engine*), который также называют движок шаблонов (*template engine*).

Вообще движков представлений в Express довольно много: *Pug, Jade, Dust, Nunjucks, EJS, Handlebars* и другие.

Вопрос выбора движка представлений - в основном вопрос предпочтений, все они предоставляют схожую функциональность, различаясь лишь в каких-то деталях.

Node.JS

Для работы с движками представлений в Express определено ряд глобальных настроек, которые мы можем установить.

Прежде всего это настройка *view engine*, которая устанавливает используемый движок предствлений, и views, которая устанавливает путь к папке с представлениями внутри проекта (если этот параметр не установлен, то по умолчанию используется папка с именем *views*).

Node.JS

Для работы с представлениями установим пакет hbs в проект с помощью команды:

```
npm install hbs –save
```

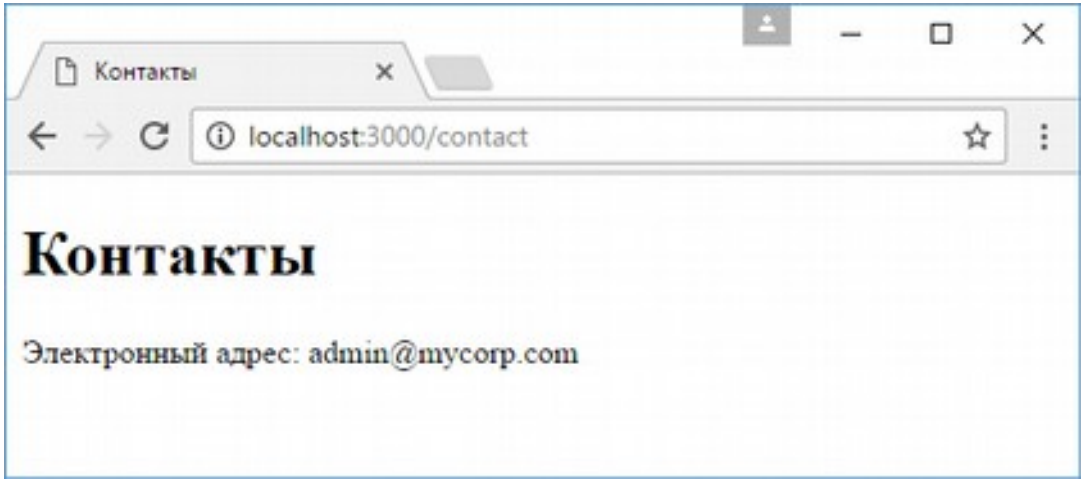
Для хранения представлений определим в проекте папку views. Затем в нее добавим новый файл contact.hbs.

Node.JS

```
const express = require("express");
const app = express();
app.set("view engine", "hbs");
app.use("/contact", function(request,
response){
    response.render("contact.hbs");
});
app.use("/", function(request, response){
    response.send("Главная страница");
});
app.listen(3000);
```

Node.JS

Для маршрута "/contact" используется функция обработчика, которая производит рендеринг представления "contact.hbs" с помощью функции response.render(). Эта функция на основе представления создает страницу html, которая отправляется клиенту.



Node.JS

Модель представления

Одним из преимуществ шаблонов является то, что мы можем передавать в представления на место шаблонов модели представления - специальные объекты, данные которые использует движок представлений для рендеринга.

Node.JS

```
app.use("/contact", function(request,
response){

    response.render("contact.hbs", {
        title: "Мои контакты",
        email: "test@sfu-kras.ru",
        phone: "+1234567890"
    });
});
```

Теперь в качестве второго параметра в функцию response.render() передается специальный объект с тремя свойствами.

Node.JS

```
<!DOCTYPE html>
<html>
<head>
    <title>{{title}}</title>
    <meta charset="utf-8" />
</head>
<body>
    <h1>{{title}}</h1>
    <p>Электронный адрес: {{email}}</p>
    <p>Телефон: {{phone}}</p>
</body>
<html>
```

Node.JS

Вместо конкретных данных в коде представления используются те данные, которые определены в модели.

Чтобы обратиться к свойствам модели в двойных фигурных скобках указывается нужное свойство: `{{title}}`.

При рендеринге представления вместо подобных выражений будут вставляться значения соответствующих свойств модели.

Node.JS

В представление передается массив:

```
app.use("/contact", function(request,
response){
    response.render("contact.hbs", {
        title: "Мои контакты",
        emailsVisible: true,
        emails: ["test@sfu-kras.ru",
"Mmedvedev@sfu-kras.ru"],
        phone: "+1234567890"
    });
});
```

Node.JS

Для вывода данных изменим представление contact.hbs:

```
{{#if emailsVisible}}
        <h3>Электронные адреса</h3>
        <ul>
            {{#each emails}}
                <li>{{this}}</li>
            {{/each}}
        </ul>
{{/if}}
```

Node.JS

Конструкция `#each` перебирает все элементы из массива .

Текущий перебираемый элемент помещается в переменную this.

Node.JS

Частичные представления

Нередко веб-страницы в приложении используют какие-то общие элементы. Это может быть меню, шапка сайта, футер, другие элементы.

Однако, если потребуется поменять этот общий элемент, то придется вносить изменения на все веб-страницы, которые используют этот элемент.

И было бы гораздо проще определить этот элемент в одном месте, а затем подключать на все страницы.

Node.JS

Решить эту проблему помогают частичные представления (partial views), которые представляют разделяемые общие элементы, которые можно добавлять на обычные представления.

К примеру, мы можем сделать общее меню и общий футер.

Для этого создадим для частичных представлений в проекте подкаталог views/partials.

Node.JS

Новый файл menu.hbs:

```
<nav><a href="/">Главная</a> | <a
href="/contact">Контакты</a></nav>
```

Добавим в views/partials новый файл footer.hbs:

```
<footer><p>MyCorp - Copyright © 2019</p></
footer>
```

Node.JS

Для вставки частичного представления применяется выражение `{{> menu}}`, в котором прописывается имя файла частичного представления без расширения.

Также добавим в папку views новое представление, которое назовем home.hbs:

```
<body>
    {{> menu}}
    <h1>Главная страница</h1>
    {{> footer}}
</body>
```

Node.JS

Таким образом, у нас два представления, которые имеют общие элементы.

Если потребуется добавить какой-нибудь новый пункт меню, то достаточно изменить файл menu.hbs.

Node.JS

В итоге весь проект будет выглядеть следующим образом:

```
app.js
node_modules
/views
     contact.hbs
     home.hbs
     /partials
          footer.hbs
          menu.hbs
```

Node.JS

Для настройки функционала частичных представлений в коде используется вызов:

```
hbs.registerPartials(__dirname +
"/views/partials");
```

который устанавливает каталог, где располагаются частичные представления.

Layout в Handlebars

Файл layout или мастер-страница позволяет определить общий макет всех веб-страниц сайта.

Благодаря чему гораздо проще обновлять сайт, определять и менять какие-то общие блоки кода.

Node.JS

Для работы с файлами layout установим в проект модуль express-handlebars с помощью следующей команды

```
npm install express-handlebars
```

Пусть в проекте в папке views/partials будут определены два частичных представления для меню и футера.

Node.JS

Создадим в проекте в папке views новый каталог layouts и определим в нем файл layout.hbs, который будет определять макет сайта:

```
<body>
    {{> menu}}

    {{{body}}}

    {{> footer}}
</body>
```

Node.JS

Здесь внедряются частичные представления menu.hbs и footer.hbs. И, кроме того, здесь также присуствует такое выражение, как `{{{body}}}`.

Вместо этого выражения будет вставлять содержимое конкретных представлений.

Затем в папке views определим два обычных представления: представление *contact.hbs* и представление *home.hbs*

Node.JS

Для использования файлов layout необходимо настроить движок hbs:

```
app.engine("hbs", expressHbs(
    {
        layoutsDir: "views/layouts",
        defaultLayout: "layout",
        extname: "hbs"
    }
))
```

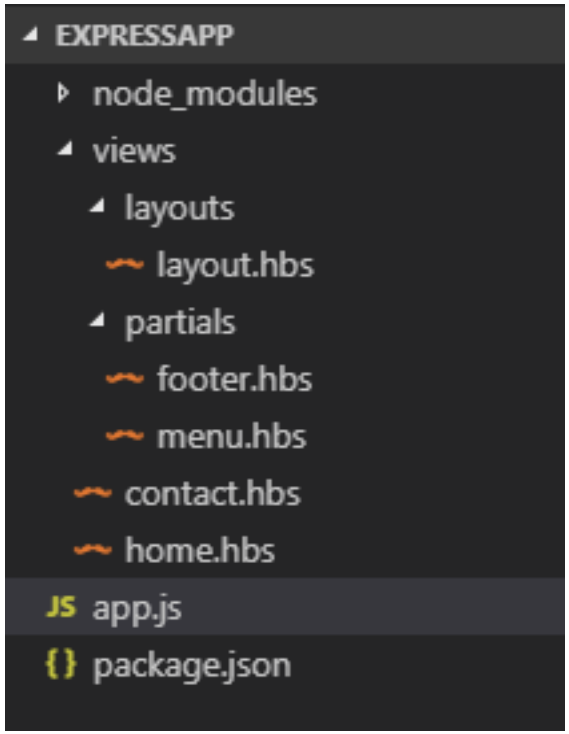
Node.JS

Функция `expressHbs` осуществляет конфигурацию движка. В частности, свойство `layoutsDir` задает путь к папке с файлами `layout` относительно корня каталога проекта.

Свойство `defaultLayout` указывает на название файла, который будет использоваться в качестве мастер-страницы. В нашем случае это файл layout.hbs, поэтому указываем название этого файла без расширения.
Третье свойство - `extname` задает расширение файлов.

Node.JS

В итоге весь проект будет выглядеть следующим образом:



Хелперы в Handlebars

Хелперы фактически представляют функции, которые возвращают некоторую строку. После получения эту строку можно добавить в любое место представления.

Строка может представлять собой в том числе и код html.

Node.JS

Хелперы позволяют оптимизировать создание кода представлений.

В частности, мы можем один раз определить функцию хелпера, а затем многократно применять ее в самых различных местах для генерации кода.

Хелпер определяется с помощью функции `hbs.registerHelper()`.
Первый параметр функции - название хелпера, а второй - функция, которая возвращает строку.

Node.JS

Далее определим представление home.hbs:

```
<html>
<head>
    <title>Главная страница</title>
    <meta charset="utf-8" />
</head>
<body>
    <h1>Главная страница</h1>
    <div>{{getTime}}</div>
</body>
<html>
```

Node.JS

Хелпер может возвращать не просто строку, но и код html.

Кроме того, хелперу можно передавать параметры, которые применяются при генерации результата.

Node.JS

```
hbs.registerHelper("createStringList",
function(array){

    var result="";
    for(var i=0; i<array.length; i++){
        result +="<li>" + array[i] +
"</li>";
    }
    return new hbs.SafeString("<ul>" +
result + "</ul>");
});
```

Node.JS

Добавлено определение хелпера `createStringList()`, который в качестве параметра принимает некоторый массив строк и из них создает элемент "<ul>".

Однако чтобы возвращаемое значение расценивалось именно как html, его надо обернуть в функцию `hbs.SafeString()`.

Node.JS

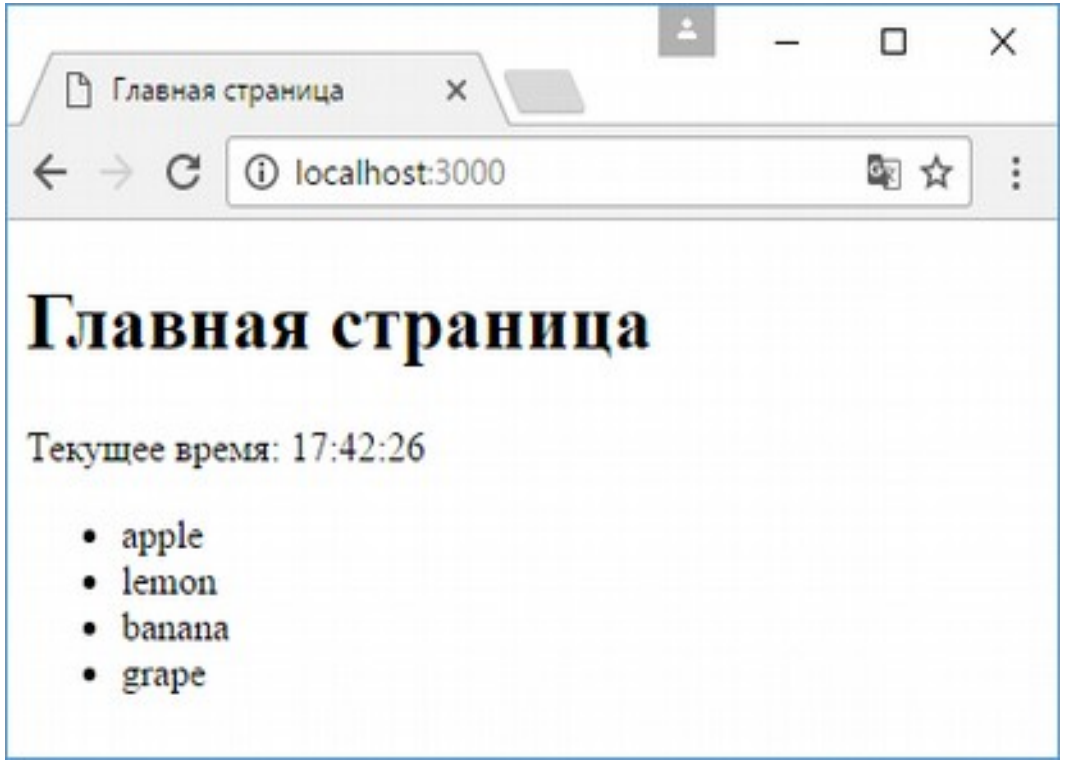
```
app.get("/", function(request, response){

    response.render("home.hbs", {
        fruit: [ "apple", "lemon",
"banana", "grape"]
    });
});
```

Node.JS

Изменим файл представления:

```
<body>
    <h1>Главная страница</h1>
    <div>{{getTime}}</div>
    <div>{{createStringList fruit}}</div>
</body>
```

Node.JS


Главная страница
localhost:3000
Главная страница
Текущее время: 17:42:26
apple
lemon
banana
grape

Node.JS

В качестве альтернативы Handlebars вкратце рассмотрим применение другого движка представлений - EJS.

Этот движок использует синтаксис, который во многом был заимствован из движка представлений Web Forms на платформе ASP.NET, который разрабатывался в компании Microsoft.

Node.JS

```
npm install ejs --save
```

```
app.set("view engine", "ejs");
```

Node.JS

Pug представляет еще один движок представлений.

Используя его можно сократить определения html-разметки.

Достаточно поместить на строке название html-элемента и затем можно определять его содержимое. Содержание элементов определяется отступами.

Node.JS

Используя Express и Node.js, мы можем реализовать полноценный API в стиле REST для взаимодействия с пользователем.
Архитектура REST (Representational State Transfer — «передача состояния представления») предполагает применение следующих методов или типов запросов HTTP для взаимодействия с сервером:
GET
POST
PUT
DELETE

Node.JS

REST-стиль особенно удобен при создании Single Page Application, которые нередко используют специальные javascript-фреймворки типа Angular, React или Knockout.

Создадим новый проект, определим в проекте файл package.json:

```
{
  "name": "webapp",
  "version": "1.0.0",
  "dependencies": {
    "body-parser": "^1.16.0",
    "express": "^4.14.0"
  }
```

Node.JS

```
[{   "id":1,
     "name":"Tom",
     "age":24
},
{     "id":2,
     "name":"Bob",
     "age":27
},
{     "id":3,
     "name":"Alice",
     "age":"23"
}]
```

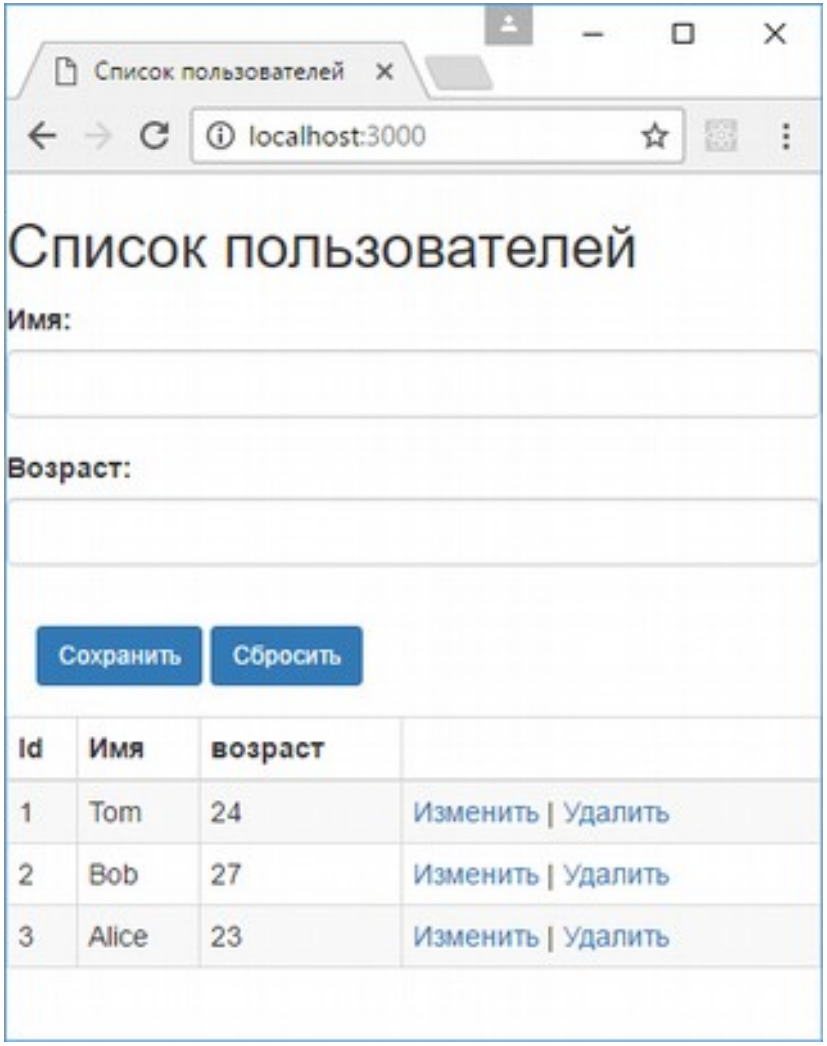
Node.JS

Для чтения и записи в этот файл мы будем использовать встроенный модуль fs.

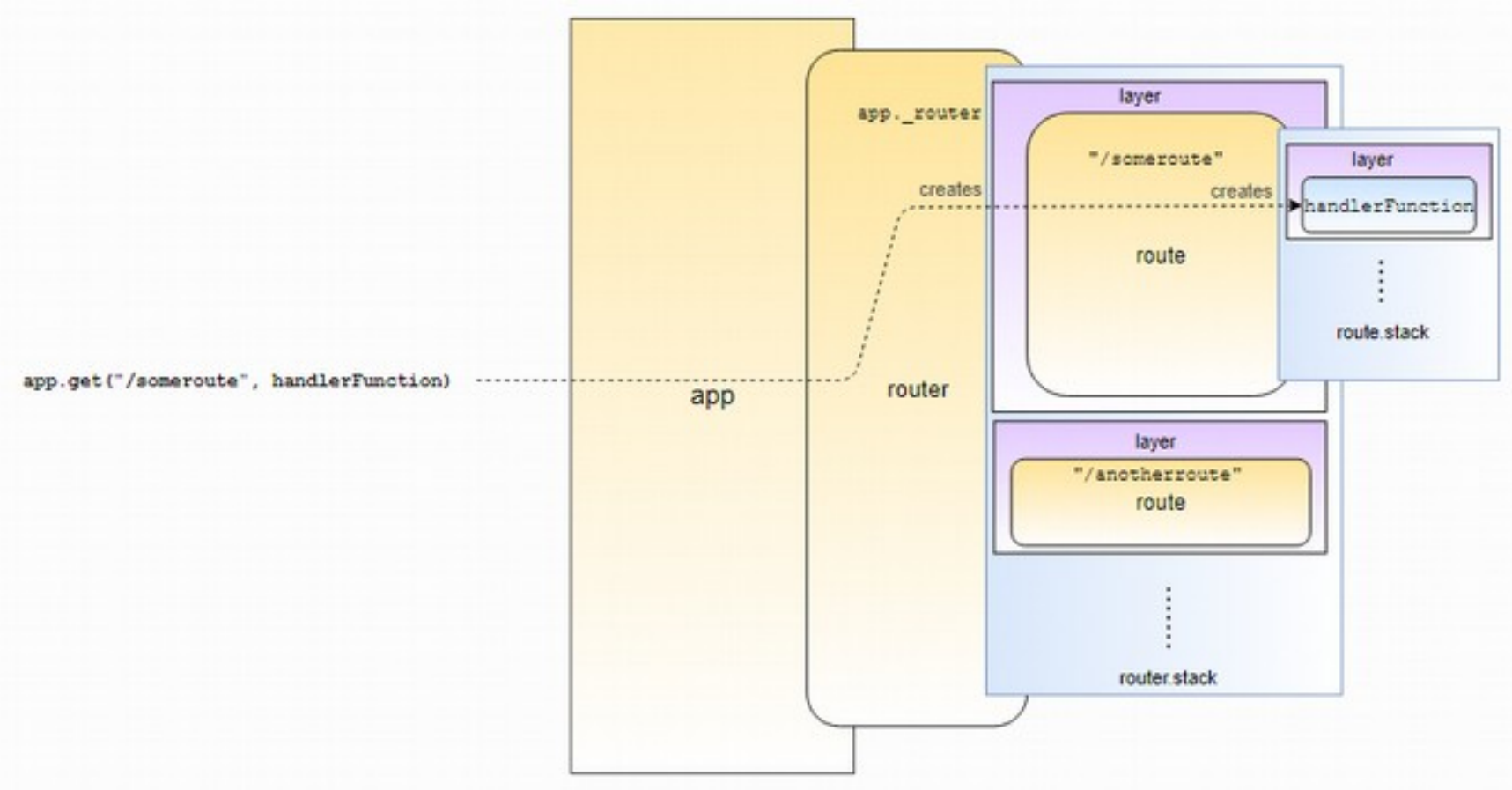
Для обработки запросов определим в проекте следующий файл app.js

Для обработки запросов определено пять методов для каждого типа запросов:
app.get()/app.post()/app.delete()/app.put()

Node.JS


Список пользователей
localhost:3000
Список пользователей
Имя:
Возраст:
Сохранить
Сбросить
Id
Имя
возраст
1
Tom
24
Изменить | Удалить
2
Bob
27
Изменить | Удалить
3
Alice
23
Изменить | Удалить

Node.JS


app.get("/someroute", handlerFunction)
app
router
app._router
creates
layer
"/someroute"
route
creates
layer
handlerFunction
route.stack
layer
"/anotherroute"
route
router.stack

Node.JS

https://habr.com/ru/company/ruvds/blog/414079/

https://metanit.com/web/nodejs/4.10.php