Desenv. BackEnd

Prof. M. Marcelo Petri

Olá! Sejam bem vindos

DISCIPLINA/UNIDADE CURRICULAR	Desenvolvimento Web
TEMA DA AULA	Programação do servidor Node.js Express
INDICADOR	1. Configura o ambiente de desenvolvimento conforme as funcionalidades e características do projeto. 2. Cria estruturas de código utilizando linguagem de programação para back-end, de acordo com os requisitos do projeto de software.
CRONOGRAMA	- Arquitetura de aplicações web: conceito de back-end e sua interação com front-end. - Ambiente de desenvolvimento: conceito, interface, configuração e compilação. - Linguagem de programação para back-end: sintaxe, palavras reservadas, variáveis e estruturas de dados; desvios condicionais e laços de repetição. - Biblioteca da linguagem: funções próprias da linguagem e suas aplicações. - Orientação a objetos na linguagem de programação back-end: classes, objetos, interfaces, entre outros
ATIVIDADE	Desenvolvimento um servidor que receba por parâmetros de URL dois dados e grave em um TXT junto com o professor e exercícios.

Objetivos

- Noções básicas de NodeJS Express
- Instalar pacotes NPM
- Exercícios

Introdução

- O Express.js, ou simplesmente Express, é um framework popular que roda em Node.js e é usado para escrever servidores HTTP que lidam com solicitações de clientes de aplicativos web.

- O Express oferece suporte a diversas maneiras de ler parâmetros enviados por HTTP.

Script inicial do servidor

- Simular um aplicativo que solicita algumas informações ao servidor.
  - Fornece uma função `echo`, que simplesmente retorna a mensagem enviada pelo cliente.
  - Informa ao cliente seu endereço IP mediante solicitação.
  - Usa cookies para identificar clientes conhecidos.

Script inicial do servidor

- Criar o arquivo JavaScript que funcionará como servidor. Usando npm, crie um diretório chamado myserver com o arquivo JavaScript:

```
$ mkdir myserver
$ cd myserver/
$ npm init
```

Script inicial do servidor

- Criar um arquivo index.js básico que será usado como ponto de entrada para o nosso servidor (para executar):

```
const express = require('express')
const app = express()
const host = "localhost"
const port = 8080
app.get('/', (req, res) => {
  res.send('Request received')
});
app.listen(port, host, () => {
  console.log(`Server ready at http://${host}:${port}`)
});
```

Script inicial do servidor

- comando curl na linha de comando para fazer uma solicitação HTTP ao servidor recém-implantado:

```
$ curl http://localhost:8080 -v
```

- Além de exibir cabeçalhos de conexão HTTP, o comando curl auxilia no desenvolvimento de aplicativos, permitindo enviar dados para o servidor usando diferentes métodos HTTP e em diferentes formatos.

- Essa flexibilidade torna mais fácil depurar quaisquer problemas e implementar novos recursos no servidor .

Rotas

- As solicitações que o cliente pode fazer ao servidor dependem de quais rotas foram definidas no arquivo index.js.
- Uma rota especifica um método HTTP e define um caminho (mais precisamente, um URI) que pode ser solicitado pelo cliente.
- Até aqui, o servidor tem apenas uma rota configurada:

```
app.get('/', (req, res) => {
  res.send('Request received')
});
```

Método POST

- Para estender a funcionalidade de nosso servidor de teste, vamos ver como definir uma rota para o método HTTP POST.
- Ele é usado pelos clientes que precisam enviar ao servidor dados extras, além dos que estão incluídos no cabeçalho da solicitação.
- A opção `--data` do comando `curl` invoca automaticamente o método POST e inclui o conteúdo que será enviado ao servidor via POST. A linha POST / HTTP/1.1 na saída a seguir mostra que o método POST foi empregado.
- No entanto, nosso servidor definiu apenas um método GET, por isso ocorre um erro quando usamos curl para enviar uma solicitação via POST:

```
$ curl http://myserver:8080/echo --data
message="This is the POST request body"
```

Método POST

- Escrevendo um programa Express capaz de tratar essa solicitação com sucesso. O Express não reconhece esse formato `application/x-www-form-urlencoded` por padrão, por isso precisamos usar o módulo express.urlencoded.

```
const express = require('express')
const app = express()
const host = "localhost"
const port = 8080
app.use(express.urlencoded({ extended: true }))
```

Método POST

- Adicionar app.post na rota existente para atender às solicitações feitas via POST e para recuperar o corpo da solicitação:

```
app.post('/', (req, res) => {
  res.send(req.body.message)
})
```

Manipulador de caminhos e funções

- Para expandir a funcionalidade echo do servidor, podemos definir uma rota usando o método POST com o caminho /echo:

```
app.post('/echo', (req, res) => {
  res.send(req.body.message)
})
```

- A resposta obtida pelo cliente para a solicitação curl mostrada anteriormente agora é bem-sucedida:

```
$ curl http://myserver:8080/echo --data
message="This is the POST request body"
This is the POST request body
```

Manipulador de caminhos e funções

- Os dados também podem ser enviados em uma string de solicitação iniciada por um ponto de interrogação: `?message=This+is+the+message`
- Os campos usados na string de solicitação estão disponíveis para o servidor na propriedade req.query.
- Outra maneira de enviar dados através do método HTTP GET é usar os parâmetros de rota do Express:

```
app.get('/echo/:message', (req, res) => {
  res.send(req.params.message)
})
```

Manipulador de caminhos e funções

- A rota corresponde às solicitações feitas com o método GET usando o caminho /echo/:message, onde :message é um espaço reservado para qualquer termo enviado com esse rótulo pelo cliente.
- Esses parâmetros estão acessíveis na propriedade req.params.
- Com esta nova rota, a função echo do servidor pode ser solicitada de forma mais sucinta pelo cliente:

```
$ curl http://myserver:8080/echo/hello
hello
```

Ajustes à resposta

- o parâmetro `res` é responsável por retornar uma resposta ao cliente.
- Além disso, o objeto `res` pode alterar outros aspectos da resposta.
- Embora as respostas que implementamos até agora sejam apenas breves mensagens de texto puro, o cabeçalho `Content-Type` das respostas está usando `text/html; charset=utf-8`.
- Embora isso não impeça que a resposta em texto puro seja aceita, será mais correto redefinir este campo no cabeçalho da resposta como `text/plain` com a configuração `res.type('text/plain')`.

Cookies

- Outros tipos de ajustes de resposta envolvem o uso de cookies, que permitem ao servidor identificar um cliente que já fez uma solicitação anteriormente.
- Os cookies são importantes para a utilização de recursos avançados, como a criação de sessões privadas que associam solicitações a um usuário específico.
- O gerenciamento de cookies deve ser instalado:

```
$ npm install cookie-parser
```

- O gerenciamento de cookies precisa ser incluído no script do servidor perto do início do arquivo:

```
const cookieParser = require('cookie-parser')
app.use(cookieParser())
```

Cookies

- Vamos modificar a função de manipulação da rota com o caminho raiz / já existente no script.
- A função de manipulação no exemplo a seguir verifica se um cookie com o nome known existe na solicitação. Se tal cookie não existir, o servidor pressupõe que se trata de um visitante que chegou ao site pela primeira vez e envia a ele um cookie com esse nome através da chamada res.cookie('known', '1').
- Atribuímos arbitrariamente o valor 1 ao cookie porque ele precisa ter algum conteúdo, mas o servidor não consulta esse valor.

Cookies

- Este aplicativo pressupõe que a simples presença do cookie indica que o cliente já solicitou esta rota antes:

```
app.get('/', (req, res) => {
  res.type('text/plain')
  if ( req.cookies.known === undefined ){
        res.cookie('known', '1')
        res.send('Welcome, new visitor!')
  }
  else
      res.send('Welcome back, visitor');
})
```

Cookies

- Por padrão, o curl não usa cookies nas transações. Mas ele tem opções para armazenar (-c cookies.txt) e enviar cookies armazenados (-b cookies.txt):

```
$ curl http://myserver:8080/ -c cookies.txt -b
cookies.txt -v
```

- Como esse comando foi o primeiro acesso desde que os cookies foram implementados no servidor, o cliente não tinha cookies para incluir na solicitação. Como seria de se esperar, o servidor não identificou o cookie na solicitação e, portanto, incluiu o cookie nos cabeçalhos de resposta, conforme indicado na linha Set-Cookie: known=1; Path=/ da saída.

Cookies

- Como habilitamos os cookies em curl, uma nova solicitação incluirá o cookie known=1 nos cabeçalhos da solicitação, permitindo que o servidor identifique a presença do cookie:

```
$ curl http://myserver:8080/ -c cookies.txt -b
cookies.txt -v
```

Segurança dos cookies

- O desenvolvedor deve estar ciente das potenciais vulnerabilidades ao usar cookies para identificar os clientes que fazem solicitações.
- Os invasores podem usar técnicas como cross-site scripting (XSS) e cross-site request forgery (CSRF) para roubar os cookies de um cliente e, assim, personificá-lo ao fazer uma solicitação ao servidor.
- De modo geral, esses tipos de ataques usam campos de comentários não validados ou URLs construídas meticulosamente para inserir código JavaScript malicioso na página.
- Quando executado por um cliente autêntico, esse código pode copiar cookies válidos e armazená-los, ou encaminhá-los para outro destino.

Segurança dos cookies

- Portanto, especialmente em aplicativos profissionais, é importante instalar e empregar recursos mais especializados do Express, conhecidos como middleware.
- Os módulos express-session ou cookiesession permitem um controle mais completo e seguro sobre a sessão e o gerenciamento de cookies.
- Esses componentes oferecem controles extras para evitar que os cookies sejam desviados de seu emissor original.

Resumo

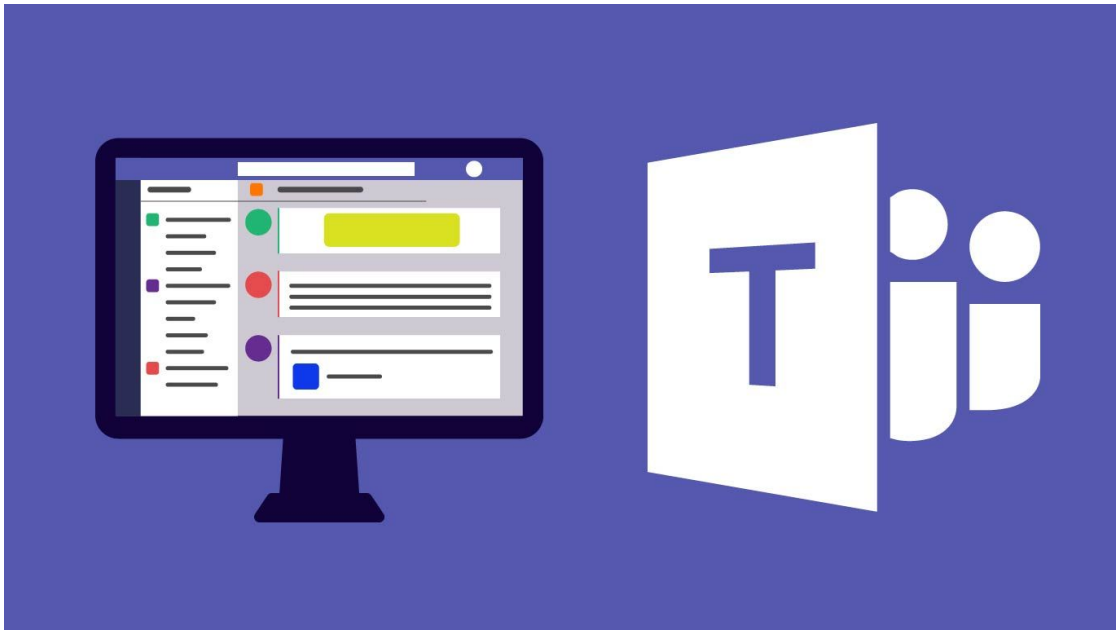
- Esta lição explica como escrever scripts do Express para receber e lidar com solicitações HTTP. O Express usa o conceito de rotas para definir os recursos disponíveis aos clientes, o que lhe dá grande flexibilidade para construir servidores para qualquer tipo de aplicativo web.
- Abordamos os seguintes conceitos e procedimentos:
  - Rotas que usam os métodos HTTP GET e HTTP POST.
  - Como os dados de formulários são armazenados no objeto request.
  - Como usar parâmetros de rota.
  - Personalização de cabeçalhos de resposta.
  - Gerenciamento básico de cookies.

Exercícios

- Escreva uma rota que use o método HTTP GET e o caminho /agent para enviar de volta ao cliente o conteúdo do cabeçalho user-agent.

Atividades

- Projeto Calculadora (Atividade 01)
- Programação do servidor NodeJS (Atividade 02)



Links

Referências

- **Básicas: [OBRIGATORIAMENTE 3 BIBLIOGRAFIAS]**
- ARAÚJO, Everton Coimbra de. Desenvolvimento para web com Java. Florianópolis: Visual Books, 2010.
- QIAN et al. Desenvolvimento Web Java. LTC, 2010.
- GOMES, Yuri Marx Pereira. Java na web com JSF, Spring, Hibernate e Netbeans 6. Rio de Janeiro, RJ: Ciência Moderna, 2008.

- **Complementar: [OBRIGATORIAMENTE 5 BIBLIOGRAFIAS]**
- GONÇALVES, Edson. Desenvolvendo aplicações web com netbeans ide 6. Rio de Janeiro, RJ: Ciência Moderna, 2008.
- LUCKOW, Décio Heinzelmann; MELO, Alexandre Altair de. Programação Java para Web. 2ª Ed. São Paulo: Novatec, 2015.
- SOUZA, Thiago Hernandes de. Java mais primefaces mais ireport: desenvolvendo um CRUD para web. Rio de Janeiro, RJ: Ciência Moderna, 2013.
- KALIN, Martin. Java Web Services: implementando. Rio de Janeiro, RJ: Alta Books, 2010.
- EIS, Diego. Guia Front-End: o caminho das pedras para ser um Dev Front-End. São Paulo: Casa do Código, 2015.
- SOUZA, Alberto. Spring MVC: domine o principal framework web Java. São Paulo: Casa do Código, 2015.