Node.js

O que é Node.js?

Node.js é uma plataforma do lado do servidor construída no motor JavaScript do Google Chrome (motor V8). O Node.js foi desenvolvido por Ryan Dahl em 2009. A definição de Node.js conforme fornecida por sua documentação oficial é a seguinte:

Node.js é uma plataforma construída no tempo de execução JavaScript do Chrome para construir facilmente aplicativos de rede rápidos e escaláveis. O Node.js usa um modelo de E / S não bloqueante e orientado por eventos que o torna leve e eficiente, perfeito para aplicativos de tempo real com muitos dados que são executados em dispositivos distribuídos.

Node.js é um ambiente de tempo de execução de plataforma cruzada de código aberto para o desenvolvimento de aplicativos do lado do servidor e de rede. Os aplicativos Node.js são escritos em JavaScript e podem ser executados no tempo de execução do Node.js no OS X, Microsoft Windows e Linux.

Node.js também fornece uma rica biblioteca de vários módulos JavaScript que simplifica o desenvolvimento de aplicativos da web usando Node.js em grande medida.

```
Node.js = Runtime Environment + JavaScript Library
```

Recursos do Node.js

A seguir estão alguns dos recursos importantes que tornam o Node.js a primeira escolha dos arquitetos de software.

- **Assíncrono e orientado a eventos** - Todas as APIs da biblioteca Node.js são assíncronas, ou seja, sem bloqueio. Basicamente, isso significa que um servidor baseado em Node.js nunca espera que uma API retorne dados. O servidor passa para a próxima API após chamá-la e um mecanismo de notificação de Eventos de Node.js ajuda o servidor a obter uma resposta da chamada de API anterior.
- **Muito rápido** - Sendo construído no motor V8 JavaScript do Google Chrome, a biblioteca Node.js é muito rápida na execução de código.
- **Single Threaded, mas altamente escalonável** - Node.js usa um modelo de thread único com loop de eventos. O mecanismo de eventos ajuda o servidor a responder de forma não bloqueadora e torna o servidor altamente escalonável, ao contrário dos servidores tradicionais que criam threads limitados para lidar com as solicitações. O Node.js usa um único programa encadeado e o mesmo programa pode fornecer serviço a um número muito maior de solicitações do que servidores tradicionais como o Apache HTTP Server.
- **Sem buffer** - os aplicativos Node.js nunca armazenam nenhum dado em buffer. Esses aplicativos simplesmente geram os dados em blocos.
- **Licença** - Node.js é lançado sob a licença MIT .

Express Framework - Visão geral

Express é uma estrutura de aplicativo web NodeJS – mínimo e flexível - que fornece um conjunto robusto de recursos para desenvolver aplicativos da web e móveis. Ele facilita o rápido desenvolvimento de aplicativos da Web baseados em Node. Observe alguns dos principais recursos da estrutura Express:

- Permite configurar middlewares para responder a solicitações HTTP.
- Define uma tabela de roteamento que é usada para realizar diferentes ações com base no método HTTP e URL.
- Permite renderizar páginas HTML dinamicamente com base na passagem de argumentos para modelos.

Antes de criar um "Hello, World!" aplicativo usando Node.js, vamos ver os componentes de um aplicativo Node.js. Um aplicativo Node.js consiste nos três componentes importantes a seguir:

- **Importar módulos necessários** - usamos a diretiva **require** para carregar módulos Node.js.
- **Criar servidor** - um servidor que ouvirá as solicitações do cliente, semelhante ao servidor HTTP Apache.
- **Solicitação de leitura e resposta de retorno** - O servidor criado em uma etapa anterior lerá a solicitação HTTP feita pelo cliente, que pode ser um navegador ou um console, e retornará a resposta.

Para criar o node server será usado o módulo *Express*

Instalando o Express

Crie uma pasta para abraigar o projeto node-express. Nomeia como quiser, por exemplo

```
>mkdir projetos-express
```

Agora, abra a pasta criada para abraigar o projeto node-express e faça a seguinte instalação:

```
>cd projetos-express
```

```
>projetos-express> npm init
```

Quando surgirem as perguntas – log após a inicialização do comando acima, basta digitar a tecla Enter para todos – até a utltima pergunta.

Na sequencia, instale a estrutura *Express* localmente (dentro da pasta que será usada para abrigar o projeto) usando NPM para que possa ser usada para criar um aplicativo da web usando terminal de node.

```
npm install express --save
```

O comando acima salva a instalação localmente no diretório **node_modules** e cria um diretório expresso dentro de node_modules. Você deve instalar os seguintes módulos importantes junto com express -

- **body-parser** - Este é um middleware node.js para lidar com dados de formulário codificados em JSON, Raw, Texto e URL.
- **cookie-parser** - Analisa o cabeçalho do cookie e preenche *req.cookies* com um objeto codificado pelos nomes dos cookies.

```
$ npm install body-parser --save
$ npm install cookie-parser --save
```

Primeiro web app com Express

A seguir está um aplicativo Express muito básico que inicia um servidor e escuta na porta 8081 para conexão. Este aplicativo responde com **Hello World!** para pedidos à página inicial. Para todos os outros caminhos, ele responderá com um **erro 404 não encontrado.**

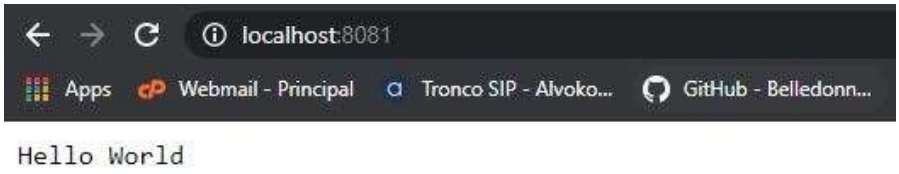
```
// primeiro app node-express
// variavel que receberá o módulo express
var express = require('express');
// objeto para fazermos uso dentro da aplicação
var app = express();


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

// aqui, vamos implementar o "escutador da requisição"
var server = app.listen(8081, function(){
    var host = server.address().address
    var port = server.address().port

    console.log('O servidor pode ser
acessado em http:// %s:%s ', host, port)
});
```

Salve o código acima em um arquivo chamado server.js e execute-o com o seguinte comando.

```
node server.js
```

Você verá a seguinte saída:

`O servidor pode ser acessado em http://::::8081;` insira a linha com o endereço ***http://localhost:8081*** em qualquer navegador para ver o seguinte resultado.



Request/Response (Pedido/Resposta)

O aplicativo Express usa uma função de retorno de chamada cujos parâmetros são objetos de **solicitação(request - req)** e **resposta(response - res)** .

```
app.get('/', function (req, res) {
   // --
})
```

- Objeto de solicitação - o objeto de solicitação representa a solicitação HTTP e tem propriedades para a string de consulta da solicitação, parâmetros, corpo, cabeçalhos HTTP e assim por diante.
- Objeto de resposta - o objeto de resposta representa a resposta HTTP que um aplicativo Express envia ao obter uma solicitação HTTP.

Você pode imprimir objetos **req** e **res** que fornecem muitas informações relacionadas à solicitação e resposta HTTP, incluindo cookies, sessões, URL, etc.

Roteamento(routing)

Vimos um aplicativo básico que atende a solicitações HTTP para a página inicial. O roteamento se refere à determinação de como um aplicativo responde a uma solicitação do cliente para um determinado ponto de extremidade, que é um URI (ou caminho) e um método de solicitação HTTP específico (GET, POST e assim por diante).

Estenderemos nosso programa Hello World para lidar com mais tipos de solicitações HTTP. No arquivo server.js do seu projeto Express/NodeJS implemente o código abaixo:

server.js

```
// este é nosso primeiro app node-express
// variavel que receberá o módulo express
var express = require('express');
// objeto para fazermos uso dentro da aplicação
```

```
var app = express();

//usando o método get
app.get('/', function(req, res){
    res.send('Hello Mundo NodeJS! Componente padrão');
});

//usando o método get com um novo endereço
app.get('/uma_lista', function(req, res){
    res.send('Esse é o conteudo para uma pagina-
componente (lista)');
})

// usando o método para um novo endereço de um novo compone
nte
app.get('/zf*zy', function(req, res){
    res.send('Novo componente acessado a partir de uma rota
 customizada!')
})

// aqui, vamos implementar o "escutador da requisição"
var server = app.listen(8081, function(){
    var host = server.address().address
    var port = server.address().port

    console.log('O servidor pode ser acessado
em http:// %s:%s ', host, port)
});
```

Salve o código acima em um arquivo e execute-o com o seguinte comando.

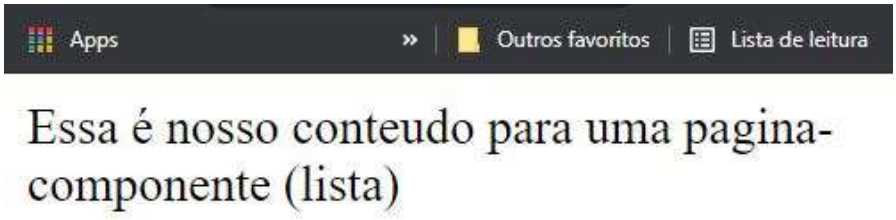
```
node server.js
```

Você verá a seguinte saída:

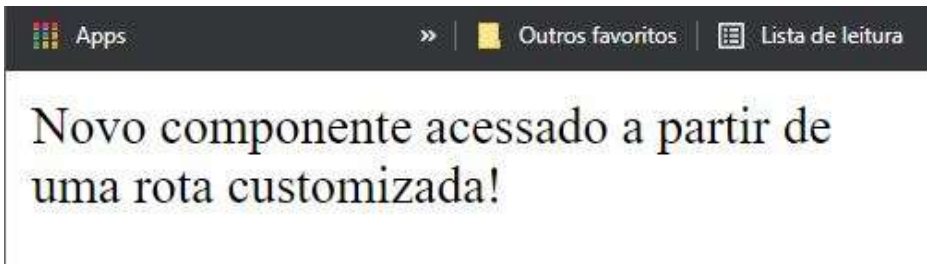
```
O servidor pode ser acessado em http://::::8081
```

Agora você pode tentar diferentes solicitações em http://localhost:8081 para ver a saída gerada por server.js. A seguir estão algumas capturas de tela mostrando diferentes respostas para diferentes URLs.

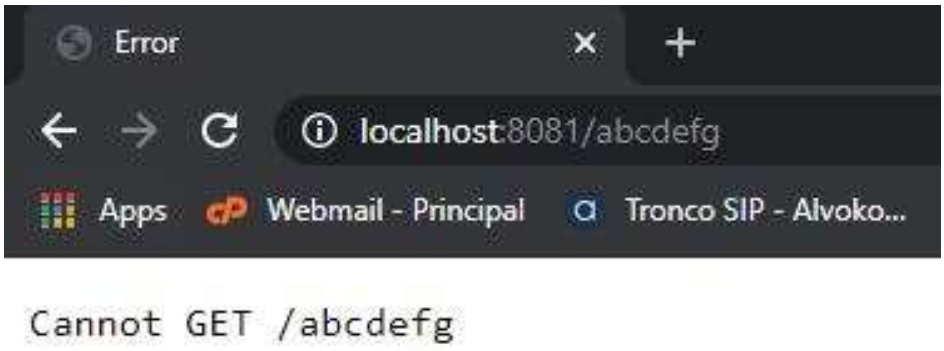
Tela mostrando novamente *http://localhost:8081/uma_lista*



Agora, acesse a rota: *http://localhost:8081/zfzy*



Tela mostrando novamente *http://localhost:8081/abcdefg*



Arquivos estáticos

È possível, rapidamente, criar a estrutura básica de um projeto Express – crinado componentes - via linha de comando, da seguinte maneira – dentro da pasta de seu projeto, via terminal de comando, siga os passos abaixo:

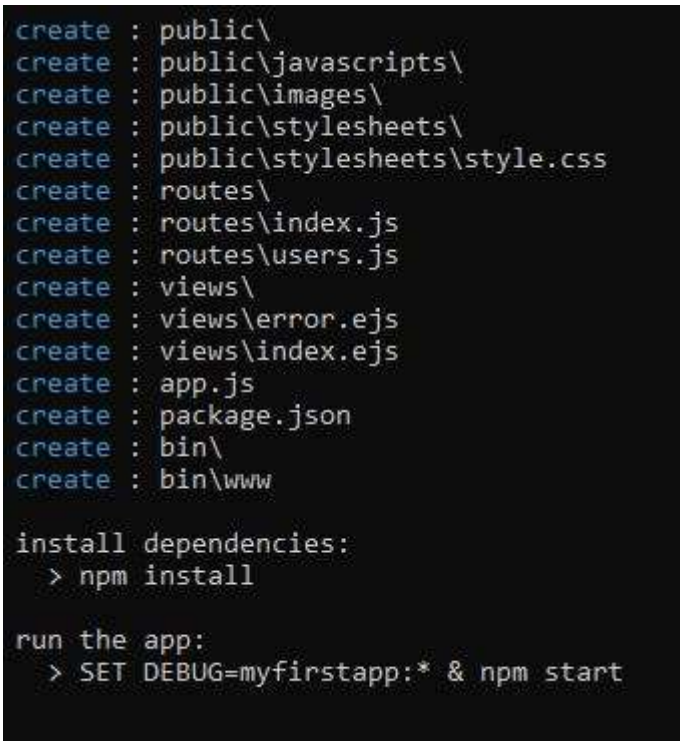
1. Insira a instrução para instalação do express-generator:

```
npm install express-generator
```

2. na sequencia, insira a seguinte instrução:

```
express --view=ejs
```

Após a instrução, você verá o seguinte resultado no prompt de comando:



Assim, teremos um projeto Express/NodeJs pronto para implementação.

O elemento “--view” é para usar a view-engine (motor de renderização) EJS, ao invés do tradicional Jade/Pug.

Depois entre na pasta e mande instalar as dependências:

```
npm install
```

Ainda no terminal de linha de comando e, dentro da pasta do projeto, digite:

```
npm start
```

Isso vai fazer com que a aplicação default inicie sua execução em ***localhost:3000***, que você pode acessar pelo seu navegador.

Para interromper a execução do servidor digite *Crtl+C.*

Entendendo o Express

O arquivo app.js – localizado dentro do diretório da sua aplicação NodeJS - é o arquivo principal da aplicação. Ao observar este arquivo é possível ver a seguinte estrutura de código:

```
var createError = require('http-errors');
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');

var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');
```

Acima estão deifinidas algumas variaveis JavaScript;estas variáveis referenciam pacotes, dependências, funcionalidades e rotas da aplicação Node JS. Rotas são uma simbiose entre *models* e *controllers.*

Na sequência é possível observar a seguinte instrução:

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

A variável app, neste contexto, é a instancia de express (módulo). A próxima de códigos usa essa instância acessar propriedades referentes ao módulo Express. Observe:

```
// view engine setup
```

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

app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use('/', indexRouter);
app.use('/users', usersRouter);
```

Acima, o arquivo ***app.js*** direciona o NodeJS a encontrar e renderizar as views do projeto a partir da *engine ejs*. Ao final do código gerado com o projeto é possível observar que o Express possibilita acessar objetos estáticos a partir da indicação de um endereço de pasta - que se encontra dentro do projeto – também gerada por padrão: *(__dirname, 'public'); o* navegador entende como se a pasta fosse parte da raiz do projeto.

```
// catch 404 and forward to error handler
app.use(function(req, res, next) {
  next(createError(404));
});

// error handler
app.use(function(err, req, res, next) {
  // set locals, only providing error in development
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') === 'development' ?
 err : {};

  // render the error page
  res.status(err.status || 500);
  res.render('error');
});
```

Acima, estão os blocos que auxiliam o projeto na manipulação de erros.

```
module.exports = app;
```

NodeJS possibilita a operação de desenvolvimento a partir do principio da componentização (um mesmo proejto divido em vários "pedaços" diferentes); uma das formas de estabelecera "comunicação" entre as parte fragmentadas –

componentes – é necessário “expor” os conjuntos de códigos para todas a aplicação. A instrução ***module.exports = app*** possibilita aos outros elementos do projeto a chamada do módulo exportado.

O tambéme é possível encontrar um conjunto de pastas que podem abrigar uma série de arquivos estáticos como imagens, arquivos de atributos de estilo, arquivos js outros.

Manteremos algumas imagens no subdiretório **public/images da** seguinte forma:

```
node_modules
public/
public/images
public/images/image.png
.
.
.
server.js
```

Modifique o arquivo *server.js* para adicionar a funcionalidade de acessar conteúdos dentro de arquivos estáticos. Observe o código abaixo e implemente como se segue:

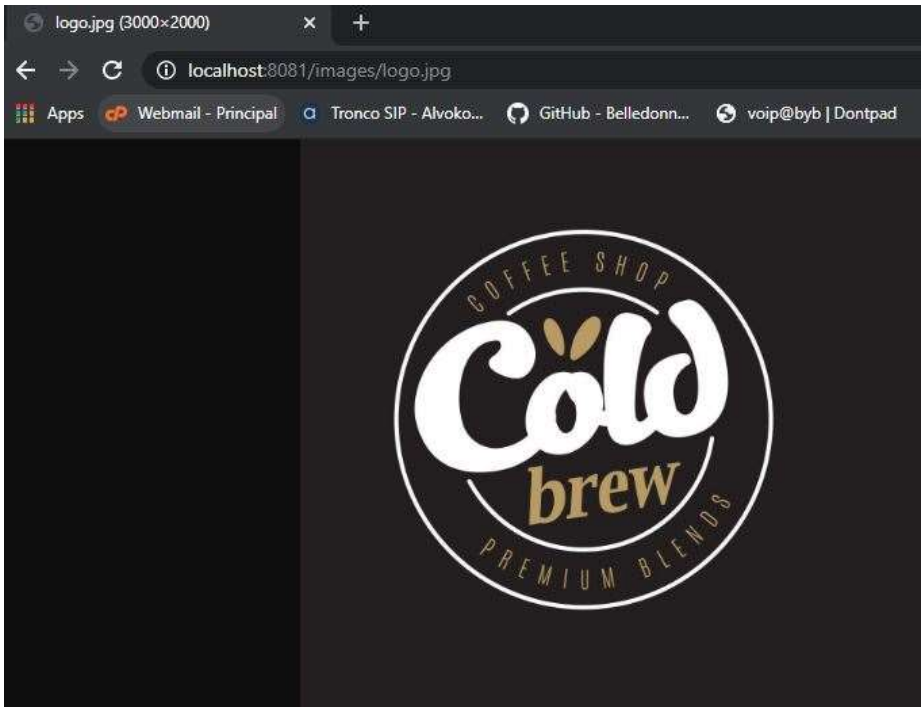
server.js

```
// este é nosso primeiro app node-express
// variavel que receberá o módulo express
var express = require('express');
// objeto para fazermos uso dentro da aplicação
var app = express();

app.use(express.static('public'))

// aqui, vamos implementar o "escutador da requisição"
var server = app.listen(8081, function(){
    var host = server.address().address
    var port = server.address().port

    console.log('Nosso servidor está sendo escutado em http
:// %s:%s ', host, port)
});
```

Salve o código acima em um arquivo chamado server.js e execute-o com o seguinte comando.

```
node server.js
```

Agora abra *http://localhost:8081/images/logo.png* em qualquer navegador e observe o seguinte resultado.



Método GET para data parsing

Abaixo é mostrada a forma mais simples de se passar dois valores usando o método *HTML FORM GET*. Vamos usar o roteamento ***data_get_parsing*** - dentro do arquivo *app.js* - para lidar com a entrada de dados. O primeiro passo é abrir o arquivo *index.ejs* e implementar o seguinte código dentro das tags *<body></body>.* Observe abaixo:

index.ejs

```
<br />
      <form action = "/data_get_parsing" method = "GET">
        Nome: <input type = "text" name = "nome">
     <br />
        Sobrenome: <input type = "text" name = "sobrenome">
        <input type = "submit" value = "Enviar">
      </form>
```

Salve o arquivo. Agora, dentro do arquivo app.js insira o seguinte código – observe, abaixo, e implemente como se segue:

app.js

```
var createError = require('http-errors');
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');

var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');
// vamos fazer uso do módulo body-parser
var bodyParser = require('body-parser')

var app = express();

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

app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use('/', indexRouter);
app.use('/users', usersRouter);

app.get('/data_get_parsing', function (req, res) {
   // a saida será no formato JSON
   response = {
      nome:req.query.nome,
      sobrenome:req.query.sobrenome
   };
   console.log(response);
   res.end(JSON.stringify(response));
})


// catch 404 and forward to error handler
app.use(function(req, res, next) {
  next(createError(404));
});

// error handler
app.use(function(err, req, res, next) {
  // set locals, only providing error in development
  res.locals.message = err.message;
```

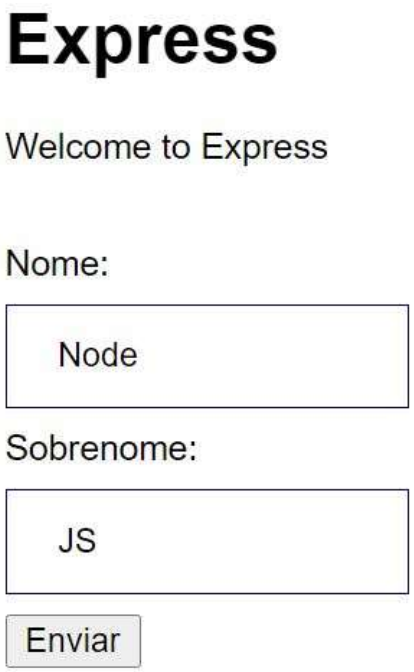
```
  res.locals.error = req.app.get('env') === 'development' ?
 err : {};

  // render the error page
  res.status(err.status || 500);
  res.render('error');
});

module.exports = app;
```

Para finalizar, implemente o seguinte conjunto de atributos CSS dentro do aquivo */stylesheets/style.css*

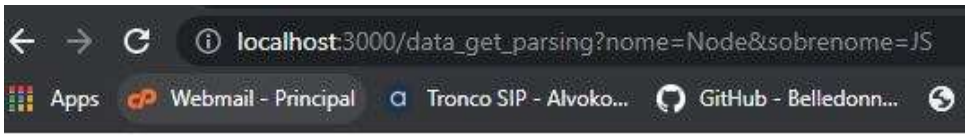
styles.css

```
input[type=text], input[type=email]{
  width: 100%;
  padding: 12px 20px;
  margin: 8px 0;
  display: inline-block;
  border: 1px solid #040670;
  box-sizing: border-box;
  }
```

Acesse a aplicação a partir do endereço *http://localhost:3000*; o formulário deve ser semelhante ao indicado abaixo:



Dessa forma, é possível inserir o nome e o sobrenome e clicar no botão enviar para ver o resultado e deve retornar o seguinte resultado:



{"nome":"Node","sobrenome":"JS"}

Método POST para data parsing

Abaixo é mostrada a forma mais simples de se passar dois valores usando o método *HTML FORM POST*. Vamos usar o roteamento ***data_post_parsing*** - dentro do arquivo *app.js* - para lidar com a entrada de dados. O primeiro passo é abrir o arquivo *index.ejs* e implementar o seguinte código dentro das tags *<body></body>.* Observe abaixo:

index.ejs

```
  <form action = "data_post_parsing" method = "POST">
     Nome: <input type = "text" name = "nome"> <br>
     Sobrenome: <input type = "text" name = "sobrenome">
     <input type = "submit" value = "Enviar">
   </form>
```

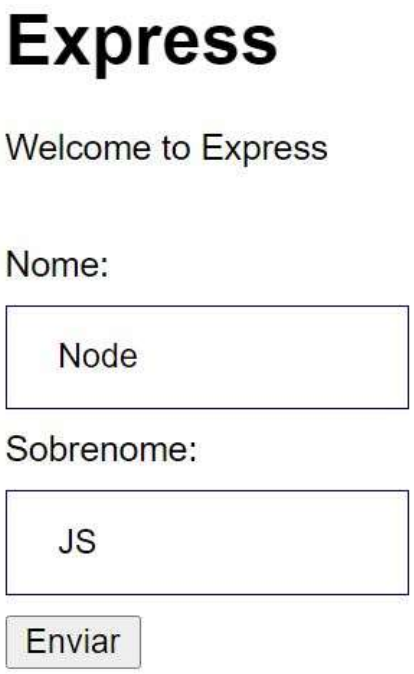
Salve o arquivo. Agora, dentro do arquivo *app.js* insira o seguinte código – observe, abaixo, e implemente como se segue:

app.js

```
var createError = require('http-errors');
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');

var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');
// vamos fazer uso do módulo body-parser
var bodyParser = require('body-parser')

var app = express();

// view engine setup
```

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

app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use('/', indexRouter);
app.use('/users', usersRouter);

app.post('/data_post_parsing', function(req, res){
    // vamos cria um objeto js com duas variaveis para
    // receber os valores inseridos nos inputs
    response = {
        nome:req.body.nome,
        sobrenome:req.body.sobrenome
    };
    console.log(response);
    res.end(JSON.stringify(response))
})

// catch 404 and forward to error handler
app.use(function(req, res, next) {
  next(createError(404));
});

// error handler
app.use(function(err, req, res, next) {
  // set locals, only providing error in development
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') === 'development' ?
 err : {};

  // render the error page
  res.status(err.status || 500);
  res.render('error');
});

module.exports = app;
```

Acesse a aplicação a partir do endereço *http://localhost:3000*; o formulário deve ser semelhante ao indicado abaixo:



Dessa forma, é possível inserir o nome e o sobrenome e clicar no botão enviar para ver o resultado e deve retornar o seguinte resultado: