4. Criando e consumindo webservices

A – Criar um webservice para inclusão e consulta de eventos

Nesta fase do projeto vamos ver como definir webservices para acesso a partir de outras aplicações ao nosso cadastro de eventos. Para tanto, criaremos um novo projeto exclusivo para o serviço.

1. Acessar a pasta **ProjetoEventos**. Nesta pasta, criar uma nova pasta chamada **WebServices**.
2. Estando na pasta **WebServices**, usando o prompt de comandos (ou o console que preferir, como o do VSCode, por exemplo), criar um novo projeto, com os comandos:

```
express apiEventos --ejs
cd apiEventos
npm install
npm install body-parser --save
npm install express-load --save
npm install mongoose --save
```

3. Abrir o VSCode apontando para a pasta **apiEventos**.
4. Criar a pasta **models**.
5. Incluir o arquivo eventos.js:

```
module.exports = function (app) {

    var mongoose = require('mongoose');
    var Schema = mongoose.Schema;

    var evento = Schema({
        descricao: { type: String, required: true },
        data: { type: Date },
        preco: { type: Number }
    });
    return mongoose.model('eventos', evento);
};
```

6. Substitua todo o conteúdo de **app.js** por este:

```
var express = require('express');
```

4. Criando e consumindo webservices

```
var load = require('express-load');

var app = express();
var bodyParser = require('body-parser');

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));

var mongoose = require('mongoose');
global.db = mongoose.connect('mongodb://localhost:27017/neventos');

load('models').into(app);

var Evento = app.models.eventos;


app.listen(3200, function () {
  console.log('ok');
});
```

7. Em **app.js**, acrescentar os métodos de serviço:

```
var express = require('express');
var load = require('express-load');

var app = express();
var bodyParser = require('body-parser');

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));

var mongoose = require('mongoose');
global.db = mongoose.connect('mongodb://localhost:27017/neventos');

load('models').into(app);

var Evento = app.models.eventos;
```

```
//método do serviço
app.get('/', function (request, response) {
  response.send('Servidor no ar');
});
app.get('/eventos', function (request, response) {

});
```

4. Criando e consumindo webservices

```
app.get('/eventos/:id', function (request, response) {

});
app.post('/eventos', function (request, response) {

});
app.put('/eventos/:id', function (request, response) {

});
app.delete('/eventos/:id', function (request, response) {

});
```

```
app.listen(3200, function () {
  console.log('ok');
});
```

8. Vamos começar a codificar as funções para os serviços. Iniciaremos pela consulta aos eventos. Atualizar os métodos **app.get()**:

```
//método do serviço
app.get('/', function (request, response) {
  response.send('Servidor no ar');
});
app.get('/eventos', function (request, response) {
  Evento.find(function (erro, eventos) {
    if (erro) {
      response.json(erro);
    }
    else {
      response.json(eventos);
    }
  });

});
app.get('/eventos/:id', function (request, response) {
  var id = request.params.id;

  Evento.findById(id, function (erro, evento) {
    if (erro) {
      response.json(erro);
    }
    else {
      response.json(evento);
    }
```

```
  });
});
```

9. Executar o comando:

```
node app.js
```

10. Testar o serviço, chamando a URL no browser (observe que colocamos uma porta diferente da aplicação. Isso é necessário porque podemos acessar este serviço em uma aplicação acessível por uma porta diferente, se estiverem no mesmo servidor):

```
localhost:3200/eventos
```

11. Verificar o id gerado para cada evento cadastrado. Copiar o id de um evento desejado, e executar a URL:

```
localhost:3200/eventos/5a6a0239dc740f18243a8294
```

12. Neste exemplo usamos o id **5a6a0239dc740f18243a8294** mas isso não significa que ela exista quando você a executar, pois são valores diferente para cada registro cadastrado.
13. Codificar os demais métodos:

```
app.post('/eventos', function (request, response) {
  var descricao = request.body.descricao;
  var data = request.body.data;
  var preco = request.body.preco;

  var evento = {
    'descricao': descricao,
    'data': data,
    'preco': preco
  };

  Evento.create(evento, function (erro, evento) {
    if (erro) {
      response.json(erro);
    }
    else {
      response.json(evento);
```

```
    }
  });

});

app.put('/eventos/:id', function (request, response) {
  var id = request.params.id;

  Evento.findById(id, function (erro, evento) {
    if (erro) {
      response.json(erro);
    }
    else {

      var evento_upd = evento;

      evento_upd.descricao = request.body.descricao;
      evento_upd.data = request.body.data;
      evento_upd.preco = request.body.preco;

      evento_upd.save(function (erro, evento) {
        if (erro) {
          response.json(erro);
        }
        else {
          response.json(evento);
        }
      });
      response.json(evento);
    }
  });


});
app.delete('/eventos/:id', function (request, response) {
  var id = request.params.id;

  Evento.findById(id, function (erro, evento) {
    if (erro) {
      response.json(erro);
    } else {
      Evento.remove(evento, function (erro, evento) {
        if (erro) {
          response.json(erro);
        }
        else {
          response.send('removido');
        }
```

```
      });
    }
  });
});
```

B – Consumir o webservice a partir da aplicação

Para demonstrar o consumo de webservices, vamos completar nossa aplicação de cadastro de eventos para contemplar esta funcionalidade.

1. No VSCode, abrir o projeto **nodeEventos**.
2. Acrescentar um item no menu de opções, em **views/eventos/menu.ejs**:

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8">
    <title>Cadastro de Eventos</title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
</head>

<body>
    <header>
        <p>Bem-vindo,
            <%= usuario.nome %>
        </p>
        <h1>Menu de Opções</h1>
        <h4>Escolha sua opção</h4>
    </header>
    <section>
        <ul>
            <li>
                <a href="/cadUsuario">Cadastro de Usuários</a>
            </li>
            <li>
                <a href="/cadEvento">Cadastro de Eventos</a>
            </li>
            <li>
                <a href="/listaEventos">Lista de Eventos</a>
            </li>
            <li>
                <a href="/listaEventosWS">Lista de Eventos
                    (WebService)</a>
            </li>
        </ul>
```

```
    </section>
    <section>
        <a href='/logout'>Logout</a>
    </section>
    <footer>
        <small>Sistema de Cadastro de Eventos e Convidados</small>
    </footer>
</body>

</html>
```

3. No arquivo routes/eventos.js, acrescentar a rota indicada:

```
module.exports = function (app) {

    var valida = require('./../middlewares/valida');
    var eventos = app.controllers.eventos;

    app.get('/menu', valida, eventos.menu);

    app.get('/cadUsuario', valida, eventos.cadastroUsuario);
    app.get('/cadEvento', valida, eventos.cadastroEvento);
    app.get('/listaEventos', valida, eventos.listaEventos);

    app.get('/listaEventosWS', valida, eventos.listaEventosWS);

    app.post('/novoEvento', eventos.novoEvento);
};
```

4. Atualizar o arquivo **controllers/eventos.js** para contemplar a nova rota, responsável pelo consumo do serviço. Para tanto, adicionar o action (cuidado com as vírgulas!):

```
 listaEventosWS: function (request, response) {
    //array para conter os eventos
    var eventos = [];

    //informações da requisição GET
    var info = {
        host: 'localhost',
        port: '3200',
        path: '/eventos',
        method: 'GET'
    };

    //chamando o serviço
```

4. Criando e consumindo webservices

```
        http.request(info, function (res) {
            res.setEncoding('utf8');
            res.on('data', function (data) {
                eventos = JSON.parse(data);

                var usuario = request.session.usuario,
                    params = { usuario: usuario, eventos: eventos };
                response.render('eventos/listaEventosWS', params);
            });
        }).end();
    },
```

5. Criar uma nova view, a ser renderizado pelo novo action no controller. A view deve ser criada em **views/eventos/listaEventosWS.ejs** (Observe que se trata de uma cópia do arquivo **listaEventos.ejs**. Mudamos apenas o título!):

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8">
    <title>Cadastro de Eventos</title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
</head>

<body>
    <header>
        <p>Bem-vindo, <%= usuario.nome %>
        </p>
        <h1>Lista de Eventos do Webservice</h1>
    </header>
    <section>
        <!--será implementado em breve-->
        <table>
            <thead>
                <tr>
                    <th>Descrição</th>
                    <th>Data</th>
                    <th>Preço</th>
                </tr>
            </thead>
            <tbody>
                <% eventos.forEach(function(evento, index) { %>
                    <tr>
                        <td>
```

```
                            <%= evento.descricao %>
                        </td>
                        <td>
                            <%= evento.data %>
                        </td>
                        <td>
                            <%= evento.preco %>
                        </td>
                    </tr>
                    <% }) %>
                </tbody>
            </table>
        </section>
        <section>
            <a href='/menu'>Voltar ao menu</a>
        </section>
        <footer>
            <small>Sistema de Cadastro de Eventos e Convidados</small>
        </footer>
    </body>

    </html>
```

6. Manter o webservice em execução, e testar a aplicação.

C – Adicionar recurso para pagamento do evento via webservice

Nesta parte da aplicação criaremos, no webservice, um recurso para pagamento de um evento com cartão de crédito. Adicionaremos um link na lista de eventos para realizar o pagamento. O processo consiste em captar a descrição do evento, o preço do evento, passa-los como parâmetro via URL para uma nova view (**pagamentos.ejs**), e a partir desta view, informaremos os dados do cartão, e enviaremos todos os dados para o webservice.

1. Alterar o arquivo listarEventosWS.ejs para acrescentar o link para a nova página. Este link informará a descrição e o preço do evento:

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8">
```

4. Criando e consumindo webservices

```
    <title>Cadastro de Eventos</title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
</head>

    <body>
        <header>
            <p>Bem-vindo, <%= usuario.nome %>
            </p>
            <h1>Lista de Eventos do Webservice</h1>
        </header>
        <section>
            <!--será implementado em breve-->
            <table>
                <thead>
                    <tr>
                        <th>Descrição</th>
                        <th>Data</th>
                        <th>Preço</th>
                    </tr>
                </thead>
                <tbody>
                    <% eventos.forEach(function(evento, index) { %>
                        <tr>
                            <td>
                                <%= evento.descricao %>
                            </td>
                            <td>
                                <%= evento.data %>
                            </td>
                            <td>
                                <%= evento.preco %>
                            </td>
                            <td>
                                <a href=
            "/pagamento/<%= evento.descricao %>/<%= evento.preco %>">
                                  comprar este evento</a>
                            </td>
                        </tr>
                        <% }) %>
                </tbody>
            </table>
        </section>
        <section>
            <a href='/menu'>Voltar ao menu</a>
        </section>
        <footer>
            <small>Sistema de Cadastro de Eventos e Convidados</small>
        </footer>
```

```
</body>

</html>
```

2. Alterar o arquivo **routes/eventos.js** para contemplar esta nova rota. Adicionaremos também a rota a ser usada via método POST para efetivação do pagamento. Observe a implementação dos parâmetros na rota (GET):

```
module.exports = function (app) {

    var valida = require('./../middlewares/valida');
    var eventos = app.controllers.eventos;

    app.get('/menu', valida, eventos.menu);

    app.get('/cadUsuario', valida, eventos.cadastroUsuario);
    app.get('/cadEvento', valida, eventos.cadastroEvento);
    app.get('/listaEventos', valida, eventos.listaEventos);

    app.get('/listaEventosWS', valida, eventos.listaEventosWS);

    app.post('/novoEvento', eventos.novoEvento);

    app.get('/pagamento/:evento/:preco', valida, eventos.pagamento);
    app.post('/novoPagamento', eventos.novoPagamento);
};
```

3. Atualizar o controller/eventos.js para incluir os dois actions para esta tarefa:

```
    pagamento: function (request, response) {
        var evento = request.params.evento,
        preco = request.params.preco,
        usuario = request.session.usuario,
            params = { usuario: usuario, evento: evento,
                       preco: preco };


        response.render('eventos/pagamento', params);
    },

    novoPagamento: function (request, response) {
        //a ser implementado
```

```
        },
```

4. Criar a view **pagamento.ejs** (**views/eventos/pagamento.ejs**):

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8">
    <title>Pagamento de Eventos</title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
</head>

<body>
    <header>
        <p>Bem-vindo, <%= usuario.nome %></p>

        <h1>Compra de Eventos</h1>
        <h4>Preencha os dados com seu cartão</h4>
    </header>
    <section>
        <form action="/novoPagamento" method="post">
            Evento:<br />
            <input type="text" name="cartao[evento]" value="<%= evento %>" />
            <br />Preço:<br />
            <input type="text" name="cartao[preco]" value="<%= preco %>" />
            <br> Num. Cartão:<br/>
            <input type="text" name="cartao[numcartao]" />
            <br> CVV: <br />
            <input type="text" name="cartao[cvv]" />
            <br>
            <button type="submit">Efetuar Pagamento</button>
        </form>
    </section>
    <footer>
        <small>Sistema de Cadastro de Eventos e Convidados</small>
    </footer>
</body>

</html>
```

5. Executar a aplicação, testar o link e verificar o valor sendo passado como parâmetro para **pagamento.ejs**.
6. Agora vamos preparar o webservice. Criar o model **pagamentos.js**:

```
module.exports = function (app) {

    var mongoose = require('mongoose');
    var Schema = mongoose.Schema;

    var pagamento = Schema({
        evento: { type: String },
        preco: { type: Number },
        numcartao: { type: String },
        cvv: { type: String }
    });
    return mongoose.model('pagamentos', pagamento);
};
```

7. Atualizar o action **novoPagamento**, em **controllers/eventos.js**. Neste action utilizaremos um novo serviço, a ser criado na próxima etapa.

```
 novoPagamento: function (request, response) {
        //a ser implementado
        var cartao = request.body.cartao;

        var cartaoPost = JSON.stringify({
            'evento': cartao.evento,
            'preco': cartao.preco,
            'numcartao': cartao.numcartao,
            'cvv': cartao.cvv
        });

        //informações da requisição POST
        var info = {
            host: 'localhost',
            port: '3200',
            path: '/pagamentos',
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
                'Content-Length': cartaoPost.length
            }
        };

        //definição do pbjeto para requisição POST
        var reqPost = http.request(info, function (res) {
            res.on('data', function (data) {
                console.log('Incluindo registros:\n');
                process.stdout.write(data);
```

```
                console.log('\n\nHTTP POST Concluído');
            });
        });

        //Gravação dos dados
        reqPost.write(cartaoPost);
        response.redirect('/menu');
        reqPost.end();
        reqPost.on('error', function (e) {
            console.error(e);
        });
    },
```

8. No projeto **apiEventos**, adicionar na pasta models o model **pagamentos.js** (o mesmo usado no projeto **nodeEventos**):

```
module.exports = function (app) {

    var mongoose = require('mongoose');
    var Schema = mongoose.Schema;

    var pagamento = Schema({
        evento: { type: String },
        preco: { type: Number },
        numcartao: { type: String },
        cvv: { type: String }
    });
    return mongoose.model('pagamentos', pagamento);
};
```

9. Atualizar o arquivo app.js, incluindo uma referencia a este model, além das funções para incluir um novo pagamento e para listar pagamentos:

```
var express = require('express');
var load = require('express-load');

var app = express();
var bodyParser = require('body-parser');

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));

var mongoose = require('mongoose');
global.db = mongoose.connect('mongodb://localhost:27017/neventos');

load('models').into(app);
```

4. Criando e consumindo webservices

```
var Evento = app.models.eventos;
var Pagamento = app.models.pagamentos;

//método do serviço

//pagamentos
app.get('/pagamentos', function (request, response) {
  Pagamento.find(function (erro, pagamento) {
    if (erro) {
      response.json(erro);
    }
    else {
      response.json(pagamento);
    }
  });
});
```

```
app.post('/pagamentos', function (request, response) {

  var evento = request.body.evento;
  var preco = request.body.preco;
  var numcartao = request.body.numcartao;
  var cvv = request.body.cvv;

  var pagamento = {
    'evento': evento,
    'preco': preco,
    'numcartao': numcartao,
    'cvv': cvv
  };

  Pagamento.create(pagamento, function (erro, pagto) {
    if (erro) {
      response.json(erro);
    }
    else {
      response.json(pagto);
    }
  });
});

app.listen(3200, function () {
  console.log('ok');
});
```

10. Testar a aplicação.

4. Criando e consumindo webservices

11. Como tarefa, implementar a lista de pagamentos no projeto **nodeEventos**.