Sails.js

Fernando Anselmo
http://fernandoanselmo.orgfree.com/wordpress/

Versão 1.0 em 27 de maio de 2018

Resumo

Sails.js[1] é um framework Web que facilita a criação de aplicativos Node.js customizados a nível empresarial. Projetado implementar a arquitetura MVC com suporte a um estilo mais moderno, orientado a dados e desenvolvimento serviços de API. Além disso, com Sails é necessário dominar somente uma linguagem de programação, porém todo o conhecimento de outras linguagens podem ser aproveitados para criar outras camadas de visões para acessar os serviços fornecidos pelo Sails.

1 Parte inicial

Sails é um framework abrangente que segue o padrão MVC para Node.js projetado especificamente para permitir um rápido desenvolvimento de aplicativos do lado do servidor e a disponibilização de serviços em JavaScript. Possui uma forte arquitetura Orientada a Serviços que fornece diferentes tipos de componentes no qual é possível utilizar para organizar o código e separar as responsabilidades.

Vejamos algumas características do produto:

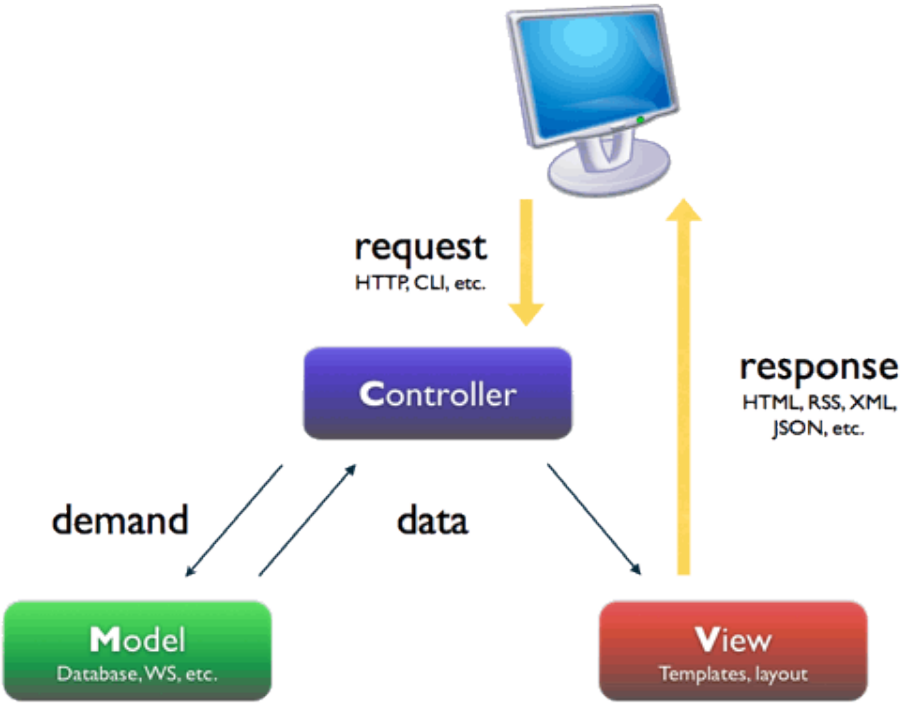
- É 100% Javascript.
- É possível usar qualquer sistema de banco de dados: Sails possui um ORM nativo, Waterline, que fornece uma camada de acesso a dados simples que funciona, não importa o banco de dados que se está usando.
- Auto-gerador de APIs REST: Sails vem com blueprints que ajudam a iniciar o backend da sua aplicação sem escrever qualquer código.
- Suporte fácil ao WebSocket: Sails traduz mensagens de socket recebidas.

- Políticas de segurança reutilizáveis: O Sails fornece uma segurança básica e controle de acesso baseado em funções por padrão que podem ser incrementadas com o uso de JWT.
- Compatível com qualquer estratégia de frontend; Seja esta Angular.js, Backbone, iOS / Object C, Android / Java, Windows Phone.
- Pipeline de ativos flexíveis: navios Sails com Grunt - o que significa que todo o fluxo de trabalho de recursos do frontend é completamente personalizável.

Sails fornece um benefício adicional de ser capaz de compartilhar seu código entre o servidor e o cliente. Isso pode ser muito útil, por exemplo, para implementar uma validação de dados onde é necessária as mesmas regras de validação no cliente e no servidor.

1.1 O que é o padrão MVC?

O Padrão de Projeto (Design Pattern) MVC define a separação de um sistema em três camadas, conforme a seguinte figura:



- **Camada de Modelo** (Model) é a responsável pela interação com o banco de dados.
- **Camada de Visão** (View) é a responsável por mostrar os dados na tela.
- **Camada de Controle** (Controller) é quem gerencia toda a comunicação entre uma Modelo e uma Visão. Não é permitido uma Visão acessar diretamente uma Modelo ou vice-versa.

Simplificadamente pense em uma empresa na qual existe um cliente que deseja falar com um determinado funcionário, porém para acessar sua sala deve passar por uma Secretária. Esse funcionario deseja obter uma determinada informação de um arquivo, para isso deve solicitá-la a Secretária. Ou seja, a Secretária é a controladora de tudo o que será permitido visualizar ou obter de informação da empresa.

1.2 O que são Serviços

A tecnologia de Serviços Web fornece uma série consideráveis benefícios para o desenvolvimento de aplicativos, uma vez que propicia a agilidade requerida pelas empresas frente às mudanças que podem ocorrer no ambiente de negócios. A grande vantagem no uso dos serviços Web provém da capacidade de permitir uma rápida e independente construção de várias 'Visões' de sistemas nas mais diversas plataformas existentes. Por exemplo, uma vez construída a camada de serviços, podemos utilizar o PhoneGap/Cordova para construir uma camada que será acessessível pela plataforma Mobile, uma outra em Java/Swing para acessar via Desktop, uma terceira em Amber.js com um sistema Web, e todas utilizando a mesma camada de Serviço.

Um Servico Web e atualmente utilizado para a integracao entre aplicações. O Web Service REST e uma das formas de se criar um Servico Web, que é utilizada com o protocolo HTTP. O conjunto de operações, suportadas pelo serviço, podem ser de quatro tipos:

- **GET**. Listar todos os membros dos recursos de coleção ou recuperar uma determinada representação de um recurso identificado como 1234.
- **POST**. Criar um novo recurso na coleção em que o ID dele seja automaticamente designado ou obter todos os recursos de uma coleção com base em um filtro.
- **PUT**. Atualizar (substituir) determinados campos de um recurso identificado como 1234.
- **DELETE**. Eliminar uma coleção ou um determinado recurso identificado como 1234.

Serviços Web se consolidaram como uma base para disponibilização de negócios eletrônicos, dentre as quais se destacam organizações que atuam em diversos mercados, tais como Google, Amazon, GM, Fedex, Governo Federal. Isso permite a construção de uma rede intra/inter-organizacional de aplicações colaborativas e distribuídas, onde os Serviços Web, na forma de módulos auto-contidos, são descritos, publicados, localizados e dinamicamente invocados através de camadas de diversas visões.

2 Instalação do Sails.js

Para proceder a instalação do Sails é necessário primeiro instalar o Node.js que pode ser obtido em `https://nodejs.org/en/` e o NPM em `https://www.npmjs.com/`

Para instalar o Sails (lembrando que 'sudo' só necessário se estivermos em ambiente Linux), em uma janela de comandos, digitar o seguinte:

```
$ sudo npm install -g sails
```

Pronto, simples assim.

2.1 Testar a instalação com um Projeto

Para criar um projeto:

```
$ sails new testProject --linker
```

A opção '–linker' faz com que quaisquer recursos sob a pasta /assets sejam copiados para a pasta .tmp/public pelo Grunt quando Sails for levantado. Uma das grandes vantagens do Sails é que também podemos criar o projeto apenas como um provedor de Serviços, isto é, sem a Camada de Visão, para isso utilizamos a seguinte opção:

```
$ sails new myApi --no-frontend
```

O próximo passo é acessar a pasta:

```
$ cd testProject
```

Disponibilizar o bower para o projeto, de modo a instalarmos novos componentes (responda todas as perguntas de forma adequada):

```
$ bower init
```

Iniciar o sails:

```
$ sails lift
```

Se tudo estiver funcionando corretamente é possível acessar o site de boas vindas no endereço:

```
http://localhost:1337/
```

Para interromper o servidor, realizar a seguinte sequencia `Ctrl+C` na janela de comandos.

2.2 Geração dos Artefatos

A maior vantagem em se utilizar o Sails é que este permite gerar as camadas MVC automaticamente, sem precisarmos perder tempo com configurações desnecessárias. A pasta /api contém todos os arquivos backend. A pasta /api/policies estão armazenadas regras para o acesso do usuário da aplicação. A pasta /api/responses contém arquivos como os erros do Servidor Web (404, 403, 500, entre outros). Adicionamos nessa pasta as funções que lidam com tarefas específicas, como decidir como gerenciar usuários com diferentes níveis de acesso. Poderia ser feito na Camada de Controle, mas não é uma boa prática controladores com um monte de lógica de negócios.

As camadas MVC do projeto gerado podem ser encontradas nas seguintes pastas:

- **Camada de Modelo** que está disponível na pasta /api/models do projeto gerado, arquivos no padrão .js
- **Camada de Controle** que está disponível na pasta /api/controllers do projeto gerado, arquivos no padrão .js
- **Camada de Visão** que está disponível na pasta /views do projeto gerado, arquivos no padrão .ejs (esta camada não é gerada automaticamente)

Gerar a Model e a Controller:

```
$ sails generate api |Nome|
```

Por exemplo:

```
$ sails generate model Pet
```

Gerar a Controller:

```
$ sails generate controller |Nome| |Ação|
```

Por exemplo:

```
$ sails generate controller Artigo create find update destroy
$ sails generate controller Comentario create destroy tag like
```

Gerar a Model:

```
$ sails generate model |Nome| |Attribute:Type|
```

Por exemplo:

```
$ sails generate model Cliente nome:string endereco:string idade:integer
```

2.3 Um pequeno teste

Vamos criar uma simples estrutura apenas para verificarmos se tudo está bem fixado até aqui. Criar um projeto conforme descrito e neste executar o seguinte comando:

```
$ sails generate api teste
```

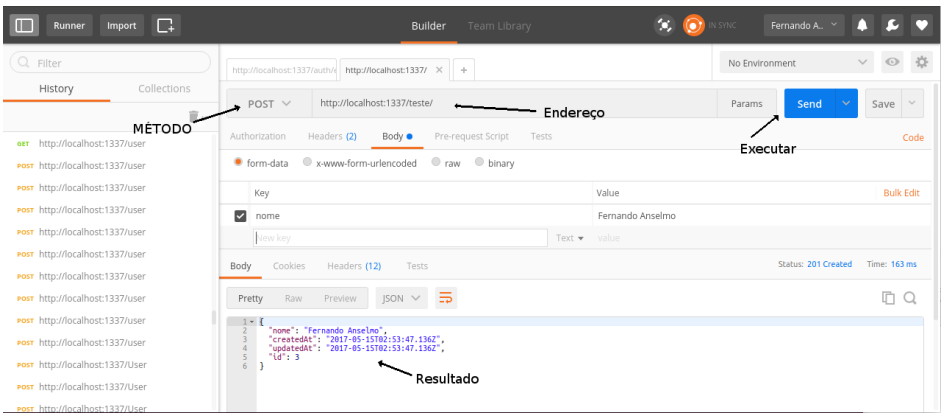
Dois arquivos foram criados, um na pasta '/api/models' chamado 'Teste.js' e outro na pasta '/api/controllers' chamado 'TesteController.js'. No primeiro arquivo modificar a codificação do modelo para:

```
1 module.exports = {
2   attributes: {
3     value: {
4       'nome': 'text'
5     }
6   }
7 };
```

E finalmente ativar o sails:

```
$ sails lift
```

Agora será preciso utilizar um cliente para acessar os serviços expostos. Existem vários pessoalmente prefiro o 'Postman':



E com ele, podemos acessar os seguintes serviços:

- **http://localhost:1337/teste** método GET que traz todos os nomes cadastrados

- **http://localhost:1337/teste/1** método GET que traz o nome com o ID igual a 1
- **http://localhost:1337/teste** método POST que ao ser passado a variável nome no BODY de um formulário esta será armazenada no banco.
- **http://localhost:1337/teste/1** método UPDATE que ao ser passado a variável nome no BODY de um formulário esta será armazenada no banco no lugar do ID igual a 1
- **http://localhost:1337/teste/1** método DELETE que irá excluir o ID com o valor igual a 1

2.4 Arquivos de Configuração

Outros arquivos importantes serão bastante utilizados são:

- **/config/sockets.js**: este arquivo contém dois métodos, "onConnect"e "onDisconnect". Utilizamos este para proceder as conexões de socket.
- **/config/routes.js**: este arquivo nos permite a definição de URLs com as Visões e endpoints para os métodos da Camada de Controle.
- **/config/models.js**: este arquivo nos permite especificar os conectores para o Banco de Dados além de definir como migar os dados.

Por padrão o Sails acessa um ORM (Object Relational Mapper) chamado Waterline que podemos utilizar para realizar testes no sistema, não se preocupe pois é possível (através do uso de adaptadores) se conectar a, basicamente, todos os bancos conhecidos do mercado.

No arquivo /config/models.js é possível definirmos o modo como a base de dados irá tratar os dados, são eles:

- **safe**. Nunca migrar automaticamente a base de dados (usado por padrão, se nada for definido).
- **alter**. Migrar os dados, mas manter os dados já existentes (experimental).
- **drop**. Cada vez que o Servidor for reiniciado, eliminar TODOS os dados e reconstruir TODOS os modelos.)

Também podemos subir o servidor com um dessas opções. Por exemplo:

```
$ sails lift --models.migrate='alter'
```

2.5 Acessar o MySQL no Docker

Antes de mais nada precisamos do banco MySQL rodando no Docker, para conseguir realizar isso assista o vídeo disponível no meu canal no endereço `https://www.youtube.com/watch?v=skx_Oxdw9i0&t=183s` e uma vez que o banco estiver configurado, acessar com o comando:

```
$ docker exec -it mybanco mysql -p
```

Usar a senha 'root' para entrar e criar a base para este exemplo com o comando

```
create database demo;
```

Sair do MySQL com o comando:

```
exit
```

Criar um projeto para o Sails conforme descrito e neste executar o seguinte comando:

```
$ sails generate api cliente
```

Modificar o arquivo 'connections.js' na pasta '/config' para a seguinte codificação:

```
module.exports.connections = {
  localDiskDb: {
    adapter: 'sails-disk'
  },
  sailsmysql: {
    adapter: 'sails-mysql',
    host: '127.0.0.1',
    port: 3306,
    user: 'root',
    password: 'root',
    database: 'demo'
  }
};
```

Modificar o arquivo 'models.js' na pasta '/config' para a seguinte codificação:

```
module.exports.models = {
  connection: 'sailsmysql',
  migrate: 'alter'
};
```

E o arquivo 'Cliente.js' na pasta '/api/models' para a seguinte codificação:

```
module.exports = {
  attributes: {
    nome: {
      type: 'string',
      required: true
    },
    email: {
      type: 'email',
      required: true,
      unique: true
    }
  }
};
```

O último passo é adicionar a biblioteca de conexão com o MySQL com o seguinte comando:

```
$ npm install sails-mysql --save
```

Executar o Sails:

```
$ sails lift
```

E execute os seguintes comandos em um navegador (como complemento para `http://localhost:1337`):

- **/cliente/create?nome=Fernando&email=anselmo@gmail.com**
  cadastrar um novo cliente com o nome: 'Fernando' e email: 'anselmo@gmail.com'.
- **/cliente/update/1?nome=Fernando Anselmo**
  modificar o cliente com o ID igual a 1 para o nome: 'Fernando Anselmo'.
- **/cliente/destroy/1**
  eliminar o cliente com o ID igual a 1 do banco.

Observe que automaticamente o Sails cria no banco alguns campos como chave primária e datas de atualização, podemos não desejar que ele se comporte dessa forma para isso, no

modelo, entre a instrução “exports” e “atributes” colocamos as seguintes instruções:

- **autoCreateAt: false,**: para não criar a data de criação do registro
- **autoUpdateAt: false,**: para não criar a data de atualização do registro
- **autoPK: false,**: para não criar a chave primária

2.6 Ajuste da saída Dados

Com o Sails podemos criar uma controller com dois passos simples (se a Model já tiver pronta então será apenas um passo) para nos devolver basicamente qualquer conjunto de dados que desejemos. Por exemplo uma nova saída com base em uma consulta SQL que agrupe os dados. Vamos aproveitar o projeto visto com o banco MySQL para realizar essa tarefa. Adicione uma nova Model e Controler com o seguinte comando:

```
$ sails generate api Conta
```

Modificar o arquivo ’Conta.js’ na pasta ’/api/models’ para a seguinte codificação:

```
module.exports = {
  attributes: {
    dtEntrada: {
      type: 'datetime',
      required: true
    },
    valor: {
      type: 'float',
      required: true
    }
  }
};
```

Crie a maior quantidade de dados que desejar variando as informações de “data de entrada” e “valor” o máximo possível. O segundo passo é adicionar um novo método de chamada no arquivo ’ContaController.js’ na pasta ’/api/controllers’ com a seguinte codificação:

```
module.exports = {
  graph2: function (req, res) {
    var myQuery = "select month(dtEntrada) as mes, sum(valor) as valor from conta "
    +
        "group by month(dtEntrada)";
    var meses = [];
    var valor = [];
    Conta.query(myQuery, function (err, contas){
    if (err) {
      return res.json({"status": 0, "error": err});
    } else {
      for (var key in contas) {
   if (contas.hasOwnProperty(key)) {
     meses.push(contas[key].mes);
     valor.push(contas[key].valor);
   }
      }
    return res.json({"mes": meses, "valor": valor});
    }
  });
}
```

Selecionamos da base de dados, mês a mês o somatório dos valores e montamos para a saída um JSON com dois arrays “mes” e “valor”, é ideal para carregar uma visão com o **Chart.js** com a montagem de um gráfico.

2.7 Adicionar o Bootstrap e a JQuery ao projeto

O Bootstrap e a JQuery consolidaram-se como padrões de mercado, para disponibilizá-los em nosso projeto devemos seguir os seguintes passos:

1. Criar na raiz do projeto um arquivo chamado .bowerrc com o seguinte conteúdo:
`{‘‘directory’’ : ‘‘assets/vendor’’ }`

2. Instalar o Bootstrap com o comando (isso já instalará a JQuery):
`$ bower install bootstrap --save --production`

3. No arquivo /assets/styles/importer.less adicionar a linha:
`@import ’../vendor/bootstrap/less/bootstrap.less’;`

4. Copiar a pasta /assets/vendor/bootstrap/fonts para embaixo da pasta /assets.

5. No arquivo /tasks/pipeline.js após a instrução ’sails.io.js’ adicione os seguintes comandos:

```
’js/dependencies/sails.io.js’,
// Adicionar a JQuery JS
’vendor/jquery/dist/jquery.min.js’,
// Adicionar o Bootstrap JS
’vendor/bootstrap/dist/js/bootstrap.min.js’,
```

Agora seu projeto já pode contar com todo o poder do Bootstrap e da JQuery.

2.8 Mais um teste da Instalação

Vamos fazer um exemplo mais completo para ver se tudo está funcionando corretamente. Na pasta /views, no arquivo ’layout.ejs’, localizar a linha com a instrução ’<%- body %>’ e alterar para:

```
    <nav class="navbar navbar-inverse navbar-fixed-top">
      <div class="container">
        <div class="navbar-header">
          <button type="button" class="navbar-toggle collapsed"
    data-toggle="collapse" data-target="#navbar" aria-expanded="false"
    aria-controls="navbar">
            <span class="sr-only">Toggle navigation</span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
          </button>
          <a class="navbar-brand" href="/">Sails Exemplo</a>
        </div>
        <div id="navbar" class="navbar-collapse collapse">
          <form class="navbar-form navbar-right">
            <div class="form-group">
              <input type="text" placeholder="Email" class="form-control">
            </div>
            <div class="form-group">
              <input type="password" placeholder="Senha" class="form-control">
            </div>
```

```
            <button type="submit" class="btn btn-success">Entrar</button>
          </form>
        </div><!--/.navbar-collapse -->
      </div>
    </nav>
    <%- body %>
    <div class="container">
      <hr />
      <footer class="footer">
        <div class="pull-right">
          <a href="http://sailsjs.com">sails.js</a>
          <div>Construído com o Sails</div>
        </div>
      </footer>
    </div>
```

Isso colocará um cabeçalho e rodapé padrão para todas as páginas do projeto. Criar uma nova pasta abaixo dessa pasta chamada '/static' e nesta crie um arquivo chamado 'index.ejs' com o seguinte conteúdo:

```
<div class="jumbotron">
  <div class="container">
    <h2>Exemplo do Sails com Bootstrap e JQuery</h2>
    <p>Verificar se tudo está funcionando corretamente...</p>
  </div>
</div>
```

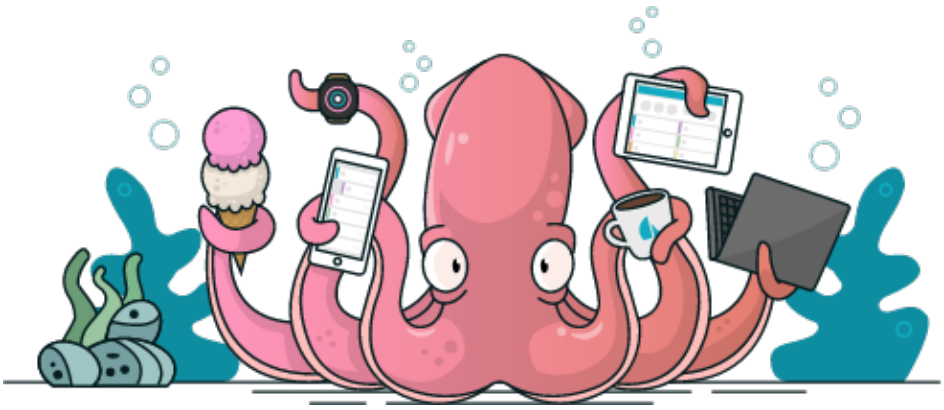
Agora no arquivo de rotas (/config/routes.js) modifique a rota padrão para:

```
  '/': {
    view: 'static/index'
  }
```

E ao levantar o Sails teremos o seguinte resultado na página inicial:

3 Modelagem de Dados

Agora que sabemos como instalar, criar e configurar corretamente um projeto no Sails. Vamos criar um exemplo mais completo para testar um modelo de dados como forma de conhecer melhor suas potencialidades.



A empresa MeuPet deseja melhorar os serviços de implantação de sua intranet com buscas no aprimoramento de disponbilização de Serviços. A estrutura da base de dados foi definida através do rascunho das seguintes tabelas:

`Cliente`

- **priNome**. String (obrigatório), primeiro nome
- **ultNome**. String (obrigatório), último nome
- **endereco**. Text, endereço de residência
- **email**. String (válido), endereço eletrônico de contato
- **telefone**. String (obrigatório), número do telefone de contato

`Pet`

- **idDono**. String (obrigatório), campo de associação com o cliente
- **genero**. String, gênero do animal que só deve permitir: 'Macho' ou 'Fêmea'
- **datNascimento**. Date, contendo a data nascimento
- **especie**. String, espécie do animal que só deve permitir: 'Cachorro' ou 'Gato'
- **raca**. String, descricao da raça do animal
- **caracteristica**. Text, campo livre para observações sobre o animal

`Servico`

- **inicial**. Date, data e hora inicial do serviço
- **idpet**. String (obrigatório), campo de associação com o animal
- **tipo**. String, tipo do serviço que só deve permitir: 'Consulta', 'Banho' ou 'Tosa'
- **descricao**. Text, descrição completa do serviço a ser realizado
- **status**. String, situação do serviço que só deve permitir: 'Agendado' ou 'Finalizado'
- **buscar**. Boolean, se é ou não para buscar o animal
- **levar**. Boolean, se é ou não para levar o animal ao término do serviço

Todas as tabelas devem ter dois campos obrigatórios que não serão informados no cadastro:

- **id**. String, chave única
- **adicionadoEm**. Date, contendo a data de cadastro

3.1 Implementando para o Banco de Dados

Como banco de dados usaremos o RethinkDB, um banco de dados ORM, open-source, escalável com foco na criação de aplicativos em tempo real mais fácil. Qual a diferença para o MogoDB? Um gerenciador que nos permite controla visualmente o banco, ou seja, temos uma maior capacidade administrativa, mas não se preocupe esse projeto pode ser adaptado para qualquer banco que deseje.

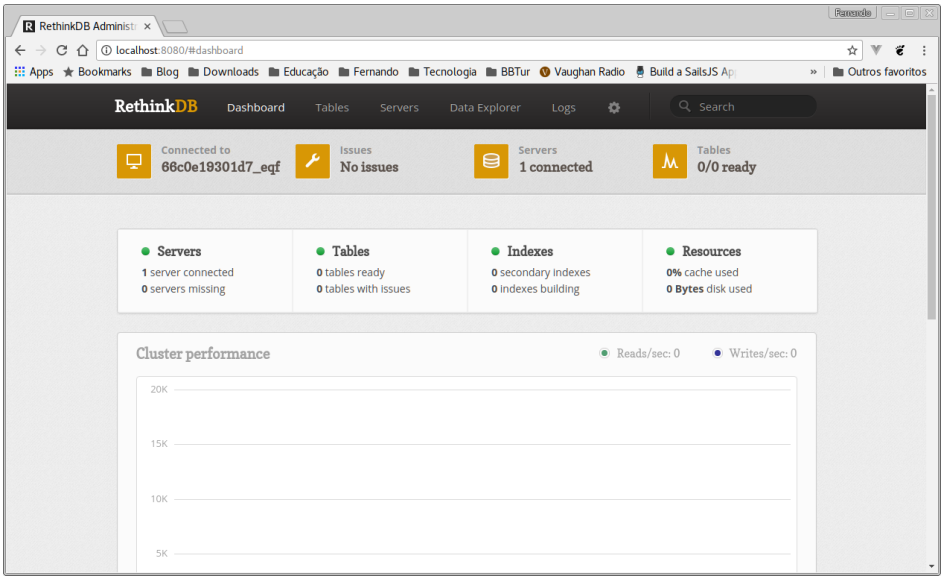
Para instalar o banco usaremo uma imagem do Docker, então com o Docker já instalado podemos digitar simplesmente o seguinte comando para baixarmos a imagem oficial:

```
$ docker pull rethinkdb
```

E o seguinte comando para rodá-la e ativar o banco:

```
$ docker run -d -p 8080:8080 -p 28015:28015 -p 29015:29015 --name
rethink1 rethinkdb
```

O banco está ativo na porta 28015, a porta 29015 é de controle do RethinkDB e a porta 8080 é sua parte administrativa, abra um navegador e acesse o endereço `http://localhost:8080` e veremos a seguinte imagem:



Se deseja parar o banco de dados basta digitar o seguinte comando:

```
$ docker stop rethink1
```

E para ativá-lo novamente:

```
$ docker start rethink1
```

3.2 Configurar o Banco de Dados

Na pasta do projeto adicione dois módulo necessários para realizar a conexão com o banco:

```
$ npm install thinky thinky-loader --save
```

No arquivo 'package.json' adicione a seguinte linha na seção 'dependencies':

```
"sails-rethinkdb":  "github:gutenye/sails-rethinkdb#master",
```

E instale esta dependência:

```
$ npm install
```

Após a instalação devemos corrigir um erro que cria tabelas duplicadas, editar a classe 'connection.js' na pasta /node_modules/sails-rethinkdb/lib e comentar as seguintes instruções na função _setupTables(tables):

```
this.db.tableCreate(name).run(this.conn, err => {
  if (err && !err.message.match(/Table `.*` already exists/))
    throw err
})
```

Com essa primeira parte pronta vamos realizar as seguintes modificações nos arquivos do projeto. No arquivo 'config/connections.js' adicionar a seguinte instrução (abaixo da expressão de 'localDiskDb'):

```
rethinkdb: {
  adapter: 'sails-rethinkdb',
  host: 'localhost',
  db: 'petshop'
}
```

Criar uma pasta chamada '/hooks' abaixo da pasta '/api'. Nesta pasta criar um arquivo chamado 'thinkhook.js', com as seguintes instruções:

```
module.exports = function(sails){
  return {
    connecting: function () {
      let orm = require('thinky-loader');
      let path = require('path');
      var dir = path.resolve(__dirname, '../models');
      let ormConfig = {
        debug     : true,
        modelsPath: dir,
        thinky    : {
          rethinkdb: {
            host        : 'localhost',
            port        : 28015,
            authKey     : '',
            db          : 'petshop',
            timeoutError: 5000,
            buffer      : 5,
            max         : 1000,
            timeoutGb   : 60 * 60 * 1000
          }
        }
      };
      orm.initialize(ormConfig)
      .then(() => console.log('RethinkDB está pronto para uso!'))
      .catch((e) => console.log(e));
    },
    initialize: function (cb) {
      this.connecting();
      sails.emit('hook:thinkhook:done');
      return cb();
    },
  }
}
```

Na inicialização também é possível passar o comando 'orm.initialize(ormConfig, thinky)' que repassa uma instância do thinky para uma configuração adicional. Nosso próximo passo é

modificar o arquivo '/config/models.js':

```
module.exports.models = {
  connection: 'rethinkdb',
  migrate: 'safe'
};
```

Se tudo está funcionando corretamente ao iniciar novamente o servidor foi recebida a seguinte mensagem:

```
Creating a pool connected to localhost:28015
RethinkDB está pronto para uso!
```

Além disso no Administrador do RethinkDB, na seção Tables será mostrada a base de dados 'petshop'.

3.3 Criação dos Modelos e Controles

Começaremos criando os três modelos/controles necessários:

```
$ sails generate api Cliente
$ sails generate api Pet
$ sails generate api Servico
```

Agora vamos alterar cada uma dos modelos (na pasta /api/models) para criar corretamente a estrutura das nossas tabelas conforme definido.

Arquivo: Cliente.js

```
module.exports = function() {
  let thinky = this.thinky;
  let validator = require('validator');
  let type = this.thinky.type;
  let models = this.models;
  return {
    tableName: 'cliente',
    schema: {
      id: type.string(),
      priNome: type.string().required(),
      ultNome: type.string().required(),
      endereco: type.string(),
      email: type.string().validator(validator.isEmail),
      telefone: type.string().required(),
      adicionadoEm: type.date().default(new Date())
    },
    options: {},
    init: function(model) { }
  };
};

```

Arquivo: Pet.js

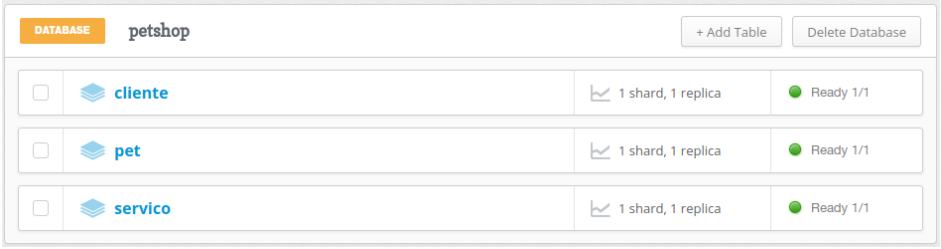
```
module.exports = function() {
  let thinky = this.thinky;
  let validator = require('validator');
  let type = this.thinky.type;
  let models = this.models;
```

```
  return {
    tableName: 'pet',
    schema: {
      id: type.string(),
      idDono: type.string().required(),
      genero: type.string().enum("Macho", "Fêmea"),
      datNascimento: type.date(),
      especie: type.string().enum("Cachorro", "Gato"),
      raca: type.string(),
      caracteristica: type.string().required(),
      adicionadoEm: type.date().default(new Date())
    },
    options: {},
    init: function(model) {
      model.belongsTo(models.cliente, "dono", "idDono", "id");
    }
  };
};
```

Arquivo: Servico.js

```
module.exports = function() {
  let thinky = this.thinky;
  let validator = require('validator');
  let type = this.thinky.type;
  let models = this.models;
  return {
    tableName: 'servico',
    schema: {
      id: type.string(),
      inicial: type.date().required(),
      idPet: type.string().required(),
      tipo: type.string().enum("Consulta", "Banho", "Tosa"),
      descricao: type.string(),
      status: type.string().enum("Agendado", "Finalizado"),
      buscar: type.boolean().default(false),
      levar: type.boolean().default(false),
      adicionadoEm: type.date().default(new Date())
    },
    options: {},
    init: function(model) {
      model.belongsTo(models.pet, "animal", "idPet", "id");
    }
  };
};
```

Se tudo está funcionando corretamente ao iniciar novamente o servidor é mostrada as mensagens de criação e inicialização das tabelas. Além disso no Administrador do RethinkDB, na seção Tables veremos na base de dados 'petshop' com as tabelas criadas conforme a seguinte imagem:



Se desejar um teste mais completo pode usar um aplicativo como o 'Postman' (ou outro aplicativo similar) para acessar os serviços que já estão criados, para cada tabela existe um CRUD completo pronto e funcionando, com métodos: GET, POST, PUT e DELETE.

4 Proteger o Acesso a Camada Controle com Token

Uma das formas de se proteger os serviços disponibilizados é através de um usuário e senha, no qual após ser realizado o acesso correto o usuário recebe um Token (chave encriptografada gerada) e para acessar os outros serviços obrigatoriamente esta chave deve ser passada.

Como primeiro passo instalar o JSON Web token:

```
$ npm install jsonwebtoken --save
```

Na pasta chamada '/api/services' criar um arquivo de configuração chamado 'jwToken.js' com a seguinte codificação:

```
var
  jwt = require('jsonwebtoken'),
  tokenSecret = "meusegredochave"; // Definir um valor unico para esta variavel
// Generar um Token
module.exports.issue = function(payload) {
  return jwt.sign(
    payload,
    tokenSecret,
    {}
  );
};
// Verificar o Token
module.exports.verify = function(token, callback) {
  return jwt.verify(
    token, tokenSecret, {}, callback
  );
};
```

Na pasta chamada '/api/policies' criar um arquivo com a política de autorização chamado 'isAuthorized.js' com a seguinte codificação:

```
module.exports = function (req, res, next) {
  var token;
  if (req.headers && req.headers.authorization) {
    var parts = req.headers.authorization.split(' ');
    if (parts.length == 2) {
      var scheme = parts[0], credentials = parts[1];
      if (/^Auth$/i.test(scheme)) {
        token = credentials;
      }
    } else {
      return res.json(401, {err: 'Formato de Autorização: Auth [token]'});
    }
  } else if (req.param('token')) {
    token = req.param('token');
    delete req.query.token;
  } else {
    return res.json(401, {err: 'Token de Autorização não encontrado'});
  }
  jwToken.verify(token, function (err, token) {
    if (err) return res.json(401, {err: 'Token Inválido!'});
```

```
    req.token = token;
    next();
  });
};
```

Configurar o arquivo de política geral em 'config/policies.js':

```
module.exports.policies = {
  '*': ['isAuthorized'], // Todos os serviços serão restritos
  'AuthController': {
    'entrar': true // Menos este serviço que permitirá o usuário logar
  },
};
```

Criar a AuthController com o comando:

`$ sails generate controller Auth`

Colocar no arquivo gerado a seguinte codificação:

```
module.exports = {
  entrar: function (req, res) {
    if (req.body.login !== 'root' || req.body.password !== 'senha') {
      return res.json(401, {err: 'Usuário ou Senha inválido!'});
    } else {
      return res.json(200, {token: jwToken.issue(12345)}); // Se tiver trabalhando com ID use-o ao inves de '12345'
    }
  }
};
```

4.1 Testar o Token

No Postman (ou outro aplicativo similar) tentar chamar qualquer controller e deve retornar: {``err'': ``No Authorization header was found'' }

Logar através da /auth/entrar, método POST com o body preenchido com os seguintes parâmetros:

- **login** com valor 'root'
- **password** com valor 'senha'

E será retornado o Token. Usar este Token no parâmetro 'Authorization' do header com o valor: `Auth [token]` em qualquer chamada de outro controller.

5 Conclusão

Sails é um framework JavaScript que facilita a criação de aplicativos com o servidor Node.js a nível empresarial fornece um monte de recursos poderosos por padrão, para que possamos começar a desenvolver um aplicativo sem ter que pensar sobre a configuração. Foi projetado com base no conhecido padrão MVC e com um suporte aos requisitos de aplicativos modernos: APIs orientadas a dados com uma arquitetura escalonável orientada a serviços. É possível usá-lo para qualquer projeto de aplicativo da Web.

Um desenvolvedor que possua experiência com aplicações frontend e está procurando um servidor ágil de JavaScript, pode encontrar no Sails uma boa solução. Este atende também

aos que possuem experiência com aplicações backend em um idioma diferente de JavaScript e está procurando expandir seus conhecimentos em Node.js. Em ambos os casos, a familiaridade com Serviços Web pode ser o quisito mais importante sobre como construir uma aplicação web.

Um conjunto de pequenos módulos trabalham em conjunto com o Sails para fornecer simplicidade, facilidade de manutenção e convenções estruturais aos aplicativos Node.js. Além disso o Sails é altamente configurável, assim não seremos forçados a manter toda uma funcionalidade que não é necessária para o projeto. Sou um entusiasta do mundo Open Source e novas tecnologias. Veja outros artigos que publico sobre tecnologia acessando meu Blog Oficial [2].

Referências

[1] Página do Sails.js
`http://sailsjs.com/`

[2] Fernando Anselmo - Blog Oficial de Tecnologia
`http://www.fernandoanselmo.blogspot.com.br/`

[3] Página do Bootstrap
`http://getbootstrap.com/`

[4] Página da JQuery
`https://jquery.com/`

[5] Página do RethinkDB
`https://www.rethinkdb.com/`