

Projeto de Interfaces WEB

Express-Crud **Aula 11**

Introdução

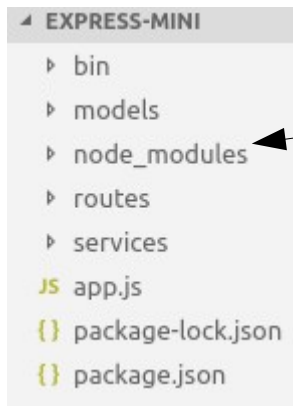
- O objetivo desse mini-projeto é:
 - Implementar um servidor Express para o objeto USER, além das operações:
 - INSERIR
 - LISTAR
 - RECUPERAR (por id e por login)
 - APAGAR (por id)
 - EDITAR (por id)

Preparando o projeto

- Instale o express-api (facilita a criação de aplicações express, gerando o projeto inicial).
 - **npm install -g express-generator-api** Depois, pra criar o projeto, simplesmente faça:
 - **express-api <nome_do_projeto>**
 - cria a pasta de fato
 - **cd <nome_do_projeto>**
 - entra na pasta
 - **npm install**
 - instala as dependências na node_modules.

Preparando o projeto

- Abra o projeto com o VScode, e crie as pastas:
 - **models** e **services**



npm install para instalar a pasta
node_modules!

Execução

- Execute o projeto
 - npm start
- Abra o navegador e digite o caminho:
 - <http://localhost:3000/api/v1/users>
- Mas de onde vem esse caminho?
 - no arquivo users.js, dentro da pasta routes, são inseridas as rotas HTTP.

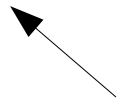
```
/* GET users listing. */  
router.get('/', function(req, res, next) {  
  res.json({users: [{name: 'Timmy'}]});  
});
```

Rota começando com /

Execução

- O caminho definido em user.js é então concatenado em app.js (arquivo principal):

```
app.use('/api/v1/users', users);
```



O “/” definido em users.js será concatenado com “/api/v1/user”, resultando na URI “/api/v1/user/”.
Obviamente podemos mudar essa URI como desejarmos.

Criando o arquivo user.models.js

- Dentro da pasta models, crie o arquivo user.models.js, com o conteúdo:

```
class UserModel{  
  constructor(id, firstName, lastName, login, email, zipcode, password){  
    this.id = id;  
    this.firstName = firstName;  
    this.lastName = lastName;  
    this.login = login;  
    this.email = email;  
    this.zipcode = zipcode;  
    this.password = password;  
  }  
}  
  
module.exports = UserModel;
```

Criando o arquivo `user.service.js`

- O arquivo de serviço irá simular uma base de dados em memória, tendo todos os métodos CRUD para um usuário e armazenando cada usuário em um vetor.
- Sendo assim, toda vez que o servidor for derrubado, perderemos todos os dados.
- Na pasta `services`, crie o arquivo:
 - `user.service.js`

user.service.js

```
const UserModel = require("../models/user.model");
```

Importando o model

```
let users = [];
```

```
let id = 0;
```

```
class UserService{
```

```
  static register(data){
```

```
    let user = new UserModel(id++,  
      data.firstName,  
      data.lastName,  
      data.login,  
      data.email,  
      data.zipcode,  
      data.password);
```

Nossa “base de dados”.

Adicionando um
usuário

```
    users.push(user);
```

```
    return user;
```

```
  }
```

```
  static list(){
```

```
    return users;
```

```
  }
```

Listando os usuários

user.service.js

```
static update(id,data){  
  for(let u of users){  
    if(u.id==id){  
      u.firstName = data.firstName;  
      u.lastName = data.lastName;  
      u.login = data.login;  
      u.zipcode = data.zipcode;  
      u.password = data.password;  
      u.email = data.email;  
      return u;  
    }  
  }  
  return null;  
}
```

Atualizando o objeto no vetor e depois retornando o objeto atualizado.

```
static delete(id){  
  
  for( let i = 0; i < users.length; i++){  
    if(users[i].id == id){  
      users.splice(i,1);  
      return true;  
    }  
  }  
  return false;  
}
```

Apagando um objeto do vetor, usando o splice.

user.service.js

```
static retrieve(id){  
    for( let i = 0; i < users.length; i++){  
        if(users[i].id == id){  
            return users[i];  
        }  
    }  
    return {};  
}
```

Recuperando por id.

```
static retrieveByLogin(login){  
    for( let i = 0; i < users.length; i++){  
        if(users[i].login == login){  
            return [users[i]];  
        }  
    }  
    return [];  
}  
}
```

Recuperando por login.

```
module.exports = UserService;
```

O arquivo de rotas

- Uma vez criado o serviço, agora é hora de disponibiliza meios de acesso a ele. Uma dessas formas é a criação de rotas HTTP.
- Vamos modificar o arquivo `user.js`, dentro da pasta `routes`. Nele iremos inserir rotas de acesso a cada um dos serviço criados em `user.service.js`

O arquivo user.js dentro de routes

```
var express = require('express');  
var router = express.Router();  
var userService = require('../services/user.service');
```

→ Criação do servidor REST express e o objeto de Rotas a partir dele.

```
router.get('/list', function(req, res, next) {  
  return res.json(userService.list());  
});
```

→ Importando o serviço que criamos!

```
router.post('/register', function(req, res, next){  
  const user = userService.register(req.body);  
  return res.json(user);  
});
```

```
router.put('/update/:id', function(req, res, next){  
  const user = userService.update(req.params.id, req.body);  
  return res.json(user);  
});
```

Criação das rotas /list, /register e /update/:id. Note os métodos get, post e put. Veja também que cada método recebe um req (request) e res (response). O req são dados vindos do cliente e o res dados de retorno ao cliente.

O arquivo user.js dentro de routes

```
router.delete('/delete/:id', function(req, res, next){  
  const ok = userService.delete(req.params.id);  
  if(ok) return res.json({"sucess":true});  
  else return res.json({"sucess":false});  
});
```

```
router.get('/retrieve/:id', function(req, res, next){  
  const out = userService.retrieve(req.params.id);  
  return res.json(out);  
});
```

```
router.get('/retrieve/login/:login', function(req, res, next){  
  const out = userService.retrieveByLogin(req.params.login);  
  return res.json(out);  
});
```

```
module.exports = router;
```

O app.js

- Temos que mudar o app.js para refletir as alterações que fizemos no código mais interno.
- O app.js é carregado pelo arquivo www

O app.js

```
var express = require('express');  
var cookieParser = require('cookie-parser');  
var bodyParser = require('body-parser');
```

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

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

```
app.use(bodyParser.json());  
app.use(bodyParser.urlencoded({ extended: false }));  
app.use(cookieParser());
```

```
app.use(function(req, res, next) {  
  res.header("Access-Control-Allow-Origin", "*");  
  res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");  
  res.header("Access-Control-Allow-Methods", "GET, POST, OPTIONS, PUT, DELETE");  
  next();  
});
```

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

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


Configuração inicial...



Permissões de acesso.



Criação da URI inicial, a qual será concatenada com cada rota definida em users.js



Testando

- Reinicie o servidor Express.
 - ctrl+c (para matar o que está rodando)
 - npm start (dentro da pasta raiz)
- Teste as URLs dentro um cliente REST.
 - GET: <http://localhost:3000/users/list>
 - POST: <http://localhost:3000/users/register> (não esqueça do body!)
 - PUT: <http://localhost:3000/users/update/1> (não esqueça do body!)
 - GET: <http://localhost:3000/users/retrieve/login/jeff>
 - GET: <http://localhost:3000/users/retrieve/1>
 - DELETE: <http://localhost:3000/users/delete/1>