



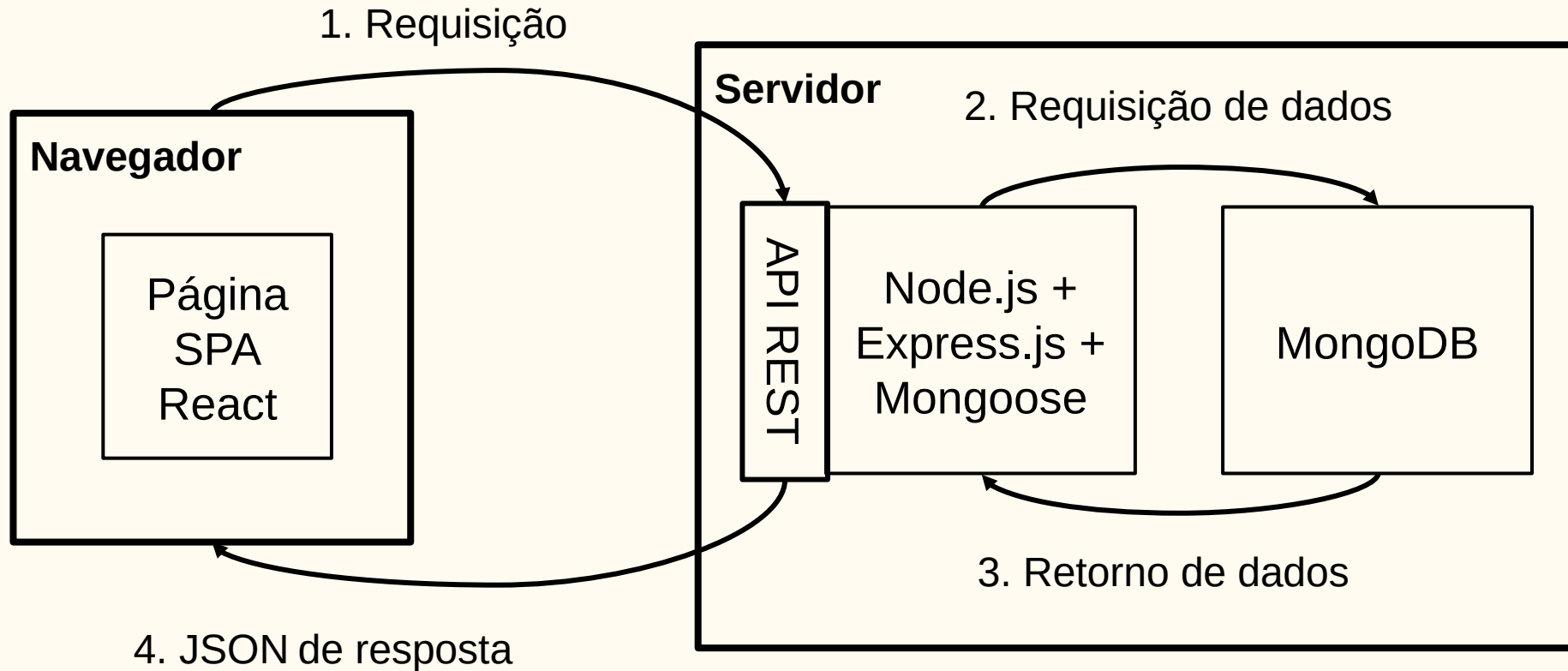
# Conexão Banco Node.js

---

Prof. Victor Farias

V 1.3

# Aplicação MERN



# MongoDB

- Banco NoSQL
  - Baseado em documentos
- Agora, é de responsabilidade da aplicação:
  - Esquema
  - Validação
  - Integridade

# Mongoose

- Biblioteca ODM (Object-Document Modeler)
  - Driver para MongoDB
  - Gerencia relações
  - Executa validações
  - Cria esquemas na aplicação


# Instalação

```
npm install mongoose@5.11 --save
```

# Arquivo de configuração

```
// config/database.js
var mongoose = require('mongoose');

module.exports = function(uri){
  mongoose.connect(uri, { useNewUrlParser: true, useUnifiedTopology: true });
  mongoose.connection.on('connected', function() {
    console.log('Mongoose! Conectado em '+uri);
  });
  mongoose.connection.on('disconnected', function() {
    console.log('Mongoose! Desconectado de ' + uri);
  });
  mongoose.connection.on('error', function(erro) {
    console.log('Mongoose! Erro na conexão: ' + erro);
  });
  mongoose.set('debug', true);
}
```



Listeners  
para os  
estados da  
conexão

# main.js

```
var http = require('http');  
var app = require('./config/express.js')();  
var db = require('./config/database.js');  
  
http.createServer(app).listen(app.get('port'), function(){  
  console.log("Servidor rodando");  
});  
db('mongodb://localhost/sistemamatricula');
```



Lançando conexão

# Criando Schemas

---



# Schema Aluno

```
// app/models/aluno.js
var mongoose = require('mongoose');
module.exports = function(){
  var schema = mongoose.Schema({
    nome: {
      type:String,
      required: true
    },
    matricula: {
      type: String,
      required: true
    },
  });
  return mongoose.model('Aluno', schema);
}();
```

Método que recebe objeto  
que representa Schema

Atributo de Aluno

Tipo do atributo

Pode-se sinalizar que atributo é obrigatório

Define nome da coleção

# Schema Aluno

```
matricula: {  
  type: [String],  
  required: true,  
  index: {  
    unique: true  
  }  
}
```

Lista de strings, caso seja necessário ter mais de uma matrícula

Assegura unicidade do atributo

# Manipulando Coleções

---

# Operações com Documentos

- Os Schemas mongoose oferecem métodos para manipular coleções
  - `.create()` - inserir objeto na coleção
  - `.find(critério)` - busca de documentos com critério
  - `.findById(id)` - busca documento pelo id
  - `.remove(criterio)` - remove documento
  - `findByIdAndRemove(id)` - remove e retorna documento
  - `findByIdAndUpdate(id, novo_doc)` - atualiza documento
  - ...
- Encadeamos esse métodos com `exec()` para executá-los
  - com exceção de `.create()`

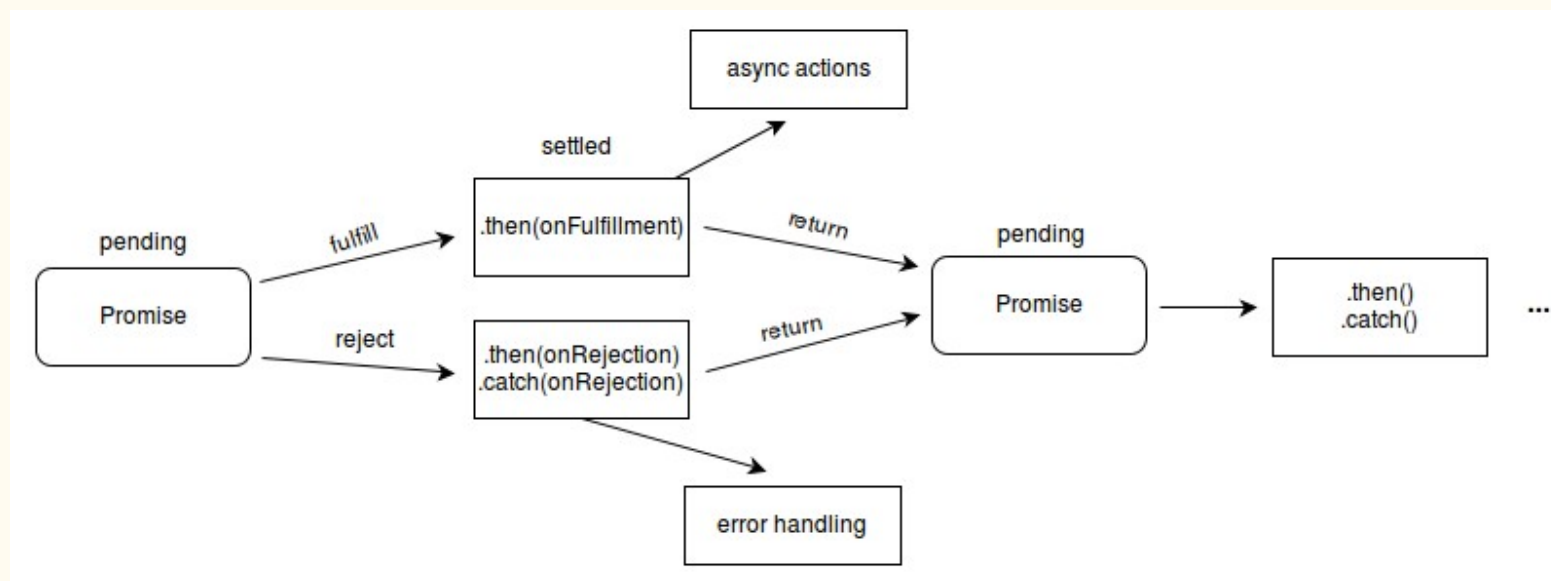
# Promises

- Métodos de manipulação retornam **promises!**
  - Promessas são objetos vão fornecer o resultado futuro de uma ação
  - Uma ação no banco de dados pode demorar muito e teremos o resultado assíncronamente
  - Além disso, a ação pode funcionar corretamente ou dar erro
- Promises vieram para combater o **callback hell**

# Promises

## ● Promises tem quatro estados:

- Pending - está processando e nada ocorre
- Fulfilled - sucesso na operação e chama função de sucesso
- Rejected - falha na operação e chama função de falha
- Settled - foi realizada ou rejeitada



# Promise

- No Mongoose, promise tem método **then** e **catch**
  - .then(fullfield)
    - fullfield é uma função que é chamada quando promise é bem sucedida
  - .catch(rejected)
    - rejected é uma função é chamada quando promise é rejeitada

# Exemplo

```
// app/controllers/alunos.js
var Aluno = require('../models/alunos.js');
module.exports.inserirAluno = function(req, res){
  let promise = Aluno.create(req.body)
  promise.then(
    function(aluno) {
      res.status(201).json(aluno);
    }
  ).catch(
    function(erro){
      res.status(500).json(erro);
    }
  );
}
```



# Exemplo

```
// app/controllers/alunos.js
...
module.exports.listarAlunos = function(req, res){
  let promise = Aluno.find().exec();
  promise.then(
    function(alunos){
      res.json(alunos)
    }
  ).catch(
    function(erro){
      res.status(500).end();
    }
  );
}
```

# Lidando com Relações

---

# Mongoose

- É possível guardar referência (id) de uma entidade em outra
- Mongoose resolve referências para gente
  - Busca objeto relacionado pelo id
- Relação **Aluno - Disciplina**
  - Muitos para muitos
  - Vamos implementar usando uma entidade intermediária **Matrícula**
  - Relação Aluno-disciplina não é o melhor jeito de implementar mas é só para dar o exemplo

# Schema Matricula

```
// app/models/matricula.js
var mongoose = require('mongoose');

module.exports = function(){
  var schema = mongoose.Schema({
    aluno:{
      type: mongoose.Schema.ObjectId,
      ref: 'Aluno'
    },
    disciplina:{
      type: mongoose.Schema.ObjectId,
      ref: 'Disciplina'
    }
  });
  return mongoose.model('Matricula', schema);
}();
```

# Inserção de Matrícula

```
module.exports.inserirMatricula = function(req, res){  
  let promise = Matricula.create(req.body);  
  promise.then(  
    (aluno) =>{  
      res.status(201).json(contato);  
    }  
  ).catch(  
    (erro) =>{  
      res.status(500).json(erro);  
    }  
  );  
}
```

# Listar Disciplinas

```
// app/controllers/disciplina.js
var Matricula = require('../models/matricula.js');
module.exports.listarMatriculas = function(req,res){
  let promise = Matricula.find().exec();
  promise.then(
    function(matriculas) {
      res.json(matriculas)
    }
  ).catch(
    function(erro){
      res.status(500).send('Erro');
    }
  );
}
```

# Listar Disciplinas - Resolvendo Referências

```
// app/controllers/disciplina.js
var Matricula = require('../models/matricula.js');
module.exports.listarMatriculas = function(req, res){
  let promise = Matricula.find()
    .populate('disciplina')
    .populate('aluno')
    .exec();

  promise.then(
    function(alunos) {
      res.json(alunos)
    }
  ).catch(
    function(erro){
      res.status(500).send('Erro');
    }
  );
};
```

} Mongoose busca referências e troca pelo objeto (olhar debug)

Perguntas?

Prof. Victor Farias