

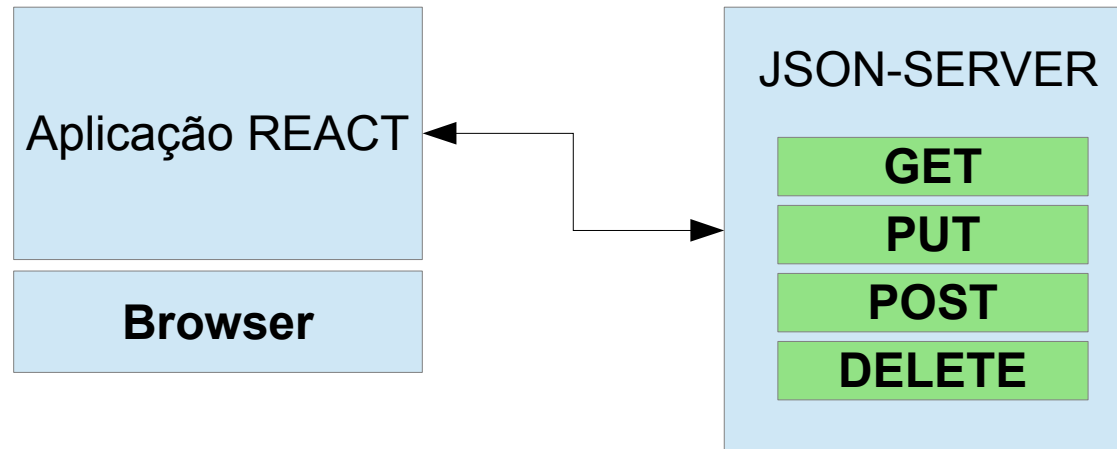
# **Desenvolvimento de Software para WEB**

O Servidor Express Puro

# **Servidor Express**

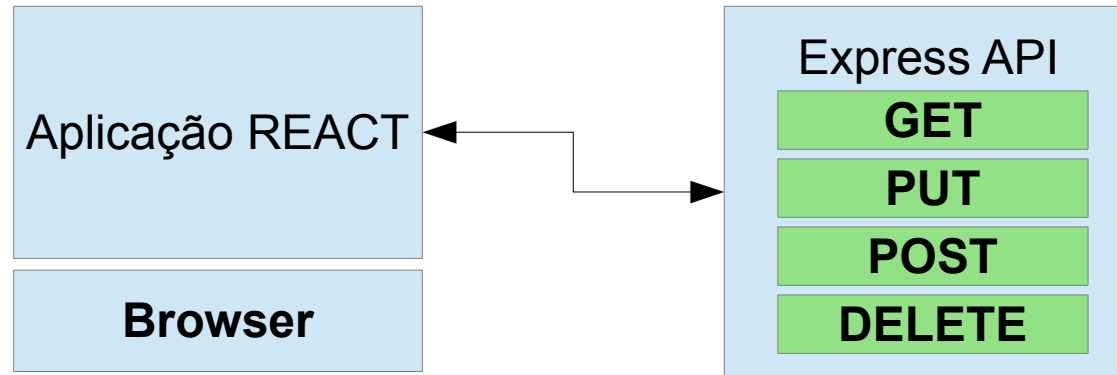
# Introdução

- Com o Express é possível criar uma API Rest que se comunique com a nossa base de dados.
- O Projeto CRUD, até o momento está desta forma:



# Introdução

- Usando o Express, o nosso projeto CRUD ficará assim:



# Introdução

- O Express substitui o json-server, o qual foi usado apenas para fins didáticos e de testes para a aplicação React.
- Como nós queremos futuramente nos comunicar com o MongoDB(ou qualquer outra base de dados), o Express fará o “meio-de-campo”.
- Então, nesta primeira versão do Express, ele **NÃO** comunicará com o MongoDB. Iremos simular um banco simples em memória (usando vetores).

# Introdução

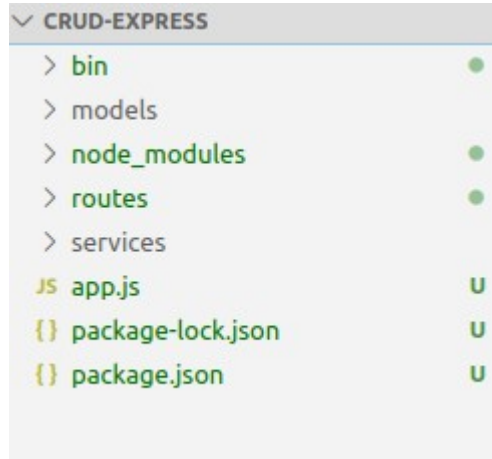
- A ideia por trás deste projeto é criar uma API REST onde será possível:
  - Criar uma entidade (CREATE)
  - Editar uma entidade (EDIT)
  - Listar as entidades criadas (LIST)
  - Apagar uma entidades criada (DELETE)
  - Mostrar uma entidade criada (RETRIEVE)

# Preparando o Projeto Express

- 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 o projeto*)
  - **cd** <nome\_do\_projeto> (*entra na pasta*)
  - **npm install** (*instala as dependências na node\_modules.*)

# Preparando o Projeto Express

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





# Preparando o Projeto Express

- Na pasta bin, abra o arquivo “www” (não se assuste, é um arquivo Javascript).
- Modifique a porta para que não entre em conflito com a porta 3000 do React:

```
/**  
 * Get port from environment and store in Express.  
 */  
  
var port = normalizePort(process.env.PORT || '3002');  
app.set('port', port);
```

# Execução

- Execute o projeto
  - **npm start**
- Abra o navegador e digite o caminho:
  - <http://localhost:3002/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/users”, resultando na URI “/api/v1/users/”.  
Obviamente podemos mudar essa URI como desejarmos.

# EstudanteModel

- Chegou a hora de implementarmos seguindo o padrão MVC-DAO.
- Vamos, inicialmente, criar os nossos “modelos”.
- Modelos, ou Models, são representações das entidades que será salvas em um banco de dados. Em outras palavras, são as tuplas.

# EstudanteModel

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

```
class EstudanteModel{
  constructor(_id,nome,curso,IRA){
    this._id = _id
    this.nome = nome;
    this.curso = curso
    this.IRA = IRA
  }
}

module.exports = EstudanteModel
```

# EstudanteService

- 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:
  - EstudanteService.js

# EstudanteService

```
const EstudanteModel = require('../models/EstudanteModel')
```

```
let estudantes = []
```

```
let _id = 0
```

```
class EstudanteService{
```

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

```
    let estudante = new EstudanteModel(  
      _id++,  
      data.nome,  
      data.curso,  
      data.IRA);
```

```
    estudantes.push(estudante);
```

```
    return estudante;
```

```
  }
```

```
  static list(){
```

```
    return estudantes;
```

```
  }
```

# EstudanteService

```
static update(_id,data){
  for(let e of estudantes){
    if(e._id == _id){
      e.nome = data.nome
      e.curso = data.curso
      e.IRA = data.IRA
      return e;
    }
  }
  return null;
}

static delete(_id){

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



# EstudianteService

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

```
module.exports = EstudianteService
```

# EstudanteRoute

- Uma vez criado o serviço, agora é hora de disponibilizar meios de acesso a ele. Uma dessas formas é a criação de rotas HTTP.
- Na pasta routes, crie o arquivo EstudanteRoute.js

# EstudianteRoute

```
var express = require('express');
var router = express.Router();
var estudianteService = require('../services/EstudianteService');

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

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

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

# EstudianteRoute

```
router.delete('/delete/:id', function (req, res, next) {  
  const ok = estudianteService.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 estudiante = estudianteService.retrieve(req.params.id);  
  return res.json(estudiante);  
});  
  
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 path = require('path');
var cookieParser = require('cookie-parser');
var bodyParser = require('body-parser');

var estudanteRoute = require('./routes/EstudianteRoute');

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('/estudiantes', estudanteRoute);

module.exports = app;
```

# 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 (ARC ou Postman).
  - GET: `http://localhost:3002/estudantes/list`
  - POST: `http://localhost:3002/estudantes/register` (não esqueça do body!)
  - PUT: `http://localhost:3002/estudantes/update/1` (não esqueça do body!)
  - GET: `http://localhost:3002/estudantes/retrieve/1`
  - DELETE: `http://localhost:3002/estudantes/delete/1`