Modularizando Acesso aos Dados

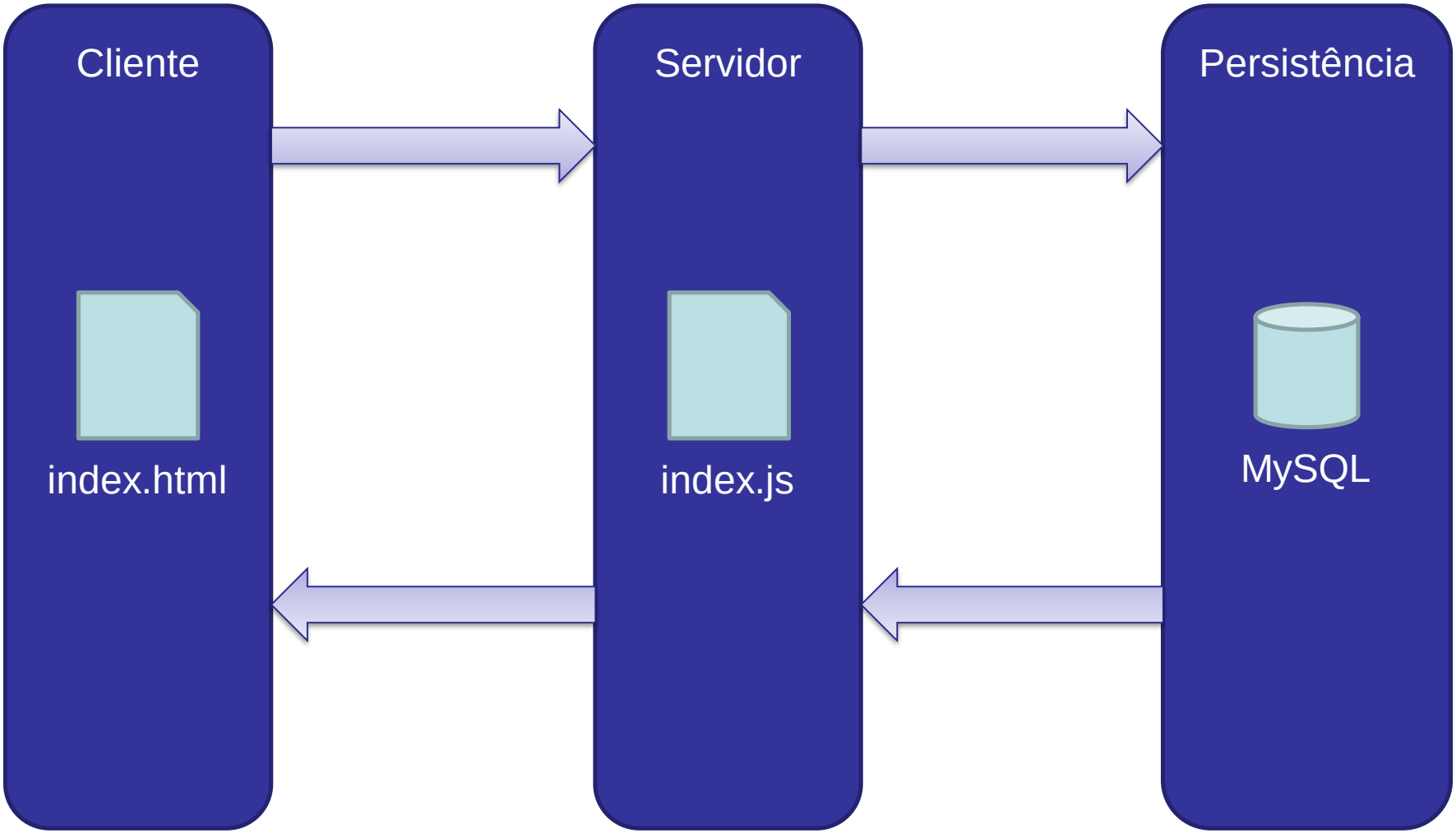
EC021 - Tópicos Avançados II
Sistemas Distribuídos

Prof. Adauto Mendes | E-mail: adauto.mendes@inatel.br

Introdução

- Até o momento toda nossa aplicação foi escrita sem nenhuma separação de módulos (por exemplo, seguindo o que determina o MVC). Tudo em um mesmo arquivo.
- Sabemos que esta abordagem não é uma boa prática independentemente do tipo de projeto, pois prejudica na manutenção do código.

Introdução



Introdução

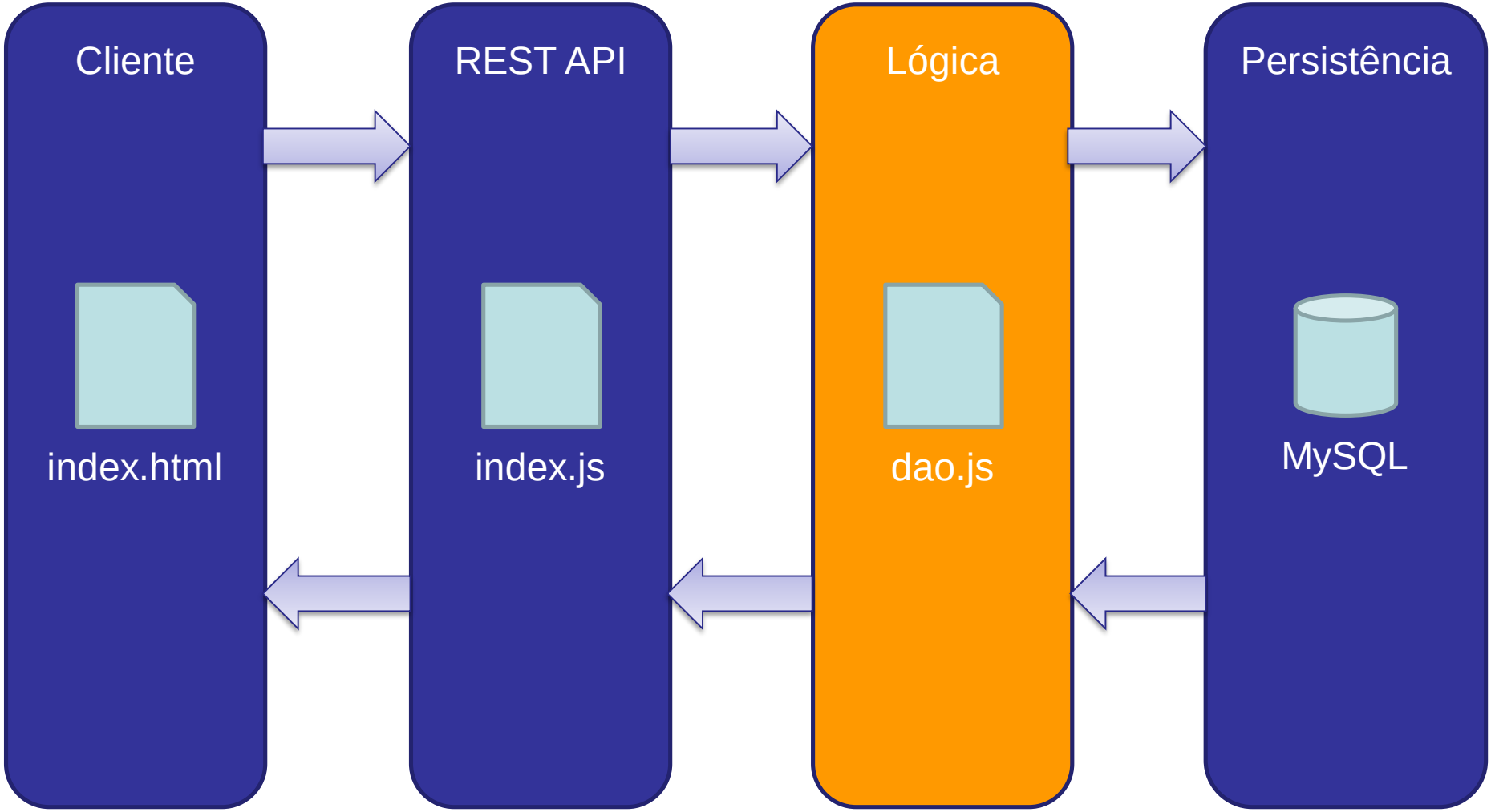
Servidor

index.js

- Rotas
- Funções acessadas pelas rotas
- Lógica de negócio da aplicação
- Conexão com MySQL
- Queries e acesso aos dados

Introdução

- Inicialmente iremos separar a lógica de negócio e acesso aos dados da camada de acesso às rotas (nossa API).



O arquivo dao.js

- Este arquivo irá funcionar como um módulo da nossa aplicação, abstraindo todas as funções de acesso e manipulação dos dados do MySQL. Sendo assim, sempre que o nosso arquivo index.js necessitar de acessar algo no BD ele deverá fazer isto via **dao.js**.

Prática

- Para expor o arquivo **dao.js** como um módulo possível de ser acessado devemos colocar todas as sua funções que serão expostas dentro da estrutura abaixo:

```
module.exports = {

}
```

- No arquivo **index.js** iremos importar como fazemos com as demais dependências:

```
var dao = require(‘./dao');
```

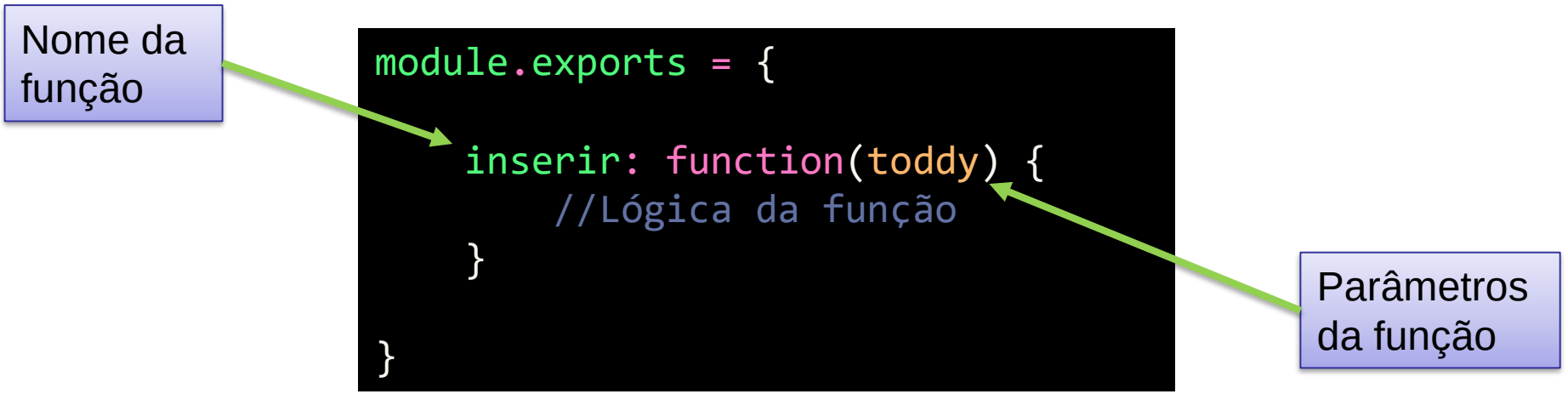
Prática – Conexão com MySQL

- Antes de mover as funções de manipulação das informações do BD, iremos mover toda configuração de conexão para o **dao.js** (como um módulo ele não deve possuir dependência de seus módulos "filhos" como é o caso do index.js):

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

/*
Criando objeto com as credenciais
de conexão com o BD
*/
var con = {
    host: 'localhost',
    user: 'root',
    password: '',
    database: 'ec021'
}

module.exports = {

}
```

Prática - Dependências

- Para criar funções no arquivo dao.js utilizaremos a sintaxe abaixo:



Prática - Inserir

- Mova toda lógica de acesso aos dados para o dao.js:
  - A função deverá receber um objeto ‘toddy’:

```
inserir: function (toddy) {
```

  - Altere a função para que retorne o resultado da query:

```
var result = null;
connection.query(strQuery, function (err, rows, fields) {
    if (!err) { //Se não houver erros
        result = rows; //Retornamos as linhas
    } else { //Caso contrário
        result = err; //Retornamos dados sobre o erro
    }
});

/** Encerrando conexão com o BD */
connection.end();

return result;
```

Prática - Inserir

- No **index.js** altere a função da rota de ‘inserir’ para que invoque o método da dao.js:

```
var toddy = {
    lote: req.body.lote,
    conteudo: req.body.conteudo,
    validade: req.body.validade
}

var result = dao.inserir(toddy);

res.json(result);
```

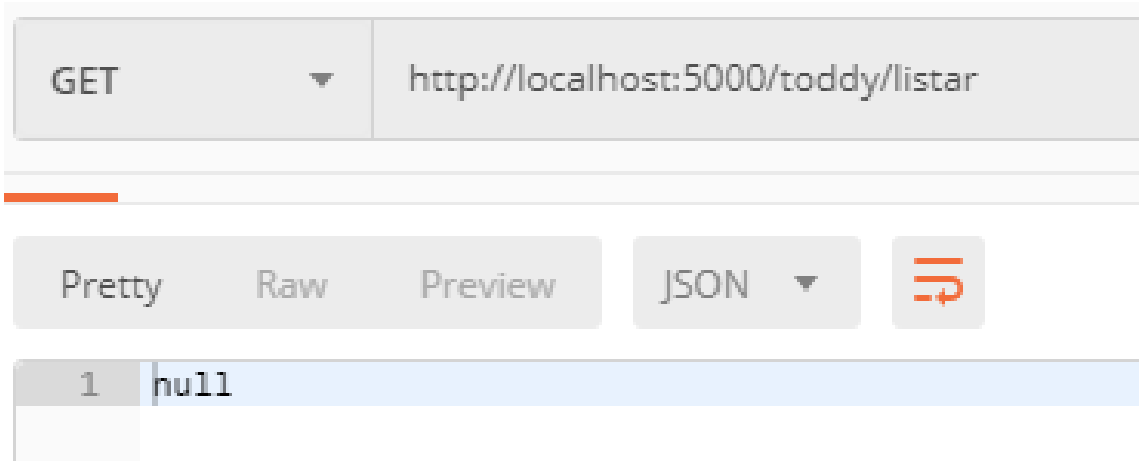
Prática

Agora é a sua vez!

- Refatore os demais métodos da sua API para que todo acesso à persistência seja feito via **dao.js**.
- Feito isto, teste sua API via Postman.

Resultado

- Quando você acessar, por exemplo, o endpoint ‘/listar’ você receberá a seguinte resposta:



Resultado

- Note que no terminal a consulta executada aparece escrita o que quer dizer que as funções da **dao.js** foram chamadas.

```
$ node index.js
CRUD - Parte 2 rodando
SELECT id, lote, conteudo, validade FROM toddy;
```

Por que isto aconteceu?