Desenvolvimento de Aplicações Web com **Ruby on Rails**

Arthur de Moura Del Esposte - esposte@ime.usp.br

Aula 02 - Modelos e Banco de Dados

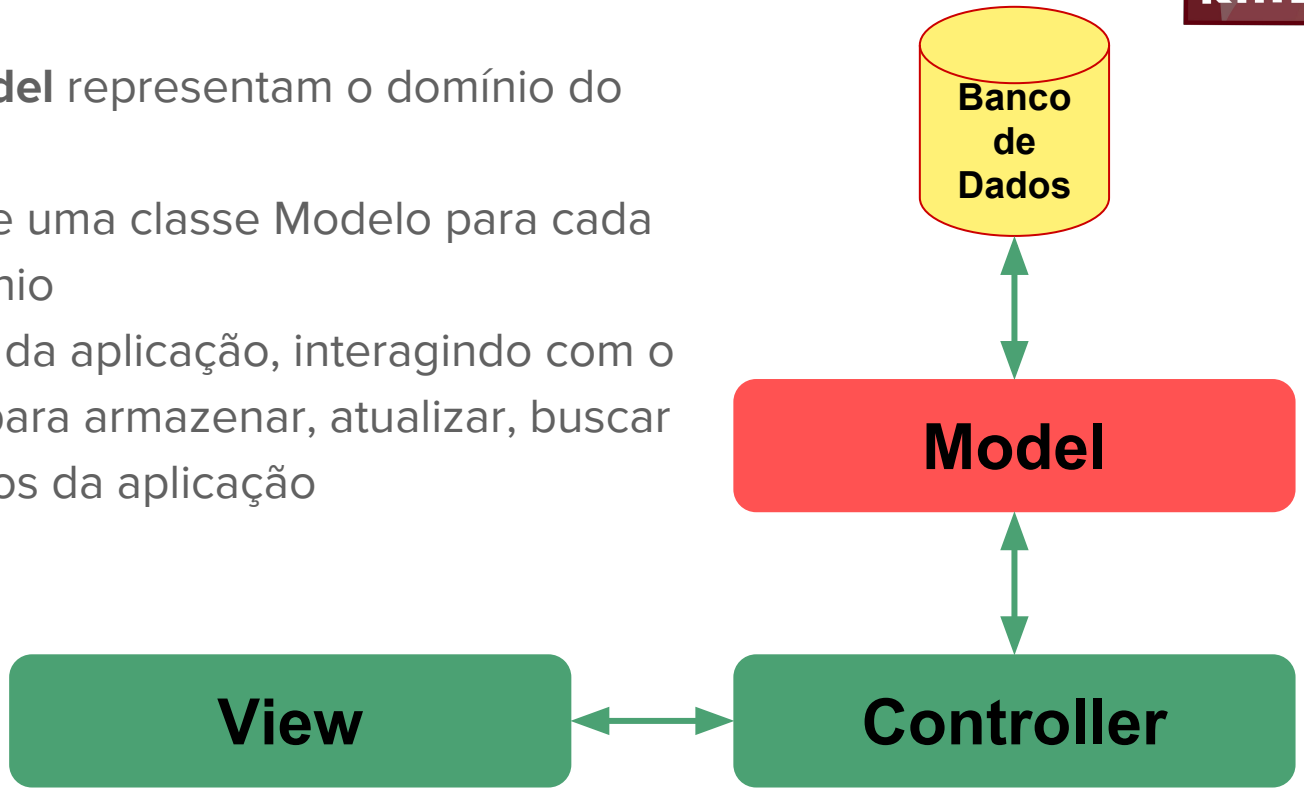
Arthur de Moura Del Esposte - esposte@ime.usp.br

Agenda

- Banco de Dados e Migrações
- Modelos: introdução ao ActiveRecord
- Modelos: CRUD
- Exercício interativo de CRUD com Rails console
- Controladoras e Visões

MVC - **M**odelos

- As classes de **Model** representam o domínio do problema
- Geralmente, existe uma classe Modelo para cada entidade do domínio
- Manipulam dados da aplicação, interagindo com o **banco de dados** para armazenar, atualizar, buscar e remover os dados da aplicação



O Banco de Dados

Bancos de Dados

- Bancos de Dados são sistemas para armazenamento durável de dados
- Em aplicações Web o uso de Banco de Dados é fundamental para o armazenamento de dados de usuários e conteúdos
- **Sistemas de Banco de Dados**, tais como **MySQL**, **Oracle**, **PostgreSQL** ajudam a gerenciar diversos bancos de dados e fornecem interfaces de acesso à esses bancos através de **SQL**
- **SQL - Structured Query Language**

SQL

- **SQL - Structured Query Language**

```
INSERT INTO users (username, email)
  VALUES ("silva", "silva@mail.com"), ("alves", "alves@mail.com")

SELECT * FROM users WHERE username = "silva@mail.com"

UPDATE users SET email = "silva2@mail.com" WHERE username = "silva@mail.com"

DELETE FROM users WHERE id=1
```

SQL

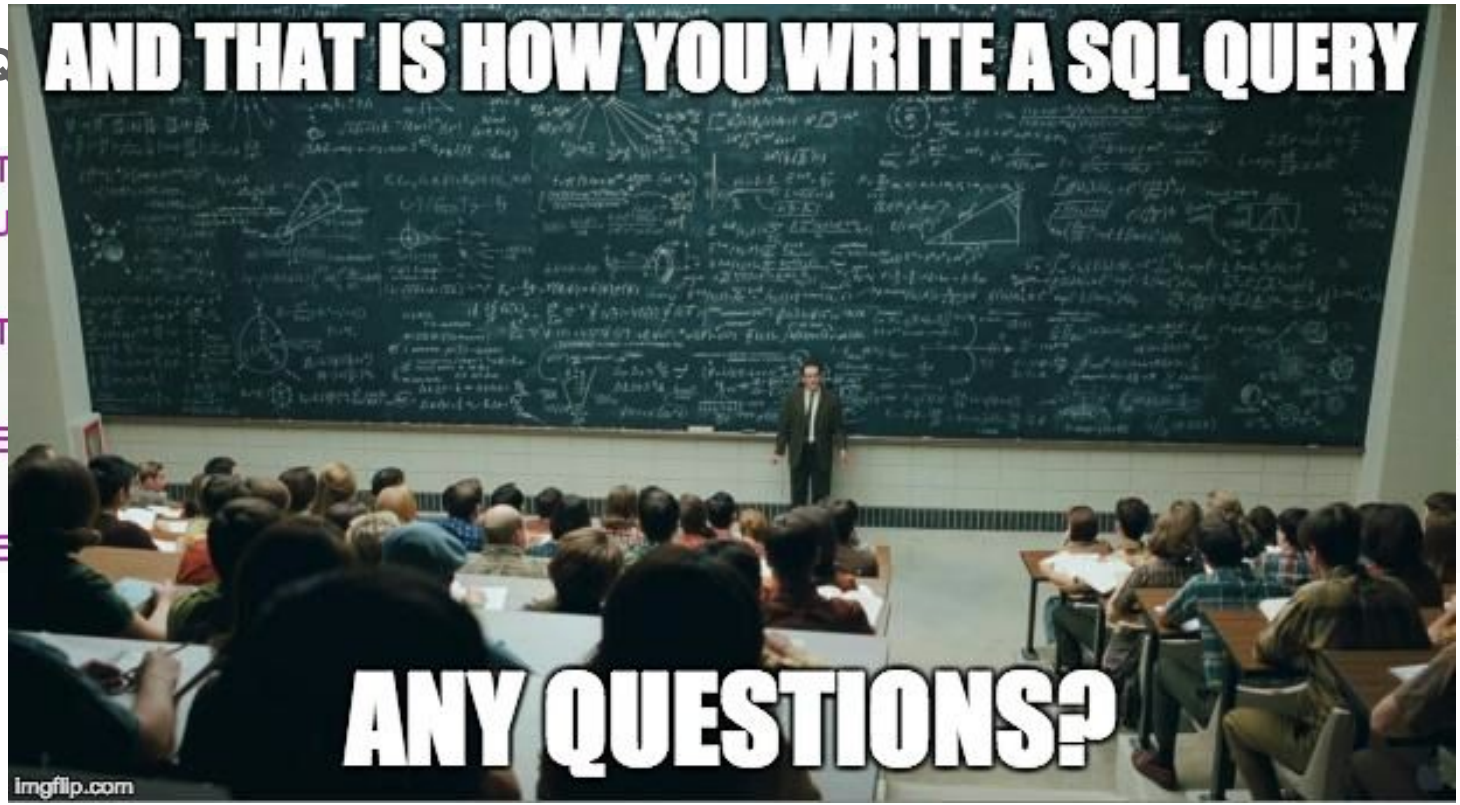
- SQ

```
INSERT
  VALU

SELECT

UPDATE

DELETE
```



Bancos de Dados

- **Banco de Dados Relacionais** oferecem estruturas para armazenamento de dados estruturados, baseados em **tabelas**
- Cada tabela representa uma entidade, onde cada coluna corresponde a um atributo dessa entidade e cada linha é uma instância dessa entidade
- A estrutura de um banco de dados está definido no **Esquema** que consiste em todos os metadados que compõem o banco
- Veja a tabela **Movies** abaixo

id	title	plot	duration	release_date
1	King Kong	“...”	120.4	05/12/2005
2	Lord of The Rings	“...”	182.10	01/01/2002

Rails **abstrai** a maior parte da complexidade no uso de Banco de Dados

SQL em Rails

- Rails gera SQL em tempo de execução, baseado no seu código Ruby
- Realizamos basicamente quatro operações no banco: CRUD
  - **C**reate - criação de novos registros
  - **R**ead - recuperação dos registros existentes
  - **U**pdate - atualização de registros existentes
  - **D**elete - remoção de registros do banco de dados

Rails e Banco de Dados

- Uma das principais vantagens do Rails é a facilidade para acessar o Banco de Dados de forma transparente para o programador
- Como por exemplo, ao criar um projeto, o Rails já cria o arquivo **config/database.yml** com uma configuração pronta para ser usada e facilmente adaptável
- Por padrão, a configuração inicial é para o uso do **SQLite3**, um Sistema de Banco de Dados Relacional leve e ótimo para **Desenvolvimento**
- Você pode alterar essa configuração para utilizar outros sistemas, como o **PostgreSQL** e MySQL

Ambientes do Rails

- Rails define três ambientes que gerenciam separadamente sua cópia de banco de dados
  - ***production*** - banco de dados de produção, com dados reais da aplicação
  - ***development*** - banco de dados que utilizamos para desenvolvimento, onde podemos manipular os registros e modificar sua estrutura sem perdas
  - ***test*** - gerenciado pelas ferramentas de teste. Ele é limpo e repovoado no início de cada execução de testes
- É possível usar diferentes Sistemas de Banco de Dados para os diferentes ambientes
- Para estudar como o Rails lida com o Banco de Dados, vamos criar um novo projeto

Projeto da Disciplina: **MyMovies**

MyMovies

- Para exercitar os conceitos de Rails, ao longo da disciplina iremos desenvolver o projeto **MyMovies** onde usuários poderão gerenciar os filmes que assistiram. O aplicativo deve conter:
- Cadastro/Edição de filmes
- Lista de filmes
- Páginas individuais de filmes
- Páginas de Diretores listando os filmes cadastrados desse diretor
- Páginas dos Atores, listando os filmes em que eles atuaram
- Login de Usuário
- Usuário poderá ter uma lista de filmes assistidos
- Usuário poderá classificar os filmes assistidos

Classes Modelos

- **Quais seriam algumas das classes Modelos que nosso sistema MyMovies deveria conter?**
- **Como essas classes se relacionam?**

MyMovies - Criando o Projeto

- **Crie um projeto na sua conta do Github**
- Posteriormente, crie o projeto Rails em seu computador na sua pasta de trabalho:

```
$ rails new my_movies

$ cd my_movies
```

MyMovies - Criando o Projeto

- Inicialize um repositório git no projeto para que possamos versionar o desenvolvimento do nosso projeto

```
$ git init
```

- Faça um commit com todas a estrutura básica criada pelo Rails

```
$ git add .

$ git commit -m “Initial commit”
```

- Adicione o repositório criado do Github como referência remota

```
$ git remote add origin <url_para_o_repositório>
```

MyMovies - Criando o Projeto

- Baixe os arquivos já criados pelo Github para que possamos subir a nossa versão para o repositório

```
$ git pull origin master
```

- Atualize suas modificações no Github empurrando seus commits para lá:

```
$ git push origin master
```

- Verifique se seus arquivos se encontram no Github

Criando o banco

- Para criar o banco de dados execute:

```
$ rake db:create
```

- Esse comando usa as configurações do arquivo config/database.yml para criar os bancos de dados de teste e desenvolvimento
- Note que inicialmente esse banco não tem nada (tabelas, registros)

Migrações

- O Rails consegue criar uma estrutura de banco de dados de duas formas: carregando um **Esquema do Banco** existente (db/schema.rb) ou através de **Migrações**
- **Migrações** são scripts que descrevem mudanças no esquema de banco de dados (layout de tabelas e colunas) de uma maneira consistente e **repetível (DRY)**
- Semelhante ao uso do Gemfile pelo **Bundler**
- Para mudar o esquema com migrações faça:
    a. Crie uma migração descrevendo quais mudanças fazer. O rails fornece um gerador de migrações que fornece o código básico para descrever migrações.
    b. Aplique a migração no banco de dados (execute a migração). Existe uma **rake task** para isso
    c. Caso a migração seja aplicada, atualize o esquema do banco de dados de testes

Gerando uma migração

- Para gerar a estrutura básica de uma migração, nós chamamos o rails e definimos o nome da migração. Veja o exemplo para criarmos a tabela **Movies**:

```
$ rails generate migration create_movies
```

- Veja os arquivos gerados
- Um arquivo de migração tem um **timestamp** e o nome da migração
- O timestamp é usado pelo Rails para saber a ordem em que as migrações devem ser aplicadas ao banco de dados
- Dessa forma, nós podemos recriar a estrutura do banco de dados várias vezes

Criando a tabela **Movies**

- Adicione o seguinte código à sua migração:

```
class CreateMovies < ActiveRecord::Migration
  def change
    create_table :movies do |t|
      t.string :title
      t.text :description
      t.datetime :release_date
      t.timestamps
    end
  end
end
```

Criando a tabela **Movies**

- Salve e aplique a migração:

```
$ rake db:migrate
```

- Caso queira reverter a última migração aplicada pode-se executar:

```
$ rake db:rollback
```

- Nem todas as migrações podem ser revertidas!

Schema

- Após aplicar as migrações, o Rails modifica o esquema do banco de dados, que é refletido no arquivo **db/schema.rb**
- O esquema pode ser usado para carregar a estrutura do banco em outros bancos de dados (ex: **production**)
- O **Schema** também possui o timestamp da última versão de migração aplicada
- Por enquanto, modificamos o banco de dados, mas ainda não temos as classes para acessar as tabelas criadas!

Modelos: Introdução ao ActiveRecord

Tabelas VS Objetos

- Nós sempre iremos trabalhar com instâncias de uma Entidade
  - O filme Exterminador do Futuro é uma instância da entidade Filme (Movie)
  - Angelina Jolie é uma instância da entidade Ator (Actor)
- Cada instância tem duas representações diferentes
  - **Objeto**: referência na memória que pode ser manipulada pela linguagem de programação cujo comportamento é definido por uma Classe
  - **Registro no Banco de Dados**: registro persistente em uma linha na tabela do banco de dados correspondente à entidade daquela instância, onde cada coluna possui o valor de algum atributo dessa instância.

Tabelas VS Objetos

- Banco de dados **Relacionais** armazenam seus registros em estruturas de **tabelas** que mantém referências à outras tabelas para definir relações entre entidades
- Por outro lado, quando tratamos esses registros em nível de programação, utilizamos o conceito de Objetos!
- **Portanto, precisamos ter mecanismos que mapeiam a nossa representação de objetos para as tabelas no banco de dados**

ActiveRecord

- O Rails faz o mapeamento de Objetos para as Entidades do Banco de Dados usando o **ActiveRecord**
- ActiveRecord é um padrão de projeto onde uma única instância de uma classe Modelo corresponde a uma única linha em uma tabela específica no Banco de Dados
- Além disso, o ActiveRecord define que a própria classe Modelo define os métodos que realizam operações na representação do objeto que está armazenado no banco de dados: **CRUD**

ActiveRecord

- Dessa forma, a model **Movie** terá a implementação dos métodos para realizar CRUD no banco de dados sobre a tabela **movies**
- Não se preocupe em implementar esses métodos! Todas as classes Modelos vão herdar da classe **ActiveRecord::Base** do Rails que já implementam esses métodos de acesso ao banco
- Então, crie o arquivo **app/model/movie.rb** onde iremos definir nossa primeira classe modelo: **Movie**

ActiveRecord

- Graças à **convenção sobre configuração**, o Rails infere automaticamente que a classe **Movie (no singular)** corresponde a tabela **movies (no plural)**
- A nossa classe **Movie** é a nossa conexão com a tabela **movies** no banco de dados
- Vamos verificar os comportamentos herdados de ActiveRecord::Base através do console do Rails:

```
$ rails console
```

- O **Rails Console** é um interpretador interativo (semelhante ao IRB) onde podemos acessar todo o ambiente do Rails

Console - **C**RUD com a modelo Movie

- starwars = Movie.create(title: "Star Wars", release_date: "25-04-1977")
- avatar = Movie.create!(title: "Avatar", release_date: "10-12-2009")
- requiem = Movie.new(title: "Requiem for a Dream")
- requiem.save!
- requiem.title
- requiem.release_date = "30-04-2012"
- requiem.save

Observações sobre o **Create**

- Na criação, os métodos **new** e **save** são diferentes
- Cada objeto salvo no banco de dados recebe um **id**
- O Rails já cria os atributos do objeto baseado nas colunas da tabela do banco de dados
- Além disso, ela já cria os métodos de acesso automaticamente para esses atributos
- **É importante reparar na diferença do registro no banco de dados com sua cópia na memória**

Console - CRUD com a modelo Movie


```
movies = Movie.where(title: "Avatar")
new_movies = Movie.where("release_date > :cutoff", cutoff: '2010-10-10')
old_movies = Movie.where("release_date < :cutoff", cutoff: '2010-10-10')
specific_movie = Movie.find(2)
specific_movie = Movie.find(-10) # => ActiveRecord::RecordNotFound
movie_by_name = Movie.find_by_title("Star Wars")
nil_movie = Movie.find_by_title("Nonexistent")
all_movies = Movie.all
```

Observações sobre o **Read**

- A busca com o **find** sempre retorna um único objeto
- A busca com **where** retorna uma relação que se comportam como uma coleção de objetos enumeráveis
- Usando metaprogramação, o Rails adiciona alguns métodos importantes para busca baseado nos atributos da classe: **find_by**_attribute
- Outros métodos também são construídos da mesma forma:
  - Movie.find_all_by_release_date
  - Movie.find_by_title_and_release_date
- Por que **where** e **find** são métodos de classe ao invés de meodo de instância?

Console - CR**U**D com a modelo Movie


```
### UPDATE
movie = Movie.find 1
movie.update_attributes(description: 'A space western')
movie.release_date = "25-05-1977"

### DELETE
movie.destroy
Movie.destroy_all
```

Suponha que façamos `movie = Movie.where(title: "Star Wars")` e então em outro console alguém modifica a **description** desse filme diretamente no banco. Quando isso ocorre, o valor de movie:

1. Será atualizado automaticamente, pois um model ActiveRecord “conecta” sua aplicação com o banco de dados
2. Será atualizado automaticamente, pois o ActiveRecord usa metaprogramação
3. Não será atualizado automaticamente, mas pode ser atualizado manualmente re-executando `movie = Movie.where(title: "Star Wars")`
4. Será indefinido ou se tornará nil (Nulo)

Suponha que façamos `movie = Movie.where(title: "Star Wars")` e então em outro console alguém modifica a **description** desse filme diretamente no banco. Quando isso ocorre, o valor de movie:

1. Será atualizado automaticamente, pois um model ActiveRecord “conecta” sua aplicação com o banco de dados
2. Será atualizado automaticamente, pois o ActiveRecord usa metaprogramação
3. ✓ Não será atualizado automaticamente, mas pode ser atualizado manualmente re-executando `movie = Movie.where(title: "Star Wars")`
4. Será indefinido ou se tornará nil (Nulo)

Populando o Banco

- Para o desenvolvimento é muito útil popular o banco com alguns dados
- Para isso, podemos criar instâncias das nossas classes modelos no arquivo **db/seeds.rb**

```
9   movies = [
10    {title: "Alladin", release_date: '25-Nov-1992'},
11    {title: "When Harry Met Sally", release_date: '21-Jul-1989'},
12    {title: "The Help", release_date: '10-Aug-2011'},
13    {title: "Raiders of the Lost Ark", release_date: '12-Jun-1981'}
14  ]
15
16  movies.each do |movie|
17    Movie.create!(movie)
18  end
```

- Para semear o banco, basta executar **rake db:seed**

Gerando Modelos

Gerando Modelos com Rails

- O Rails possui alguns scripts que geram código automaticamente para agilizar o desenvolvimento
- Vamos utilizar o Rails para gerar a modelo **Professional**
- Como ambos **Actor** e **Director** possuem atributos semelhantes, ambos serão uma especialização da classe **Professional**
- Inicialmente, vamos gerar a model Professional:

```
$ rails generate model Professional name:string birthdate:datetime gender:string country:string type:string
```

Gerando Modelos com Rails

- Essa forma de criar models já gera:
  - Migração
  - Classe Modelo
  - Testes Unitários
- Rails usa convenção para gerar o nome da migração e das classes!
- Sempre abra a migração antes de aplicar para ver se tudo está correto

```
$ rake db:migrate
```

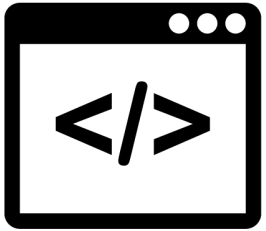
Herança com Rails

- Crie as classes **Actor** e **Director**. Ambas devem herdar de **Professional**
- Com isso, o Rails utiliza a mesma tabela para registrar tanto os atores quanto os diretores, porém usa a coluna **type** que criamos para distinguir o tipo específico de um registro

Console - Herança

- Actor.create!(name: "Angelina Jolie", gender: "female")
- Actor.create!(name: "Will Smith", gender: "male")
- Director.create!(name: "Quentin Tarantino", gender: "male")
- Actor.count
- Director.count
- Professional.count
- Professional.where(gender: "male")
- Actor.where(gender: "male")

Exercício



- Altere o arquivo db/seed.rb para alimentar alguns atores e diretores no banco de dados:
  - Atriz: Angelina Jolie
  - Ator: Will Smith
  - Atriz: Margot Robbie
  - Diretor: Quantin Tarantino
  - Diretor: Mel Gibson

MyMovies - Commitando Modificações

- Registre uma nova versão com as modificações feitas até agora:

```
$ git add .

$ git commit -m “Initial commit”
```

- Atualize suas modificações no Github empurrando seus commits para lá:

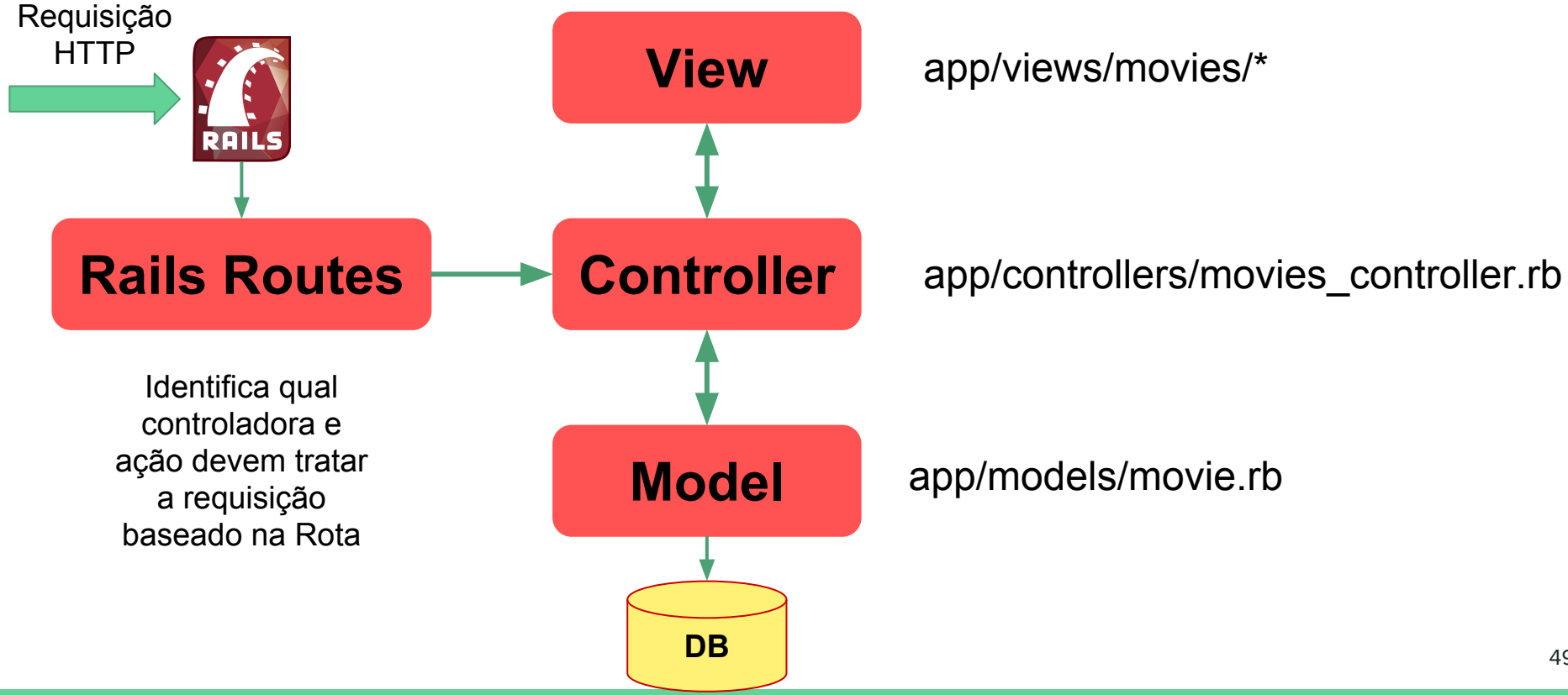
```
$ git push origin master
```

Acessando as Modelos através de Controladoras e Views

Acessando as Modelos

- No geral, queremos sempre fazer quatro operações com as classes Modelos - **CRUD**
- Para isso, temos que criar as páginas necessárias para que usuários possam interagir com a aplicação e manipular nossas classes
- Para uma mesmo Modelo, temos uma única Controladora que responde à todos as requisições comuns às operações CRUD e apresentação dos recursos
- Por outro lado, para um mesmo Modelo e Controladora, temos várias Visões diferentes que permitem:
    - Listar todos os filmes
    - Mostrar a página de um filme
    - Página para criar um novo filme

MVC


Requisição
HTTP
RAILS
Rails Routes
Identifica qual
controladora e
ação devem tratar
a requisição
baseado na Rota
View
Controller
Model
DB
app/views/movies/*
app/controllers/movies_controller.rb
app/models/movie.rb

Gerando a Controladora e Visão

- Podemos facilitar a criação da Controladora e Visão utilizando o gerador de código do Rails!

```
$ rails generate controller Movies index show new create
```

- Veja os arquivos criados
- A ação **index** irá renderizar a lista de todos os filmes
- A ação **show** irá renderizar a página de um filme específico
- A ação **new** irá renderizar um formulário para criar um novo filme
- A ação **create** irá tratar a criação do filme e redirecionar para a página do novo filme cadastrado

Comunicação entre Controladores e Visão

- As Controladoras podem acessar as classes Modelos
- Implemente a ação Movies#index da seguinte forma:

```
def index
  @movies = Movie.all
end
```

- A Visão tem acesso as variáveis definidas na Controladora correspondente
- Para isso, os arquivos de Visão precisam fazer chamadas ao código Ruby
- Portanto, ao invés de trabalhar com arquivos HTML puros, vamos trabalhar com arquivos cuja extensão será **.html.erb**
- **ERB = Embedded Ruby**
- Ou seja, podemos ter código Ruby nesse tipo de arquivo que será posteriormente transformado em um HTML puro para responder ao cliente

Comunicação entre Controladores e Visão

- Adicione o seguinte código no arquivo **app/views/movies/index.html.erb**

```
<h1>Movies</h1>

<table>
  <tr>
    <th>Title</th>
    <th>Release</th>
  </tr>
  <% @movies.each do |movie| %>
    <tr>
      <td><%= movie.title %></td>
      <td><%= movie.release_date %></td>
    </tr>
  <% end %>
</table>
```

MyMovies - Commitando Modificações

- Registre uma nova versão com as modificações feitas até agora:

```
$ git commit -am “Initial commit”
```

- Atualize suas modificações no Github empurrando seus commits para lá:

```
$ git push origin master
```

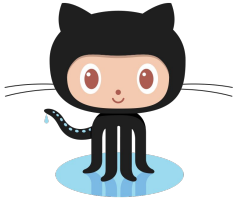
Atividades Sugeridas!

Exercícios/Desafios de Programação

1. Leia a documentação de Rails para Migrações e crie uma nova migração que adiciona os campos **budget** (orçamento) e **box_office** (receita) na tabela movies. O que acontece com os filmes que já estavam no Banco de Dados?
2. Crie a classe **Award** (Prêmio) com os atributos **name** e **category** que no futuro serão utilizados para apresentar os prêmios que um filme ganhou (ex: Oscar)
3. Crie a página que liste todos os Atores
4. Crie a página que liste todos os Diretores
5. Estude como você pode estar utilizando o CSS em Rails para melhorar as páginas criadas e então aplique melhorias visuais na página movies/index

Contato

https://gitlab.com/arthurmde



https://github.com/arthurmde

http://bit.ly/2jvND12

http://bit.ly/2j0llo9

Centro de Competência em Software Livre - CCSL

esposte@ime.usp.br

Obrigado!