

# INE5646 Programação para Web

- Tópico :

Processamento no Cliente - MVC

(estes slides fazem parte do material didático da disciplina  
INE5646 Programação para Web)

# Sumário

- Navegador como plataforma
- Padrão de Projeto MVC
- MVC e aplicações para web
- Backbone.js
- Backbone.js: exemplo

# Navegador como Plataforma

- A expansão da web 2.0 torna cada vez mais importante o processamento na camada 1 (navegador):
  - Se no início (web 1.0) o papel do navegador era apenas renderizar um documento HTML agora ele funciona como uma plataforma para execução de programas.
- O desenvolvimento de programas que são executados nos navegadores deve seguir as mesmas recomendações válidas para aplicações tradicionais:
  - Quanto mais complexo for o programa mais difícil (caro) será sua manutenção.

# Padrão de Projeto MVC

- O padrão de projeto MVC (Model-View-Control) facilita a manutenção de uma aplicação porque prega a separação do problema de visualização dos dados do problema de representação destes dados.
- O papel do controle é garantir a sincronia entre os dados e sua representação: alterações que ocorrem nos dados, decorrentes da execução de um algoritmo, devem ser refletidas na visualização dos dados. Da mesma forma, alterações decorrentes de ações do usuário (edição de uma informação) devem ser propagadas para o modelo.
- Este padrão de projeto existe há muito tempo, muito antes do surgimento das aplicações para web.

# MVC e aplicações para web

- MVC é apenas um padrão de projeto. É aplicado de formas diferentes mas sempre com o mesmo objetivo.
- Na web 1.0 o padrão é usado na camada 2. Exemplo típico no contexto Java:
  - **M**: classes Java
  - **V**: páginas JSP ou JSF
  - **C**: servlet
- Na web 2.0 o padrão tende a ser mais usado na camada 1.

# MVC e aplicações para web

- Há duas abordagens para uso do padrão MVC na camada 1:
  - O padrão é uma filosofia de trabalho e o desenvolvedor define a sua maneira de como representar os 3 elementos. Em outras palavras, o padrão é um compromisso que o desenvolvedor assume.
  - O padrão é materializado em uma biblioteca ou framework. O desenvolvedor, neste caso, é “obrigado” a seguir a filosofia MVC adotada pela biblioteca ou framework.
- Independentemente da abordagem, quatro tecnologias estão tipicamente envolvidas no padrão MVC:
  - HTML/JSON/Javascript: representação dos dados (M)
  - CSS: aparência dos dado (V)
  - Javascript: sincronização dos dados com sua aparência (C)

# Backbone.js

- O backbone.js é uma biblioteca Javascript para implementação da camada 1 de aplicações para web.
- Site: <http://backbonejs.org/>
- MVC:
  - **M**: extensão da classe Backbone.Model
  - **V**: extensão da classe Backbone.View + HTML + CSS + jQuery
  - **C**: incorporado na view via funções Javascript

# Backbone.js

- Por ser uma biblioteca (e não um framework) o desenvolvedor tem bastante liberdade para construir suas aplicações.
- **Eventos:**
  - Objetos podem gerar e reagir a eventos.
  - A sincronização entre os modelos e as views pode ser feita via módulo de eventos. Na situação típica, uma view atua como um “listener” para eventos gerados pelo modelo: sempre que o valor de um atributo do modelo for alterado a view executará uma função (normalmente com o objetivo de atualizar o que é mostrado na página HTML).



# Backbone.js: Exemplo

- Na aplicação exemplo, o objetivo é mostrar uma relação de filmes cadastrados (obtidos da camada 2). O usuário seleciona um dos filmes e a aplicação mostra alguns dados (título original, ano de lançamento e o cartaz (imagem)).
- A aplicação, bastante simples, é do tipo SPA (single page application).
- A aplicação explora apenas uma parte das possibilidades do Backbone, o suficiente para dar uma ideia de como ele funciona.

# Backbone.js: Exemplo

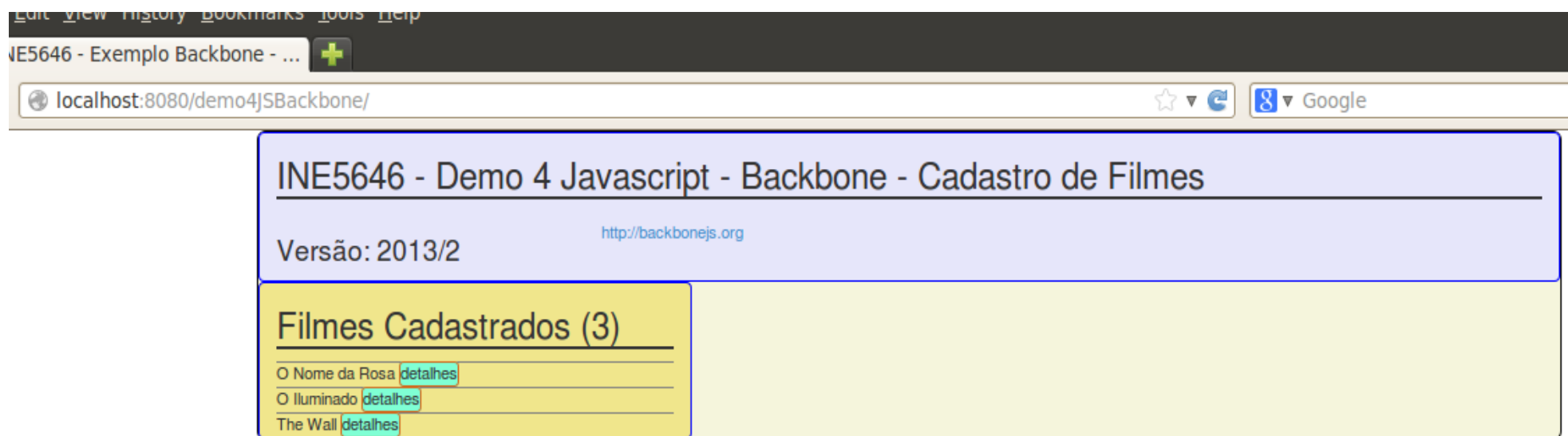
- Arquivos envolvidos na **camada 1**:
  - **index.html**: única página da aplicação. Ao ser carregada, traz também os demais arquivos (CSS e Javascript). Utiliza Bootstrap 3.0 para definição do layout da página.
  - **css/bootstrap.min.css**: estilos do Bootstrap
  - **css/estilos.css**: estilos específicos da aplicação
  - **imgs/\*.jpg**: imagens das capas dos 3 filmes
  - **js/lib/jquery-2.0.3.min.js**: jQuery, necessário para o Backbone
  - **js/lib/underscore-min.js**: Underscore, necessário para o Backbone
  - **js/lib/backbone-min.js**: Backbone
  - **js/app.js**: implementação dos modelos e views da aplicação.

# Backbone.js: Exemplo

- Arquivos envolvidos na **camada 2**:
  - **Filme.java**: classe Java que representa um filme. Esta classe contém apenas atributos e métodos setters/getters.
  - **Servicos.java**: servlet que implementa um serviço do tipo REST. O serviço consiste em retornar um array de objetos JSON contendo os 3 filmes cadastrados.
    - Obs: a transformação de objetos Java em objetos JSON é feita usando-se a biblioteca **Gson** (<http://code.google.com/p/google-gson/>).




# Backbone.js: Exemplo

- Conteúdo inicial de index.html:



# Backbone.js: Exemplo

- Ao clicar em “detalhes” ao lado de “O nome da rosa” surge a descrição do filme:

no4JSBackbone/    Google

---

INE5646 - Demo 4 Javascript - Backbone - Cadastro de Filmes

Versão: 2013/2 <http://backbonejs.org>

### Filmes Cadastrados (3)

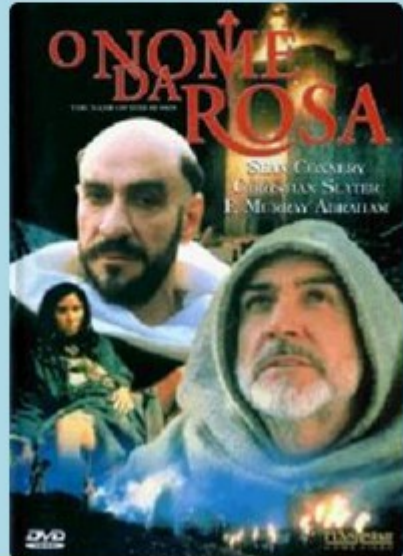
O Nome da Rosa	<a href="#">detalhes</a>
O Iluminado	<a href="#">detalhes</a>
The Wall	<a href="#">detalhes</a>

### Descrição do Filme

Título: **O Nome da Rosa**

Título Original: **Der name der rose**

Ano: **1986**



# Backbone.js: Exemplo – Filme.java

- Um POJO tradicional.

```
Filme.java
1 package demo4jsbackbone.servicos.dados;
2
3 public class Filme {
4     String id;
5     String tituloEmPortugues;
6     String tituloOriginal;
7     String genero;
8     Integer ano;
9     String cartaz;
10
11     public Filme(String id, String tituloEmPortugues, String tituloOriginal, String genero, Integer ano, String cartaz) {
12         this.id = id;
13         this.tituloEmPortugues = tituloEmPortugues;
14         this.tituloOriginal = tituloOriginal;
15         this.genero = genero;
16         this.ano = ano;
17         this.cartaz = cartaz;
18     }
19
20     public String getId() {
21         return id;
22     }
23 }
```

# Backbone.js: Exemplo – Servicos.java

- Serviços é um servlet.

```
Servicos.java x
1 package demo4jsbackbone.servicos;
2
3 import com.google.gson.Gson;
4 import demo4jsbackbone.servicos.dados.Filme;
5 import java.io.IOException;
6 import java.io.PrintWriter;
7 import java.util.ArrayList;
8 import java.util.List;
9 import javax.servlet.ServletException;
10 import javax.servlet.annotation.WebServlet;
11 import javax.servlet.http.HttpServlet;
12 import javax.servlet.http.HttpServletRequest;
13 import javax.servlet.http.HttpServletResponse;
14
15 @WebServlet(name = "Servicos", urlPatterns = {"/filmes"})
16 public class Servicos extends HttpServlet {
17
```

# Backbone.js: Exemplo – Servicos.java

- Quando o servlet receber uma requisição do tipo GET retornará um array de filmes no formato JSON.

```
18  @Override
19  protected void doGet(HttpServletRequest request, HttpServletResponse response)
20      throws ServletException, IOException {
21      response.setContentType("application/json;charset=UTF-8");
22      PrintWriter out = response.getWriter();
23      try {
24          Gson gson = new Gson();
25          String arrayDeFilmes = gson.toJson(obtenhaFilmes());
26          out.print(arrayDeFilmes);
27      } finally {
28          out.close();
29      }
30  }
```



# Backbone.js: Exemplo – Servicos.java

- No exemplo, por uma questão de simplicidade, os filmes são gerados a cada requisição. Em uma aplicação completa deveriam, por exemplo, estar armazenados em um banco de dados.

```
32 private List<Filme> obtenhaFilmes() {  
33     List<Filme> filmes = new ArrayList<Filme>();  
34     filmes.add(new Filme("1", "O Nome da Rosa", "Der name der rose", "policial", 1986, "o_nome_da_rosa.jpg"));  
35     filmes.add(new Filme("2", "O Iluminado", "The Shining", "terror", 1980, "o_iluminado.jpg"));  
36     filmes.add(new Filme("3", "The Wall", "The Wall", "musical", 1982, "the_wall.jpg"));  
37  
38     return filmes;  
39 }
```

# Backbone.js: Exemplo – estilos.css

- Estilos CSS usados na aplicação.

```
9  #app {
10    border: black thin solid;
11    border-radius: 8px;
12    background-color: beige;
13  }
14
15  #cabecalho {
16    border: blue thin solid;
17    border-radius: 5px;
18    background-color: lavender;
19  }
20
21  #conteudo {
22  }
23
24  #filmes {
25    border: blue thin solid;
26    border-radius: 5px;
27    background-color: khaki;
28  }
```

```
30  #detalhes {
31    border: blue thin solid;
32    border-radius: 5px;
33    background-color: lightblue;
34  }
35
36  .mostrarDetalhes {
37    border: chocolate thin solid;
38    border-radius: 4px;
39    background-color: aquamarine;
40    cursor: pointer;
41  }
42
43  .fechar {
44    background-color: aqua;
45    font-size: 0.5em;
46    cursor: pointer;
47    border: black solid;
48  }
49
50  h2 {
51    border-bottom-style: solid;
52    border-bottom-width: medium;
53  }
54
55  .filme {
56    border-top: gray thin solid;
57  }
```

# Backbone.js: Exemplo – index.html

- HTML5 (linha 1).

```
index.html x
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>INE5646 - Exemplo Backbone - Cadastro de Filmes</title>
5     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
6     <link href="css/estilos.css" type="text/css" rel="stylesheet">
7     <link href="css/bootstrap.min.css" type="text/css" rel="stylesheet">
8   </head>
9   <body>
```

# Backbone.js: Exemplo – index.html

- Utilização de divs para definir o layout da interface com o usuário.
- As classes CSS (container, row, col-sm-12, col-sm-4 e col-sm-8) são definidas pelo Bootstrap e implementam o conceito de design responsivo (se adaptam automaticamente ao tamanho da tela).

```
11 <div class="container">
12   <div id="app" class="row">
13     <div id="cabecalho" class="col-sm-12"></div>
14     <div id="filmes" class="col-sm-4"></div>
15     <div id="detalhes" class="col-sm-8"></div>
16   </div>
17 </div>
```

# Backbone.js: Exemplo – index.html

- Cada template define um fragmento de código HTML que representa a aparência de uma view. As expressões delimitadas por `<%` e `%>` definem a parte dinâmica do template. Equivalem aos scriptlets de JSP.
- O mecanismo de template usado no exemplo é o que existe na biblioteca Underscore. Mas pode-se usar qualquer outra biblioteca (Mustache, Ember, etc).

```
20 <script id="cabecalhoTemplate" type="text/template">
21   <div class="row">
22     <div class="col-sm-12"><h2><%= titulo %></h2></div>
23     <div class="col-sm-3"><h3>Versão: <%= versao %></h3></div>
24     <div class="col-sm-9"><h5><a href="<%= link %>" target="_blank"><%= link %></a></h5></div>
25   </div>
26 </script>
27
28 <script id="filmeResumidoTemplate" type="text/template">
29   <%= tituloEmPortugues %> <span class="mostrarDetalhes">detalhes</span>
30 </script>
31
```

# Backbone.js: Exemplo – index.html

- A inclusão de todos os templates dentro do arquivo index.html é apenas uma conveniência. Cada template poderia ser definido em um arquivo Javascript próprio.

```
32 <script id="filmeCompletoTemplate" type="text/template">
33   <div class="row">
34     <div class="col-sm-12">
35       <h2>Descrição do Filme <span class="fechar">X</span></h2>
36     </div>
37     <div class="col-sm-6">
38       <h3>Título: <span class="label label-primary"><%= tituloEmPortugues %></span></h3>
39       <h3>Título Original: <span class="label label-primary"><%= tituloOriginal %></span></h3>
40       <h3>Ano: <span class="label label-primary"><%= ano %></span></h3>
41     </div>
42     <div class="col-sm-6">
43       
44     </div>
45   </div>
46 </script>
47
48 <script id="filмотecaTemplate" type="text/template">
49   <h2>Filmes Cadastrados (<%= qtd %>)</h2>
50 </script>
```

# Backbone.js: Exemplo – index.html

- Inclusão das bibliotecas utilizadas (linhas 52 a 54). A ordem é fundamental! O arquivo com o Backbone deve ser o último pois depende do jQuery e do Underscore.
- O arquivo app.js, que depende dos anteriores, possui a implementação Javascript dos modelos e views Backbone da aplicação.

```
52     <script src="js/lib/jquery-2.0.3.min.js"></script>
53     <script src="js/lib/underscore-min.js"></script>
54     <script src="js/lib/backbone-min.js"></script>
55
56     <script src="js/app.js"></script>
57 </body>
58 </html>
```



# Backbone.js: Exemplo – app.js

```
10 var App = {
11     // armazenará todas as views
12     views: {},
13     // armazenará todos os modelos
14     models: {},
15     // armazenará todas as coleções
16     collections: {},
17     // armazenará todos os templates
18     templates: {},
19     // corresponde ao método main de Java
20     main: function() {
21         new App.views.AppView();
22     }
23 };
24
```



# Backbone.js: Exemplo – app.js

```
36 // Certamente um exagero definir os dados do cabeçalho como sendo um objeto.
37 // Seu propósito é apenas centralizar os dados para eventuais atualizações.
38 App.models.DadosDoCabeçalho = Backbone.Model.extend({
39     defaults: {
40         titulo: "INE5646 - Demo 4 Javascript - Backbone - Cadastro de Filmes",
41         versao: "2013/2",
42         link: "http://backbonejs.org"
43     }
44 });
45
```

# Backbone.js: Exemplo – app.js

```
46 // Representa um filme..
47 // Seu propósito é deixar explícito quais os atributos de um filme.
48 App.models.Filme = Backbone.Model.extend({
49   defaults: {
50     tituloEmPortugues: "??",
51     tituloOriginal: "??",
52     genero: "??",
53     ano: "??",
54     cartaz: "??"
55   }
56 });
```

# Backbone.js: Exemplo – app.js

```
58 // Uma filмотeca é uma coleção de filmes
59 App.collections.Filмотeca = Backbone.Collection.extend({
60   model: App.models.Filme,
61   // URL usada para recuperar os filmes cadastrados no servidor.
62   // No caso, a URL será http://localhost:8080/demo4JSBackbone/filmes
63   url: "filmes"
64 });
65
```

# Backbone.js: Exemplo – app.js

- Observar o uso de jQuery para obter o conteúdo dos templates.

```
72 // Armazena todos os templates que serão usados pelas views.
73 // A função "template" da biblioteca "_" cria uma função que, quando executada,
74 // gerará o fragmento de código HTML definido pelo template.
75 App.templates.cabecalhoTemplate = _.template($("#cabecalhoTemplate").html());
76 App.templates.filmeResumidoTemplate = _.template($("#filmeResumidoTemplate").html());
77 App.templates.filmeCompletoTemplate = _.template($("#filmeCompletoTemplate").html());
78 App.templates.filмотecaTemplate = _.template($("#filмотecaTemplate").html());
79
```

# Backbone.js: Exemplo – app.js

```
81 // Exibe o cabeçalho da aplicação
82 App.views.CabecalhoView = Backbone.View.extend({
83   // Onde, na página HTML, o cabeçalho será renderizado.
84   // O valor de el deve ser um elemento DOM já presente na página.
85   el: '#cabecalho',
86   // executado quando um objeto da classe CabecalhoView for instanciado
87   initialize: function() {
88     this.template = App.templates.cabecalhoTemplate;
89     this.render();
90   },
91   // Inclui no elemento el o código HTML gerado pela função template
92   // a partir dos dados contidos no modelo. Em outras palavras, exibe o modelo
93   // no formato HTML.
94   render: function() {
95     this.$el.html(this.template(this.model.attributes));
96     return this;
97   }
98 });
```

# Backbone.js: Exemplo – app.js

```
100 // Exibe, de forma resumida, os dados de um filme.
101 // Como o atributo el não está definido, gerará uma div contendo
102 // o código HTML.
103 App.views.FilmeResumidoView = Backbone.View.extend({
104     // incluirá a class CSS filme na div
105     className: "filme",
106     initialize: function() {
107         this.template = App.templates.filmeResumidoTemplate;
108         this.render();
109     },
110     render: function() {
111         this.$el.html(this.template(this.model.attributes));
112         return this;
113     },
```

# Backbone.js: Exemplo – app.js

- (continuação da view anterior)

```
114 // quando o usuário clicar sobre o elemento associado à classe CSS
115 // mostrarDetalhes então o método mostreDetalhes deve ser executado.
116 // Lista completa de eventos: http://api.jquery.com/category/events/
117 events: {
118   'click .mostrarDetalhes': 'mostreDetalhes'
119 },
120 // os detalhes do filme serão exibidos pela view detalhesView
121 mostreDetalhes: function(e) {
122   // o evento gerado nesta view não deve ser propagado para cima na
123   // hierarquia DOM
124   e.stopPropagation();
125   this.detalhesView.mostr(e);
126 },
127 // define qual a view mostrará os detalhes do filme
128 setDetalhesView: function(view) {
129   this.detalhesView = view;
130 }
131 });
```



# Backbone.js: Exemplo – app.js

```
133 // Os detalhes de um filme serão exibidos por esta view
134 App.views.FilmeCompletoView = Backbone.View.extend({
135     // o código HTML relacionado a esta view será anexado ao elemento DOM
136     // cujo atributo id seja igual a detalhes.
137     el: '#detalhes',
138     initialize: function() {
139         this.template = App.templates.filmeCompletoTemplate;
140         this.mostrar = false;
141         this.render();
142     },
143     // define se e como mostrar a view
144     render: function() {
145         if (this.mostrar) {
146             this.$el.html(this.template(this.model.attributes));
147             this.$el.fadeIn();
148         } else
149             this.$el.fadeOut();
150         return this;
151     },
```



# Backbone.js: Exemplo – app.js

- (continuação da view anterior)

```
152 // quando o usuário clicar sobre o elemento DOM associado à classe
153 // CSS fechar então o método fecha deve ser executado
154 events: {
155     'click .fechar': 'fecha'
156 },
157 // faz com que a view mostre os detalhes do filme passado como
158 // parâmetro.
159 mostre: function(filme) {
160     this.model = filme;
161     this.mostrar = true;
162     this.render();
163 },
164 fecha: function(e) {
165     e.preventDefault();
166     this.mostrar = false;
167     this.render();
168 }
169 });
170
```

# Backbone.js: Exemplo – app.js

```
172 // View encarregada de exibir todos os filmes
173 App.views.FilмотecaView = Backbone.View.extend({
174   // o código HTML associado à view será incluído no elemento DOM associado
175   // ao id filmes
176   el: '#filmes',
177   initialize: function() {
178     this.template = App.templates.filмотecaTemplate;
179     // execute o método render sempre que a collection for atualizada
180     // com dados vindos do servidor. Outra forma de dizer: sempre que a
181     // collection gerar um evento chamado sync então faça com que a própria
182     // view execute o método render.
183     this.listenTo(this.collection, "sync", this.render);
184     this.render();
185   },
```

# Backbone.js: Exemplo – app.js

- (continuação da view anterior)

```
186     render: function() {
187         // comece mostrando quantos filmes a collection contém.
188         this.$el.html(this.template({qtd: this.collection.length}));
189         // esta é a view a ser usada para exibir os detalhes dos filmes
190         var detalhesView = new App.views.FilmeCompletoView();
191         // para cada filme contido na collection faça:
192         this.collection.each(function(filme) {
193             // crie uma view para exibir os dados resumidos do filme
194             var frv = new App.views.FilmeResumidoView({model: filme});
195             // associe as duas views
196             frv.setDetalhesView(detalhesView);
197             // adicione o conteúdo HTML associado à view frv
198             this.$el.append(frv.el);
199         }, this);
200         return this;
201     }
202 });
```

# Backbone.js: Exemplo – app.js

```
204 // Representa a view da aplicação inteira.
205 App.views.AppView = Backbone.View.extend({
206     // define os dados usados pela aplicação
207     initialize: function() {
208         this.dadosCabecalho = new App.models.DadosDoCabecalho();
209         // observar que inicialmente a filмотeca está vazia (zero filmes)
210         this.filмотeca = new App.collections.Filмотeca();
211         this.render();
212     },
```

# Backbone.js: Exemplo – app.js

- (continuação da view anterior)

```
213  render: function() {
214      // instancia e renderiza a view associada ao cabeçalho da aplicação
215      this.cabecalhoView = new App.views.CabecalhoView({model: this.dadosCabecalho});
216      // instancia e renderiza a view associada à filмотeca da aplicação
217      this.filмотecaView = new App.views.FilмотecaView({collection: this.filмотeca});
218      // solicita, ASSÍNCRONAMENTE, ao servidor que envie um array de objetos JSON
219      // contendo os filmes cadastrados. O atributo reset indica para apagar
220      // quaisquer filmes que por ventura estiverem na collection. Neste caso não
221      // seria realmente necessário. Quando a resposta HTTP chegar a coleção vai
222      // ser atualizada e a filмотeca lançará o evento "sync". Se houver algum
223      // listener ele será acionado (observar que a FilмотecaView se define como
224      // sendo um listener para este tipo de evento.
225      this.filмотeca.fetch({reset: true});
226      return this;
227  }
228  });
```



# Backbone.js: Exemplo – app.js

```
230 //-----  
231 // main  
232 //-----  
233  
234 // Inicia a execução da aplicação na camada 1.  
235 App.main();  
236
```