More Models

- LEVEL 1 -

Re Introducing Backbone.js

“Get your truth out of the DOM”

- Jeremy Ashkenas

Provides client-side app structure

Models to represent data

Views to hook up models to the DOM

Synchronizes data to/from server

Reviewing how Model Data is Fetched from Server

```
var TodoItem = Backbone.Model.extend({urlRoot: '/todos'});
```

RESTful Web Service (Rails Flavor)

Fetch todo with id = 1

```
var todoItem = new TodoItem({id: 1})
```

```
todoItem.fetch();
```

GET /todos/1

```
---> { id: 1, description: 'Pick up milk', status: 'incomplete' }
```

Parsing non-standard JSON into your Models

```
todoItem.fetch();
```

A non-standard Response from Server

```
{todo:{id: 1, description: 'Pick up milk', status: 'incomplete'}}
```

```
var TodoItem = Backbone.Model.extend({
  parse: function(response){
    return response;
  }
})
```

Default Backbone just returns response

```
todoItem.attributes;
```

---> { }

Server Breaking Conventions

```
todoItem.fetch();
```

Response from Server

```
{todo:{id: 1, description: 'Pick up milk', status: 'incomplete'}}
```

```
var TodoItem = Backbone.Model.extend({
  parse: function(response){
    return response.todo;
  }
})
```

```
todoItem.attributes;
```

---> `{ id: 1, description: 'Pick up milk', status: 'incomplete' }`

Instantiating Models doesn't call parse by Default

```
var todoItem = new TodoItem({
  todo: {id: 1, description: 'Pick up milk', status: 'incomplete'}
});
```

```
todoItem.attributes;
```

---> { }

```
var todoItem = new TodoItem({
  todo:{id: 1, description: 'Pick up milk', status: 'incomplete'}
}, { parse: true });
```

```
todoItem.attributes;
```

---> { id: 1, description: 'Pick up milk', status: 'incomplete' }

Changing Attribute Names

Response from Server

```
{todo:{id: 1, desc: 'Pick up milk', status: 'incomplete'}}
```

```
var TodoItem = Backbone.Model.extend({
  parse: function(response){
    response = response.todo;
    response.description = response.desc;
    delete response.desc;
    return response;
  }
})
```

```
todoItem.attributes;
```

---> { id: 1, description: 'Pick up milk', status: 'incomplete' }

Sending JSON back to the Server

Update the todo

```
todoItem.set({description: 'Pick up cookies.'});
```

```
todoItem.save();
```

PUT /todos/1

Sends toJSON() to server

```
{id: 1, description: 'Pick up cookies.', status: 'incomplete'}
```

Server expects JSON with root and desc

```
{todo:{id: 1, desc: 'Pick up cookies.', status: 'incomplete'}}
```

So how to do we modify the JSON before we send it to the server?

Overriding the toJSON Function

```
var TodoItem = Backbone.Model.extend({
  toJSON: function(){
    return _.clone(this.attributes);
  }
});
```

default backbone

```
{
  id: 1,
  description: 'Pick up cookies.',
  status: 'incomplete'
}
```

```
var TodoItem = Backbone.Model.extend({
  toJSON: function(){
    return { todo: _.clone(this.attributes) };
  }
});
```

```
{
  todo: { id: 1, description: 'Pick up cookies.', status: 'incomplete' }
}
```

Overriding the toJSON Function

```
var TodoItem = Backbone.Model.extend({
  toJSON: function(){
    var attrs = _.clone(this.attributes);
    attrs.desc = attrs.description;
    attrs = _.pick(attrs, 'desc', 'status');
    return { todo: attrs };
  }
});
```

returns an object with only the 'desc' and 'status' properties

```
todoItem.toJSON();
```

```
{
  todo: { desc: 'Pick up cookies.', status: 'incomplete' }
}
```

Now we can't use toJSON in View

Render View with Attributes

```
var TodoView = Backbone.View.extend({
  render: function(){
    this.$el.html(this.template(this.model.toJSON()));
    return this;
  }
});
```

toJSON no longer works for templates

Use model.attributes instead

```
var TodoView = Backbone.View.extend({
  render: function(){
    this.$el.html(this.template(this.model.attributes));
    return this;
  }
});
```

Unconventional ID Attribute

```
var todoItem = new TodoItem({id: 1})
```

```
todoItem.fetch();
```

---> { _id: 1, description: 'Pick up milk', status: 'incomplete' }

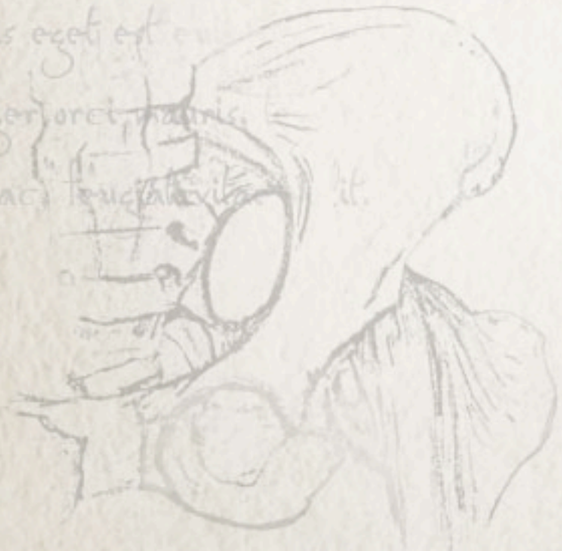
```
todoItem.id
```
---> undefined

```
var TodoItem = Backbone.Model.extend({
  idAttribute: '_id'
});
```

```
todoItem.id
```
---> 1

Customizing Collections

- LEVEL 2 -



Review Fetching Data from the Server

```
var TodoItems = Backbone.Collection.extend({
  url: '/todos'
});
```

URL to get JSON data from

populate collection from server

```
todoItems.fetch()
```

GET /todos

```
[
  {description: 'Pick up milk.', status: 'incomplete', id: 1},
  {description: 'Get a car wash.', status: 'incomplete', id: 2}
]
```

```
todoItems.length
```

---> 2

Handling non-standard Response from Server

```
todoItems.fetch()
```

Response from Server

```
{
  "total": 25, "per_page": 10, "page": 2,
  "todos": [ {"id": 1, ... }, {"id": 2, ... } ]
}
```

```
var TodoItems = Backbone.Collection.extend({
  parse: function(response){
    return response;
  }
})
```

```
todoItems.toJSON(); ---> [ ]
```

Handling non-standard Response from Server

Response from Server

```
{
  "total": 25, "per_page": 10, "page": 2,
  "todos": [ {"id": 1}, {"id": 2} ]
}
```

```
var TodoItems = Backbone.Collection.extend({
  parse: function(response){
    this.perPage = response.per_page;
    this.page = response.page;
    this.total = response.total;
    return response.todos;
  }
})
```

```
todoItems.perPage --> 10
todoItems.page    --> 2
todoItems.total   --> 25
```

These aren't special properties

```
todoItems.toJSON(); ---> [{ id: 1, description: 'Pick up milk.' ... }]
```

How do we fetch with extra params?

```
todoItems.fetch()
```

GET /todos

```
{
  "total": 100, "per_page": 10, "page": 1,
  "todos": [ {"id": 1}, {"id": 2} ]
}
```

GET /todos?page=6

```
{
  "total": 100, "per_page": 10, "page": 6,
  "todos": [ {"id": 61}, {"id": 62} ]
}
```

Fetching with extra params

```
todoItems.fetch({data: { page: 6 }})
```

GET /todos?page=6

```
{
  "total": 100, "per_page": 10, "page": 6,
  "todos": [ {"id": 61}, {"id": 62} ]
}
```

Get next page

```
todoItems.page
```

---> 1

```
todoItems.fetch({data: {page: todoItems.page + 1}});
```

GET /todos?page=2

```
{
  "total": 100, "per_page": 10, "page": 2,
  "todos": [ {"id": 11}, {"id": 21} ]
}
```

Review our Collection View

```
var TodosView = Backbone.View.extend({
  initialize: function(){
    this.collection.on('reset', this.render, this);
  },
  render: function(){
    this.addAll();
    return this;
  },
  addAll: function(){
    this.$el.empty();
    this.collection.forEach(this.addOne);
  },
  addOne: function(todoItem){
    var todoView = new TodoView({model: todoItem});
    this.$el.append(todoView.render().el);
  }
});
```

How do we add a "next" link?

Render the Next Link

```
var TodosView = Backbone.View.extend({
  template: _.template('<a href="#/todos/p<%= page %>">next page</a>'),
  initialize: function(){
    this.collection.on('reset', this.render, this);
  },
  render: function(){
    this.addAll();
    this.$el.append(this.template({page: this.collection.page + 1});
    return this;
  },
  ...
});
```

Now all we have to create a route for “/todos/p:page”

Review our Router

```
var TodoApp = new (Backbone.Router.extend({
  routes: {
    "": "index"
  },
  initialize: function(){
    this.todoItems = new TodoItems();
    this.todosView = new TodosView({collection: this.todoItems});
    this.todosView.render();
    $('#app').append(this.todosView.el);
  },
  index: function(){
    this.todoItems.fetch();
  }
});
```

Implementing the Page Route

```
var TodoApp = new (Backbone.Router.extend({
  routes: {
    "todos/p:page": "page",
    "": "index"
  },
  page: function(page){
    this.todoItems.fetch({data: {page: page}});
  },
  initialize: function(){
    this.todoItems = new TodoItems();
    this.todosView = new TodosView({collection: this.todoItems});
    this.todosView.render();
    $('#app').append(this.todosView.el);
  },
  index: function(){
    this.todoItems.fetch();
  }
});
```

Sorting Collections by Comparator

Sort by Status

```
var TodoItems = Backbone.Collection.extend({
  comparator: 'status'
});
```

```
todoItems.fetch();
```

```
[
  {description: 'Get a car wash.', status: 'complete', id: 2},
  {description: 'Pick up milk.', status: 'incomplete', id: 1}
]
```

Sorting Collections by function

Passing a function in as a comparator

```
var TodoItems = Backbone.Collection.extend({
  comparator: function(todo1, todo2) {
    return todo1.get('status') < todo2.get('status')
  }
});
```

Sort by status in reverse order

```
todoItems.fetch()
```

```
[
  {description: 'Pick up milk.', status: 'incomplete', id: 1},
  {description: 'Get a car wash', status: 'complete', id: 2}
]
```

Aggregate Values

Display amount of completed todo items

```
var TodoItems = Backbone.Collection.extend({
  completedCount: function() {
    return this.where({status: 'complete'}).length;
  }
});
```

Returns filtered array of models

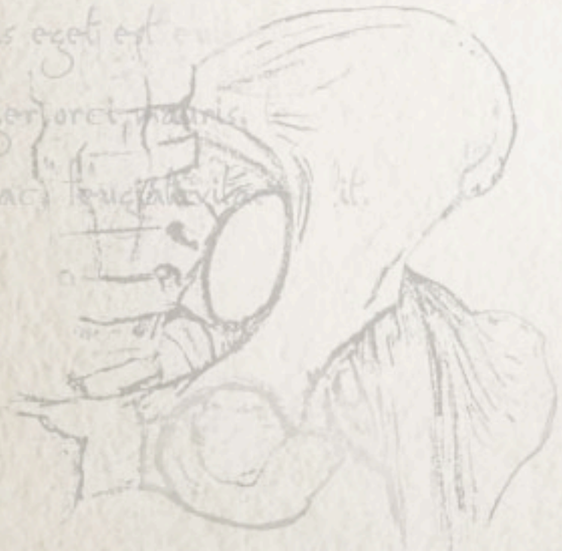
```
[
  {description: 'Pick up milk.', status: 'incomplete', id: 1},
  {description: 'Get a car wash', status: 'complete', id: 2}
]
```

```
todoItems.completedCount()
```

----> 1

Customizing Collections

- LEVEL 2 -



Real Routes

- LEVEL 3 -

Optional Routes

Search route with query and optional page parameter

```
var TodoRouter = new (Backbone.Router.extend({
  routes: {
    'search/:query': 'search',
    'search/:query/p:page': 'search'
  },
  search: function(query, page) {
    page = page || 0;
    console.log(query);
    console.log(page);
  }
}));
```

Duplicate Routes

```
TodoRouter.navigate('search/milk', {trigger: true});
```
--> `"milk", 0`

```
TodoRouter.navigate('search/milk/p2', {trigger: true});
```
--> `"milk", 2`

Optional Routes

```
var TodoRouter = new (Backbone.Router.extend({
  routes: {
    'search/:query(/p:page)': 'search',
  },
  search: function(query, page) {
    page = page || 0;
    console.log(query);
    console.log(page);
  }
}));
```

Optional Route Part

```
TodoRouter.navigate('search/milk', {trigger: true});
```
--> `"milk", 0`

```
TodoRouter.navigate('search/milk/p2', {trigger: true});
```
--> `"milk", 2`

What if we want to match ‘search/milk/p2/”?

Optional Trailing Slash

```
var TodoRouter = new (Backbone.Router.extend({
  routes: {
    'search/:query(/p:page)(/)': 'search',
  },
  search: function(query, page) {
    page = page || 0;
    console.log(query);
    console.log(page);
  }
}));
```

Optional trailing slash

```
TodoRouter.navigate('search/milk/p2/', {trigger: true});
```

--> `"milk", 2`

URI with Spaces Gotcha

```
var TodoRouter = new (Backbone.Router.extend({
  routes: {
    'search/:query(/p:page)(/)': 'search',
  },
  search: function(query, page) {
    page = page || 0;
    console.log(query);
    console.log(page);
  }
}));
```

Encoded space

Search for “Hello World”

```
TodoRouter.navigate('search/Hello%20World/p2', {trigger: true})
```

```
"Hello%20World", 2
```

Not decoded

URI with Spaces Gotcha

```
var TodoRouter = new (Backbone.Router.extend({
  routes: {
    'search/:query(/p:page)(/)': 'search',
  },
  search: function(query, page) {
    page = page || 0;
    query = decodeURIComponent(query);
    console.log(query);
    console.log(page);
  }
}));
```

Encoded space

```
TodoRouter.navigate('search/Hello%20World/p2', {trigger: true})
```

--->

```
"Hello World", 2
```

Decoded

Regex in Routes

Restrict parameter to numeric input

```
var TodoRouter = new (Backbone.Router.extend({
  routes: {
    'todos/:id': 'show'
  },
  show: function(id) {
    console.log("id = " + id);
  }
}));
```

```
TodoRouter.navigate('todos/2', {trigger: true})
```

---> id = 2

```
TodoRouter.navigate('todos/notanid', {trigger: true})
```

-> id = notanid

Shouldn't trigger show

Regex in Routes

```
var TodoRouter = new (Backbone.Router.extend({
  show: function(id) {
    console.log("id = " + id);
  }
}));
```

Each Regex Capture Group becomes a param

```
TodoRouter.route(/^todos\/(\d+)$/, 'show');
```

```
TodoRouter.navigate('todos/2', {trigger: true})
```
---> `id = 2`

```
TodoRouter.navigate('todos/notanid', {trigger: true})
```

Doesn't trigger show

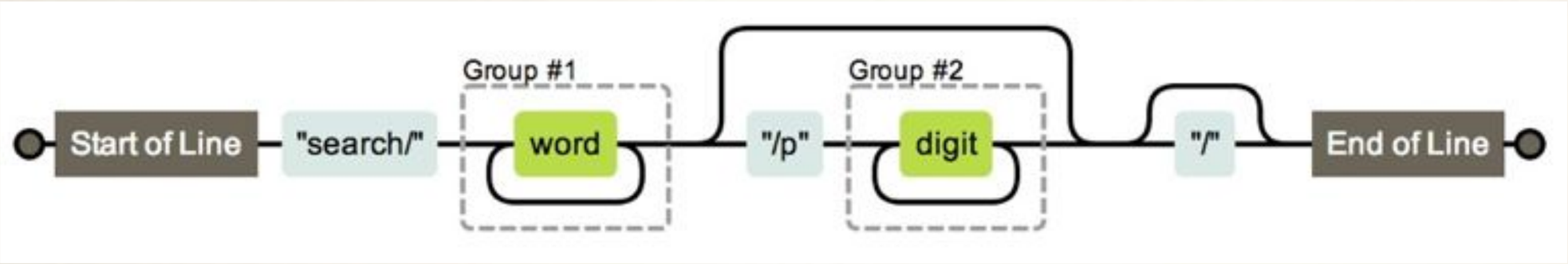
Regex in Routes Refactor

```
var TodoRouter = new (Backbone.Router.extend({
  initialize: function(){
    this.route(/^todos\/(\d+)$/, 'show');
  },
  show: function(id) {
    console.log("id = " + id);
  }
}));
```

Keep routes inside the router

Advanced Regex Routes

```
/^search\/(\w+)(?:\/p(\d+))?\/?$/
```
--> `'search/:query(/p:page)'`



Matches

```
'search/milk'
'search/milk/p2'
'search/milk/p2/'
```

Doesn't Match

```
'searches/milk'
'search/milk/p2/p1'
'search/milk/2'
```

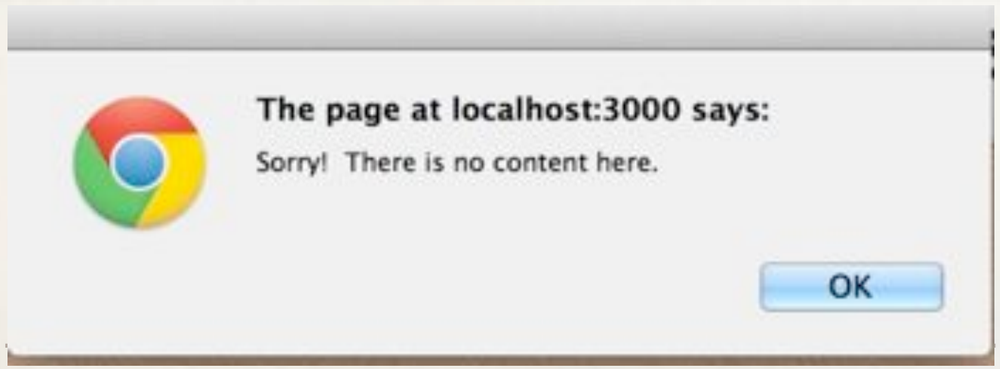
http://www.regexper.com/

Catch-all Routes

Alert user when no route matches

```
var TodoRouter = Backbone.Router.extend({
  routes: {
    '*path': 'notFound'
  },
  notFound: function(path) {
    alert('Sorry!  There is no content here.');
  }
});
```

```
TodoRouter.navigate('nothinghere', {trigger: true})
```



File Path Route

Accept a file path and get its parts

```
var TodoRouter = new (Backbone.Router.extend({
  routes: {
    'file/*path': 'file'
  },
  file: function(path) {
    var parts = path.split("/");
    console.log(parts);
  }
}));
```

```
TodoRouter.navigate("file/this/is/a/file.txt", {trigger: true});
```

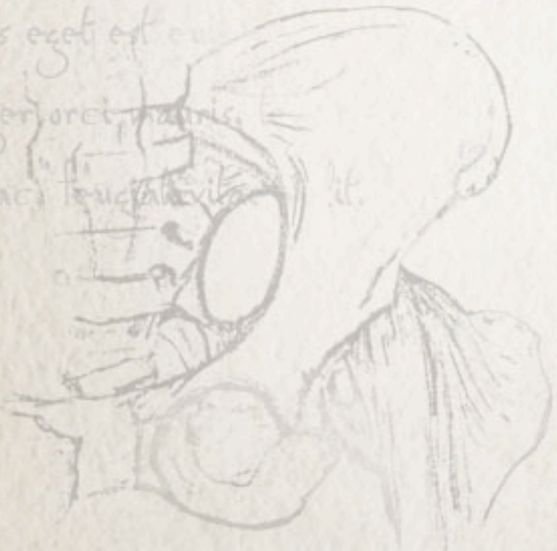
```
["this", "is", "a", "file.txt"]
```

Real Routes

- LEVEL 3 -

Varying Views

- LEVEL 4 -



View Initialization Options Review

Pass in the model

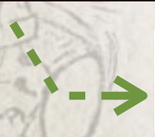
```
var todoView = new TodoView({
  model: todoItem
});
```

```
todoView.model == todoItem
```

What other options can you pass into View initialization?

Or the collection

```
var todoView = new TodoView({
  collection: todoItems
});
```



```
todoView.collection == todoItems
```

Using Existing DOM Elements

```
var TodoView = Backbone.View.extend({
  template: _.template("<%= description %>"),
  render: function(){
    this.$el.html(this.template(this.model.attributes));
    return this;
  }
});
```

```
var todoView = new TodoView({model: todoItem});
```

```
<div class="todo">
</div>
```

```
$('.todo').html(todoView.render().el);
```

--->

```
<div class="todo">
  <div>Pickup Milk.</div>
</div>
```

Using Existing DOM Elements

```
var TodoView = Backbone.View.extend({
  template: _.template("<%= description %>"),
  render: function(){
    this.$el.html(this.template(this.model.attributes));
    return this;
  }
});
```

```
var todoView = new TodoView({model: todoItem, el: $('.todo')});
```

```
<div class="todo">
</div>
```

```
todoView.render()
```

--->

```
<div class="todo">
   Pickup Milk.
</div>
```

Custom Initialization Options

Pass in an extra option

```
var todoView = new TodoView({
  model: todoItem,
  user: currentUser
});
```

```
todoView.options ---> { "user": currentUser }
```

Access extra options in initialize

```
var TodoView = Backbone.View.extend({
  initialize: function(options){
    this.user = options.user;
  }
});
```

Escaping User Content

Rendering user supplied strings can lead to XSS attack

```
var TodoView = Backbone.View.extend({
  template: _.template('<%= description %>'),
  render: function(){
    this.$el.html(this.template(this.model.attributes));
    return this;
  }
});
```

```
var todoView = new TodoView({model: todoItem});
```

```
todoItem.set('description', "<script src='attack.js' />")
```

```
todoView.render().el
```

--->

```
<div>
  <script src='attack.js' />
</div>
```

Escaping User Content

Rendering user supplied strings can lead to XSS attack

```
var TodoView = Backbone.View.extend({
  template: _.template('<%= model.escape("description") %>'),
  render: function(){
    this.$el.html(this.template({model: this.model}));
    return this;
  }
});
```

```
var todoView = new TodoView({model: todoItem});
```

```
todoItem.set('description', "<script src='attack.js' />")
```

```
todoView.render().el
```

--->

```
<div>
  &lt;script src=&#x27;attack.js&#x27; /&gt;
</div>
```

Passing extra options to event handlers

```
var TodoView = Backbone.View.extend({
  template: _.template('<span><%= description %></span>'),
  initialize: function(){
    this.model.on('change:description', this.change, this);
  },
  render: function(){
    this.$el.html(this.template(this.model.attributes));
  },
  change: function(model, value){
    this.$('span').html(value);
    this.$el.effect("highlight", {}, 1000);
  },
});
```

```
todoItem.set({description: "Pickup Kids"});
```

---> `'change:description'`

How do make a change without firing off events?

Passing extra options to event handlers

```
var TodoView = Backbone.View.extend({
  template: _.template('<span><%= description %></span>'),
  initialize: function(){
    this.model.on('change:description', this.change, this);
  },
  render: function(){
    this.$el.html(this.template(this.model.attributes));
  },
  change: function(model, value){
    this.$('span').html(value);
    this.$el.effect("highlight", {}, 1000);
  },
});
```

```
todoItem.set({description: "Pickup Kids"}, {silent: true});
```

How do we stop just the highlight from happening?

Passing extra options to event handlers

```
var TodoView = Backbone.View.extend({
  template: _.template('<span><%= description %></span>'),
  initialize: function(){
    this.model.on('change:description', this.change, this);
  },
  render: function(){
    this.$el.html(this.template(this.model.attributes));
  },
  change: function(model, value, options){
    this.$('span').html(value);
    if (options.highlight !== false){
      this.$el.effect("highlight", {}, 1000);
    }
  },
});
```

```
todoItem.set({description: "Pickup Kids"}, {highlight: false});
```

View Event Cleanup

```
var TodoView = Backbone.View.extend({
  initialize: function() {
    this.model.on('change', this.render, this);
  }
});
```

```
todoView.remove();
```

Model still holds reference to view

```
var TodoView = Backbone.View.extend({
  remove: function() {
    Backbone.View.prototype.remove.apply(this, arguments);
    this.model.off(null, null, this);
  }
});
```

Old Backbone

View Event Cleanup

```
var TodoView = Backbone.View.extend({
  initialize: function() {
    this.listenTo(this.model, 'change', this.render);
  }
});
```

View “listens to” a model

Stop all listeners for this view

```
todoView.stopListening();
```

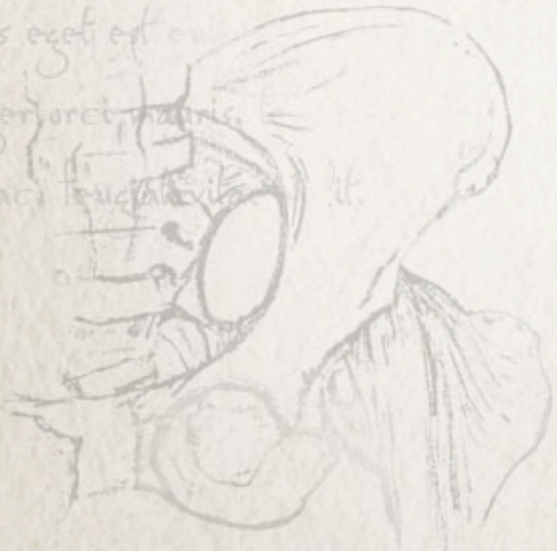
Will automatically call stopListening()

```
todoView.remove();
```

Added in Backbone 0.9.9

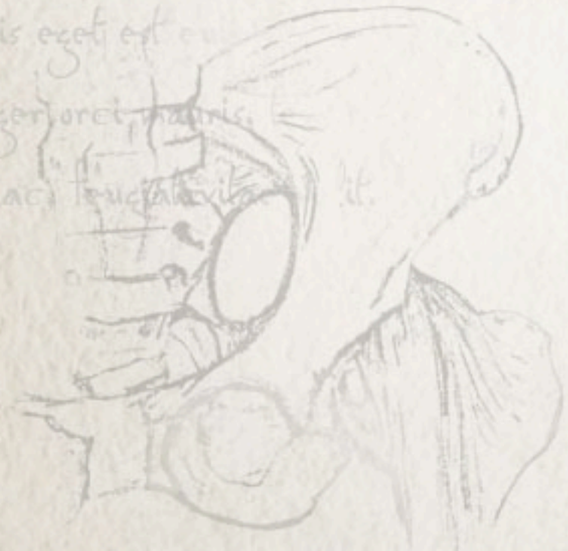
Varying Views

- LEVEL 4 -



Working with Forms

- LEVEL 5 -



Create an Ajax form for creating new Todos

```
<form action="/todos" method="POST">
  <input name=description value="What do you need to do?" />
  <button>Save</button>
</form>
```

Default description

Use plain jQuery

```
$('button').click(function(e){
  e.preventDefault();
  var uri = $('form').attr('action');
  var description = $('input[name=description]').val();
  $.post(uri, {description: description});
});
```

Wait, shouldn't we be creating Todos using Backbone?

--> **POST /todos** `{ description: 'Pickup Kids.' }`

Convert to using Backbone

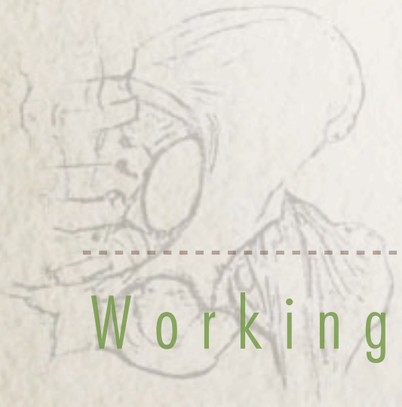
```
var TodoItem = Backbone.Model.extend({
  urlRoot: "/todos"
});
```

Default description

```
var todoItem = new TodoItem({description: "What do you need to do?"});
todoItem.save({description: 'Pickup Kids.'});
```

--> **POST /todos** `{ description: 'Pickup Kids.' }`

What about the form?



Build Form with Backbone View

```
var TodoForm = Backbone.View.extend({
  template: _.template('<form>' +
    '<input name=description value="<%= description %>" />' +
    '<button>Save</button></form>'),
  render: function(){
    this.$el.html(this.template(this.model.attributes));
    return this;
  }
});
```

```
var todoItem = new TodoItem({description: "What do you need to do?"});
var todoForm = new TodoForm({model: todoItem});
$('#app').html(todoForm.render().el);
```

```
<form><input name=description value="What do you need to do?" />
<button>Save</button></form>
```

Capture Button Click to Save Model

```
var TodoForm = Backbone.View.extend({
  template: _.template('<form>' +
    '<input name=description value="<%= description %>" />' +
    '<button>Save</button></form>'),
  events: {
    'click button': 'save'
  },
  save: function(e) {
    e.preventDefault();
    var newDescription = this.$('input[name=description]').val();
    this.model.save({description: newDescription});
  }
});
```

POST /todos `{ description: 'Pickup Kids.' }`

How do we save when pressing enter?

Capture Return Key to Save Model

```
var TodoForm = Backbone.View.extend({
  template: _.template('<form>' +
    '<input name=description value="<%= description %>" />' +
    '<button>Save</button></form>'),
  events: {
    submit: 'save'
  },
  save: function(e) {
    e.preventDefault();
    var newDescription = this.$('input[name=description]').val();
    this.model.save({description: newDescription});
  }
});
```

Will call save on either click or pressing Enter/Return

Reusing Form to Edit existing TodoItem

Get existing TodoItem from already fetched collection

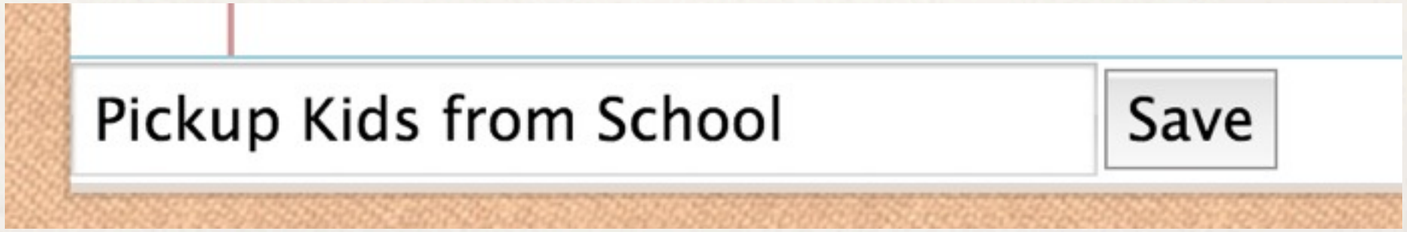
```
var todoItem = todoItems.get(1);
```

Pass in existing model

```
var editTodoForm = new TodoForm({model: todoItem});
```

Replace #app with the HTML of the form

```
$('#app').html(editTodoForm.render().el);
```



Review of our App's Router

```
var TodoApp = new (Backbone.Router.extend({
  routes: {
    "": "index"
  },
  initialize: function(){
    this.todoItems = new TodoItems();
    this.todosView = new TodosView({collection: this.todoItems});
  },
  index: function(){
    this.todoItems.fetch();
    $('#app').html(this.todosView.render().el);
  }
}));
```

Let's add a route to render the TodoForm with an existing TodoItem

Add Route to render Edit Form

```
var TodoApp = new (Backbone.Router.extend({
  routes: {
    "": "index",
    "todos/:id/edit": "edit"
  },
  initialize: function(){
    this.todoItems = new TodoItems();
    this.todosView = new TodosView({collection: this.todoItems});
  },
  index: function(){
    this.todoItems.fetch();
    $('#app').html(this.todosView.render().el);
  }
  edit: function(id){
    var todoForm = new TodoForm({model: this.todoItems.get(id) });
    $('#app').html(todoForm.render().el);
  }
}));
```

Add Route to render New Form

```
var TodoApp = new (Backbone.Router.extend({
  routes: {
    "": "index",
    "todos/:id/edit": "edit",
    "todos/new": "newTodo"
  },
  ...

  newTodo: function(){
    var todoItem = new TodoItem({description: "What do you have to do?"});
    var todoForm = new TodoForm({model: todoItem});
    $('#app').append(todoForm.render().el);
  },

  edit: function(id){
    var todoForm = new TodoForm({model: this.todoItems.get(id) });
    $('#app').html(todoForm.render().el);
  }
}));
```

Get Back to the List after saving

```
var TodoForm = Backbone.View.extend({
  ...
  save: function(e) {
    e.preventDefault();
    var newDescription = this.$('input[name=description]').val();
    this.model.save({description: newDescription});
    Backbone.history.navigate('', { trigger: true });
  }
});
```

We should only call this if the model saved successfully

Get Back to the List after saving success

```
var TodoForm = Backbone.View.extend({
  ...
  save: function(e) {
    e.preventDefault();
    var newDescription = this.$('input[name=description]').val();
    this.model.save({description: newDescription}, {
      success: function(model, response, options){
        Backbone.history.navigate('', { trigger: true });
      }
    });
  }
});
```

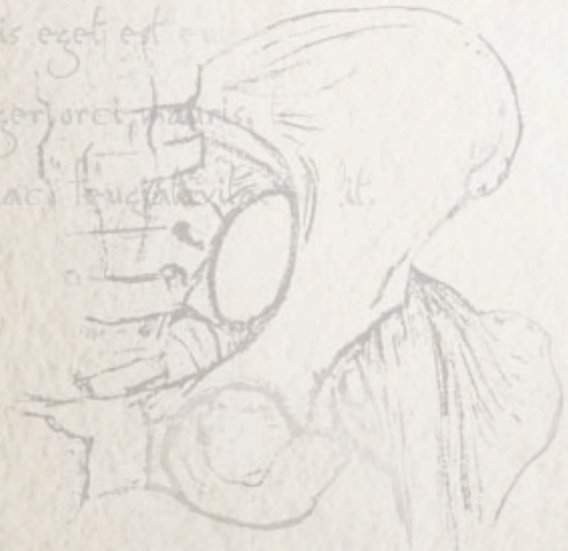
What happens if the Todo doesn't save?

Alert User to save error

```
var TodoForm = Backbone.View.extend({
  ...
  save: function(e) {
    e.preventDefault();
    var newDescription = this.$('input[name=description]').val();
    this.model.save({description: newDescription}, {
      success: function(model, response, options){
        Backbone.history.navigate('', { trigger: true });
      }, error: function(model, xhr, options){
        var errors = JSON.parse(xhr.responseText).errors;
        alert('Oops, something went wrong with saving the TodoItem: ' + errors);
      }
    );
  }
});
```

Working with Forms

- LEVEL 5 -



App Organization

- LEVEL 6 -

Class naming

Everything in the Global Scope

```
var TodoItem = Backbone.Model.extend({});
var TodoItemView = Backbone.View.extend({});
var TodoItems = Backbone.Collection.extend({});
var TodoItemsView = Backbone.View.extend({});
var TodoRouter = Backbone.Router.extend({});
```

- Leads to naming collisions
- Need to put “what kind of object” it is in the name e.g. “TodoItemView”
- Maintainability doesn’t scale with large applications

Use a Global Object for Namespace

```
var App = {
  Models: {},
  Views: {},
  Collections: {}
}
```

Create a single global object where everything is stored

```
App.Models.TodoItem = Backbone.Model.extend({});
App.Views.TodoItem = Backbone.View.extend({});
App.Collections.TodoItems = Backbone.Collection.extend({});
App.Views.TodoItems = Backbone.View.extend({});
App.TodoRouter = Backbone.Router.extend({});
```

Store one-off objects on App

Reference classes with the namespace

```
var todoItem = new App.Models.TodoItem({...})
```

Handle Links Outside of Backbone Views

```
<ul>
  <li><a href="/completed">Show Completed</a></li>
  <li><a href="/support">Support</a></li>
</ul>
```

Using plain jQuery

```
$('a').click(function(e){
  e.preventDefault();
  Backbone.history.navigate(e.target.pathname, {trigger: true});
});
```

What if we wanted backbone to handle this?

Handle Links Outside of Backbone Views

```
var App = {
  Models: {},
  Views: {},
  Collections: {},
  start: function(){
    $('a').click(function(e){
      e.preventDefault();
      Backbone.history.navigate(e.target.pathname, {trigger: true});
    });

    Backbone.history.start({pushState: true});
  }
}
```

```
$(function(){ App.start(); })
```

Outside of Backbone Views

Handle Links // Outside of Backbone Views

```
var App = Backbone.View.extend({
  Models: {},
  Views: {},
  Collections: {},
  events: {
    'click a': function(e){
      e.preventDefault();
      Backbone.history.navigate(e.target.pathname, {trigger: true});
    }
  },
  start: function(){
    Backbone.history.start({pushState: true});
  }
})
```

App is now a View Class

Use View event handling for clicks

```
var app = new App({el: document.body});
```

Using body will capture everything

Handle Links Outside of Backbone Views

```
var App = new (Backbone.View.extend({
  ...
  events: {
    'click a': function(e){
      e.preventDefault();
      Backbone.history.navigate(e.target.pathname, {trigger: true});
    }
  },
  start: function(){
    Backbone.history.start({pushState: true});
  }
}))({el: document.body});
```

Create instance without creating a class

```
$(function(){ App.start(); })
```

Skip links

```
<ul>
  <li><a href="/completed">Show Completed</a></li>
  <li><a href="/support">Support</a></li>
</ul>
```

Event Handler

```
events: {
  'click a': function(e){
    e.preventDefault();
    Backbone.history.navigate(e.target.pathname, {trigger: true});
  }
}
```

Captures all link clicks

Skip links

```
<ul>
  <li><a href="/completed" data-internal="true">Show Completed</a></li>
  <li><a href="/support">Support</a></li>
</ul>
```

Event Handler

```
events: {
  'click a[data-internal]': function(e){
    e.preventDefault();
    Backbone.history.navigate(e.target.pathname, {trigger: true});
  }
}
```

Captures only the links we want it to

Build Initial HTML

```
var App = new (Backbone.View.extend({

  ...

  template: _.template('<h1>ToDo List!</h1>' +
    '<div id="app"></div>'),

  render: function(){
    this.$el.html(this.template());
  }

}))({el: document.body});
```

```
$(function(){
  App.render();
  App.start();
})
```

```
<body>
  <h1>ToDo List!</h1>
  <div id="app"></div>
</body>
```

Add Initial HTML to the body

Object Initialization

```
var App = new (Backbone.View.extend({
  ...
  start: function(){
    var todos = new App.Collections.TodoItems();
    var todosView = new App.Views.TodoItems({collection: todos});
    this.$el.append(todosView.render().el);
    todos.fetch();
  }
}))({el: document.body});
```

Ajax slows down initial render

```
$(function(){ App.start(); })
```

Bootstrap Model Data

```
var App = new (Backbone.View.extend({
  start: function(bootstrap){
    var todos = new App.Collections.TodoItems(bootstrap.todos);
    var todosView = new App.Views.TodoItems({collection: todos});
    this.$el.append(todosView.render().el);
  }
}))({el: document.body});
```

```
var bootstrap = {
  todos: [
    {id: 1, description: "Pickup Milk.", status: "complete"},
    {id: 2, description: "Pickup Kids.", status: "incomplete"},
  ]
}
```

Bootstrap data comes from rendered HTML page

```
$(function(){ App.start(bootstrap); })
```

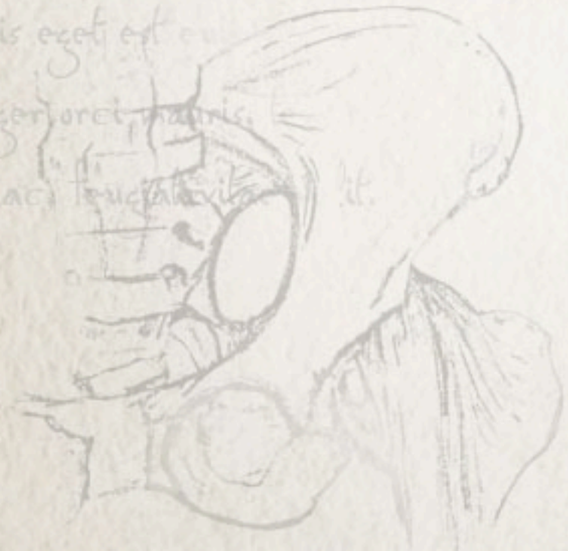
Pass in data to start

App Organization

- LEVEL 6 -

Customizing Backbone

- LEVEL 7 -



Using other template engines

Views don't care how you use templates

```
var TodoItemView = Backbone.View.extend({
  banana: _.template("<span><%= description %></span" +
    "<em><%= assigned_to %></em>"),

  render: function(){
    this.$el.html(this.banana(this.model.attributes));
  }
});
```

Very easy to use other template libraries

Mustache.js handlebars

EJS Google Clojure Templates

Using Mustache.js

Underscore template

```
_.template("<span><%= description %></span" +
  "<em><%= assigned_to %></em>")
```

{{mustache template}}

```
Mustache.compile("<span>{{ description }}</span" +
  "<em>{{ assigned_to }}</em>")
```

Mustache doesn't allow arbitrary js

Using Mustache.js in Backbone View

```
var TodoItemView = Backbone.View.extend({
  template: Mustache.compile("<span>{{ description }}</span" +
    "<em>{{ assigned_to }}</em>"),

  render: function(){
    this.$el.html(this.template(this.model.toJSON()));
  }
});
```

Mustache.compile() returns a function just like _.template()

Mustache.js templates

View

```
{ "name": "Eric" }
```

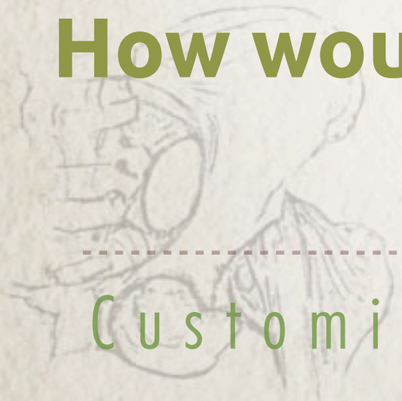
undercore.js template

```
<li><%= name %></li>
```

Mustache.js template

```
<li>{{name}}</li>
```

How would you render an array of names?



Mustache.js Sections

Render array of names

```
{ "names": ["Eric", "Nate", "Jacob"] }
```

undercore.js template

```
<% _.each(names, function(name) { %>
  <li><%= name %></li>
<% }); %>
```

Strange and verbose

Mustache.js template

```
{{#names}}
  <li>{{.}}</li>
{{/names}}
```

"." refers to each string in the array

Mustache.js templates cont.

```
{
  "people": [
    { name: "Eric", hairColor: "brown" },
    { name: "Nate", hairColor: "blond" },
    { name: "Jacob", hairColor: "blue" },
  ]
}
```

```
<% _.each(people, function(person) { %>
  <li><%= person.name %> has <%= person.hairColor %> hair</li>
<% }); %>
```

undercore.js

```
{{#people}}
  <li>{{ name }} has {{ hairColor }} hair</li>
{{/people}}
```

Mustache.js

More Mustache.js Sections

View

```
{ "completed": false }
```

Template

```
Are you done?
{{#completed}}
 <em>Done!</em>
{{/completed}}
```

Output

```
Are you done?
```

View

```
{ "names": [] }
```

Template

```
<ul>
{{#names}}
 <li>{{.}}</li>
{{/names}}
</ul>
```

Output

```
<ul>
</ul>
```

Inverted

View

```
{ "completed": false }
```

Template

```
Are you done?
{{^completed}}
 <em>Nope!</em>
{{/completed}}
```

Output

```
Are you done?
<em>Nope!</em>
```

Mustache.js Function Sections

View

```
{
  name: "Eric",
  header: function(){
    return function(text, render){
      return "<h1>" + render(text) + "</h1>";
    }
  }
}
```

```
Hello {{name}}.
```

Template

```
{{#header}}
Hello {{name}}.
{{/header}}
```

Output

```
<h1>Hello Eric.</h1>
```

Default RESTful Persistence Strategy

```
var todoItem = new TodoItem({id: 1})
```

Read

```
todoItem.fetch();
```

GET /todos/1

Update

```
todoItem.save();
```

PUT /todos/1

Delete

```
todoItem.destroy();
```

DELETE /todos/1

Create

```
(new TodoItem({description: "Pickup Kids"})).save()
```

POST /todos

How do we make TodoItem read-only?

Make Read-Only Model

```
var TodoItem = Backbone.Model.extend({
  sync: function(method, model, options){
    if (method === "read"){
      Backbone.sync(method, model, options);
    }else{
      console.error("You can not " + method + " the TodoItem model");
    }
  }
});
```

`method` = `"read"`, `"create"`, `"update"`, or `"delete"`

```
todoItem.fetch();
```

```
todoItem.save();
```

You can not update the TodoItem model

Completely Replace Persistence Strategy

```
var TodoItem = Backbone.Model.extend({
  sync: function(method, model, options){
    options || (options = {});

    switch(method){
      case 'create':
      break;
      case 'read':
      break;
      case 'update':
      break;
      case 'delete':
      break;
    }
  }
});
```

How would we replace a server with localStorage?

Persistent Key/Value Store for the Web

```
localStorage.setItem(<key>, <value>)
```

```
localStorage.setItem("animal", "Dog")
```

```
localStorage.getItem("animal") ---> "Dog"
```

```
localStorage.removeItem("animal")
```

```
localStorage.getItem("animal") ---> undefined
```

Object Syntax

```
localStorage["animal"] = "Cat"
```

```
localStorage["animal"] ---> "Cat"
```

Browser Support

IE 8.0+
Firefox 3.5+
Safari 4.0+
Chrome 4.0+
Opera 10.5+
iPhone 2.0+
Android 2.0+

Implement the Create Method

```
case 'create':
  var key = "TodoItem-" + model.id;
  localStorage.setItem(key, JSON.stringify(model));
break;
```

TodoItem-1

Automatically calls toJSON on model

```
(new TodoItem({id: 1, description: "Pickup Kids"})).save()
```

Key	Value
TodoItem-1	{"id":1,"description":"Pickup Kids"}

Implement the Read Method

```
case 'read':
  var key = "TodoItem-" + model.id;
  var result = localStorage.getItem(key);
  if (result){
    result = JSON.parse(result);
    options.success && options.success(result);
  }
break;
```

Pass result to success callback

```
var todoItem = new TodoItem({id: 1})
```

```
todoItem.fetch();
```

```
todoItem.attributes; ---> { "id": 1, "description": "Pickup Kids" }
```

Implement the Read Method

```
case 'read':
  var key = "TodoItem-" + model.id;
  var result = localStorage.getItem(key);
  if (result){
    result = JSON.parse(result);
    options.success && options.success(result);
  }else if (options.error){
    options.error("Couldn't find TodoItem id=" + model.id);
  }
break;
```

```
var todoItem = new TodoItem({id: 2})
```

```
todoItem.fetch({error: function(m){ alert(m) }});
```

Pass message to error callback

Backbone.localStorage

```
<script src="backbone-localstorage.js" />
```

https://github.com/jeromegn/Backbone.localStorage

```
var TodoItems = Backbone.Collection.extend({
  model: TodoItem,
  localStorage: new Backbone.LocalStorage("TodoItems")
});
```

Key	Value
TodoItems	1,2
TodoItems-1	{"id":1,"description":"Pickup Milk.","status":"incomplete"}
TodoItems-2	{"id":2,"description":"Pickup Kids.","status":"incomplete"}

Customizing Backbone

- LEVEL 7 -