Sinatra Recipes

Community contributed recipes and techniques

Fork me on GitHub

Select a topic

Models ¶↑

Models are a common pattern for abstracting your data out into an Object-relational mapper or other low-level interface for data transport. This section will cover recipes for integrating these into your Sinatra application. The suggested practice for this is using datamapper, which is covered in the book.

- Active Record
- Couchdb
- Data Mapper
- Mongo
- Ohm
- Sequel

Did we miss something?

It's very possible we've left something out, that's why we need your help!This is a community driven project after all. Feel free to fork the project and send us a pull request to get your recipe or tutorial included in the book.

See the README for more details.

Top

Sinatra Recipes

Community contributed recipes and techniques

Select a topic

Chapters

ActiveRecord ¶ ↑

First require ActiveRecord gem in your application, then give your database connection settings:

```
require 'rubygems'
require 'sinatra'
require 'active_record'

ActiveRecord::Base.establish_connection(
  :adapter => 'sqlite3',
  :database =>  'sinatra_application.sqlite3.db'
)
```

Now you can create and use ActiveRecord models just like in Rails (the example assumes you already have a 'posts' table in your database):

```
class Post < ActiveRecord::Base
end

get '/' do
  @posts = Post.all()
  erb :index
end
```

This will render ./views/index.erb:

```
<% for post in @posts %>
  <h1><%= post.title %></h1>
<% end %>
```

Did we miss something?

It's very possible we've left something out, that's why we need your help!This is a community driven project after all. Feel free to fork the project and send us a pull request to get your recipe or tutorial included in the book.

See the README for more details.

Top

Sinatra Recipes

Community contributed recipes and techniques

Select a topic

Chapters

CouchDB ¶ ↑

There are several data modelling libraries for CouchDB. We're going to introduce CouchRest Model.

Require couchrest_model gem in your app:

```
require 'couchrest_model'
```

Specify the database URL:

```
configure do
  $COUCH = CouchRest.new ENV["COUCHDB_URL"]
  $COUCH.default_database = ENV["COUCHDB_DEFAULT_DB"]
  $COUCHDB = $COUCH.default_database
end
```

Create the Model class:

```
class Post < CouchRest::Model::Base
  use_database $COUCHDB

  property :title, String
  property :body, String

  design do
    view :by_title
  end
end
```

Save a new instance of your model from a route:

```
post '/post' do
  @post = Post.create :title => params[:title], :body => params[:body]
  redirect "/posts/#{@post.title}"
end
```

Find and render instances of your model matching a criteria:

```
get '/posts/:title' do
  @posts = Post.by_title(:key => params[:title])
  erb :posts
end
```

This will render ./views/posts.erb:

```
<% for post in @posts %>
  <div>
    <h1><%= post.title %></h1>
    <p><%= post.body %></p>
  </div>
<% end %>
```

Full documentation on CouchRest and CouchRest Model.

Did we miss something?

It's very possible we've left something out, that's why we need your help!This is a community driven project after all. Feel free to fork the project and send us a pull request to get your recipe or tutorial included in the book.

See the README for more details.

Top

Sinatra Recipes

Community contributed recipes and techniques

Select a topic

Chapters

DataMapper ¶ ↑

Start out by getting the DataMapper gem if you don't already have it, and then making sure it's in your application. A call to `setup` as usual will get the show started, and this example will include a 'Post' model.

```
require 'rubygems'
require 'sinatra'
require 'data_mapper' # metagem, requires common plugins too.

# need install dm-sqlite-adapter
DataMapper::setup(:default, "sqlite3://#{Dir.pwd}/blog.db")

class Post
  include DataMapper::Resource
  property :id, Serial
  property :title, String
  property :body, Text
  property :created_at, DateTime
end

# Perform basic sanity checks and initialize all relationships
# Call this when you've defined all your models
DataMapper.finalize

# automatically create the post table
Post.auto_upgrade!
```

Once that is all well and good, you can actually start developing your application!

```
get '/' do
```

```
  # get the latest 20 posts
  @posts = Post.all(:order => [ :id.desc ], :limit => 20)
  erb :index
end
```

Finally, the view at `./view/index.html`:

```
<% @posts.each do |post| %>
  <h3><%= post.title %></h3>
  <p><%= post.body %></p>
<% end %>
```

For more information on DataMapper, check out the project documentation.

Did we miss something?

It's very possible we've left something out, that's why we need your help!This is a community driven project after all. Feel free to fork the project and send us a pull request to get your recipe or tutorial included in the book.

See the README for more details.

Top

Sinatra Recipes

Community contributed recipes and techniques

Fork me on GitHub

Select a topic

Chapters

MongoDB ¶ ↑

There are many ORMs out there for Mongo (or *ODMs* in this case). This page will go over just a few.

Mongo ¶ ↑

Looking for the official driver?

MongoMapper ¶ ↑

```
require 'mongomapper'
```

Create the Model class

```
class Link
  include MongoMapper::Document
  key :title, String
  key :link, String
end
```

Create the route:

```
get '/' do
  @links = Link.all
  haml :links
end
```

Mongoid ¶ ↑

```
require 'mongoid'
```

Create the Model class

```
class Link
  include Mongoid::Document
  field :title, :type => String
  field :link, :type => String
end
```

Create the route:

```
get '/' do
  @links = Link.all
  haml :links
end
```

Candy ¶ ↑

```
require 'candy'
```

Create the Model class

```
class Link
  include Candy::Piece
end

class Links
  include Candy::Collection
  collects :link   # Declares the Mongo collection is 'Link'
end

Link.connection # => Defaults to localhost port 27017
Link.db         # => Defaults to your username, or 'candy' if unknown
Link.collection # => Defaults to the class name ('Link')
```

Create the route:

```
get '/' do
  @links = Links.all
  haml :links
end
```

Mongomatic ¶ ↑

```
require 'mongomatic'
```

Create the Model class

```
class Link < Mongomatic::Base
  def validate
    self.errors.add "title", "blank" if self["title"].blank?
    self.errors.add "link",  "blank" if self["link"].blank?
  end
end
```

Create the route:

```
get '/' do
  @links = Link.all
  haml :links
end

def validate
  self.errors.add "name", "blank" if self["name"].blank?
  self.errors.add "email", "blank" if self["email"].blank?
  self.errors.add "address.zip", "blank" if (self["address"] || {})["zip"].blank?
end
```

MongoODM ¶ ↑

```
require 'mongo_odm'
```

Create the Model class

```
class Link
  include MongoODM::Document
  field :title
  field :link
end
```

Create the route:

```
get '/' do
  @links = Link.find.to_a
  haml :links
end
```

GitHub links

- MongoMapper
- Mongoid
- Candy
- Mongomatic
- MongoODM

Did we miss something?

It's very possible we've left something out, that's why we need your help!This is a community driven project after all. Feel free to fork the project and send us a pull request to get your recipe or tutorial included in the book.

See the README for more details.

Top

Sinatra Recipes

Community contributed recipes and techniques

Fork me on GitHub

Select a topic

Chapters

Ohm ¶ ↑

Ohm is an object hash-mapping library for the Redis database.

Require the Ohm gem in your app:

```
require 'rubygems'
require 'sinatra'
require 'ohm'
```

Configure Ohm for your environment:

```
configure :production do
  Ohm.connect(:url => ENV["MY_REDIS_URL"])
end
```

Create a model class and a Redis index to allow fast lookups:

```
class Post < Ohm::Model
  attribute :title
  attribute :body
  index :title
end
```

Save a new instance of your model from a route:

```
post '/post' do
  Post.create :title => params[:title],
              :body => params[:body]
end
```

Find and render instances of your model matching a criteria:

```
get '/posts/:title' do
  @posts = Post.find(:title => params[:title])
  erb :index
end
```

This will render ./views/index.erb:

```
<% for post in @posts %>
  <h1><%= post.title %></h1>
<% end %>
```

Did we miss something?

It's very possible we've left something out, that's why we need your help!This is a community driven project after all. Feel free to fork the project and send us a pull request to get your recipe or tutorial included in the book.

See the README for more details.

Top

Sinatra Recipes

Community contributed recipes and techniques

Select a topic

Chapters

Sequel ¶ ↑

Require the Sequel gem in your app:

```
require 'rubygems'
require 'sinatra'
require 'sequel'
```

Use a simple in-memory DB:

```
DB = Sequel.sqlite
```

Create a table:

```
DB.create_table :links do
  primary_key :id
  varchar :title
  varchar :link
end
```

Create the Model class:

```
class Link < Sequel::Model; end
```

Create the route:

```
get '/' do
  @links = Link.all
  haml :links
```

```
end
```

Did we miss something?

It's very possible we've left something out, that's why we need your help!This is a community driven project after all. Feel free to fork the project and send us a pull request to get your recipe or tutorial included in the book.

See the README for more details.

Top

Sinatra Recipes

Community contributed recipes and techniques

Select a topic

Chapters

Mongo ¶ ↑

Mongo is a document-oriented database. Though Object Relational Mappers (ORMs) are often used to connect to databases, you will see here that it is very easy to connect your applications to a Mongo database without the use of an ORM, though several exist. See the Mongo models page for a discussion of some of the ORMs available.

Install the required gems:

```
gem install mongo
gem install bson_ext
```

The first step is in connecting your application to an instance of Mongo is to create a connection. You can do this in your *configure* block:

Note that there has been a change in the Ruby API post v 1.8.x. The following examples use the newer API

```
require 'rubygems'
require 'sinatra'
require 'mongo'
require 'json/ext' # required for .to_json

include Mongo

configure do
  conn = MongoClient.new("localhost", 27017)
  set :mongo_connection, conn
```

```
  set :mongo_db, conn.db('test')
end
```

With a connection “in hand” you can connect to any database and collection in your Mongo instance.

```
get '/collections/?' do
  settings.mongo_db.collection_names
end

helpers do
  # a helper method to turn a string ID
  # representation into a BSON::ObjectId
  def object_id val
    BSON::ObjectId.from_string(val)
  end

  def document_by_id id
    id = object_id(id) if String === id
    settings.mongo_db['test'].
      find_one(:_id => id).to_json
  end
end
```

Finding Records

```
# list all documents in the test collection
get '/documents/?' do
  content_type :json
  settings.mongo_db['test'].find.to_a.to_json
end

# find a document by its ID
get '/document/:id/?' do
  content_type :json
  document_by_id(params[:id]).to_json
end
```

Inserting Records

```
# insert a new document from the request parameters,
# then return the full document
post '/new_document/?' do
  content_type :json
  new_id = settings.mongo_db['test'].insert params
  document_by_id(new_id).to_json
end
```

Updating Records

```
# update the document specified by :id, setting its
# contents to params, then return the full document
put '/update/:id/?' do
  content_type :json
  id = object_id(params[:id])
  settings.mongo_db['test'].update(:_id => id, params)
  document_by_id(id).to_json
end

# update the document specified by :id, setting just its
# name attribute to params[:name], then return the full
# document
put '/update_name/:id/?' do
  content_type :json
  id   = object_id(params[:id])
  name = params[:name]
  settings.mongo_db['test'].
    update(:_id => id, {"$set" => {:name => name}})
  document_by_id(id).to_json
end
```

Deleting Records

```
# delete the specified document and return success
delete '/remove/:id' do
  content_type :json
  settings.mongo_db['test'].
    remove(:_id => object_id(params[:id]))
  {:success => true}.to_json
end
```

For more information on using the Ruby driver without an ORM take a look at MongoDB's tutorial.

Did we miss something?

It's very possible we've left something out, that's why we need your help!This is a community driven project after all. Feel free to fork the project and send us a pull request to get your recipe or tutorial included in the book.

See the README for more details.

Top