Adrian Mejia's [code]Blog

var life = { "work_hard","have_fun","make_history" };

Ruby on Rails Architectural Design

| Comments

1 Introduction

Ruby on Rails (RoR) is open source web framework written in the Ruby programming language, and all the applications in Rails are written in Ruby. Ruby on Rails is focused on productivity and enforces agile web development.

Rails framework leverages the features of the Ruby language. Yukishiro Matzumoto designed this language in 1995 influence by Perl, Eiffel, Python, Smalltalk and others. It's a dynamically typed, fully object-oriented, general-purpose scripting language. Ruby was designed to have an elegant syntax and made as human readable as possible, for instance it does not need colons and parenthesis around parameters. Some parts of the code are read like English declarations.

The Ruby on Rails framework was designed for database-backed web applications. It was created as a response to heavy web frameworks such as J2EE and the .NET framework. In order to make the development process faster, Ruby on Rails uses conventions and assumptions that are considered best ways to accomplish tasks, and it's designed to encourage those. This convention eliminates configuration code and increases productivity. Many of the common tasks for web development are built-in in the framework to work out-of-the-box. This includes email management, object-database mappers, file structures, code generation, how the elements are named and organized and so on. All of these conventions allow developers to write less code and develop agile applications. Additionally, the enhance maintainability and understandability around the Ruby on Rails developers' community.

Ruby on Rails architecture has the following features:

- Model-View-Controller architecture.
- Representational State Transfer (REST) for web services.
- Supports the major databases (MySQL, Oracle, MS SQL Server, PostgreSQL, IBM DB2, and more).
- Open-source server side scripting language.
- Convention over configuration
- Scripts generators to automate tasks.
- Use of YAML machine, which is a human-readable data serialization format.

The above-described features are distributed in the following Rails' components and the Fig. 2 shows the interaction between some of these components:

- Action Mailer
- Action Pack
  - Action Controller
  - Action Dispatcher
  - Action View
- Active Model
- Active Record
- Active Resource
- Active Support
- Railties

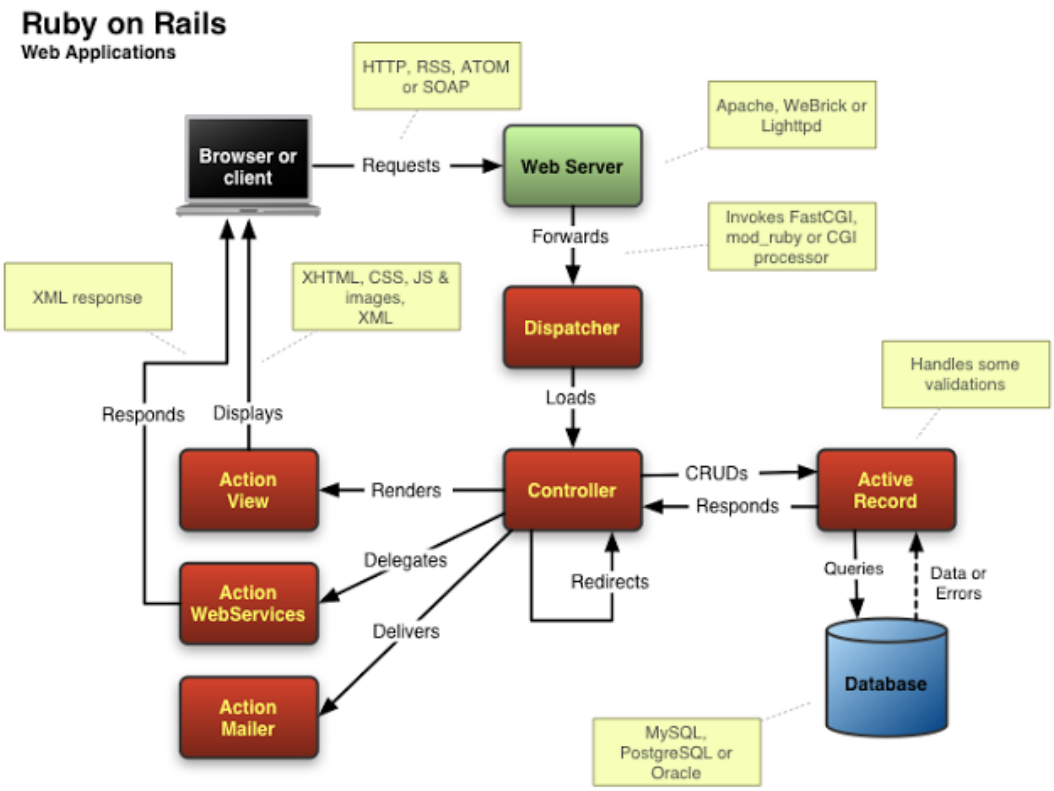

Figure 2. Reference model depicting the overall framework architecture [source]

1.1 Model-View-Controller Pattern

Ruby on Rails uses the Model-View-Controller (MVC) architectural pattern in order to improve the maintainability of the application. The Model centralizes the business logic, the View manages the display logic, while the Controller deals with the application flow. The MVC allows a clean separation of concerns, in the way that it keeps the business logic separated from HTML views. Additionally, it improves decoupling and testing.

1.1.1 Model

The Model layer carries the business logic of the application and the rules to manipulate the data. In Ruby on Rails, the models are used to manage the interaction with their corresponding elements in the database. The Models represent the information in the database and do the appropriate validations.

1.1.2 View

The view is the front-end of the application, representing the user interface. In Ruby on Rails, views are HTML files with embedded Ruby code. The embedded Ruby code in the HTMLs is fairly simple (loops and conditionals). It is only used to display data to the user in the form of views. Views are used to provide the data to the browsers that requested the web pages. Views can server content in several formats, such as HTML, PDF, XML, RSS and more.

1.1.3 Controller

Controllers interact with models and views. The incoming requests from the browsers are processed by the controllers, which process the data from the models and pass it to the views for presentation.

1.2 Rails Modules

1.2.1 Action Mailer

This module is responsible for providing e-mail services. It processes incoming mails and creates new ones. This module can handle simple text or complex rich-format emails. Also it has common tasks built-in, such as, sending out forgotten passwords, welcome messages, and fulfilling any other written-communication's need. Action Mailer is wrapped around the Action Controller. It provides ways to make email with templates in the same way that Action View uses it to render web pages.

1.2.2 Action Pack

The Action Pack module provides the controller and view layers of the MVC patterns. These modules capture the user requests made by the browser and map these requests to actions. These actions are defined in the controllers layer and later the actions render a view that is displayed in the browser. Action Pack is divided in 3 sub-modules, which are: Action Dispatch, Action Controller, and Action View.

- **Action Dispatch:** handles routing of web browser request. It parses the web request and does advanced processing around HTTP, such as handling cookies, sessions, request methods, and so forth.
- **Action Controller:** after the action dispatch has processed the request it makes the routing to its corresponding controller. This module provides a base controller from which all the other controllers can inherit. Action Controller contains actions to controls model and view. This module makes data available as needed, controls views rendering and redirection. Additionally, it manages the user sessions, application flow, caching features, helper modules and implement filters for the pre, during and post processing hooks.
- **Action View:** it is call by the Action Controller. It renders the presentation of the web page requested. Action View provides master layouts, templates lookups and view helpers that assist the generation of the HTML, feeds and other presentation formats. There are three templates schemas in Rails, which are rhtml, rxml, and rjs. The rhtml format generates HTML views to the users with ERB (embedded ruby code in HTML). The rxml is used to construct XML documents using Ruby, and rjs allow creating dynamic JavaScript code in Ruby useful to implement AJAX functionality.

1.2.3 Active Model

Define the interface between the Action Pack and the Active Record modules. Also, Action Record interfaces can be used outside of Rails framework to provide Object-relational mapping (ORM) functionalities.

1.2.4 Active Record

Active record is an architectural pattern used to manage data in relational databases through objects. In Ruby on Rails the Active Record module provides object-relational mapping to classes. This module builds the Model layer that connects the database tables with its representation in ruby classes. Rails provide tools to implement the CRUD functionality with zero-configuration. CRUD allows creating, reading, updating and deleting records from the database through ruby objects. An object represents each row in a database tables. Additionally, it also provides advance search capabilities and the ability to create relationships or associations between models. Active Records relies heavily on conventions on how the classes should be named, the tables in the database, the foreign keys and primary keys. However, the database mapping can be accomplished using configuration, but it is highly encouraged to follow the rails convention, such as active record modules.

This modules is used to create model classes, which contains the business logic, handle validations and relationships, automatically maps to a table and encapsulates data access, provides getters and setters, callbacks and also supports several databases.

1.2.5 Active Resource

Active Resource module is used for managing the connection between RESTful web services and business objects. It follows the same principle of Active Record that is to reduce the amount of code needed to map resources. Active Resources maps model classes to remote REST resources in the same way that Active Record maps model classes to database tables. Active Resource leverages the HTTP protocol and adds code conventions to make it easy to infer complex structures and relations. Active Record also provides proxy capabilities between an Active Resource (client) and a RESTful service. This is accomplished implementing an object-relational mapping for REST web services. When a request to a remote resource is made, a REST XML is generated and transmitted, and then the result is parsed into a ruby object.

1.2.6 RESTful Architecture

REST stands for Representational State Transfer. REST is an alternative to web services, such as SOAP and WSDL. It relies in the HTTP protocol for all the CRUD operations: create, read, update and delete. RESTful web services are appropriated when the web services are completely stateless, limited bandwidth (it's very useful for mobile devices since it doesn't the the overhead of other protocols like SOAP), when the data is not generated dynamically so it could be cached to improve performance and when there is a mutual understanding between the service producer and the consumer.

1.2.7 Active Support

It is a collection of utility classes and standard Ruby libraries extensions that are useful for the development on Ruby on Rails. It includes a rich support for multi-bytes strings, internationalization, time zones and testing.

1.2.8 Railties

Railties is the Rails' core code that builds new applications. It is responsible for "glue"-ing all the above-describe modules all together. Additionally, it handles all the bootstrapping process, the command line interface and provides the Rails' code generators. Rake is one of the command lines used to perform database tasks, deployment, documentation, testing and cleanups. Rails also supply a built-in testing framework that generates test stubs automatically when code is generated, provides unit testing, functional testing for views and controls, test fixtures and supply test data using YAML.

2 Architectural Solution

The selected views to analyze this architecture are Module Views and Component and Connector (C&C) Views. The Module View will contain UML diagrams which represent a static view of all the components, while the Component & Connector View will show the UML diagram that presents a run-time view of a system's architecture: what components exist at run-time and how do these components communicate with one another.

2.1 Connect and Component View

The Component-and-Connector View is a dynamic view of the system and presents the components, interfaces, connectors and systems. In the interfaces representation we choose to present the Interfaces as UML interfaces. This provides a compact representation of the interfaces and avoids crowding the diagram. The connector types will be represented as associations and connector instances as links to be the consistent with the UML notation that we are using the for the interfaces representation. In the same way the systems will be presented as UML subsystems. Fig. 3

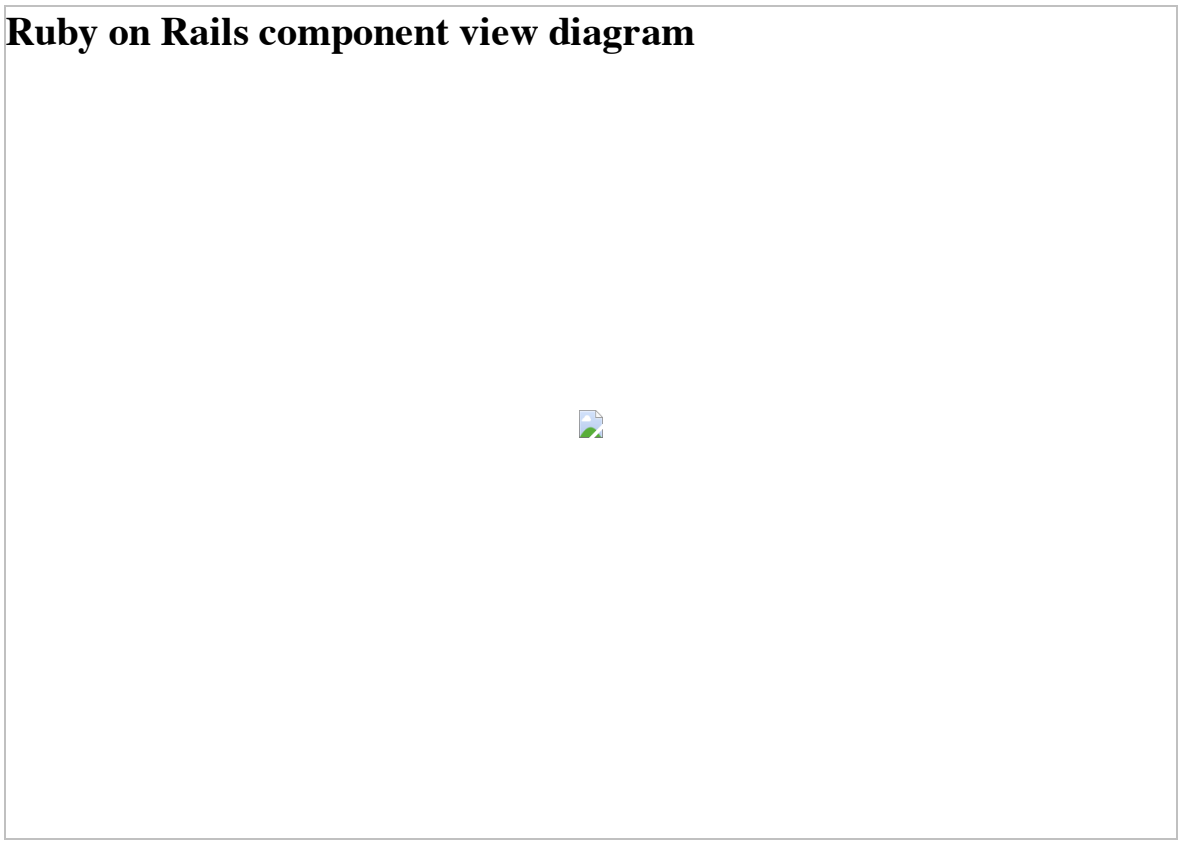

Figure 3. Dynamic View: Ruby on Rails Connector and Component View

2.2 Modules View

The Module View shows the code or implementation as modules and presents the interfaces and their relations between each other. The Module View contains the following: modules, interfaces, aggregations, generalizations and dependencies. These views are an

appropriate way to show a static view of the architecture. Fig. 4

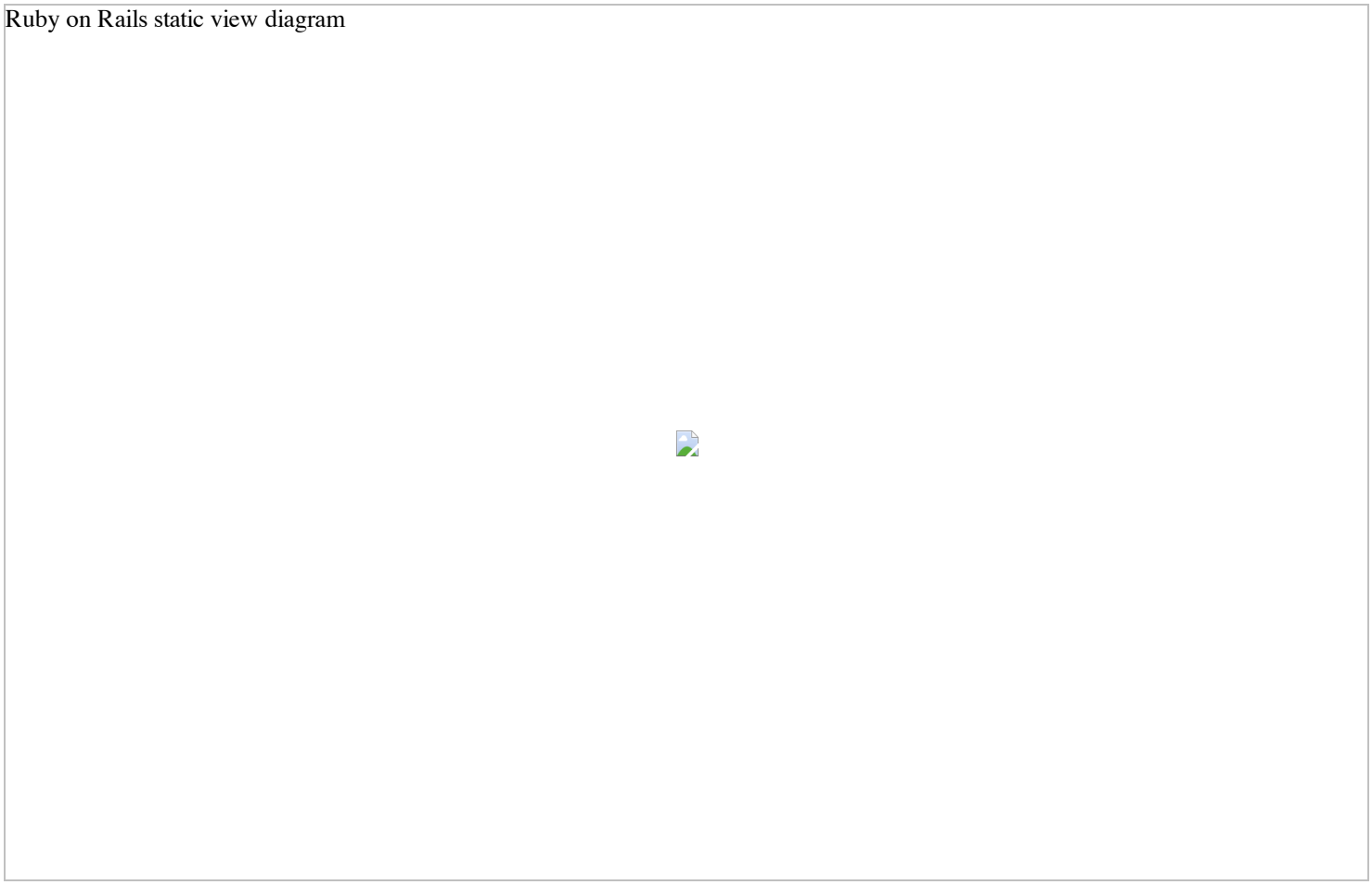

Figure 4. Static View: Ruby on Rails Module View

The Module View shows the code or implementation as modules and presents the interfaces and their relations between each other. The Module View contains the following: modules, interfaces, aggregations, generalizations and relationships. This view is appropriate way to show static view of the architecture.

3 Pitfalls of the Ruby on Rails Architecture

Since Rails is built on the Ruby language it inherits the goodness and weakness of that language. Ruby is a dynamic scripting language with an elegant syntax and fully object-oriented. Because it is an interpreted language it is slower than other languages that are compiled like Java or C++. In most cases, this difference in speed is not a problem but when the web application needs to scale to millions of concurrent users this performance starts to degrade. The Ruby language is not suited for high concurrency applications, because it is not optimized for speed computing. It was designed to be elegant, simply and for rapid development. Ruby doesn't have good thread support yet and, like many other scripting languages, Ruby has trouble dealing with long-lived processes. But other languages, like Java, are really good at that because they have been optimized for year to handle threads efficiently. Another weakness is the Ruby's garbage collector is not as good as Java's in that each process requires much more memory. In terms of deployment, a web application on Ruby on Rails could be harder to deploy than sites that are using more common technologies, such as PHP. That's because not all the hosting providers support Rails, but in time the support of Rails is increasing.

There are some kinds applications in which Ruby on Rails is not suitable for:

- Static web sites.
- Content Management Systems.
- Sites that depend on large amount of existing code or legacy code.
- Sites with legacy databases.
- Multi-site servers, situations that require dozens or hundreds of websites residing in a single server.

- Very large-scale web applications (millions of concurrent users) that need to deliver high-performance.

The Rails architecture sacrifices some flexibility in configuration to increase productivity based on conventions that are considered best practices. This could make experienced developers in other languages feel tied to things in "the Rails way". On the other hand, Ruby sacrifices speed and resource management to favor elegant syntax, productivity and maintainability. However, most of the small and mid-size enterprise web applications do little computing and factors such as latency, bandwidth and database performance are not very crucial. These factors start to matter in high-traffic websites with millions of concurrent users.

3.1 Twitter case study

A study case of high-traffic web application using Rails is Twitter. They started using Ruby on Rails but they reached a point that the scaling of their platform was not cost-effective. This was mainly because Ruby on Rails has poor multi-threading support. As in 2011, they have more than 1 billion tweets per week and 200 million users. In 2011 they reported [1], that are moving their back-end applications from Ruby on Rails to Java, which perform 3x faster searches and handle high concurrency better.

As the Twitter Engineering team stated in [1] and [2]:"*As part of the effort, we launched a new real-time search engine, changing our back-end from MySQL to a real-time version of Lucene. Last week, we launched a replacement for our Ruby-on-Rails front-end: a Java server we call Blender. We are pleased to announce that this change has produced a 3x drop in search latencies and will enable us to rapidly iterate on search features in the coming months.*" … And later… "*In April 2011, we launched a replacement, called Blender, which improved our search latencies by 3x, gave us 10x throughput, and allowed us to remove Ruby on Rails from the search infrastructure. Today, we are indexing an average of 2,200 TPS while serving 18,000 QPS (1.6B queries per day!). More importantly, Blender completed the infrastructure necessary to make the most significant user-facing change to Twitter search since the acquisition of Summize.*"

3.2 Yellopages.com case study

Yellowpages.com on the other hand moved from Java to Ruby on Rails framework. The rationale behind this decision is that they pursued to increase their maintainability, absolute control of URLs, no sessions, and be more agile. They have 170 million page views per month and 2.5 million searches per day.

Brian Burridge summarizes the process in the following way "YellowPages.com's move from Java to Rails. They serve up to 23 million visitors a month. The conversion resulted in 20,000 lines of Ruby code instead of 125,000 lines of Java code, and most importantly eased the difficulty they had in maintaining it. Once complete, and optimized their site is now faster than before. They also completed the rewrite in three months with four developers."

4 Summary

Ruby on Rails has received widespread support throughout the software development industry, more specifically the open-source community. This support reflects, to a certain extent, that the framework was able to provide a fairly robust architecture somewhat consistent with the goals that the architect envisioned.

The framework successfully exhibits must of its intended quality attributes. The responsibility of maintaining some of these qualities are deferred to third party components, as is the case of the performance quality, which is deferred to whichever web-server is configured at development or deployment time. The biggest strength of the framework is the way it leverages the Convention over Configuration and Don't Repeat Yourself principles. This principle enables intuitive approaches when dealing with certain tactics to achieve some of the quality attributes. For instance, modifiability becomes less hassle due to the isolation or centralization of certain changes since the framework relies on default places where certain functionality should be located.

On the other hand, Rails inherits some of the weaknesses of the language used to build it. More specifically, poor concurrency support hinders the scalability potential of an application built using Ruby on Rails. Furthermore, memory management is not as efficient as other languages available. For instance, Java exhibits a much better garbage collector behavior when compared with Ruby on Rails.

Overall, it would be safe to say that the framework architecture meets most of its intended goals, but not without its caveats. The level and speed of adoption in industry and the open-source community could be used as validation of the potential the framework has, but at the same time we see limitations emerging when using the framework. Some of these limitations are part of the driven force behind the ongoing evolution of Ruby on Rails.

Posted by Adrian Mejia agile frameworks, ruby on rails, software architecture, twitter, web development, web frameworks

Was it useful? There are many ways of saying thank you :)

« Microsoft Zune Failure Analysis Git auto-commit with Crontab »

Comments

About Me

{
"name": "Adrian Mejia",
"location": "Boston, MA",
"title": "Software Engineer",
"technologies": ["Ruby on Rails", "Ruby", "JavaScript", "Java", "jQuery", "HTML5", "etc."],
"interests": ["Web development", "Software Architecture", "NoSQL Databases", "Algorithms"],
}

Email

Linkedin

Github

Recent Posts

- Cheap Airplay Receiver With Raspberry Pi
- Algorithms for Dummies (Part 1): Big-O Notation and Sorting
- Backbone.js for Absolute Beginners - Getting Started (Part 4: Routers)
- Backbone.js for Absolute Beginners - Getting Started (Part 3: CRUD)
- Backbone.js for Absolute Beginners - Getting Started (Part 2: Models, Collections and Views)

GitHub Repos

- jerkyll-drupal7-migration

  Migrate drupal7 blog to jerkyll/octopress blog.

- soshop

  social shop

- impulsideas

  Crowdfunding platform

- algorithms_part1

- tweet_scheduler

@amejiarosario on GitHub

On Delicious

- Bootswatch: Free themes for Twitter Bootstrap
- David Hauser - 25 Entrepreneurs Tell What They Wish They'd Known before Founding Their First Startup
- Learn HTML5, CSS3, Javascript | CSSDeck

My Delicious Bookmarks »

Latest Tweets

- Status updating...

conventions and