Admin API Ruby on Rails *Backend system*

Software Development Documentation

FrontEnd | BackEnd | FullStack

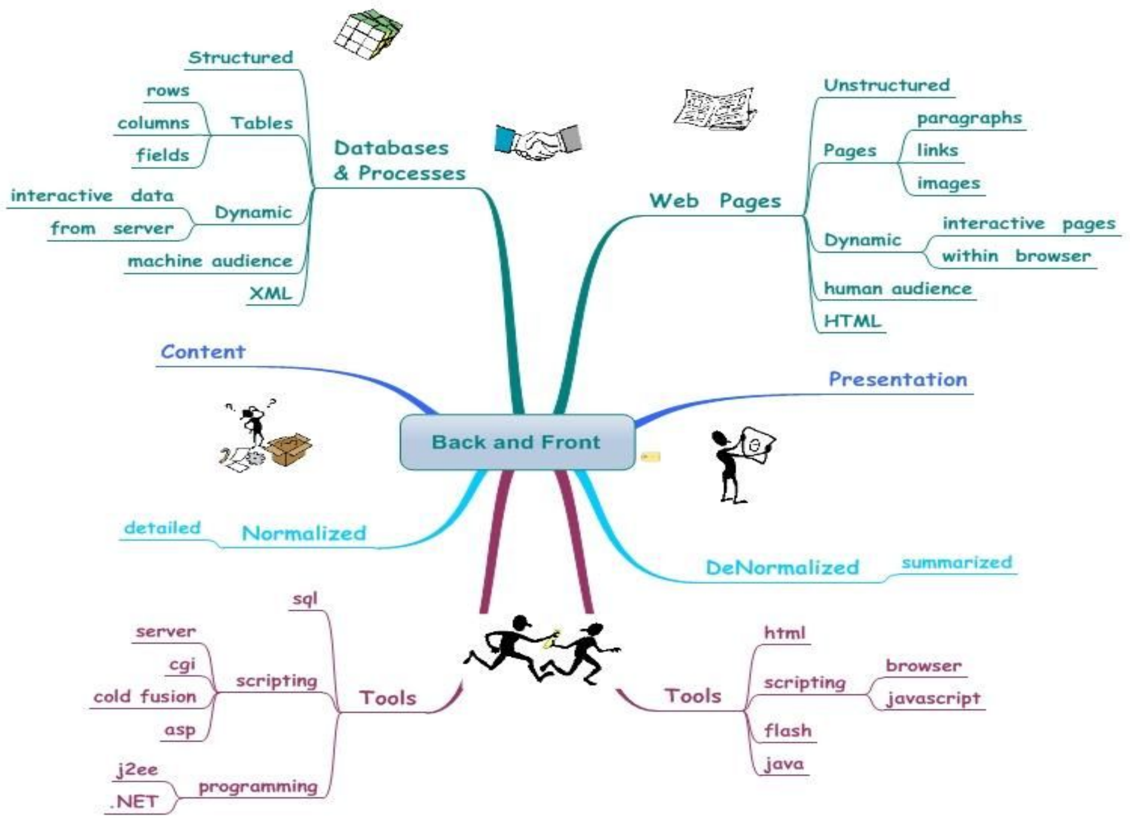
Built with ❤ by Joseph M Mwania

Introduction

In the Web Development Industry, developer-client hindrances do arise vis-à-vis technology mismatch, ideal frameworks to use, frameworks in beta release, SEO, target reach, design process, end-product, tweaking the nitty-gritty to wow both client and end user, etc.

Many a client who are in need of a Website App find themselves in a dilemma on who to hire, what tech to be used, and above all what the development process will entail. So the client would do what most people do. Google. They type and hit the search button only to realize that there are a myriad ways to solve their problems and approaching a skilled developer would be the best thing. So they search for portfolios (by the way, most designers/developers rarely keep them as they have already built a customer base) and eventually contact the seemingly best.

After a series of e-mailing, they realise that the dude or dudess(I'm all for affirmative action fellows :-) does only half of the work. The designer would normally do FrontEnd Design work, the developer would do BackEnd Development whilst a FullStack Developer has the sweet spot of doing both FrontEnd and BackEnd work.



From my experience, I have come to realize that many are not aware what it means by Client Side and Server Side. They always say; I just want it to look "Amazing", "Beautiful!", "Great!", or even "Awesome!". This is what has led me to do this project in tranches using SoC as an eye opener of what it means to work on the Client (FrontEnd) and Server (BackEnd).

Later on I'll amalgamate the FrontEnd and BackEnd systems to make a fully functional website App.

Many newbies in the industry never explain themselves so much on what to expect on a BackEnd design, only to wake up and find a deluge of e-mails telling them that the design lacks styling, responsiveness, etc. Then after explaining how the different platforms work, the client would realize that s/he is pointing at a FrontEnd issue, which has nothing to do with the BackEnd.

It is vital to know basic job descriptions in web development, the technologies to be used and what to expect. Accepting the agreement without knowing the terminologies used can be a setback and a time loss.

In order to visualize the server side functionalities, with the database (DB) and how the App interacts with data, I have created snapshots of the DB to try and visualize what is happening behind the scenes.

The source code is provided in the app and because the software design pattern that I have used to implement the UI is the Model View Controller MVC, then 90% of the code will reside within the following paths:

app/assets Mostly FrontEnd work that I will do subsequent to this whilst merging the two projects.
app/controllers To link data from the Model and send to view and vice versa.
app/views The view. (Which will be seen from your favorite browser.)
app/models The DB models
app/db The DB migrations, schema, seed data etc.

I'll restrict myself within the Model part of **MVC**, because that’s where the DB resides and coding to interact with it is done.

When people ask about the structure of the DB (which makes sense because it's part of Server Side Coding), the DB paths are usually sent to them and since code is not everyone's language, they accept the project only to revert back on the same issue creating a loop of correspondence.

That is why I have decided to create snapshots of the DB architecture using external software to visualize the whole process of how data is stored and accessed via MVC.

I would like to repeat that this project has both a Client Side, a Server Side and the full fledge Website App which will have both Technologies merged together. No sooner do I code and merge the third stack than I will push it to GitHub. (Possibly start of 2017 or there about)

I've tested the overall performance on different connections like GPRS, 2G, 3G, 4G, DSL, ISDN and 802.11(n and ac) and the load time is impressive given that there aren't images or videos at this point.

Anyway, the FrontEnd design runs on Bootstrap 3 whilst the BackEnd system runs on Ruby on Rails. I refrained to use many gem plugins like Simple Form, Device among others that make coding easier, and tried to use vanilla code, touching each part whilst keenly observing it's functionality and intractability. Another reason is because it helps me come with ideas and broaden my thinking. Oh yes! Programming makes you think.

Why Ruby on Rails?

Ruby is an Object Oriented Programming Language and almost every code is an object. It's easy to code, share with other developers and de-bug across platforms. Its well supported by the Open-Source community in GitHub and BitBucket and it's versatile in solving many problems in organizations. (*At the onset of 2016, a colleague of mine and I took a challenge to build a gem that would interact with Ruby to access, sort and classify Big Data in a certain firm. We saved the firm over 400 hrs of work if they had done that manually*)

Ruby is compatible with Frameworks like Rails, Sinatra, Padrino, Cuba etc and employs the Don’t Repeat Yourself (DRY) principle of coding.

When you get the hang of Ruby and a Framework of your choice you become very productive and organized. Security can be enhanced using a one line of code, and Cross-site Scripting (XSS) are rare in Ruby Applications.

Been an Open-Source enthusiast, I can't help mentioning that Ruby is Open Source like Linux, AForge.NET, Blender, OpenCog etc which is the future of Software Development Engineering in my opinion.

Software Engineering Technologies & Methodology

During the Software Development Process (SDP), many designers may elect to use Prototyping, Spiral or Waterfall methodologies to organize their ideas and thoughts on how to solve problems. They define specific deliverables to be done by a project team and later assemble and test the product.

As you will notice, my design approach leans more on Iterative and Incremental Development (IID). Combining cyclic processes of prototyping, testing and analyzing coupled with phase testing as the software grows; thus allowing partial UI/UX feel and early rectifications and enhancements.

RubyGems are designed and tested in the open-source arena, which are exemplary for IID from a design perspective; the use of Byebug, Kaminari, Bootstrap-sass and Faker enhanced different functionalities streamlining the design process.

Table of Contents

Ruby and Rails Environment

My coding environment.

```
ruby 2.3.1p112 (2016-04-26 revision 54768) [x86_64-darwin15] Rails 5.0.0.1 irb
version 0.9.6(09/06/30)  git version 2.6.4  node version v6.6.0 npm version 3.10.3
npm -cli version 3.10.3
```

Byebug minitest for Testing and Debugging

Byebug is a simple to use, feature rich debugger for Ruby 2. It uses the new TracePoint API for execution control and the new Debug Inspector API for call stack navigation, so it doesn't depend on internal core sources. It's developed as a C extension, so it's fast. And it has a full test suite so it's reliable.

It allows you to see what is going on *inside* a Ruby program while it executes and offers many of the traditional debugging features such as:

- Stepping: Running your program one line at a time.
- Breaking: Pausing the program at some event or specified instruction, to examine the current state.
- Evaluating: Basic REPL functionality, although [pry] does a better job at that.
- Tracking: Keeping track of the different values of your variables or the different lines executed by your program.

Build Status

Linux

Windows

Requirements

- Required: MRI 2.0.0 or higher. For debugging ruby 1.9.3 or older, use [debugger].
- Recommended:
  - MRI 2.1.8 or higher.
  - MRI 2.2.4 or higher.
  - MRI 2.3.0 or higher.

Kaminari

Kaminari

A Scope & Engine based, clean, powerful, customizable and sophisticated paginator for modern web app frameworks and ORMs

Clean

Does not globally pollute +Array+, +Hash+, +Object+ or `AR::Base`.

Easy to use

Just bundle the gem, then your models are ready to be paginated. No configuration required. Don't have to define anything in your models or helpers.

Simple scope-based API

Everything is method chainable with less "Hasheritis". You know, that's the Rails 3 way. No special collection class or anything for the paginated values, instead using a general `AR::Relation` instance. So, of course you can chain any other conditions before or after the paginator scope.

Customizable engine-based I18n-aware helper

As the whole pagination helper is basically just a collection of links and non-links, Kaminari renders each of them through its own partial template inside the Engine. So, you can easily modify their behaviour, style or whatever by overriding partial templates.

Bootsrap-sass

Dropped it in rails using asset pipeline.

```
gem 'bootstrap-sass', '~> 3.3.6' gem 'sass-rails', '>= 3.2'  # Imported Bootstrap
and it's sprockets in the application.scss @import "bootstrap-sprockets"; @import
"bootstrap";
```

Faker

Faker

This gem is a port of Perl's Data::Faker library that generates fake data.

It comes in very handy for taking screenshots (taking screenshots for my project, Catch the Best was the original impetus for the creation of this gem), having real-looking test data, and having your database populated with more than one or two records while you're doing development.

I used this gem to populate my database. It's as handy as Lorem Ipsum is handy in generating blind text. We developers use it for good purposes, that is populating data in our db and testing it's functionality while other people use it to generate thousands of e-mails to junk people's inboxes.

```
30.times do   post = Post.create(     title: Faker::Lorem.sentence(20),
content: Faker::Lorem.paragraph,     publish: true,     moderator: moderator)
tag = Tag.create(name: Faker::Lorem.word)    post_tag = PostTag.create(post: post,
tag: tag)     visitor = Visitor.create(fullname: Faker::Name.name,
email: Faker::Internet.email)   comment = Comment.create(message:
Faker::Lorem.paragraph,                            status: [true, false].sample,
post: post,                            visitor: visitor)   message =
Message.create(             content: Faker::Lorem.paragraph,            status:
[true, false].sample,            visitor: visitor)    notifiable = [visitor,
comment].sample    notification = Notification.create(     notifiable_id:
notifiable.id,     notifiable_type: notifiable.class.name) end
```

Gems

The following table shows the gems I used for this project. Most of the default gems were updated with current versions via Ruby Gems whilst others worked well in their older versions. I grouped some of the gems from their default state to development test or production so as not to mix things up.

Group	Gemfile	Version Type
default	'rails'	5.0.0.1
default	puma	3.4.0
default	bootstrap-sass	~> 3.3.6
default	sass-rails	>= 3.2
default	bcrypt	~> 3.1', '>= 3.1.11
default	faker	~> 1.6', '>= 1.6.6
default	kaminari	~> 0.17.0
default	uglifier	3.0.0
default	coffee-rails	4.2.1
default	jquery-rails	4.1.1
default	turbolinks	5.0.1
default	jbuilder	2.4.1
development	sqlite3	1.3.11
development	byebug	9.0.0
development	web-console	3.1.1
development	listen	3.0.8
development	spring	1.7.2
development	spring-watcher-listen	0.1.1
test	rails-controller-testing	2.4.1
test	minitest-reporters	1.1.9
test	guard	2.13.0
test	guard-minitest	2.4.4
production	pg	0.18.4 0.8.1
production	PostgreSQL	

Database Snapshots

Hashed Password

My DB's use SHA-256 storage of passwords and even though I may maintain some client servers, there is no way I can access user passwords as they are stored as a hash. To crack an hash algorithm a black hat hacker would need to find a collision within the hash using a preimage attack, possibly with the help of a cryptographic hash function. That would take O(2^n/2) time in case of integers where n ∈ ß (ß = Length of hash fn in bits)

SHA-2 outputs 512 bits and thus would take O(2^256) times. That means that if you had particulates of wheat flour numbering 1e+80, they would estimate to 1.185711e+80 which can give a sense of what O(2^256) means. For a 12 bytes password, it would roughly take ~0,22s (~2^-2s) for 65536 (1.185711e+80) computations to crack the hash which would take;

```
1.7668471e+72 * 2^-2 = 4.4171177e+71 ~ 1e+72s ~ 3,17 * 1e+64 years
```

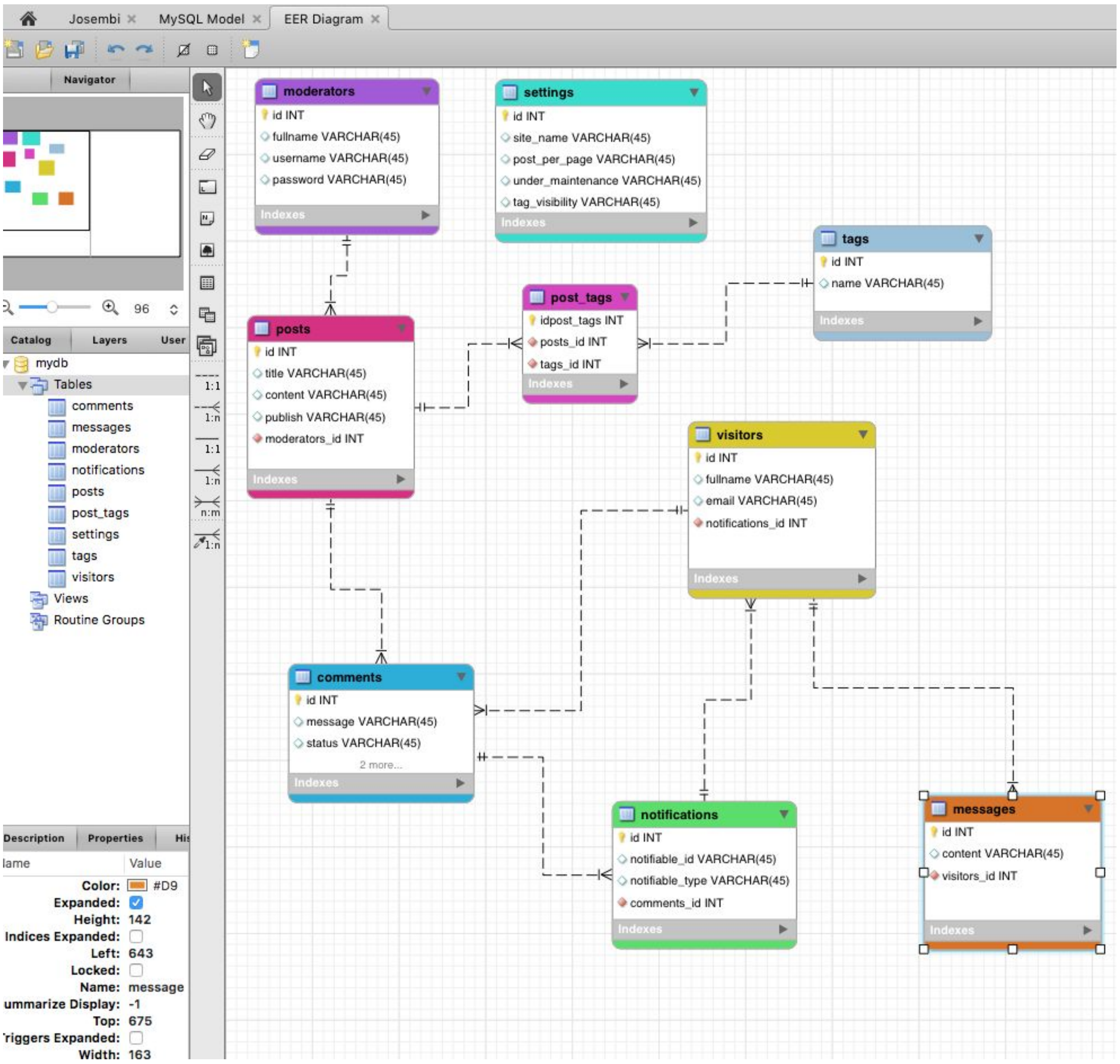
Even the best super-computers in the world that some folks are using to mint Bitcoins wouldn't be able to brute-force successfully that hash. That's why Ruby on Rails in built security features are efficient in detering attacks. Malicious scripting, CSRF, DoS, DDoS etc are often targeted in vulnerable web platfroms.

Active Record Models Tables

I created different models eg Comments, Messages, Moderators etc and made associations between them to simplify things by hitting the DB in a straight forward way without overloading it. This is very useful especially when the client is adding or deleting data from the View.

Some static images have been separated from my main repo because I had hosted them in Amazon CloudFront. I later tried Akamai which is also a cool CDN and had the same issue because apparently GitHub wants me to commit and track them. Meanwhile I'll use my private server host to target images on a popup upon clicking a link while I search for a solution.
Image problem solved via GitHub Cloud Services.*

Below is the Database Architecture Mockup.



lame	Value
Color:	#D9
Expanded:	✓
Height:	142
Indices Expanded:	☐
Left:	643
Locked:	☐
Name:	message
ummarize Display:	-1
Top:	675
'riggers Expanded:	☐
Width:	163

I have paired the models to illustrate which active record belongs where with a clear illustration of Associations (has_one, has_and_belongs_to_many, has_and_belongs, etc).

I also made a twist of the DB architecture design to create a Polymorphic Association. If you look at the "belongs_to" Notifications model declaration, you see that the model itself has an ID of type Integer, a notifiable_id which is a VARCHAR(45), an notifiable_type of VARCHAR(45) and a comments_id which is an integer {*I wont go in depth about SQL Data Types here*}. The vital thing to note is the ability to use polymorphism to save two models in one table.

Active Record Models Schema

The Ruby DSL within the Active Record is amazing in creating dynamic DB's. The RDBMS that Ruby uses is easily queried via the Interactive Ruby Shell (IRB) while programming. After creating a Mockup of how I intended the database to look, I created migrations and the schema for each model is illustrated below.

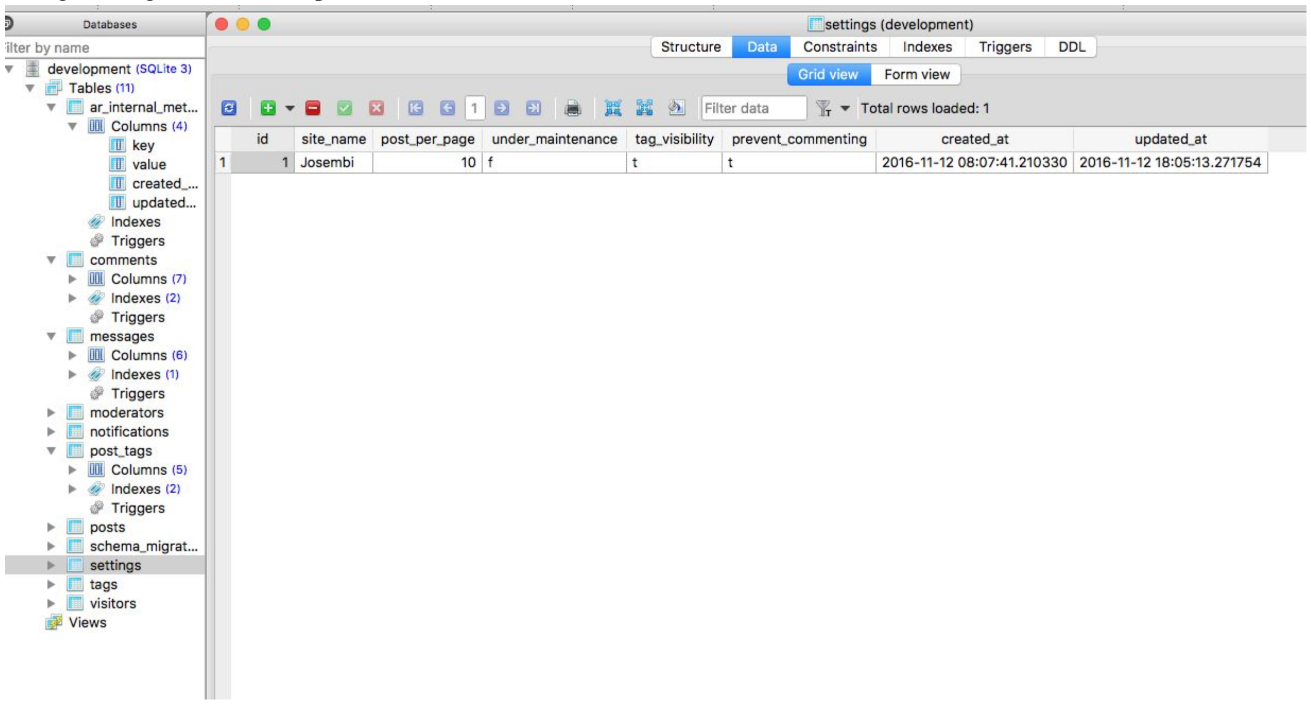
```ruby
ActiveRecord::Schema.define(version: 20161112093722) do

  create_table "comments", force: :cascade do |t|
    t.text     "message"
    t.boolean  "status",     default: false
    t.integer  "post_id"
    t.integer  "visitor_id"
    t.datetime "created_at",                 null: false
    t.datetime "updated_at",                 null: false
    t.index ["post_id"], name: "index_comments_on_post_id"
    t.index ["visitor_id"], name: "index_comments_on_visitor_id"
  end

  create_table "messages", force: :cascade do |t|
    t.text     "content"
    t.integer  "visitor_id"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.boolean  "status"
    t.index ["visitor_id"], name: "index_messages_on_visitor_id"
  end

  create_table "moderators", force: :cascade do |t|
    t.string   "fullname"
    t.string   "username"
    t.string   "password_digest"
    t.datetime "created_at",      null: false
    t.datetime "updated_at",      null: false
  end

  create_table "notifications", force: :cascade do |t|
    t.string   "notifiable_type"
    t.integer  "notifiable_id"
    t.datetime "created_at",      null: false
    t.datetime "updated_at",      null: false
    t.index ["notifiable_type", "notifiable_id"], name: "index_notifications_on_notifiable_type_and_notifiable_id"
  end

  create_table "post_tags", force: :cascade do |t|
    t.integer  "post_id"
    t.integer  "tag_id"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.index ["post_id"], name: "index_post_tags_on_post_id"
    t.index ["tag_id"], name: "index_post_tags_on_tag_id"
  end

  create_table "posts", force: :cascade do |t|
    t.string   "title"
    t.text     "content"
    t.boolean  "publish"
    t.integer  "moderator_id"
    t.datetime "created_at",   null: false
    t.datetime "updated_at",   null: false
    t.index ["moderator_id"], name: "index_posts_on_moderator_id"
  end

  create_table "settings", force: :cascade do |t|
    t.string   "site_name"
    t.integer  "post_per_page"
    t.boolean  "under_maintenance"
    t.boolean  "tag_visibility"
    t.boolean  "prevent_commenting"
    t.datetime "created_at",         null: false
    t.datetime "updated_at",         null: false
  end
  create_table "tags", force: :cascade do |t|
    t.string   "name"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
  end

  create_table "visitors", force: :cascade do |t|
    t.string   "fullname"
    t.string   "email"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
  end

end
```

Active Record Models Data

I won't go into details explaining what is happening on each individual model due to the complexities of data interaction within the MVC, but the important thing is to portray how data is stored, accessed, created and modified. From the UI of the website, the owner can grant administrator priviledges to third parties who would still act as a moderator.

Settings

The Settings model access is accessed only by the moderator via login credentials and s/he can approve posts, change page settings among other CRUD operations.

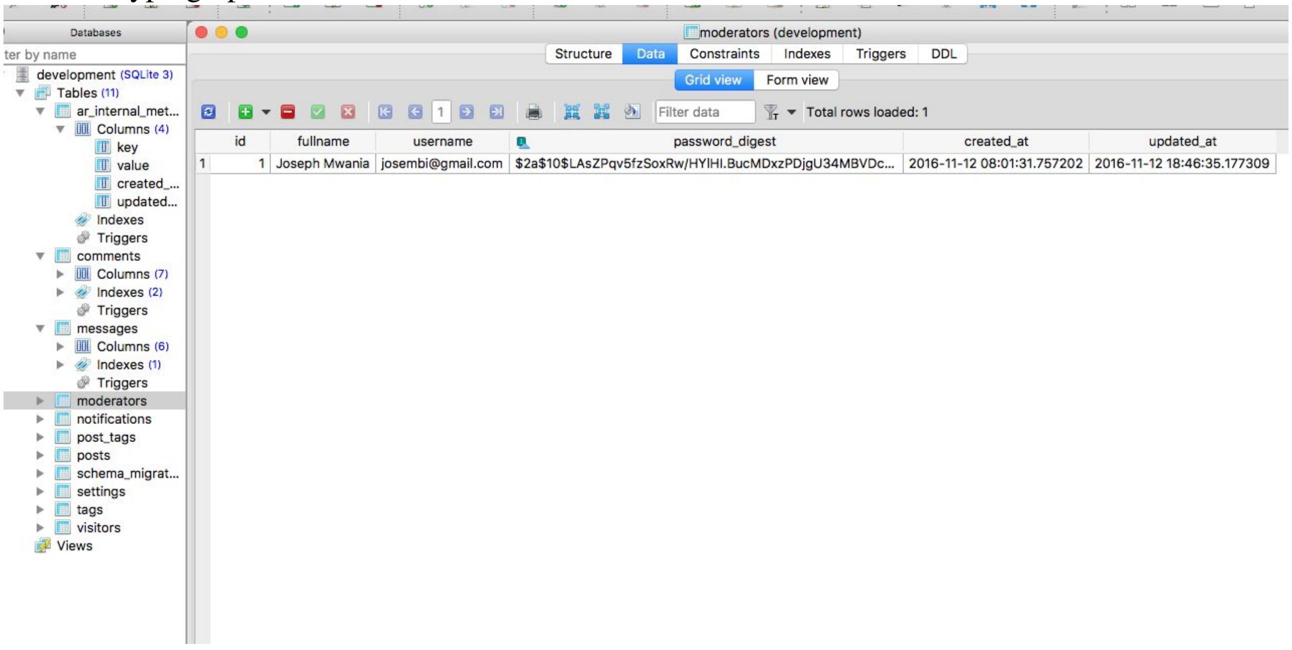


	id	site_name	post_per_page	under_maintenance	tag_visibility	prevent_commenting	created_at	updated_at
1	1	Josembi	10	f	t	t	2016-11-12 08:07:41.210330	2016-11-12 18:05:13.271754

The Website App name is displayed under the site_name. Number of posts can be limited per page and a boolean of true can trigger the Web App's status to under_maintenance sending it offline.

Moderators

A clear view of how the password_digest stores passwords. They cannot be viewed as string entity but as a cryptographic hash function

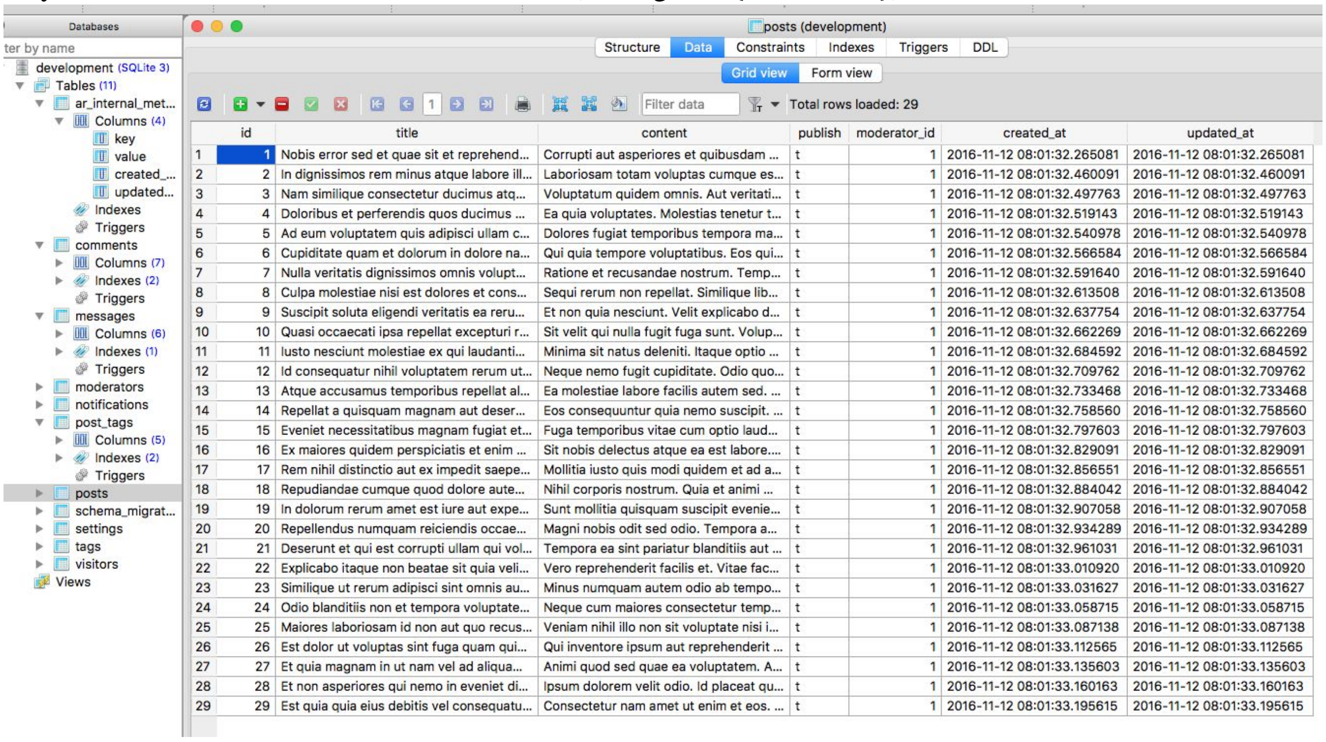


Total rows loaded: 1

	id	fullname	username	password_digest	created_at	updated_at
1	1	Joseph Mwania	josembi@gmail.com	$2a$10$LAsZPqv5fzSoxRw/HYlHI.BucMDxzPDjgU34MBVDc...	2016-11-12 08:01:31.757202	2016-11-12 18:46:35.177309

Posts

The Posts model can accept many comments (has_many), can accept many tags (has_many) and only the moderator can make modifications. (belongs_to {moderator})



Total rows loaded: 29

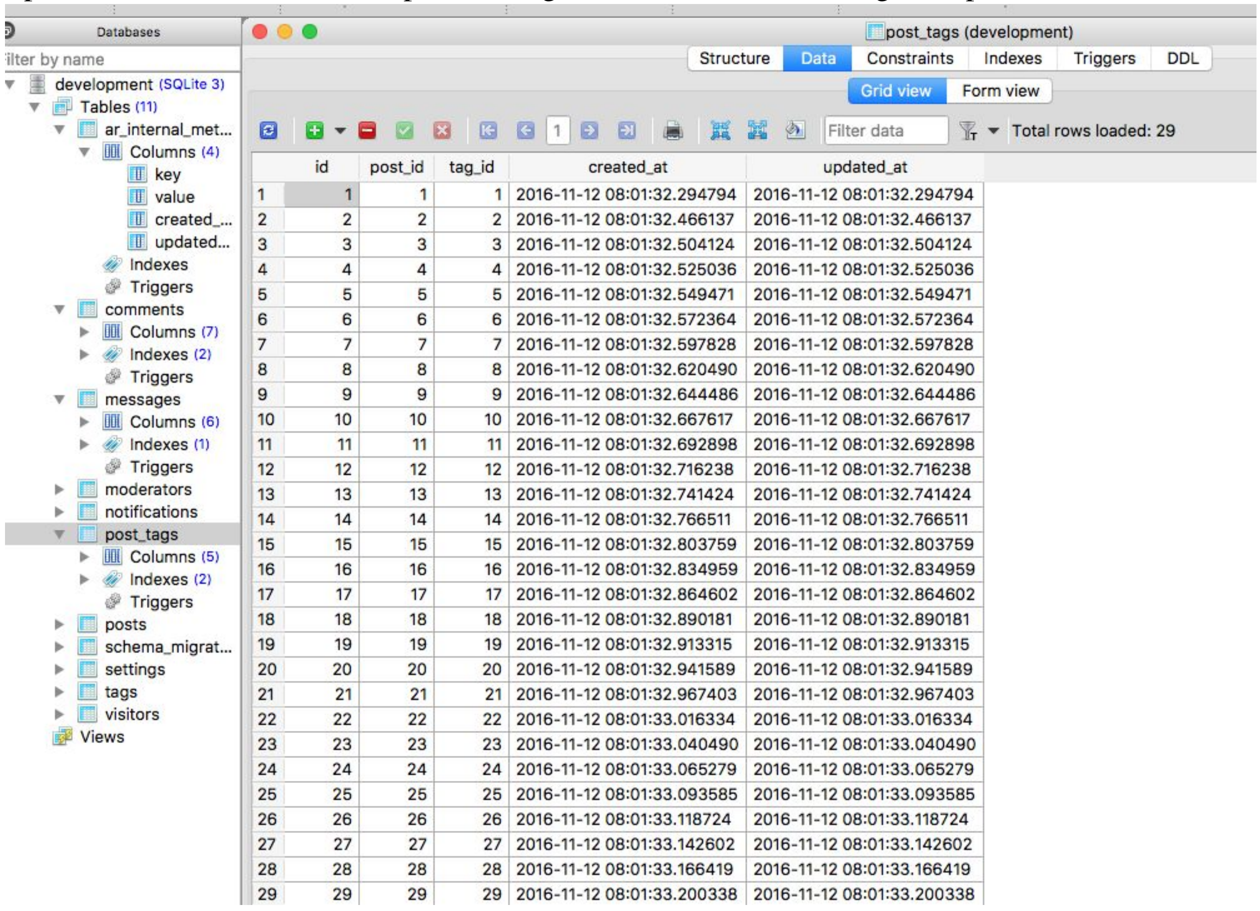
	id	title	content	publish	moderator_id	created_at	updated_at
1	1	Nobis error sed et quae sit et reprehend...	Corrupti aut asperiores et quibusdam ...	t	1	2016-11-12 08:01:32.265081	2016-11-12 08:01:32.265081
2	2	In dignissimos rem minus atque labore ill...	Laboriosam totam voluptas cumque es...	t	1	2016-11-12 08:01:32.460091	2016-11-12 08:01:32.460091
3	3	Nam similique consectetur ducimus atq...	Voluptatum quidem omnis. Aut veritati...	t	1	2016-11-12 08:01:32.497763	2016-11-12 08:01:32.497763
4	4	Doloribus et perferendis quos ducimus ...	Ea quia voluptates. Molestias tenetur t...	t	1	2016-11-12 08:01:32.519143	2016-11-12 08:01:32.519143
5	5	Ad eum voluptatem quis adipisci ullam c...	Dolores fugiat temporibus tempora ma...	t	1	2016-11-12 08:01:32.540978	2016-11-12 08:01:32.540978
6	6	Cupiditate quam et dolorum in dolore na...	Qui quia tempore voluptatibus. Eos qui...	t	1	2016-11-12 08:01:32.566584	2016-11-12 08:01:32.566584
7	7	Nulla veritatis dignissimos omnis volupt...	Ratione et recusandae nostrum. Temp...	t	1	2016-11-12 08:01:32.591640	2016-11-12 08:01:32.591640
8	8	Culpa molestiae nisi est dolores et cons...	Sequi rerum non repellat. Similique lib...	t	1	2016-11-12 08:01:32.613508	2016-11-12 08:01:32.613508
9	9	Suscipit soluta eligendi veritatis ea reru...	Et non quia nesciunt. Velit explicabo d...	t	1	2016-11-12 08:01:32.637754	2016-11-12 08:01:32.637754
10	10	Quasi occaecati ipsa repellat excepturi r...	Sit velit qui nulla fugit fuga sunt. Volup...	t	1	2016-11-12 08:01:32.662269	2016-11-12 08:01:32.662269
11	11	Iusto nesciunt molestiae ex qui laudanti...	Minima sit natus deleniti. Itaque optio ...	t	1	2016-11-12 08:01:32.684592	2016-11-12 08:01:32.684592
12	12	Id consequatur nihil voluptatem rerum ut...	Neque nemo fugit cupiditate. Odio quo...	t	1	2016-11-12 08:01:32.709762	2016-11-12 08:01:32.709762
13	13	Atque accusamus temporibus repellat al...	Ea molestiae labore facilis autem sed. ...	t	1	2016-11-12 08:01:32.733468	2016-11-12 08:01:32.733468
14	14	Repellat a quisquam magnam aut deser...	Eos consequuntur quia nemo suscipit. ...	t	1	2016-11-12 08:01:32.758560	2016-11-12 08:01:32.758560
15	15	Eveniet necessitatibus magnam fugiat et...	Fuga temporibus vitae cum optio laud...	t	1	2016-11-12 08:01:32.797603	2016-11-12 08:01:32.797603
16	16	Ex maiores quidem perspiciatis et enim ...	Sit nobis delectus atque ea est labore....	t	1	2016-11-12 08:01:32.829091	2016-11-12 08:01:32.829091
17	17	Rem nihil distinctio aut ex impedit saepe...	Mollitia iusto quis modi quidem et ad a...	t	1	2016-11-12 08:01:32.856551	2016-11-12 08:01:32.856551
18	18	Repudiandae cumque quod dolore aute...	Nihil corporis nostrum. Quia et animi ...	t	1	2016-11-12 08:01:32.884042	2016-11-12 08:01:32.884042
19	19	In dolorum rerum amet est iure aut expe...	Sunt mollitia quisquam suscipit evenie...	t	1	2016-11-12 08:01:32.907058	2016-11-12 08:01:32.907058
20	20	Repellendus numquam reiciendis occae...	Magni nobis odit sed odio. Tempora a...	t	1	2016-11-12 08:01:32.934289	2016-11-12 08:01:32.934289
21	21	Deserunt et qui est corrupti ullam qui vol...	Tempora ea sint pariatur blanditiis aut ...	t	1	2016-11-12 08:01:32.961031	2016-11-12 08:01:32.961031
22	22	Explicabo itaque non beatae sit quia veli...	Vero reprehenderit facilis et. Vitae fac...	t	1	2016-11-12 08:01:33.010920	2016-11-12 08:01:33.010920
23	23	Similique ut rerum adipisci sint omnis au...	Minus numquam autem odio ab tempo...	t	1	2016-11-12 08:01:33.031627	2016-11-12 08:01:33.031627
24	24	Odio blanditiis non et tempora voluptate...	Neque cum maiores consectetur temp...	t	1	2016-11-12 08:01:33.058715	2016-11-12 08:01:33.058715
25	25	Maiores laboriosam id non aut quo recus...	Veniam nihil illo non sit voluptate nisi i...	t	1	2016-11-12 08:01:33.087138	2016-11-12 08:01:33.087138
26	26	Est dolor ut voluptas sint fuga quam qui...	Qui inventore ipsum aut reprehenderit ...	t	1	2016-11-12 08:01:33.112565	2016-11-12 08:01:33.112565
27	27	Et quia magnam in ut nam vel ad aliqua...	Animi quod sed quae ea voluptatem. A...	t	1	2016-11-12 08:01:33.135603	2016-11-12 08:01:33.135603
28	28	Et non asperiores qui nemo in eveniet di...	Ipsum dolorem velit odio. Id placeat qu...	t	1	2016-11-12 08:01:33.160163	2016-11-12 08:01:33.160163
29	29	Est quia quia eius debitis vel consequatu...	Consectetur nam amet ut enim et eos. ...	t	1	2016-11-12 08:01:33.195615	2016-11-12 08:01:33.195615

Ruby is pretty readable and the code Associations can illustrate that below.

```ruby
class Post < ActiveRecord::Base
  has_many :comments, dependent: :destroy
  has_many :post_tags, dependent: :destroy
  has_many :tags, through: :post_tags
  belongs_to :moderator

  validates :title, presence: true
  validates :content, presence: true

  scope :published, -> { where(publish: true).order(id: :desc) }

  def self.josembi search
    where("title LIKE ? OR content LIKE ?", "%#{search}%", "%#{search}%")
  end

  def self.joe_beppi_tags param_tag
    includes(:tags).where(publish: true, tags: {name: param_tag}).order(id: :desc)
  end
end
```

Post Tags

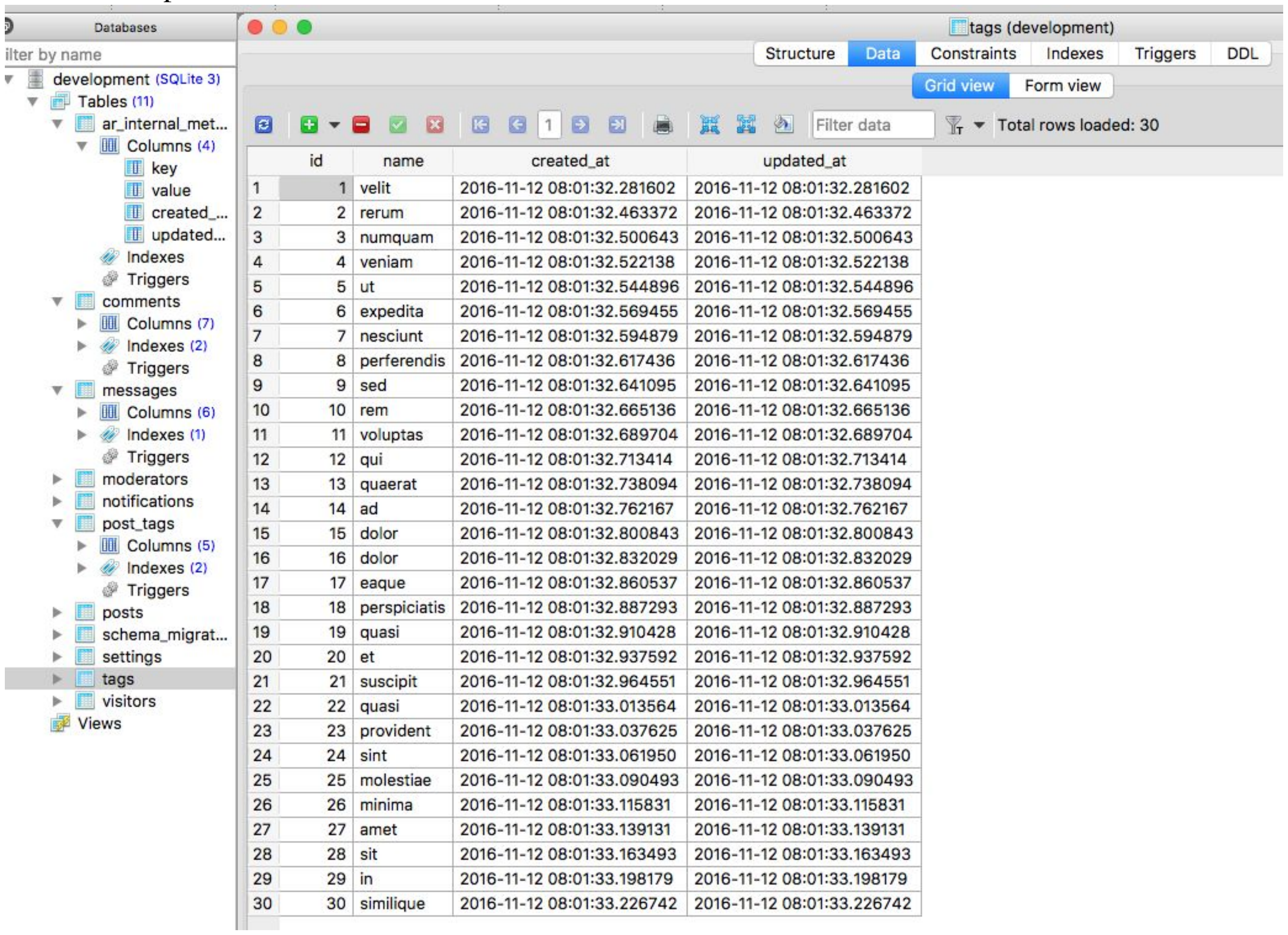
Apart from the main table id, a post and tag id have been added to tag each post.



	id	post_id	tag_id	created_at	updated_at
1	1	1	1	2016-11-12 08:01:32.294794	2016-11-12 08:01:32.294794
2	2	2	2	2016-11-12 08:01:32.466137	2016-11-12 08:01:32.466137
3	3	3	3	2016-11-12 08:01:32.504124	2016-11-12 08:01:32.504124
4	4	4	4	2016-11-12 08:01:32.525036	2016-11-12 08:01:32.525036
5	5	5	5	2016-11-12 08:01:32.549471	2016-11-12 08:01:32.549471
6	6	6	6	2016-11-12 08:01:32.572364	2016-11-12 08:01:32.572364
7	7	7	7	2016-11-12 08:01:32.597828	2016-11-12 08:01:32.597828
8	8	8	8	2016-11-12 08:01:32.620490	2016-11-12 08:01:32.620490
9	9	9	9	2016-11-12 08:01:32.644486	2016-11-12 08:01:32.644486
10	10	10	10	2016-11-12 08:01:32.667617	2016-11-12 08:01:32.667617
11	11	11	11	2016-11-12 08:01:32.692898	2016-11-12 08:01:32.692898
12	12	12	12	2016-11-12 08:01:32.716238	2016-11-12 08:01:32.716238
13	13	13	13	2016-11-12 08:01:32.741424	2016-11-12 08:01:32.741424
14	14	14	14	2016-11-12 08:01:32.766511	2016-11-12 08:01:32.766511
15	15	15	15	2016-11-12 08:01:32.803759	2016-11-12 08:01:32.803759
16	16	16	16	2016-11-12 08:01:32.834959	2016-11-12 08:01:32.834959
17	17	17	17	2016-11-12 08:01:32.864602	2016-11-12 08:01:32.864602
18	18	18	18	2016-11-12 08:01:32.890181	2016-11-12 08:01:32.890181
19	19	19	19	2016-11-12 08:01:32.913315	2016-11-12 08:01:32.913315
20	20	20	20	2016-11-12 08:01:32.941589	2016-11-12 08:01:32.941589
21	21	21	21	2016-11-12 08:01:32.967403	2016-11-12 08:01:32.967403
22	22	22	22	2016-11-12 08:01:33.016334	2016-11-12 08:01:33.016334
23	23	23	23	2016-11-12 08:01:33.040490	2016-11-12 08:01:33.040490
24	24	24	24	2016-11-12 08:01:33.065279	2016-11-12 08:01:33.065279
25	25	25	25	2016-11-12 08:01:33.093585	2016-11-12 08:01:33.093585
26	26	26	26	2016-11-12 08:01:33.118724	2016-11-12 08:01:33.118724
27	27	27	27	2016-11-12 08:01:33.142602	2016-11-12 08:01:33.142602
28	28	28	28	2016-11-12 08:01:33.166419	2016-11-12 08:01:33.166419
29	29	29	29	2016-11-12 08:01:33.200338	2016-11-12 08:01:33.200338

Tags

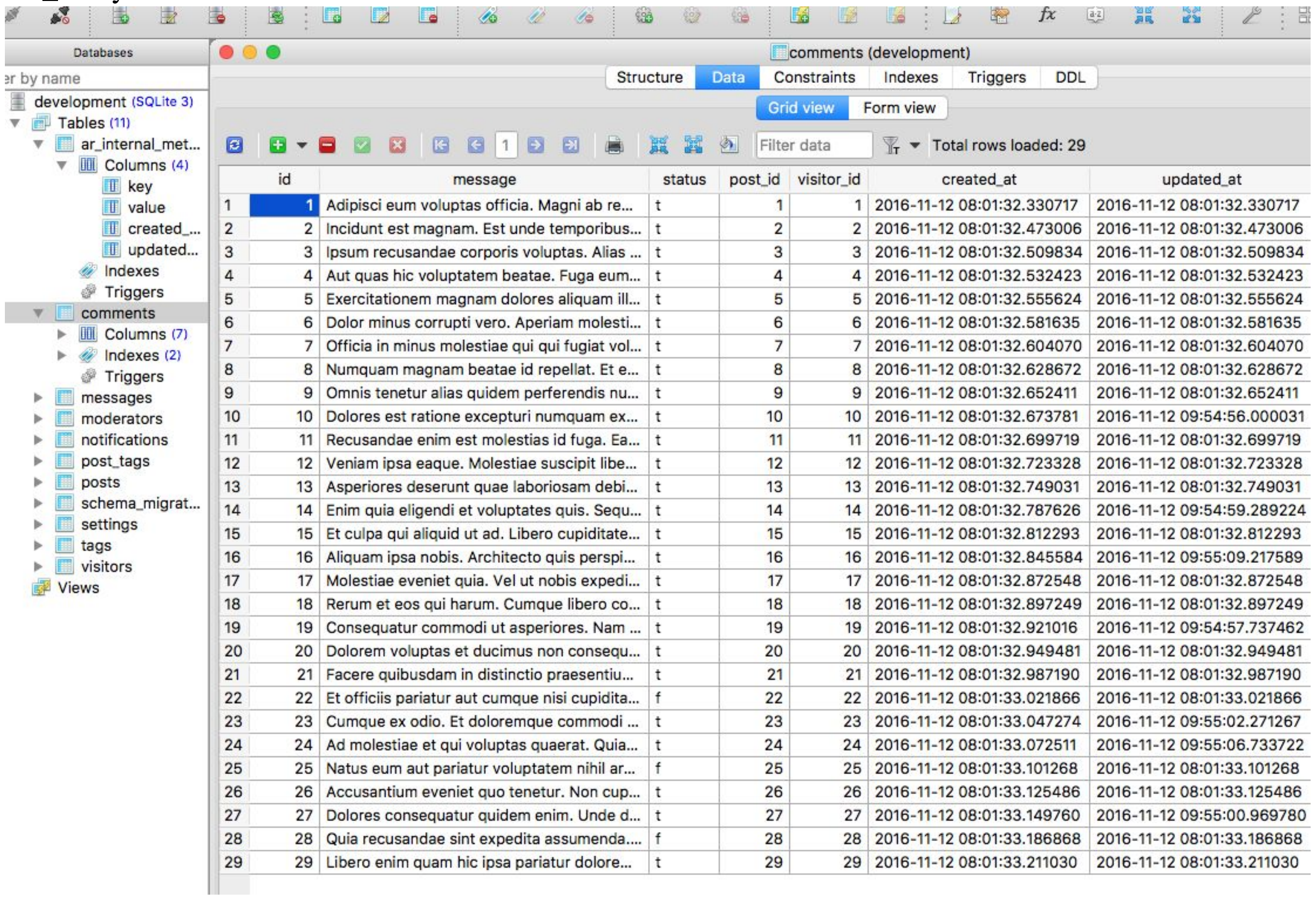
A tag name of value string has been added. This can be handy when searching the database. In present day, a database can hit the one million records mark within days if not hours in cases of social media platforms.



	id	name	created_at	updated_at
1	1	velit	2016-11-12 08:01:32.281602	2016-11-12 08:01:32.281602
2	2	rerum	2016-11-12 08:01:32.463372	2016-11-12 08:01:32.463372
3	3	numquam	2016-11-12 08:01:32.500643	2016-11-12 08:01:32.500643
4	4	veniam	2016-11-12 08:01:32.522138	2016-11-12 08:01:32.522138
5	5	ut	2016-11-12 08:01:32.544896	2016-11-12 08:01:32.544896
6	6	expedita	2016-11-12 08:01:32.569455	2016-11-12 08:01:32.569455
7	7	nesciunt	2016-11-12 08:01:32.594879	2016-11-12 08:01:32.594879
8	8	perferendis	2016-11-12 08:01:32.617436	2016-11-12 08:01:32.617436
9	9	sed	2016-11-12 08:01:32.641095	2016-11-12 08:01:32.641095
10	10	rem	2016-11-12 08:01:32.665136	2016-11-12 08:01:32.665136
11	11	voluptas	2016-11-12 08:01:32.689704	2016-11-12 08:01:32.689704
12	12	qui	2016-11-12 08:01:32.713414	2016-11-12 08:01:32.713414
13	13	quaerat	2016-11-12 08:01:32.738094	2016-11-12 08:01:32.738094
14	14	ad	2016-11-12 08:01:32.762167	2016-11-12 08:01:32.762167
15	15	dolor	2016-11-12 08:01:32.800843	2016-11-12 08:01:32.800843
16	16	dolor	2016-11-12 08:01:32.832029	2016-11-12 08:01:32.832029
17	17	eaque	2016-11-12 08:01:32.860537	2016-11-12 08:01:32.860537
18	18	perspiciatis	2016-11-12 08:01:32.887293	2016-11-12 08:01:32.887293
19	19	quasi	2016-11-12 08:01:32.910428	2016-11-12 08:01:32.910428
20	20	et	2016-11-12 08:01:32.937592	2016-11-12 08:01:32.937592
21	21	suscipit	2016-11-12 08:01:32.964551	2016-11-12 08:01:32.964551
22	22	quasi	2016-11-12 08:01:33.013564	2016-11-12 08:01:33.013564
23	23	provident	2016-11-12 08:01:33.037625	2016-11-12 08:01:33.037625
24	24	sint	2016-11-12 08:01:33.061950	2016-11-12 08:01:33.061950
25	25	molestiae	2016-11-12 08:01:33.090493	2016-11-12 08:01:33.090493
26	26	minima	2016-11-12 08:01:33.115831	2016-11-12 08:01:33.115831
27	27	amet	2016-11-12 08:01:33.139131	2016-11-12 08:01:33.139131
28	28	sit	2016-11-12 08:01:33.163493	2016-11-12 08:01:33.163493
29	29	in	2016-11-12 08:01:33.198179	2016-11-12 08:01:33.198179
30	30	similique	2016-11-12 08:01:33.226742	2016-11-12 08:01:33.226742

Comments

The Comments model inherits from the ApplicationRecord and belongs_to Visitor, Post and has_many notifications.

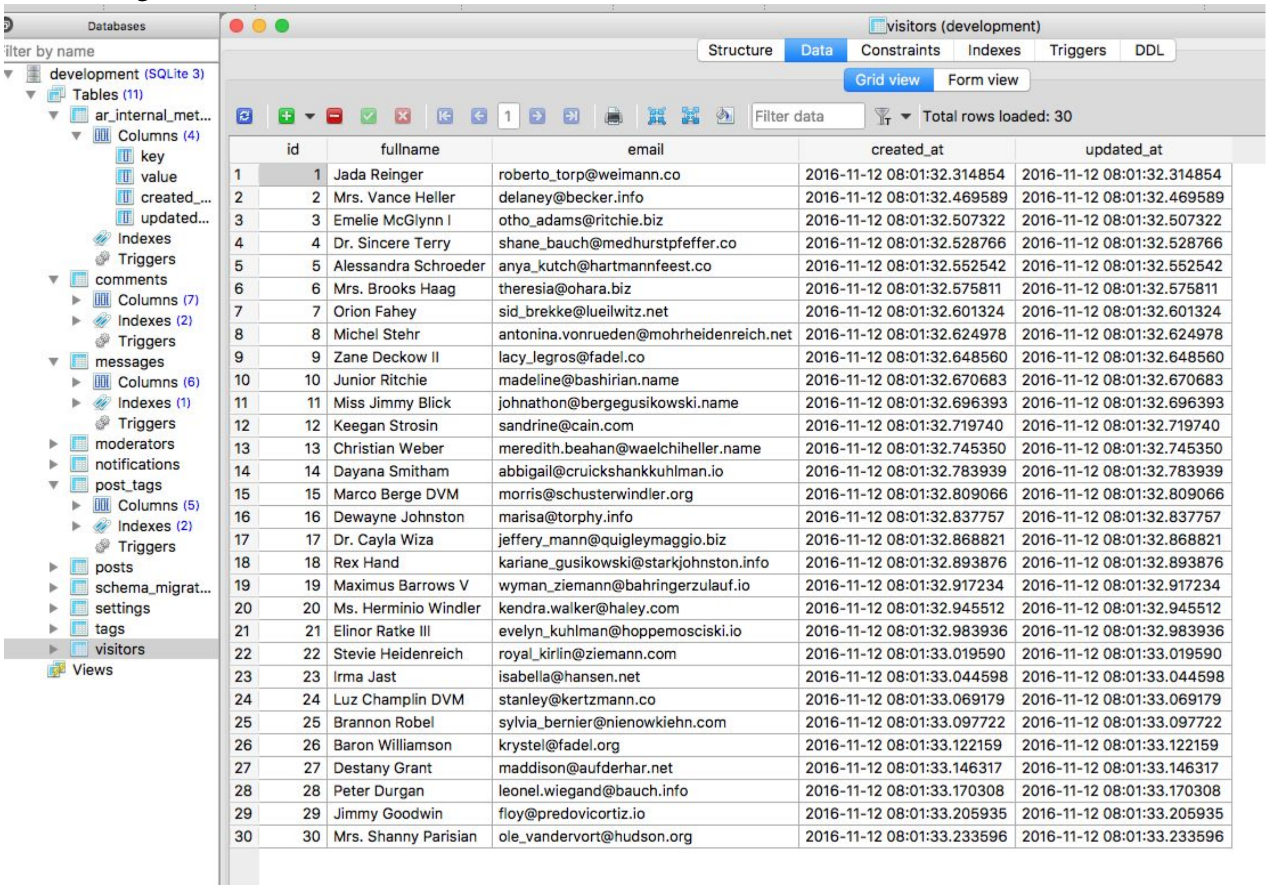


	id	message	status	post_id	visitor_id	created_at	updated_at
1	1	Adipisci eum voluptas officia. Magni ab re...	t	1	1	2016-11-12 08:01:32.330717	2016-11-12 08:01:32.330717
2	2	Incidunt est magnam. Est unde temporibus...	t	2	2	2016-11-12 08:01:32.473006	2016-11-12 08:01:32.473006
3	3	Ipsum recusandae corporis voluptas. Alias ...	t	3	3	2016-11-12 08:01:32.509834	2016-11-12 08:01:32.509834
4	4	Aut quas hic voluptatem beatae. Fuga eum...	t	4	4	2016-11-12 08:01:32.532423	2016-11-12 08:01:32.532423
5	5	Exercitationem magnam dolores aliquam ill...	t	5	5	2016-11-12 08:01:32.555624	2016-11-12 08:01:32.555624
6	6	Dolor minus corrupti vero. Aperiam molesti...	t	6	6	2016-11-12 08:01:32.581635	2016-11-12 08:01:32.581635
7	7	Officia in minus molestiae qui qui fugiat vol...	t	7	7	2016-11-12 08:01:32.604070	2016-11-12 08:01:32.604070
8	8	Numquam magnam beatae id repellat. Et e...	t	8	8	2016-11-12 08:01:32.628672	2016-11-12 08:01:32.628672
9	9	Omnis tenetur alias quidem perferendis nu...	t	9	9	2016-11-12 08:01:32.652411	2016-11-12 08:01:32.652411
10	10	Dolores est ratione excepturi numquam ex...	t	10	10	2016-11-12 08:01:32.673781	2016-11-12 09:54:56.000031
11	11	Recusandae enim est molestias id fuga. Ea...	t	11	11	2016-11-12 08:01:32.699719	2016-11-12 08:01:32.699719
12	12	Veniam ipsa eaque. Molestiae suscipit libe...	t	12	12	2016-11-12 08:01:32.723328	2016-11-12 08:01:32.723328
13	13	Asperiores deserunt quae laboriosam debi...	t	13	13	2016-11-12 08:01:32.749031	2016-11-12 08:01:32.749031
14	14	Enim quia eligendi et voluptates quis. Sequ...	t	14	14	2016-11-12 08:01:32.787626	2016-11-12 09:54:59.289224
15	15	Et culpa qui aliquid ut ad. Libero cupiditate...	t	15	15	2016-11-12 08:01:32.812293	2016-11-12 08:01:32.812293
16	16	Aliquam ipsa nobis. Architecto quis perspi...	t	16	16	2016-11-12 08:01:32.845584	2016-11-12 09:55:09.217589
17	17	Molestiae eveniet quia. Vel ut nobis expedi...	t	17	17	2016-11-12 08:01:32.872548	2016-11-12 08:01:32.872548
18	18	Rerum et eos qui harum. Cumque libero co...	t	18	18	2016-11-12 08:01:32.897249	2016-11-12 08:01:32.897249
19	19	Consequatur commodi ut asperiores. Nam ...	t	19	19	2016-11-12 08:01:32.921016	2016-11-12 09:54:57.737462
20	20	Dolorem voluptas et ducimus non consequ...	t	20	20	2016-11-12 08:01:32.949481	2016-11-12 08:01:32.949481
21	21	Facere quibusdam in distinctio praesentiu...	t	21	21	2016-11-12 08:01:32.987190	2016-11-12 08:01:32.987190
22	22	Et officiis pariatur aut cumque nisi cupidita...	f	22	22	2016-11-12 08:01:33.021866	2016-11-12 08:01:33.021866
23	23	Cumque ex odio. Et doloremque commodi ...	t	23	23	2016-11-12 08:01:33.047274	2016-11-12 09:55:02.271267
24	24	Ad molestiae et qui voluptas quaerat. Quia...	t	24	24	2016-11-12 08:01:33.072511	2016-11-12 09:55:06.733722
25	25	Natus eum aut pariatur voluptatem nihil ar...	f	25	25	2016-11-12 08:01:33.101268	2016-11-12 08:01:33.101268
26	26	Accusantium eveniet quo tenetur. Non cup...	t	26	26	2016-11-12 08:01:33.125486	2016-11-12 08:01:33.125486
27	27	Dolores consequatur quidem enim. Unde d...	t	27	27	2016-11-12 08:01:33.149760	2016-11-12 09:55:00.969780
28	28	Quia recusandae sint expedita assumenda....	f	28	28	2016-11-12 08:01:33.186868	2016-11-12 08:01:33.186868
29	29	Libero enim quam hic ipsa pariatur dolore...	t	29	29	2016-11-12 08:01:33.211030	2016-11-12 08:01:33.211030

```ruby
class Comment < ApplicationRecord
  belongs_to :post
  belongs_to :visitor
  has_many :notifications, as: :notifiable,  dependent: :destroy

  validates :message, presence: true

  scope :approved, -> { where status: true }

  def self.josembi_wa_kimeu params
    joins(:visitor).where("fullname LIKE ? OR message LIKE ?", "%#{params}%", "%#{params}%")
  end

  include Josembi
end
```

Visitors

Hopefully the trend is becoming clear now. The image shows generated data of the Visitor model. Each visitor has a fullname, an e-mail and the date s/he was created and/or updated. Each visitor will accept nested attributes from Comments & Messages and has_many (Notifications, Comments and Messages).



Total rows loaded: 30

	id	fullname	email	created_at	updated_at
1	1	Jada Reinger	roberto_torp@weimann.co	2016-11-12 08:01:32.314854	2016-11-12 08:01:32.314854
2	2	Mrs. Vance Heller	delaney@becker.info	2016-11-12 08:01:32.469589	2016-11-12 08:01:32.469589
3	3	Emelie McGlynn I	otho_adams@ritchie.biz	2016-11-12 08:01:32.507322	2016-11-12 08:01:32.507322
4	4	Dr. Sincere Terry	shane_bauch@medhurstpfeffer.co	2016-11-12 08:01:32.528766	2016-11-12 08:01:32.528766
5	5	Alessandra Schroeder	anya_kutch@hartmannfeest.co	2016-11-12 08:01:32.552542	2016-11-12 08:01:32.552542
6	6	Mrs. Brooks Haag	theresia@ohara.biz	2016-11-12 08:01:32.575811	2016-11-12 08:01:32.575811
7	7	Orion Fahey	sid_brekke@lueilwitz.net	2016-11-12 08:01:32.601324	2016-11-12 08:01:32.601324
8	8	Michel Stehr	antonina.vonrueden@mohrheidenreich.net	2016-11-12 08:01:32.624978	2016-11-12 08:01:32.624978
9	9	Zane Deckow II	lacy_legros@fadel.co	2016-11-12 08:01:32.648560	2016-11-12 08:01:32.648560
10	10	Junior Ritchie	madeline@bashirian.name	2016-11-12 08:01:32.670683	2016-11-12 08:01:32.670683
11	11	Miss Jimmy Blick	johnathon@bergegusikowski.name	2016-11-12 08:01:32.696393	2016-11-12 08:01:32.696393
12	12	Keegan Strosin	sandrine@cain.com	2016-11-12 08:01:32.719740	2016-11-12 08:01:32.719740
13	13	Christian Weber	meredith.beahan@waelchiheller.name	2016-11-12 08:01:32.745350	2016-11-12 08:01:32.745350
14	14	Dayana Smitham	abbigail@cruickshankkuhlman.io	2016-11-12 08:01:32.783939	2016-11-12 08:01:32.783939
15	15	Marco Berge DVM	morris@schusterwindler.org	2016-11-12 08:01:32.809066	2016-11-12 08:01:32.809066
16	16	Dewayne Johnston	marisa@torphy.info	2016-11-12 08:01:32.837757	2016-11-12 08:01:32.837757
17	17	Dr. Cayla Wiza	jeffery_mann@quigleymaggio.biz	2016-11-12 08:01:32.868821	2016-11-12 08:01:32.868821
18	18	Rex Hand	kariane_gusikowski@starkjohnston.info	2016-11-12 08:01:32.893876	2016-11-12 08:01:32.893876
19	19	Maximus Barrows V	wyman_ziemann@bahringerzulauf.io	2016-11-12 08:01:32.917234	2016-11-12 08:01:32.917234
20	20	Ms. Herminio Windler	kendra.walker@haley.com	2016-11-12 08:01:32.945512	2016-11-12 08:01:32.945512
21	21	Elinor Ratke III	evelyn_kuhlman@hoppemosciski.io	2016-11-12 08:01:32.983936	2016-11-12 08:01:32.983936
22	22	Stevie Heidenreich	royal_kirlin@ziemann.com	2016-11-12 08:01:33.019590	2016-11-12 08:01:33.019590
23	23	Irma Jast	isabella@hansen.net	2016-11-12 08:01:33.044598	2016-11-12 08:01:33.044598
24	24	Luz Champlin DVM	stanley@kertzmann.co	2016-11-12 08:01:33.069179	2016-11-12 08:01:33.069179
25	25	Brannon Robel	sylvia_bernier@nienowkiehn.com	2016-11-12 08:01:33.097722	2016-11-12 08:01:33.097722
26	26	Baron Williamson	krystel@fadel.org	2016-11-12 08:01:33.122159	2016-11-12 08:01:33.122159
27	27	Destany Grant	maddison@aufderhar.net	2016-11-12 08:01:33.146317	2016-11-12 08:01:33.146317
28	28	Peter Durgan	leonel.wiegand@bauch.info	2016-11-12 08:01:33.170308	2016-11-12 08:01:33.170308
29	29	Jimmy Goodwin	floy@predovicortiz.io	2016-11-12 08:01:33.205935	2016-11-12 08:01:33.205935
30	30	Mrs. Shanny Parisian	ole_vandervort@hudson.org	2016-11-12 08:01:33.233596	2016-11-12 08:01:33.233596

For a cleaner coding environment, I've created services, one of them called "visitor_jose_service.rb" that will ensure efficient communication by the controller to the database. The rule of the thump is to have a fast access of data. In an e-commerce website where upselling is vital, you dont want users to switch to another website due to a slow platform.

```ruby
class VisitorJoseService
    attr_reader :params
  def initialize(params)
    @params = params
  end

  def visitor
    build_existing_visitor_message || build_new_visitor_message
  end

  private
  def existing_visitor
    @visitor ||= Visitor.find_by(email: params[:email])
  end

  def build_new_visitor_message
    Visitor.new(params)
  end

  def message
    params[:messages_attributes]['0']
  end

  def build_existing_visitor_message
    return unless existing_visitor
    existing_visitor.tap do |v|
      v.messages << Comment.new(message)
  end
end
end
```

The instance variable @visitor which is under the visitor function will manage the VisitorJoseService under the class declaration above. Under the main class of MessagesController below, data inheritance is from the ApplicationController and the visitor method instantiates @visitor and points it to the aforementioned VisitorJoseService.

```ruby
class MessagesController < ApplicationController
  def new
    @visitor_message = Visitor.new(messages: [Message.new])
  end

  def create
    if visitor.save
      flash[:notice] = "Successfully sent your message"
      redirect_to new_message_path
    else
      @visitor_message = visitor
      render :new
    end
  end

  private

  def strong_params
    params.require(:visitor).permit(:fullname, :email, messages_attributes: [:content])
  end

  def visitor
    @visitor ||= VisitorJoseService.new(strong_params).visitor
  end
end
```

For security reasons (attacks, malicious scripts, etc), I created two controllers for Messages, Comments and Posts. Controllers performing CRUD operations will inherits data from the Admin::ApplicationController, former ActionController::Base in previous Rails versions.

```ruby
class Admin::MessagesController < Admin::ApplicationController

  def index
    if params[:search].present?
      @messages = Message.josembi_the_rubist_find_content(params[:search]).page params[:page]
    else
      @messages = Message.all.order(id: :desc).page params[:page]
    end
  end

  def show
    @message = Message.find(params[:id])
    @message.kasyula_msg_read
  end

  def update
    @message = Message.find(params[:id])
    @message.update(status: params[:status])

    redirect_to :back, notice: 'Successfullly updated message'
  end

  def destroy
    @message = Message.find(params[:id])
    @message = Message.destroy

    redirect_to :back, notice: 'Successfully deleted Visitor'
  end
end
```

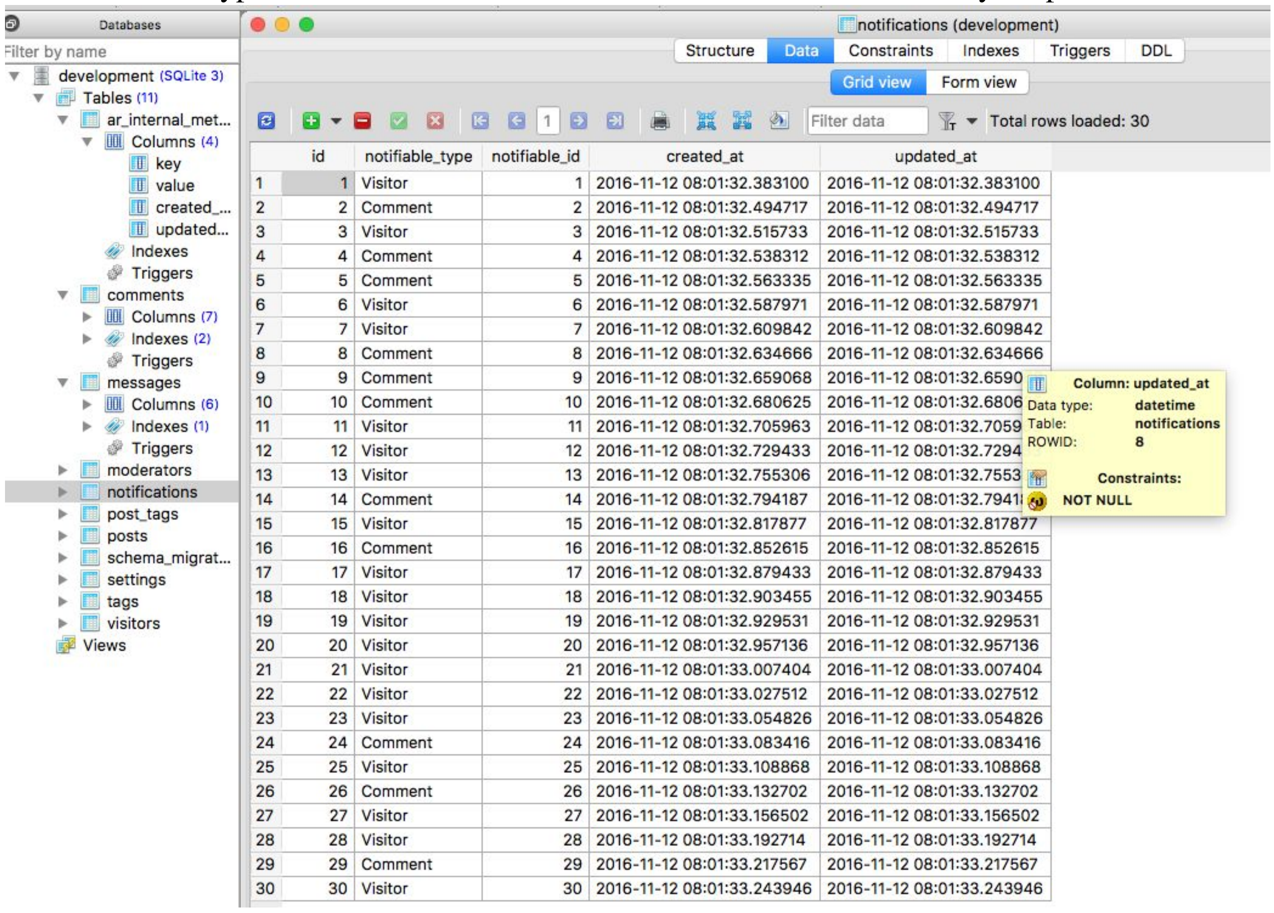
After the request is made by the user, a form will be displayed on the browser(View) and data will be inputted. The data will reach again the controller via the service and then the model will ensure that it's saved well within the database tables and the cycle will start all over again.

Below is an html form embedded in ruby.

```
<h1>Messages#new</h1>

<% form_for @visitor_message, url: messages_url do |∑ß_rebase_form| %>

  <% if @visitor_message.errors.any? %>
    <div id="error_explanation">
      <h2>
        <% pluralize(@visitor_message.errors.count, "error") %> prohibited this comment from sending:
      </h2>
      <ul>
        <% @visitor_message.errors.full_messages.each do |message| %>
          <li><% message %></li>
          <% end %>
      </ul>
    </div>
  <% end %>

  <p>
    <% ∑ß_rebase_form.label :fullname %>
    <% ∑ß_rebase_form.text_field :fullname %>
  </p>
  <p>
    <% ∑ß_rebase_form.label :email %>
    <% ∑ß_rebase_form.text_field :email %>
  </p>
  <% ∑ß_rebase_form.fields_for :messages do |∑ß∂_recursive| %>
    <p>
      <% ∑ß∂_recursive.label :content %>
      <% ∑ß∂_recursive.text_area :content %>
    </p>
  <% end %>
  <p><% ∑ß_rebase_form.submit 'Send Message' %></p>

<% end %>
```

Notifications

Has a notifiable type of either Visitor or Comment, a clear illustration of Polymorphism.



```
Databases
Filter by name
▼ development (SQLite 3)
  ▼ Tables (11)
    ▼ ar_internal_met...
      ▼ Columns (4)
          key
          value
          created_...
          updated...
        Indexes
        Triggers
    ▼ comments
      ▶ Columns (7)
      ▶ Indexes (2)
        Triggers
    ▼ messages
      ▶ Columns (6)
      ▶ Indexes (1)
        Triggers
    ▶ moderators
    ▶ notifications
    ▶ post_tags
    ▶ posts
    ▶ schema_migrat...
    ▶ settings
    ▶ tags
    ▶ visitors
  Views
```

notifications (development)

Structure | Data | Constraints | Indexes | Triggers | DDL

Grid view | Form view

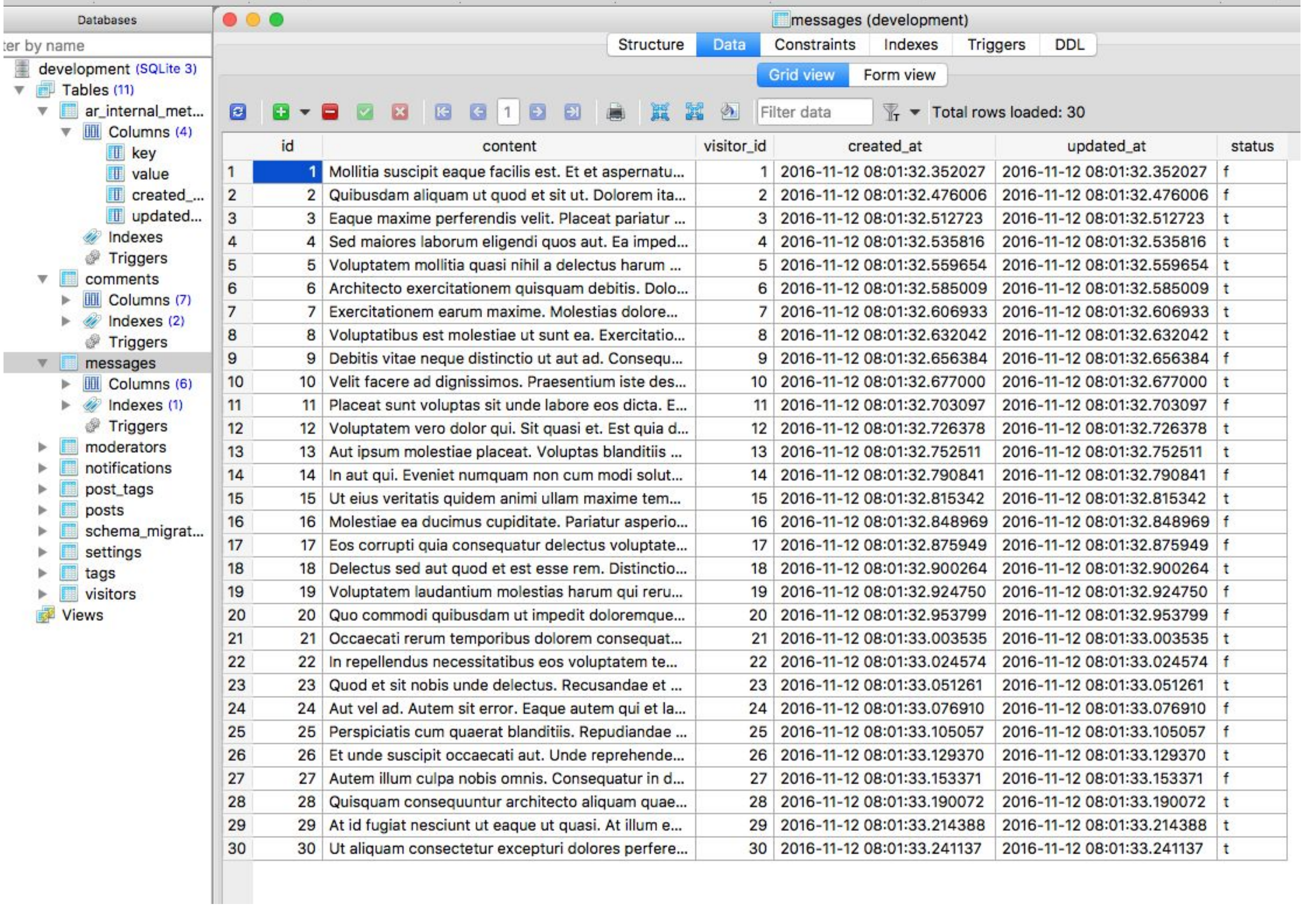
Filter data — Total rows loaded: 30

	id	notifiable_type	notifiable_id	created_at	updated_at
1	1	Visitor	1	2016-11-12 08:01:32.383100	2016-11-12 08:01:32.383100
2	2	Comment	2	2016-11-12 08:01:32.494717	2016-11-12 08:01:32.494717
3	3	Visitor	3	2016-11-12 08:01:32.515733	2016-11-12 08:01:32.515733
4	4	Comment	4	2016-11-12 08:01:32.538312	2016-11-12 08:01:32.538312
5	5	Comment	5	2016-11-12 08:01:32.563335	2016-11-12 08:01:32.563335
6	6	Visitor	6	2016-11-12 08:01:32.587971	2016-11-12 08:01:32.587971
7	7	Visitor	7	2016-11-12 08:01:32.609842	2016-11-12 08:01:32.609842
8	8	Comment	8	2016-11-12 08:01:32.634666	2016-11-12 08:01:32.634666
9	9	Comment	9	2016-11-12 08:01:32.659068	2016-11-12 08:01:32.6590
10	10	Comment	10	2016-11-12 08:01:32.680625	2016-11-12 08:01:32.6806
11	11	Visitor	11	2016-11-12 08:01:32.705963	2016-11-12 08:01:32.7059
12	12	Visitor	12	2016-11-12 08:01:32.729433	2016-11-12 08:01:32.7294
13	13	Visitor	13	2016-11-12 08:01:32.755306	2016-11-12 08:01:32.7553
14	14	Comment	14	2016-11-12 08:01:32.794187	2016-11-12 08:01:32.7941
15	15	Visitor	15	2016-11-12 08:01:32.817877	2016-11-12 08:01:32.817877
16	16	Comment	16	2016-11-12 08:01:32.852615	2016-11-12 08:01:32.852615
17	17	Visitor	17	2016-11-12 08:01:32.879433	2016-11-12 08:01:32.879433
18	18	Visitor	18	2016-11-12 08:01:32.903455	2016-11-12 08:01:32.903455
19	19	Visitor	19	2016-11-12 08:01:32.929531	2016-11-12 08:01:32.929531
20	20	Visitor	20	2016-11-12 08:01:32.957136	2016-11-12 08:01:32.957136
21	21	Visitor	21	2016-11-12 08:01:33.007404	2016-11-12 08:01:33.007404
22	22	Visitor	22	2016-11-12 08:01:33.027512	2016-11-12 08:01:33.027512
23	23	Visitor	23	2016-11-12 08:01:33.054826	2016-11-12 08:01:33.054826
24	24	Comment	24	2016-11-12 08:01:33.083416	2016-11-12 08:01:33.083416
25	25	Visitor	25	2016-11-12 08:01:33.108868	2016-11-12 08:01:33.108868
26	26	Comment	26	2016-11-12 08:01:33.132702	2016-11-12 08:01:33.132702
27	27	Visitor	27	2016-11-12 08:01:33.156502	2016-11-12 08:01:33.156502
28	28	Visitor	28	2016-11-12 08:01:33.192714	2016-11-12 08:01:33.192714
29	29	Comment	29	2016-11-12 08:01:33.217567	2016-11-12 08:01:33.217567
30	30	Visitor	30	2016-11-12 08:01:33.243946	2016-11-12 08:01:33.243946

Column: updated_at
Data type: datetime
Table: notifications
ROWID: 8
Constraints:
NOT NULL

Messages

Message content is shown in the image and it's relative validations and associations in the code.



Total rows loaded: 30

	id	content	visitor_id	created_at	updated_at	status
1	1	Mollitia suscipit eaque facilis est. Et et aspernatu...	1	2016-11-12 08:01:32.352027	2016-11-12 08:01:32.352027	f
2	2	Quibusdam aliquam ut quod et sit ut. Dolorem ita...	2	2016-11-12 08:01:32.476006	2016-11-12 08:01:32.476006	f
3	3	Eaque maxime perferendis velit. Placeat pariatur ...	3	2016-11-12 08:01:32.512723	2016-11-12 08:01:32.512723	t
4	4	Sed maiores laborum eligendi quos aut. Ea imped...	4	2016-11-12 08:01:32.535816	2016-11-12 08:01:32.535816	t
5	5	Voluptatem mollitia quasi nihil a delectus harum ...	5	2016-11-12 08:01:32.559654	2016-11-12 08:01:32.559654	t
6	6	Architecto exercitationem quisquam debitis. Dolo...	6	2016-11-12 08:01:32.585009	2016-11-12 08:01:32.585009	t
7	7	Exercitationem earum maxime. Molestias dolore...	7	2016-11-12 08:01:32.606933	2016-11-12 08:01:32.606933	t
8	8	Voluptatibus est molestiae ut sunt ea. Exercitatio...	8	2016-11-12 08:01:32.632042	2016-11-12 08:01:32.632042	t
9	9	Debitis vitae neque distinctio ut aut ad. Consequ...	9	2016-11-12 08:01:32.656384	2016-11-12 08:01:32.656384	f
10	10	Velit facere ad dignissimos. Praesentium iste des...	10	2016-11-12 08:01:32.677000	2016-11-12 08:01:32.677000	t
11	11	Placeat sunt voluptas sit unde labore eos dicta. E...	11	2016-11-12 08:01:32.703097	2016-11-12 08:01:32.703097	f
12	12	Voluptatem vero dolor qui. Sit quasi et. Est quia d...	12	2016-11-12 08:01:32.726378	2016-11-12 08:01:32.726378	t
13	13	Aut ipsum molestiae placeat. Voluptas blanditiis ...	13	2016-11-12 08:01:32.752511	2016-11-12 08:01:32.752511	t
14	14	In aut qui. Eveniet numquam non cum modi solut...	14	2016-11-12 08:01:32.790841	2016-11-12 08:01:32.790841	f
15	15	Ut eius veritatis quidem animi ullam maxime tem...	15	2016-11-12 08:01:32.815342	2016-11-12 08:01:32.815342	t
16	16	Molestiae ea ducimus cupiditate. Pariatur asperio...	16	2016-11-12 08:01:32.848969	2016-11-12 08:01:32.848969	f
17	17	Eos corrupti quia consequatur delectus voluptate...	17	2016-11-12 08:01:32.875949	2016-11-12 08:01:32.875949	f
18	18	Delectus sed aut quod et est esse rem. Distinctio...	18	2016-11-12 08:01:32.900264	2016-11-12 08:01:32.900264	t
19	19	Voluptatem laudantium molestias harum qui reru...	19	2016-11-12 08:01:32.924750	2016-11-12 08:01:32.924750	f
20	20	Quo commodi quibusdam ut impedit doloremque...	20	2016-11-12 08:01:32.953799	2016-11-12 08:01:32.953799	f
21	21	Occaecati rerum temporibus dolorem consequat...	21	2016-11-12 08:01:33.003535	2016-11-12 08:01:33.003535	t
22	22	In repellendus necessitatibus eos voluptatem te...	22	2016-11-12 08:01:33.024574	2016-11-12 08:01:33.024574	f
23	23	Quod et sit nobis unde delectus. Recusandae et ...	23	2016-11-12 08:01:33.051261	2016-11-12 08:01:33.051261	t
24	24	Aut vel ad. Autem sit error. Eaque autem qui et la...	24	2016-11-12 08:01:33.076910	2016-11-12 08:01:33.076910	f
25	25	Perspiciatis cum quaerat blanditiis. Repudiandae ...	25	2016-11-12 08:01:33.105057	2016-11-12 08:01:33.105057	f
26	26	Et unde suscipit occaecati aut. Unde reprehende...	26	2016-11-12 08:01:33.129370	2016-11-12 08:01:33.129370	t
27	27	Autem illum culpa nobis omnis. Consequatur in d...	27	2016-11-12 08:01:33.153371	2016-11-12 08:01:33.153371	f
28	28	Quisquam consequuntur architecto aliquam quae...	28	2016-11-12 08:01:33.190072	2016-11-12 08:01:33.190072	t
29	29	At id fugiat nesciunt ut eaque ut quasi. At illum e...	29	2016-11-12 08:01:33.214388	2016-11-12 08:01:33.214388	t
30	30	Ut aliquam consectetur excepturi dolores perfere...	30	2016-11-12 08:01:33.241137	2016-11-12 08:01:33.241137	t

```ruby
class Message < ApplicationRecord
  belongs_to :visitor

  validates :content, presence: true

  def self.josembi_the_rubist_find_content params
    joins(:visitor).where("fullname LIKE ? OR content LIKE ?", "%#{params}%", "%#{params}%" )
  end

  def kasyula_msg_read
    update(status: :true) if status == false
  end
end
```

Deployment instructions

I usually deploy to Heroku by doing the following.

- 1. Version Control (Git).
- 2. Bundle, Commit and Push to the Server.
- 3. Access using URL to view App.
- 4. Access logs by inputting the heroku logs.

Developer

Joseph M Mwania

Contacts

http://www.theappwebtech.com/

https://github.com/appwebtech

https://twitter.com/appwebtech

https://www.facebook.com/theappwebtech/

https://it.pinterest.com/appwebtech/

License

Disclaimer: I apologize for some of my commit messages. "They are raw and un-edited" with some un-professional lingo. When working on Mock-ups/Prototypes, I usually push after switching branches and editing commit messages on github is effective only with "reset --hard" or "--force-with-lease" which can really mess with project history. Once again, sorry for that, It does happen when coding late at night if the sole intention of making the project is to be viewed without the source code or when running a hackathon with fellow developers.