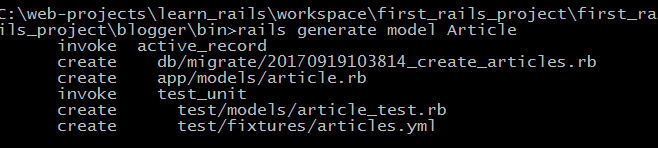
Making a Blog wit rails

Saved in C:\web-projects\learn\_rails\workspace\first\_rails\_project\first\_rails\_project\blogger

1. I initiated a repo on github and wrote an initial read.me file
2. Ran rails new blogger- this command create a nasic rails app which I can personalise
3. All commands are now run from the bin directory
4. Rails server- this command (in a new terminal) will run a server from my machine allowing me to open my rails app with my browser to see the welcome to rails page at local host 3000.

Setting up the Article Model

1. The blog will contain articles so I need to create a model which will be able to interact with the articles in the database. To do this I use rails generate model Article. This command will create all the files I need to interact with the articles in the database.

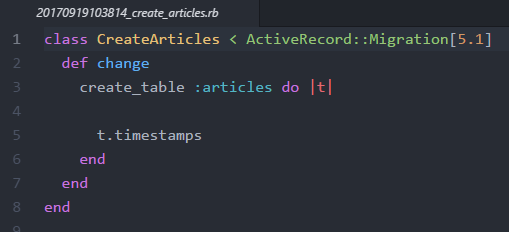


The db migrate (first line) is a database migration. It has a time stamp. A migration is able to alter our database for us. Each migration creates a new version of the database, altering tables, columns or entries This migration will be able to create the articles table in the database.

app/models/article.rb- this ruby file will contain the code for the model

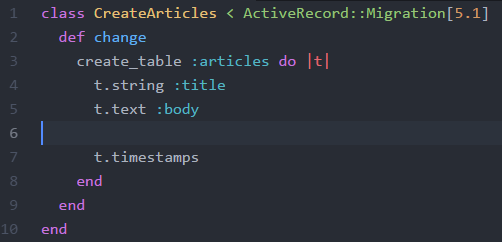
The last two files are associated with testing

**The Article Migration**



This is the current file for the migration. It is named with a timestamp which allows the migrations to be kept in chronological order. It has only one method-change. Within the change method creat\_table has been called automatically. This is a method which will create a table! The generator has passes the create\_table method the symbol :article as a parameter. This will create a table called Articles. There is then a block which assigns the variable t to the table which is created.

,t is important because we call methods on t to create the columns in the articles table. We will need a column called title and a column named body to hold the data from our articles. The title will be defined as a string and the body is defined as text. Text is data type which the rails adapters can change to what ever type our database will understand without us having to worry about it. This could be varchar or text depending on the database used.



We edit the change method to include t.string :title and t.text :body. This will create our columns with the correct titles and data types. Rails also automatically makes two columns using t.timestamps. These columns are titled created and and updated\_at. These record the times that articles are uploaded and when they are subsequently updated. Rails will do the recording for us.

We are now ready to run the migration (to create our table in the database). We do this with the command, rake rb:migrate. Rake is used to carry out maintainance functions and testing. The command says, “look in your functions for databases and run the migrate function”. The migrate action will then find all the migrations in db/migrations folder. It then looks at a table in the databses which says which migrations haven’t been run. It will then run the migrations which haven’t bee run yet.

In ourcase CreateArticles was run and made a table named articles.

WORKING WITH A MODEL IN THE CONSOLE-Side step 😉

If we want to we can access all parts of our app with the command line. If we run rails console it initiates an IRB. This means we don’t need to go through a web interface to update our app eg create articles. We can then type Article.new to create an empty article. It shows that we have the headings title, body and two timestamp headings

LOOKING AT THE MODEL

The code for the model is kept in app/models/article.rb. Currently this file is pretty empty with no attributes defined. Rails is really clever in that it already knows that an article should have a title and a body. It does this using a technique called reflection. Rails looks at the database, sees the column headings and assumes that these headings should be the attributes in the model. There will also be an attributed called id. Rails gives each entry a unique id so that we can refer to it if needed.

To make an new article in the console we type the below line by line

a = Article.new

a.title =”sample title”

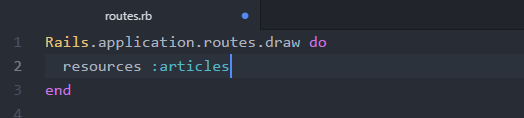
a.body = “sample text in the body”

a.save

This input will create our first article and save it! As well as filling in the title and body, rails will fill in the id and timestamps for me. The id was 1. I can view the article using Article.all

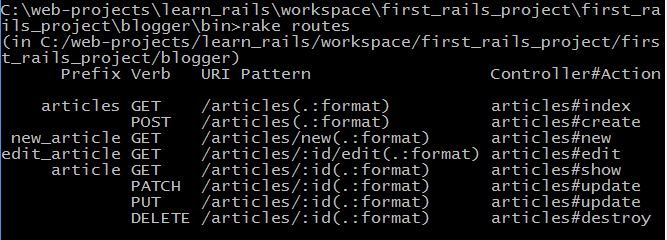
SETTING UP A ROUTER

Out of the MVC architecture we have made the model. Now we need to create the view and the controller. When a browser speaks to a rails app the first thing rails does is go to the router. The router is able to decided what the request is trying to do and what it needs. It figures it out using the address it is requesting and other http parameter such as Get or Put. We find the router file in confige/routes.rb. This file is pretty empty until I start building routes.



We edit the file to something like the above. I added resources :articles. What this block is doing is telling the router to expect requests which follow RESTful principles What this means is that when I get a request like “localhost3000/articles/” the router will understand that I want a list of articles. I don’t have to specifically tell it(?) Also it would know that “localhost3000/articles/new/” means I want to write a new article.

Line two, although simple will create all the typical routes I will need(in accordance with rails’ restfull principles). I can view them by doing Rake routes and I get the below.If I delete line two my routes disappear!



Lets look at the first row. Articles is the first prefix in the list. The router will provide two methods using “articles”, the articles\_path and article\_url. The \_path version is relative while the \_url version uses a full url. The path version is always preferred.

The second column says “get”, this is the verb for the HTTP route. Web borwsers typically send requests with the verbs get or post.

The third column- *I don’t understand it* but the part in brackets is optional. The uri pattern in the example will match articles/, articles, and articles.json.

The fourth column shos where the route will map to in the application. In our example the route will map to the index method of the Articlescontroller class.

Now we have our router knows how to handle requests about articles we need a place to send the requests-the controller