

MoneyDashboard App

Brief: to create a web based app using Ruby to control a SQL database in order to allow for tracking of expenditure by different Merchant and purchase type.

UX Preparation

User Persona:

Name: Susan Johnston

Demographics:

Age: 18

Occupation: Student

Resides in: Edinburgh

Income bracket: Low

Nationality: British

Behaviours:

- **Not 'tight' with money (sometimes spends recklessly)**
- **Works part time to fund studies**
- **Living on a student grant**
- **Expensive hobbies (so sometimes goes over budget)**

Needs and goals:

- **Needs to be able to create, read, update and delete the entries on her app**
- **Needs to be alerted when she goes over budget**

- **Would like to see breakdown of her expenditures by purchase type (called 'Tag Type' in the app).**
- **Would like to see the history of her expenditures broken down by month**

User Needs: similar to the needs and goals above.

- As a general user, she wants to be able to create, read, update and delete the transaction records on her app, so that she can store details and view them later, or edit/update them as required.
- She would like to be able to view lists of previous transactions, so that she can see where her money is going and if she has any areas where she could trim her spending.
- As a user on a limited budget, she would like to be given alerts if she runs over her budget limits, so she can improve her financial behaviours and plan for the future.
- She would like to be able to see a breakdown of transactions by Tag, so that she can analyse the areas where she may be overspending.
- She would like to see her total expenditure broken down by Tag transaction date, in order to view the months when her expenditure is greatest.

User Journey

- User action: runs web app file
- System response: opens home page

- User action: selects Transaction, Merchant or TagType from home page menu.
- System response: opens list of all current Transaction, Merchant or TagType entries.

- User action: clicks on button to create a new Transaction, Merchant or TagType from links on the respective pages
- System response: adds a new entry to the database, unless the Merchant or TagType entry is already present (in which case don't add to the database and inform the user). Also increases the total expenditure amount and checks that the user's spending limit has not been breached. If it has been, display a screen informing the user of this (but still update the database).

- User action: clicks on button to show an existing Transaction, Merchant or TagType from links on the respective pages
- System response: displays a screen with the selected item only displayed, rather than the whole list.

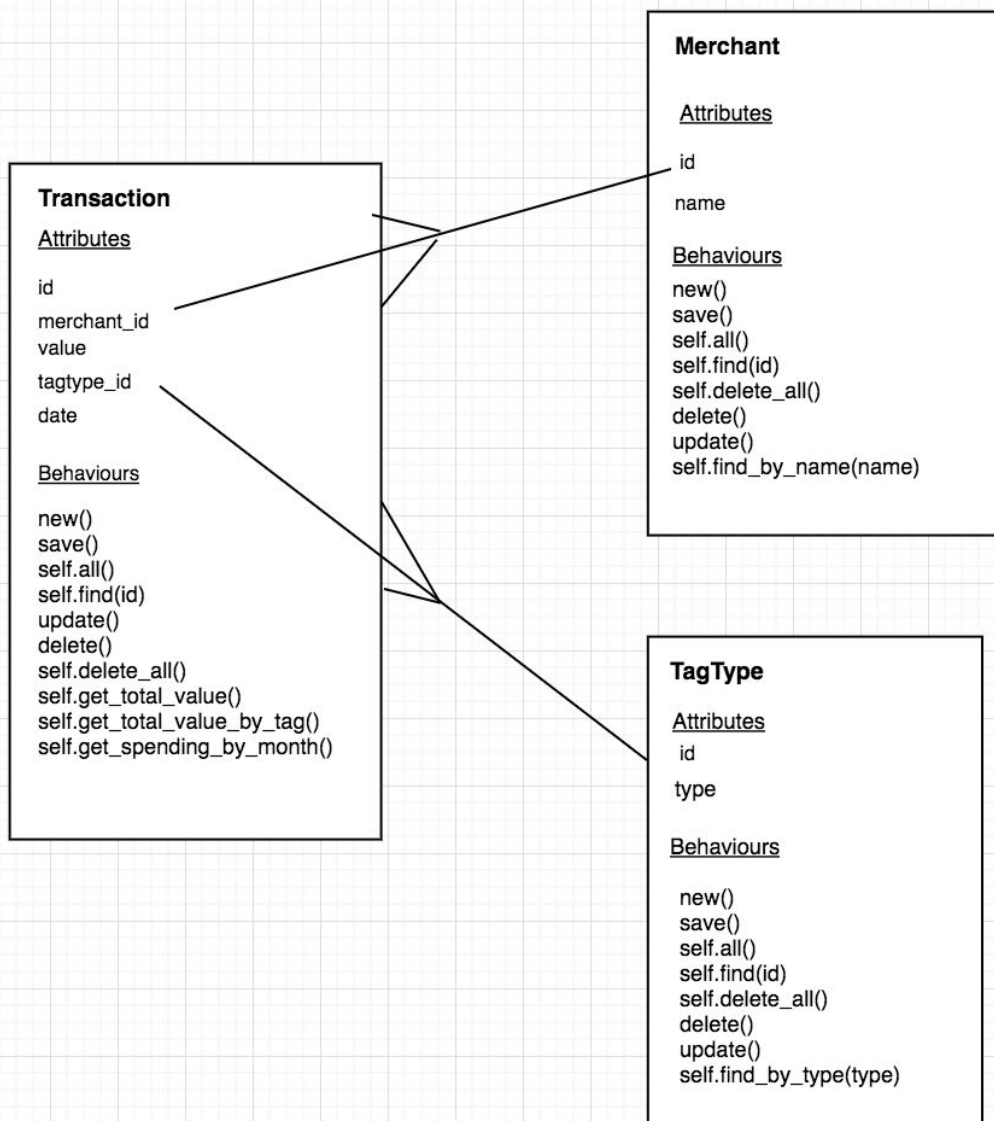
- User action: clicks on button to delete an existing Transaction, Merchant or TagType from links on the respective pages.
- System response: updates the required database, updates the total expenditure, and displays the updated list to the user.

- User action: clicks on button from Transaction screen, to display total expenditure broken down by Tag.
- System response: displays a table containing Tag Type as one column, and total expenditure for the Tag Type as the other column.

- User action: clicks on button from Transaction screen, to display total expenditure broken down by month.
- System response: displays a table containing month of transaction as one column, and total expenditure for the month as the other column.

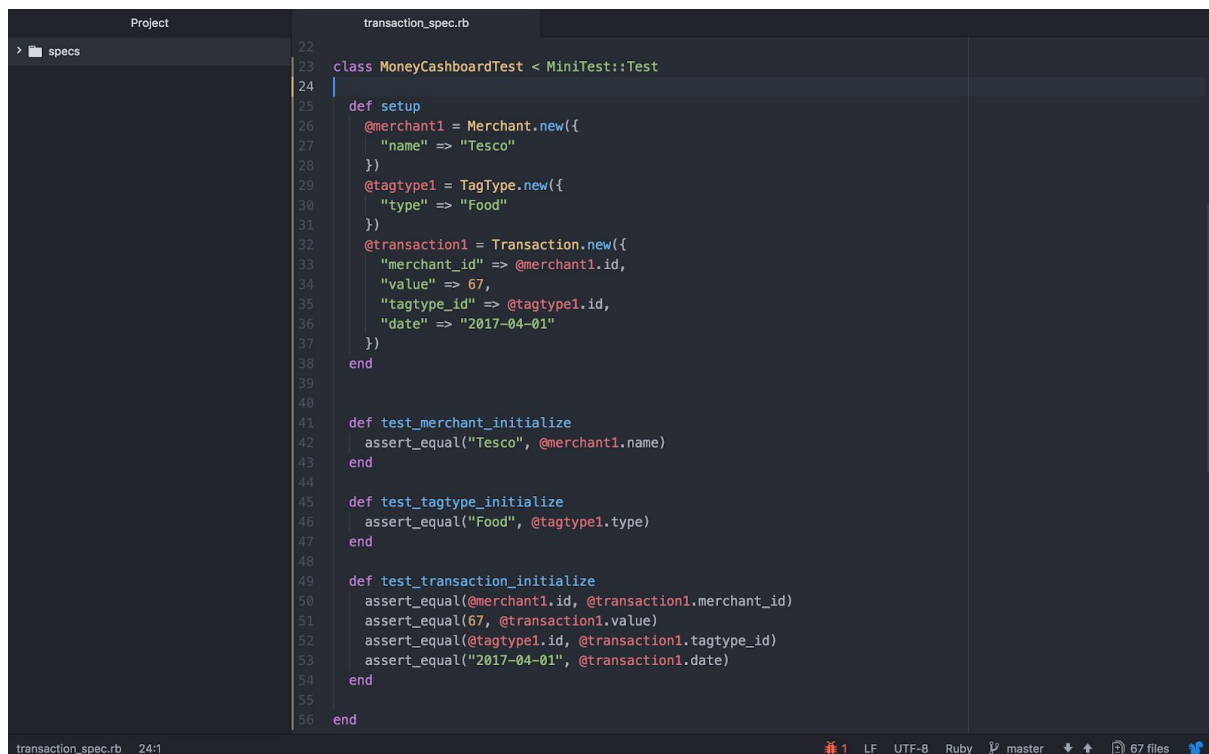
UML Diagram

MoneyCashboard UML



Test Driven Development

- All of the functions for each class used SQL database queries, apart from the 'new' functions.
- Tested that the functions created Merchant, TagType and Transaction records, and that the fields were correctly populated for one example of each class.
- Tested that the overall total transaction value matched the totals from the breakdowns by Tag Type and by month.



```
22
23 class MoneyCashboardTest < MiniTest::Test
24
25   def setup
26     @merchant1 = Merchant.new({
27       "name" => "Tesco"
28     })
29     @tagtype1 = TagType.new({
30       "type" => "Food"
31     })
32     @transaction1 = Transaction.new({
33       "merchant_id" => @merchant1.id,
34       "value" => 67,
35       "tagtype_id" => @tagtype1.id,
36       "date" => "2017-04-01"
37     })
38   end
39
40
41   def test_merchant_initialize
42     assert_equal("Tesco", @merchant1.name)
43   end
44
45   def test_tagtype_initialize
46     assert_equal("Food", @tagtype1.type)
47   end
48
49   def test_transaction_initialize
50     assert_equal(@merchant1.id, @transaction1.merchant_id)
51     assert_equal(67, @transaction1.value)
52     assert_equal(@tagtype1.id, @transaction1.tagtype_id)
53     assert_equal("2017-04-01", @transaction1.date)
54   end
55
56 end
```

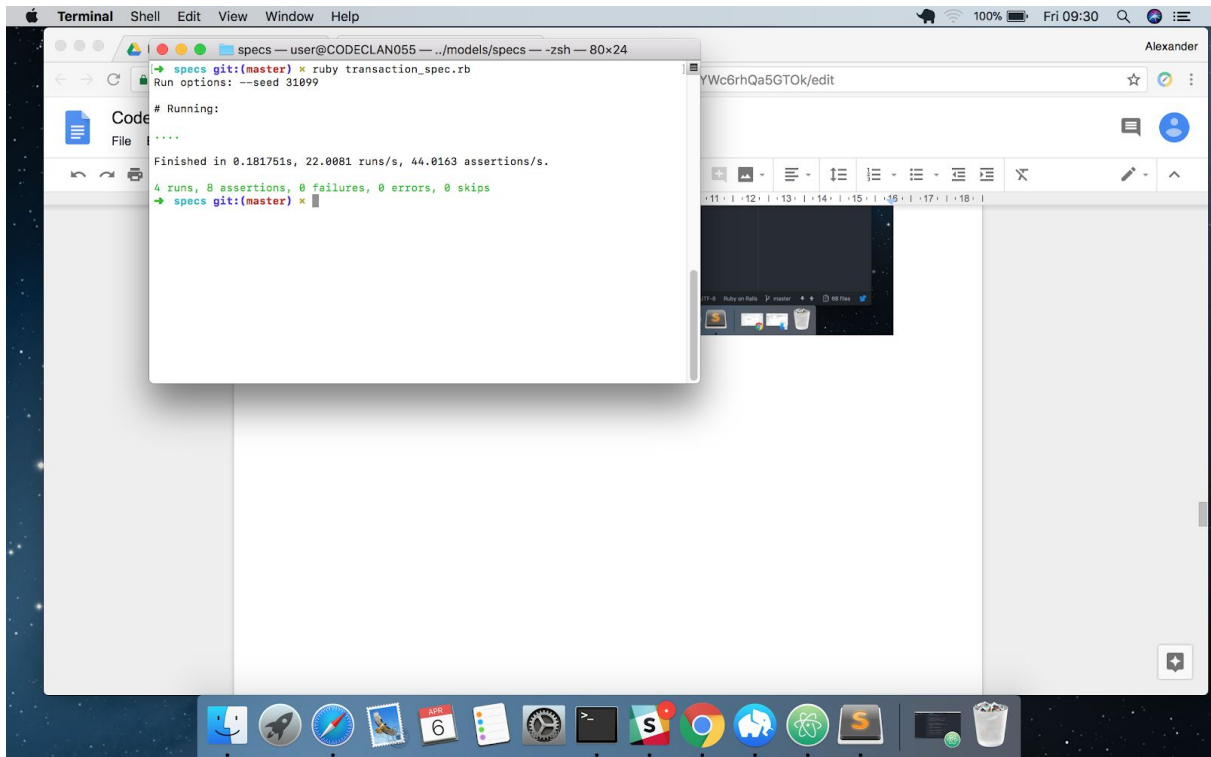
Project

specs

transaction_spec.rbtransaction.rb

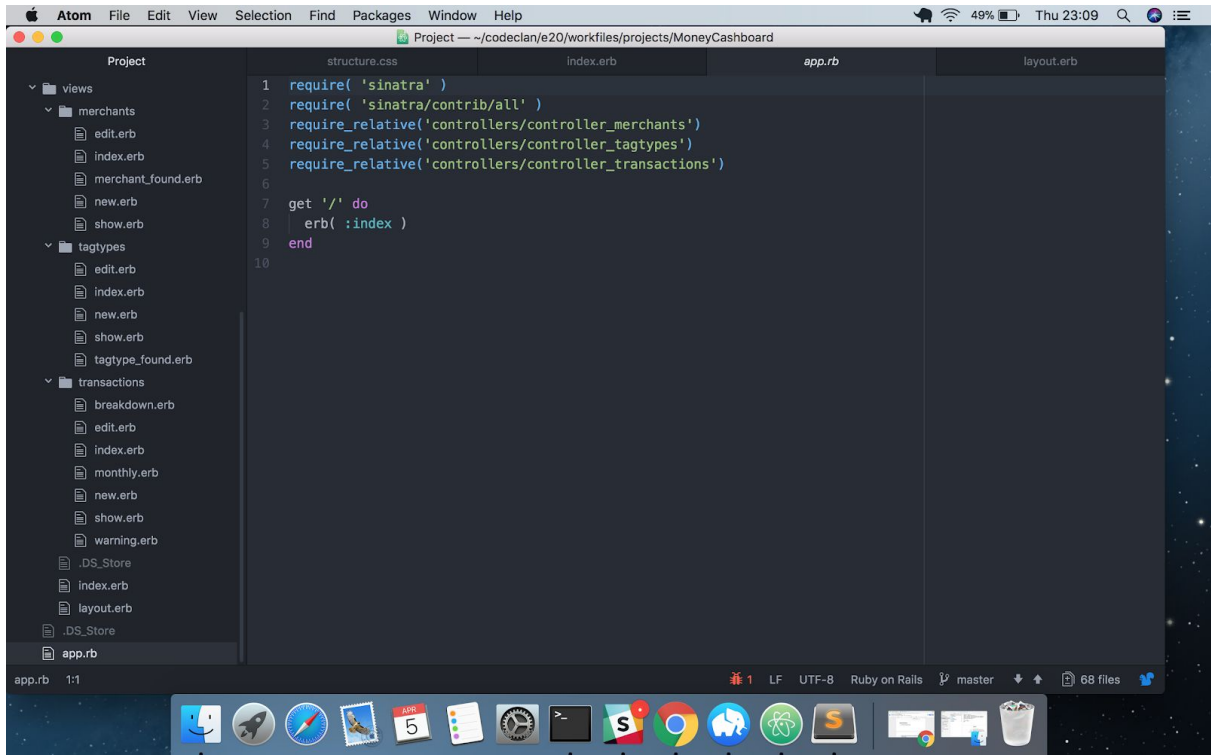
```
45 def test_tagtype_initialize
46   assert_equal("Food", @tagtype1.type)
47 end
48
49 def test_transaction_initialize
50   assert_equal(@merchant1.id, @transaction1.merchant_id)
51   assert_equal(67, @transaction1.value)
52   assert_equal(@tagtype1.id, @transaction1.tagtype_id)
53   assert_equal("2017-04-01", @transaction1.date)
54 end
55
56 def test_totals_match_breakdown_totals
57   overall_total = Transaction.get_total_value()
58   tagtype_total_array = Transaction.get_total_values_by_tag()
59   tagtype_total = 0
60
61   for total_hash in tagtype_total_array
62     tagtype_total += total_hash["total"]
63   end
64
65   month_totals_array = Transaction.get_spending_by_month()
66   months_total = month_totals_array.sum
67
68   assert_equal(overall_total, tagtype_total)
69
70   assert_equal(overall_total, months_total)
71
72 end
73
74 end
75
```

transaction_spec.rb 62:43 1 LF UTF-8 Ruby master 1 file

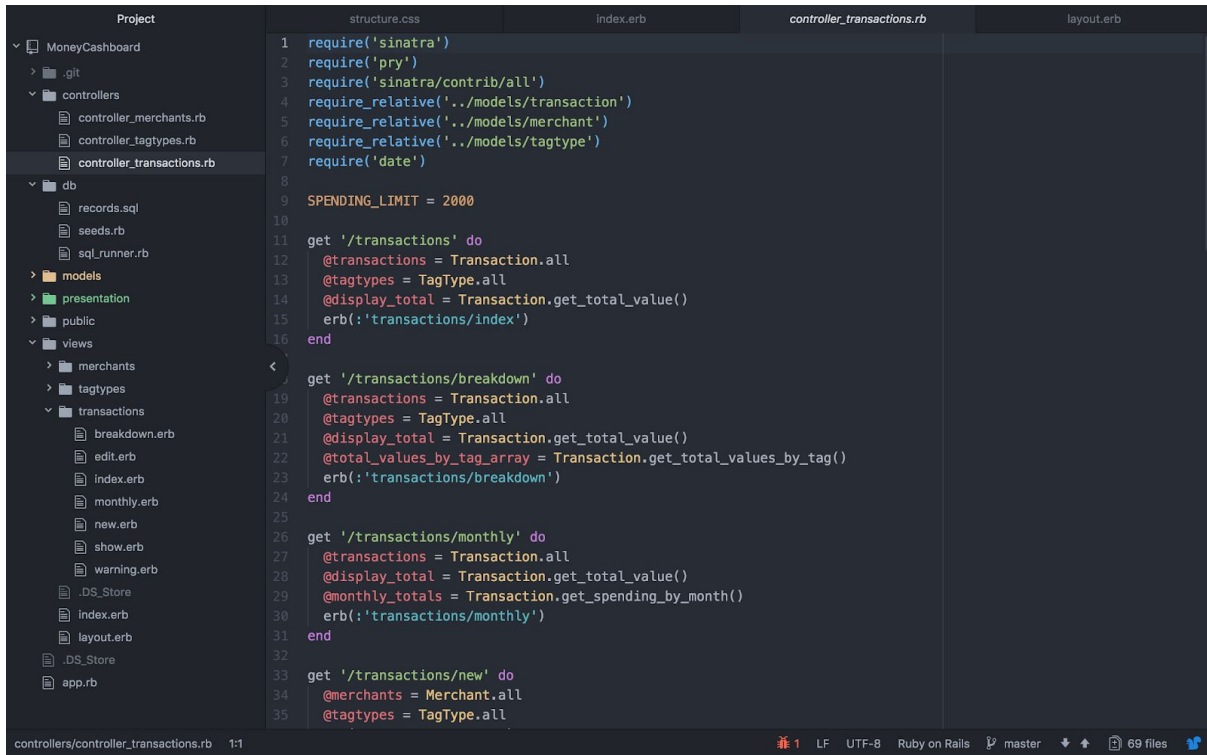


Outline Code Structure:

- All runs from single app.rb file



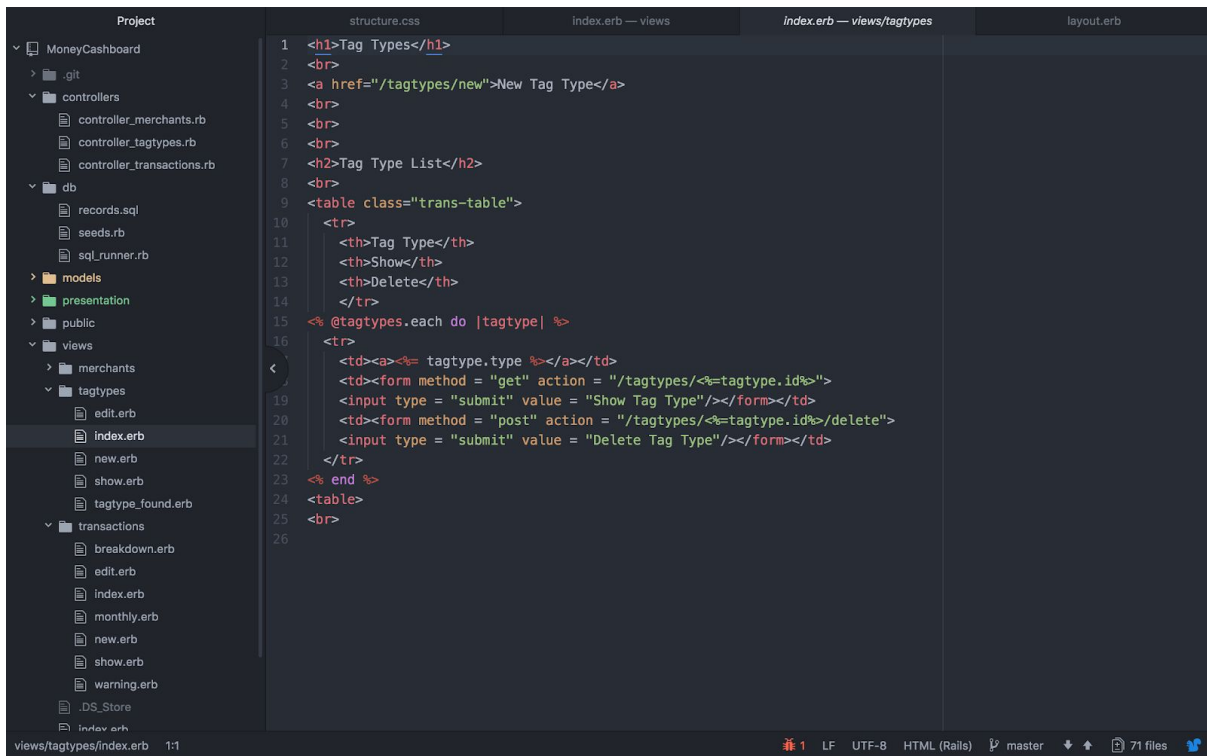
- Three controllers, one for each of the Transaction, Merchant and TagType classes:



The screenshot shows a code editor with a sidebar on the left displaying a project structure for 'MoneyCashboard'. The sidebar includes folders like .git, controllers, db, models, presentation, public, and views, along with various files like records.sql, seeds.rb, sql_runner.rb, and transaction-related files. The main editor area shows the content of 'controller_transactions.rb', which includes require statements for Sinatra, Pry, and various models, a SPENDING_LIMIT constant, and three GET routes for transactions: index, breakdown, and monthly. The status bar at the bottom indicates the file path 'controllers/controller_transactions.rb', line 1:1, and other details like LF, UTF-8, Ruby on Rails, master branch, and 69 files.

```
1 require('sinatra')
2 require('pry')
3 require('sinatra/contrib/all')
4 require_relative('../models/transaction')
5 require_relative('../models/merchant')
6 require_relative('../models/tagtype')
7 require('date')
8
9 SPENDING_LIMIT = 2000
10
11 get '/transactions' do
12   @transactions = Transaction.all
13   @tagtypes = TagType.all
14   @display_total = Transaction.get_total_value()
15   erb(:'transactions/index')
16 end
17
18 <
19 get '/transactions/breakdown' do
20   @transactions = Transaction.all
21   @tagtypes = TagType.all
22   @display_total = Transaction.get_total_value()
23   @total_values_by_tag_array = Transaction.get_total_values_by_tag()
24   erb(:'transactions/breakdown')
25 end
26
27 get '/transactions/monthly' do
28   @transactions = Transaction.all
29   @display_total = Transaction.get_total_value()
30   @monthly_totals = Transaction.get_spending_by_month()
31   erb(:'transactions/monthly')
32 end
33
34 get '/transactions/new' do
35   @merchants = Merchant.all
36   @tagtypes = TagType.all
```

- Separate RESTful routes for each class. Views with HTML code styled using CSS for each of the RESTful routes, e.g.:

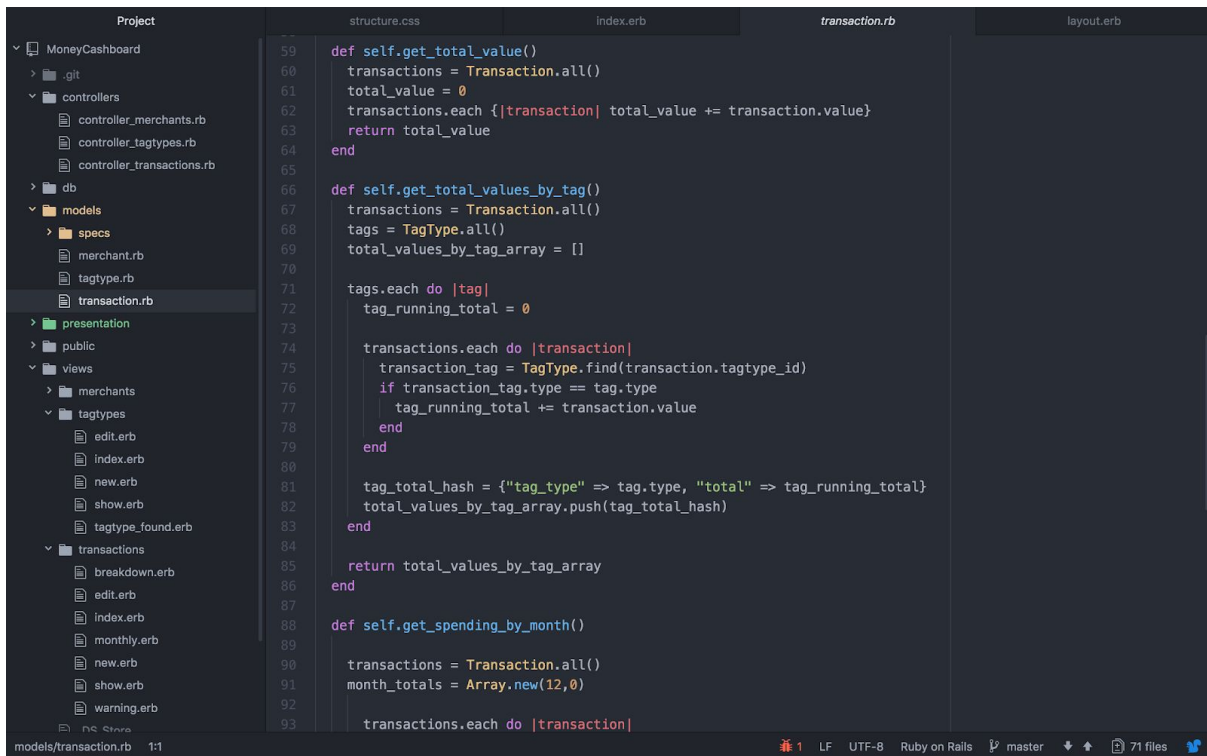


The screenshot shows a code editor with a project structure on the left and a view file on the right. The project structure is for a Rails application named 'MoneyDashboard'. The view file, 'index.erb', is located in the 'views/tagtypes' directory. The code in the view file is as follows:

```
1 <h1>Tag Types</h1>
2 <br>
3 <a href="/tagtypes/new">New Tag Type</a>
4 <br>
5 <br>
6 <br>
7 <h2>Tag Type List</h2>
8 <br>
9 <table class="trans-table">
10 <tr>
11 <th>Tag Type</th>
12 <th>Show</th>
13 <th>Delete</th>
14 </tr>
15 <%= @tagtypes.each do |tagtype| %>
16 <tr>
17 <td><a href="/tagtypes/<%=tagtype.id%>"><%= tagtype.type %></a></td>
18 <td><form method = "get" action = "/tagtypes/<%=tagtype.id%>">
19 <input type = "submit" value = "Show Tag Type"/></form></td>
20 <td><form method = "post" action = "/tagtypes/<%=tagtype.id%>/delete">
21 <input type = "submit" value = "Delete Tag Type"/></form></td>
22 </tr>
23 <%= end %>
24 </table>
25 <br>
26
```

The status bar at the bottom indicates the file is 'views/tagtypes/index.erb' at line 1, column 1. The editor is using the 'HTML (Rails)' syntax highlighter and is on the 'master' branch.

- All 3 classes have the usual functions for Create, Read, Update, Delete functionality. Transaction class has a few extra functions to allow it to find the total transaction value, total value by tag type and total value by month:



```
Project
MoneyCashboard
  .git
  controllers
    controller_merchants.rb
    controller_tagtypes.rb
    controller_transactions.rb
  db
  models
    specs
      merchant.rb
      tagtype.rb
      transaction.rb
  presentation
  public
  views
    merchants
    tagtypes
      edit.erb
      index.erb
      new.erb
      show.erb
      tagtype_found.erb
    transactions
      breakdown.erb
      edit.erb
      index.erb
      monthly.erb
      new.erb
      show.erb
      warning.erb
  DE Shiva

models/transaction.rb 1:1

structure.css
index.erb
transaction.rb
layout.erb

59 def self.get_total_value()
60   transactions = Transaction.all()
61   total_value = 0
62   transactions.each {|transaction| total_value += transaction.value}
63   return total_value
64 end
65
66 def self.get_total_values_by_tag()
67   transactions = Transaction.all()
68   tags = TagType.all()
69   total_values_by_tag_array = []
70
71   tags.each do |tag|
72     tag_running_total = 0
73
74     transactions.each do |transaction|
75       transaction_tag = TagType.find(transaction.tagtype_id)
76       if transaction_tag.type == tag.type
77         tag_running_total += transaction.value
78       end
79     end
80
81     tag_total_hash = {"tag_type" => tag.type, "total" => tag_running_total}
82     total_values_by_tag_array.push(tag_total_hash)
83   end
84
85   return total_values_by_tag_array
86 end
87
88 def self.get_spending_by_month()
89
90   transactions = Transaction.all()
91   month_totals = Array.new(12,0)
92
93   transactions.each do |transaction|
```

Do Differently/Issues:

- RESTful routes - better site planning would have helped with this.
- Draw site map earlier in planning.
- Just stuck to a simple website design to help with this.
- Styling with CSS - still got a lot to learn on Developer Tools!

Would like to have had time to implement:

- User input for spending limit. Tried this, but insufficient time.
- More CSS - add Flexbox, more background themes and motifs
- Graphical displays of results (bar chart, pie charts) which could have been done using the Ruby 'Gruff' gem.

Demo of App