# Experimenting with Ruby, Sinatra and PostgreSQL: a Message Board App

László Harri Németh

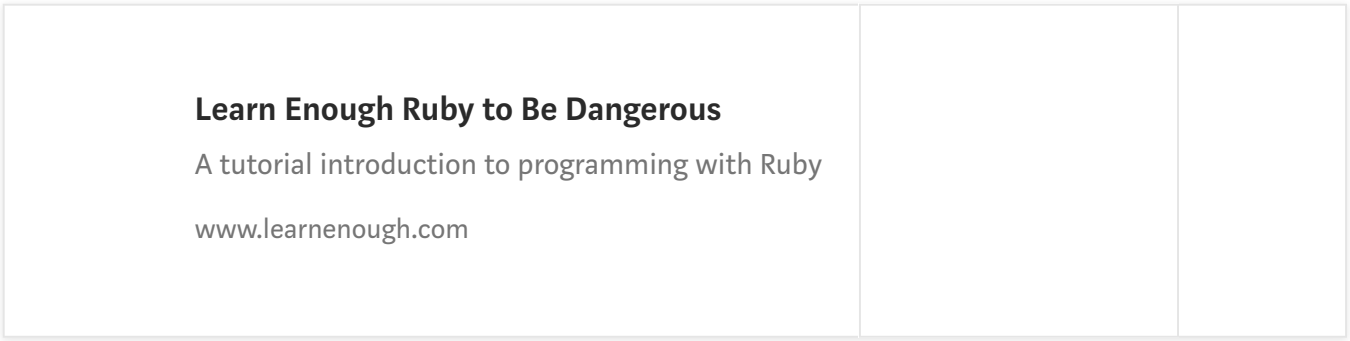
Dec 18, 2018 · 11 min read

Although it's 2018, and the first version of Ruby was released in 1995, 23 years ago, nowadays it seems Ruby has a reasonably large number of employment opportunities. The latest version of Ruby is 2.5.3 (it was released on October 18, 2018). In the present article I create a simple Message Board App using Ruby and PostgreSQL.

Photo by Jason Leung on Unsplash

I've found the following book very helpful for understanding the basics of the language.

I will use Sinatra framework in this article.




. . .

Keywords: Ruby, Embedded Ruby, HTML, CSS, SQL, erb, Sinatra, PostgreSQL, Linux, Ubuntu, rerun, Message Board, yield, HTTP, regular expressions, regex

. . .



. . .

## Table of Content

he app will be built step by step. The article is divided into the following sections:

## Prerequisites

For this article I will use the following programming languages and technology:

- HTML
- CSS
- Ruby
- SQL
- Sinatra (Ruby library)
- PostgreSQL
- Linux

## Preparation

This step is about:

- installing Sinatra for Ruby
- testing Ruby with a simple hello world app

- starting PostgreSQL server
- installing rerun for Ruby (rerun is a Ruby library)

## Installing Sinatra

I assume that your system already contains Ruby. I use `Ubuntu 14.04.5 LTS (trusty)` and my version of Ruby is

```
ruby 2.4.0p0 (2016-12-24 revision 57164) [x86_64-linux]
```

I use Cloud 9 environment.

For these examples we will also need Sinatra. Sinatra is a Domain Specific Language for quickly creating web applications in Ruby with minimal effort. To check if Sinatra is already installed, you can use

```
gem list sinatra
```

It should return something like this, containing the version of Sinatra installed in your system:

```
*** LOCAL GEMS ***
sinatra (2.0.4)
```

If Sinatra is not installed, you should install it:

```
gem install sinatra
```

To try if everything works, create the following Ruby file. Ruby source code has the extension `.rb`.

```
1  require 'sinatra'
2
3  # Listen on all interfaces in the development environment
```

```ruby
# This is needed when we run from Cloud 9 environment
# source: https://gist.github.com/jhabdas/5945768
set :bind, '0.0.0.0'
set :port, 8080

get '/' do
    'hello world!'
end
```

ruby-hello-world-sinatra.rb hosted with ❤ by GitHub view raw

Run this file with Ruby:

```
ruby ruby-hello-world-sinatra.rb
```

In my system the command returns the following. This will start a web server and run the Ruby script.

```
nlharri:~/workspace/ruby $ ruby ruby-hello-world-sinatra.rb
[2018-11-21 09:53:58] INFO  WEBrick 1.3.1
[2018-11-21 09:53:58] INFO  ruby 2.4.0 (2016-12-24) [x86_64-linux]
== Sinatra (v2.0.4) has taken the stage on 8080 for development with
backup from WEBrick
[2018-11-21 09:53:58] INFO  WEBrick::HTTPServer#start: pid=2433
port=8080
```

Open your web browser and go to http://localhost:8080. You need to see a hello world text in the browser.

To stop the server, press `CTRL+C` in the terminal. I will get back to the hello world example later.

## Start PostgreSQL server

I will use PostgreSQL version `9.3.18` . If PostgreSQL server is not starting automatically, and you would like to start it, you can use

```
sudo update-rc.d postgresql enable
```

If you would like to start the PostgreSQL server, you can use

```
sudo service postgresql start
```

By default, PostgreSQL sets up the user and database “postgres” upon a new installation. We interact with the postgres database software through an interface called “psql.”

You can login with user “postgres” to database “postgres” with the following command:

```
sudo –u postgres psql postgres
```

(You can exit with `CTRL+D` .)

## Install rerun

We will install `rerun` , which will make it possible to make changes to the code and see the results without restarting the server.

For this, we prepare a `Gemfile` in the project.

```
touch Gemfile
```

And the `Gemfile` should contain the following:

```
source 'https://rubygems.org'

gem 'sinatra', '2.0.4'
gem 'rerun',   '0.13.0'
```

We need to install these packages (we already installed `sinatra` earlier).

```
bundle install
```

(If the system does not contain `bundler`, you need to install it with `sudo gem install bundler`)

Now we need to start the server, but for this we use the following command:

```
bundle exec rerun ruby-hello-world-sinatra.rb
```

Earlier we needed to quit and restart the server in order to see changes to the app, but thanks to the `rerun` the app is updated automatically, after we refresh the browser by hand.

## Message Board Web App

What will the app know? The app will show the already posted messages, and the nickname of the person who posted the message. It will allow to post new messages. The data will be stored in a database, and the app will read the entries from the database, and save the new posts there.

The app is created step by step.

. . .

## 1st step: List hard-coded values

The first version of the app will show some hard-coded messages without using the database. (Later we will enhance the app to read up messages from the database.) In the first version we will also not able to post messages.

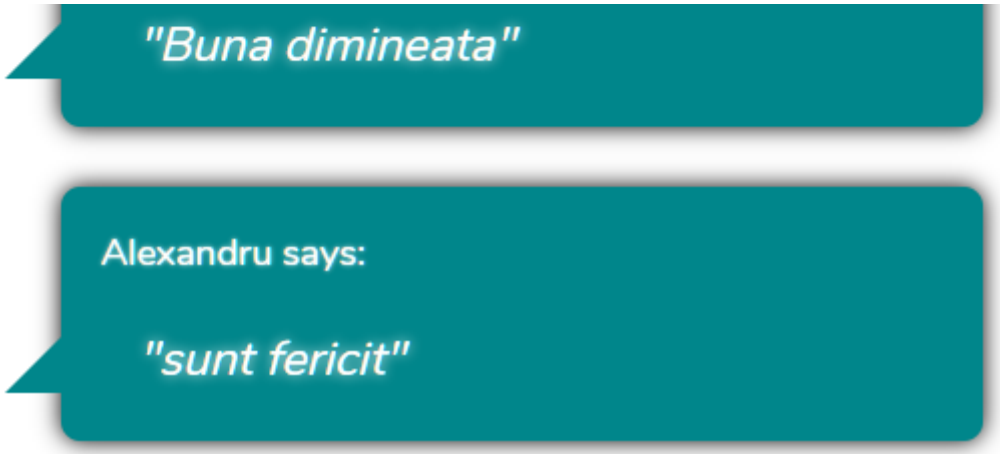

## Design

I tried to define a nice-looking but simple design with CSS. I used Bubbly example which was very helpful for me to define the design.

**Bubbly - CSS speech bubbles made easy**

leaverou.github.io

The basic layout and design looks like the following.




I also put this to codepen.io.

Message Board
A PEN BY nlharri

Run Pen

Basic Design of the Message Board App

## Implementation with Embedded Ruby

We will hard-code values of the messages data, and use templating of Sinatra to show the data. The data is passed to the erb, for which this stackoverflow post was very useful to understand.

Here you can check the coding. The `l_main.erb` and the `v_message.erb` should be in the `views` folder so that Sinatra will find them.

With `:locals` we can pass data to the `erb`.

```ruby
require 'sinatra'

# Listen on all interfaces in the development environment
# This is needed when we run from Cloud 9 environment
# source: https://gist.github.com/jhabdas/5945768
set :bind, '0.0.0.0'
set :port, 8080

get '/' do
  t_msg = [
    { nick: "Harri", msg: "Hello World!" },
    { nick: "Ioana", msg: "Buna dimineata" },
    { nick: "Alexandru", msg: "sunt fericit" }
  ]
  erb :v_message, :layout => :l_main, :locals => {:t_msg => t_msg}
end
```

app.rb hosted with ❤ by GitHub view raw

```erb
<!doctype html>
<html lang=en>
  <head>
    <link rel="stylesheet" type="text/css" href="/stylesheets/main.css">
    <link href="https://fonts.googleapis.com/css?family=Nunito" rel="stylesheet">
    <meta charset=utf-8>
    <title>Message Board</title>
  </head>
  <body>
    <div class="pageheader">
      <h1>Message Board</h1>
    </div>

    <div class="speech-bubble">
      <form action="/new" id="newmessage">
        <p class="nick"><input type="text" placeholder="Nickname" name="nickname"></p>
        <p class="message"><textarea placeholder="Message" name="message" form="newmess
        <p class="submit"><button class="submit" type="submit" form="newmessage" value=
      </form>
    </div>

    <%= yield %>

  </body>
```

```
24    </body>
25  </html>
```

l_main.erb hosted with ❤ by GitHub view raw

```
1  <% t_msg.each do |msg| %>
2    <div class="speech-bubble">
3      <p class="nick"><%= msg[:nick] %> says:</p>
4      <p class="message">   "<%= msg[:msg] %>"</p>
5    </div>
6  <% end %>
```

v_message.erb hosted with ❤ by GitHub view raw

. . .

## 2nd step: Read messages from the db

In this step we enhance the app to read the messages from the db. For this we need to define a database and a table, and put some entries to it.

Photo by Nikolay Tarashchenko on Unsplash

## Define database and table

For the next version of the app we need a database to store the data.

For this step the following tutorials were very useful.

**How To Create, Remove & Manage Tables in PostgreSQL on a Cloud Server |...**

PostgreSQL is a database management system that uses the SQL querying language. It is a very...

www.digitalocean.com

**PostgreSQL Ruby tutorial**

This is PostgreSQL tutorial. This tutorial covers the basics of PostgreSQL programming in Ruby language.

zetcode.com

In this step we will:

- create a new user `messageboarduser` in the system for accessing the database
- create a new user `messageboarduser` in PostgreSQL
- define database `messageboard`
- define database table `messageboardmessages`

## Create a new user in the system

We create a new user with the following command

```
sudo adduser messageboarduser
```

You need to provide a password. (I used messageboarduser also for the password, but of course you should not do the same in a productive environment.)

## Create a new user and database in PostgreSQL

Log into the default PostgreSQL user (called “postgres”) to create a database and assign it to the new user:

```
sudo su - postgres
psql
```

Create a new user that matches the system user you created. Then create a database managed by that user:

```
CREATE USER messageboarduser WITH PASSWORD 'messageboarduser';
CREATE DATABASE messageboard OWNER messageboarduser;
```

Exit out of the interface with the following command:

```
\q
```

Exit out of the default “postgres” user account and log into the user you created with the following commands:

```
exit
sudo su - messageboarduser
```

Sign into the database you created with the following command:

```
psql messageboard
```

## Define database table for the app

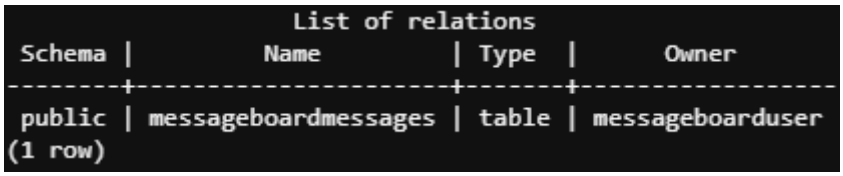
We will define the `messageboardmessages` db table with the following command:

```
CREATE TABLE messageboardmessages (
 message_id varchar(36) PRIMARY KEY,
 nickname varchar(30) NOT NULL,
 message varchar(200) NOT NULL,
 timestamp varchar(50) NOT NULL
);
```

We can see the new table by typing “\d” into the prompt:

```
\d
```

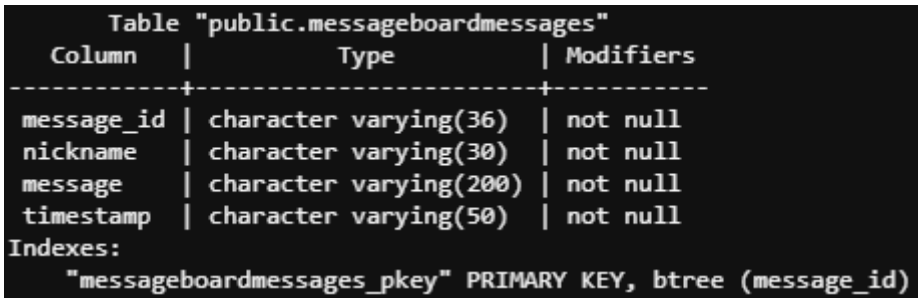
The result:




The following command prints the definition of the table

```
\d messageboardmessages
```

The result:




## Add entries to the table

We will add one sample entry to the newly defined table. (Here the value of `message_id` is a unique identifier which I generated manually. We will get back to unique identifier generation later.)

```
INSERT INTO messageboardmessages(message_id, nickname, message)
VALUES ('201519e8-f3e7-4fcf-9349-3f81b94e908c', 'PostgreSQLUser',
'Hello from PostgreSQL', '2018-11-30 10:48:19 +0000');
```

Let’s query the data to see if it is really there:

```
SELECT * FROM messageboardmessages;
```

The result:

```
             message_id              |    nickname    |        message        |          timestamp
-------------------------------------+----------------+-----------------------+----------------------------
 201519e8-f3e7-4fcf-9349-3f81b94e908c | PostgreSQLUser | Hello from PostgreSQL | 2018-11-30 10:48:19 +0000
(1 row)
```

So we have the entry there.

## Check adjust PostgreSQL config file (optional)

It might happen that when working on a Linux machine of the Ubuntu flavor and setting up a PostgreSQL 9.3 database, you run into the error ***'PG::ConnectionBad: FATAL: Peer authentication failed for user'*** when trying to connect to a database from a web application. The most common fix for this error in a development or staging environment is to loosen the local permissions up a bit. For this you need to follow the following tutorial.

> **PostgreSQL Database: PG::ConnectionBad: FATAL: Peer authentication failed for user**
>
> If, when working on a Linux machine of the Ubuntu flavor and setting up a PostgreSQL 9.3 database, you run into the...
>
> memorytin.com

- locate the file `/etc/postgresql/9.3/main/pg_hba.conf` and open it using `sudo` ( `sudo nano /etc/postgresql/9.3/main/pg_hba.conf` )
- scroll down through the file (almost to the bottom) until you find the section that starts with `#Database administrative login by Unix domain socket`
- directly below that you will find `local all all peer` or `local all postgres peer` — change it to `local all all trust`
- save and close the file
- restart the PostgreSQL server: `sudo service postgresql restart`

## Enhance the app with reading from the db

For this we need the `pg` Gem. Let's enhance the `Gemfile` with this. So the `Gemfile` looks like this:

```
source 'https://rubygems.org'

gem 'sinatra', '2.0.4'
gem 'rerun',   '0.13.0'
gem 'pg',      '1.1.3'
```

Install the Gem:

```
bundle install
```

We need to include `pg` in the `app.rb` to use it.

```
require 'pg'
```

We will also need to enhance the `get '/' do` part to query the data from the database. The final version of `app.rb` can be seen here:

```
1  require 'sinatra'
2  require 'pg'
3
4  # Listen on all interfaces in the development environment
5  # This is needed when we run from Cloud 9 environment
6  # source: https://gist.github.com/jhabdas/5945768
7  set :bind, '0.0.0.0'
8  set :port, 8080
9
10 get '/' do
11   t_msg = []
12
13   begin
14     # connect to the database
15     conection = PG.connect :dbname => 'messageboard', :user => 'messageboarduser', :pas
16
17     # read data from the database
18     t_messages = conection.exec 'SELECT * FROM messageboardmessages'
19
20     # map data to t_msg, which is provided to the erb later
```

```
    # map data to t_msg, which is provided to the erb later
    t_messages.each do |s_message|
      t_msg.push({ nick: s_message['nickname'], msg: s_message['message'] })
    end

  rescue PG::Error => e
    val_error = e.message

  ensure
    conection.close if conection

  end

  # call erb, pass parameters to it
  erb :v_message, :layout => :l_main, :locals => {:t_msg => t_msg, :val_error => val_er
end
```

app.rb hosted with ❤ by GitHub view raw

After this modification, when the server is restarted and the page is reloaded, you should see the following:

# 3rd step: Enable posting of messages and save them in the database

This step is about:

- generating unique identifiers
- validate input
- generate time stamp
- writing data to the database

## Generating unique identifiers

Enhance the `Gemfile` with uuidtools:

```
source 'https://rubygems.org'

gem 'sinatra', '2.0.4'
gem 'rerun',   '0.13.0'
gem 'pg', '1.1.3'
gem 'uuidtools', '2.1.5'
```

This `uuidtools` package can be used to generate a new unique identifier for the new message entry.

The main `app.rb` will be enhanced with the generation of the new identifier.

## Validation of input to avoid SQL injection

We need to avoid SQL injection, therefore we allow only a limited set of characters to be used in the input fields. For this we will use a regular expression check, similar to the following:

```
if ( input =~ /^[a-zA-Z0-9 ]*$/ )
 puts "input is OK!"
end
```

This will allow using only characters of the English alphabet, numbers, and space.

This is just an example for the app, and in a productive environment you should use more robust techniques to avoid SQL injection!

Regular expression tester for Ruby:

**Rubular: a Ruby regular expression editor and tester**

Rubular is a Ruby-based regular expression editor and tester. It's a handy way to test regular expressions as you write...

rubular.com

A good guide to regular expressions can be found here:

**Ruby Regular Expressions**

Ruby Regular Expressions - Learn Ruby in simple and easy steps starting from basic to advanced...

www.tutorialspoint.com

. . .

Guides about SQL injections regarding Ruby can be found here:

**Preventing SQL injections in Ruby (and other vulnerabilities) - Sqreen Blog |...**

Ruby is a wonderful language for beginner coders to start with and scale to large, distributed Web...

blog.sqreen.io

**Fixing SQL Injection Vulnerabilities in Ruby/Rails**

Details on what a SQL Injection vulnerability is, why you need to fix it, and how to fix it!

gavinmiller.io

**Rails SQL Injection Examples**

This page lists many query methods and options in ActiveRecord which do not sanitize raw SQL arguments and are not...

rails-sqli.org

. . .

## Generating time stamp

For the timestamp generation we will use default feature of Ruby:

```
timestamp = Time.now
```

Note that in this example we use the time of the server. This will be saved in the database, and not the time on the clients' computer.

## Writing the data to the database

Database insertion will be done with the following implementation:

```
# connect to the database
connection = PG.connect :dbname => 'messageboard', :user =>
'messageboarduser', :password => 'messageboarduser'

# generate new UUID
val_uuid = UUIDTools::UUID.random_create.to_s

# insert data into the database
timestamp = Time.now
connection.exec "INSERT INTO messageboardmessages(message_id,
nickname, message, timestamp) VALUES ('#{val_uuid}', '#
{params[:nickname]}', '#{params[:message]}', '#{timestamp}');"
```

## Summary of the 3rd step

We needed to modify:

- `Gemfile`
- `app.rb`
- `main.css`

- `l_main.erb`
- `v_message.erb`

In this example implementation there are many repetitions. According to the DRY principle we should eliminate repetitions. I need to refactor this code later. In this example I just would like to show a quick example of using Ruby and Sinatra.

The modified implementation looks like the following:

```
require 'sinatra'
require 'pg'
require 'uuidtools'

# Listen on all interfaces in the development environment
# This is needed when we run from Cloud 9 environment
# source: https://gist.github.com/jhabdas/5945768
set :bind, '0.0.0.0'
set :port, 8080

get '/' do
  t_msg = []
  t_val_error = []

  begin
    # connect to the database
    connection = PG.connect :dbname => 'messageboard', :user => 'messageboarduser', :p

    # read data from the database
    t_messages = connection.exec 'SELECT * FROM messageboardmessages'

    # map data to t_msg, which is provided to the erb later
    t_messages.each do |s_message|
      t_msg.unshift({ nick: s_message['nickname'], msg: s_message['message'], timestam
    end

  rescue PG::Error => e
    t_val_error.unshift(e.message.to_s)

  ensure
    connection.close if connection

  end

  # call erb, pass parameters to it
  erb :v_message, :layout => :l_main, :locals => {:t_msg => t_msg, :t_val_error => t_v
```

```
37
38  end
39
40  post '/newmessage' do
41    t_msg = []
42    t_val_error = []
43
44    # validate input
45    val_input_regex = /^[a-zA-Z0-9 ]*$/
46    if ( ( params[:nickname] != "" ) and
47         ( params[:message]  != "" ) and
48         ( params[:nickname] =~ val_input_regex ) and
49         ( params[:message]  =~ val_input_regex ) )
50
51      begin
52        # connect to the database
53        connection = PG.connect :dbname => 'messageboard', :user => 'messageboarduser',
54
55        # generate new UUID
56        val_uuid = UUIDTools::UUID.random_create.to_s
57
58        # get time stamp
59        timestamp = Time.now
60
61        # insert data into the database
62        connection.exec "INSERT INTO messageboardmessages(message_id, nickname, message,
63                         VALUES ('#{val_uuid}', '#{params[:nickname]}', '#{params[:messa
64
65        # read data from the database
66        t_messages = connection.exec 'SELECT * FROM messageboardmessages'
67
68        # map data to t_msg, which is provided to the erb later
69        t_messages.each do |s_message|
70          t_msg.unshift({ nick: s_message['nickname'], msg: s_message['message'], timest
71        end
72
73      rescue PG::Error => e
74        t_val_error.unshift(e.message.to_s)
75
76      ensure
77        connection.close if connection
78
79      end
80
81      if t_val_error.empty?
82        # redirect to the root
83        redirect to("/")
84      else
```

```
85          # call erb, pass parameters to it
86          erb :v_message, :layout => :l_main, :locals => {:t_msg => t_msg, :t_val_error =>
87        end
88
89      else
90
91        # return error message
92        t_val_error.unshift("Nickname and message should not be empty, and can contain onl
93
94        # read the data again
95        begin
96          # connect to the database
97          connection = PG.connect :dbname => 'messageboard', :user => 'messageboarduser',
98
99          # read data from the database
100         t_messages = connection.exec 'SELECT * FROM messageboardmessages'
101
102         # map data to t_msg, which is provided to the erb later
103         t_messages.each do |s_message|
104           t_msg.unshift({ nick: s_message['nickname'], msg: s_message['message'], timest
105         end
106
107       rescue PG::Error => e
108         t_val_error.unshift(e.message.to_s)
109
110       ensure
111         connection.close if connection
112
113       end
114
115       # call erb, pass parameters to it
116       erb :v_message, :layout => :l_main, :locals => {:t_msg => t_msg, :t_val_error => t
117
118     end
119
120   end
```

app.rb hosted with ❤ by GitHub view raw

```
1  source 'https://rubygems.org'
2
3  gem 'sinatra', '2.0.4'
4  gem 'rerun',   '0.13.0'
5  gem 'pg', '1.1.3'
6  gem 'uuidtools', '2.1.5'
```

Gemfile hosted with ❤ by GitHub view raw

```
1  <!doctype html>
```

```
<html lang=en>
  <head>
    <link rel="stylesheet" type="text/css" href="/stylesheets/main.css">
    <link href="https://fonts.googleapis.com/css?family=Nunito" rel="stylesheet">
    <meta charset=utf-8>
    <title>Message Board</title>
  </head>
  <body>
    <div class="pageheader">
      <h1>Message Board</h1>
    </div>
    <div class="pagesubheader">
      <h3>&copy; 2019 nlharri</h3>
    </div>

    <% t_val_error.each do |error_msg| %>
    <div class="speech-bubble-error">
      <p class="nick">Error: <%= error_msg %></p>
    </div>
    <% end %>

    <div class="speech-bubble">
      <form method="POST" action="/newmessage" id="newmessage">
        <p class="nick"><input id="nick" type="text" placeholder="Nickname" name="nickr
        <p class="message"><textarea id="message" placeholder="Message" name="message"
        <p class="submit"><button id="button" class="submit" type="submit" form="newmes
      </form>
    </div>

    <%= yield %>

  </body>
</html>
```

l_main.erb hosted with ❤️ by GitHub view raw

```
/*
adapted from:
- https://leaverou.github.io/bubbly/
- https://alligator.io/css/styling-form-input-validity/
*/

/* speech bubble */
.speech-bubble {
  position: relative;
  background: #00868B;
  border-radius: .4em;
  box-shadow: 0px 0px 18px #000000;
}
```

```
14
15  .speech-bubble:after {
16    content: '';
17    position: absolute;
18    left: 0;
19    top: 70%;
20    width: 0;
21    height: 0;
22    border: 28px solid transparent;
23    border-right-color: #00868B;
24    border-left: 0;
25    border-bottom: 0;
26    margin-top: -14px;
27    margin-left: -28px;
28  }
29
30  div.speech-bubble {
31    max-width: 420px;
32    padding-top: 1px;
33    padding-right: 20px;
34    padding-bottom: 1px;
35    padding-left: 20px;
36    margin-top: 30px;
37    margin-bottom: 30px;
38    margin-right: 20px;
39    margin-left: 50px;
40    text-shadow: 0px 0px 10px #ffffff;
41    font-family: 'Nunito', sans-serif;
42    color: #FFFFFF;
43    font-size: 18pt;
44  }
45
46  /* ---- error area */
47  .speech-bubble-error {
48    position: relative;
49    background: #ff0000;
50    border-radius: .4em;
51    box-shadow: 0px 0px 18px #000000;
52  }
53
54  .speech-bubble-error:after {
55    content: '';
56    position: absolute;
57    left: 0;
58    top: 70%;
59    width: 0;
60    height: 0;
61    border: 28px solid transparent;
```

```
    border-right-color: #ff0000;
    border-left: 0;
    border-bottom: 0;
    margin-top: -14px;
    margin-left: -28px;
  }

  div.speech-bubble-error {
    max-width: 420px;
    padding-top: 1px;
    padding-right: 20px;
    padding-bottom: 1px;
    padding-left: 20px;
    margin-top: 30px;
    margin-bottom: 30px;
    margin-right: 20px;
    margin-left: 50px;
    text-shadow: 0px 0px 10px #ffffff;
    font-family: 'Nunito', sans-serif;
    color: #FFFFFF;
    font-size: 18pt;
  }

  /* others */
  input, textarea {
    margin: 0 .25rem;
    resize:vertical;
    width: 95%;
    max-width: 400px;
    font-size: 18pt;
    font-family: 'Nunito', sans-serif;
    border: 1px solid #000000;
    border-left: 5px solid;
    border-radius: 7px;
  }

  .nick input {
    font-size: 14pt;
  }

  .message textarea {
    font-style: italic;
  }

  .speech-bubble p.nick {
    font-size: 14pt;
    text-shadow: 0px 0px 3px #ffffff;
```

```
}

.speech-bubble p.message {
  font-style: italic;
}

.speech-bubble p.submit {
  text-align: right;
}

.speech-bubble p.submit button.submit {
  font-size: 16pt;
  font-family: 'Nunito', sans-serif;
  background-color: #FFFFFF;
  border-radius: .4em;
  margin: 0;
  padding: 10px;
  border: 0;
  box-shadow: 0px 0px 8px #ffffff;
}

.speech-bubble p.submit button.submit:hover {
  box-shadow: 0px 0px 3px #ffffff;
  cursor: pointer;
}

.speech-bubble p.submit button.submit:active {
  box-shadow: 0px 0px 3px #fffcbb;
  cursor: pointer;
}

.speech-bubble p.submit button.submit:disabled {
  box-shadow: 0px 0px 3px #ffffff;
  cursor: wait;
}

.speech-bubble p.nick input:disabled, .speech-bubble p.message textarea:disabled {
  cursor: wait;
}


div.pageheader {
  max-width: 500px;
  text-align: center;
  text-shadow: 0px 0px 10px #000000, 0px 0px 10px #000000, 0px 0px 10px #000000;
  font-family: 'Nunito', sans-serif;
  color: #ffffff;
}
```

```
157
158  div.pagesubheader {
159    max-width: 500px;
160    text-align: right;
161    text-shadow: 0px 0px 10px #00868B, 0px 0px 10px #000000, 0px 0px 10px #000000;
162    font-family: 'Nunito', sans-serif;
163    color: #ffffff;
164  }
```

main.css hosted with ❤️ by GitHub view raw

```
1  <% t_msg.each do |msg| %>
2    <div class="speech-bubble">
3      <p class="nick"><%= msg[:nick] %> (<%= msg[:timestamp] %>) says:</p>
4      <p class="message">   "<%= msg[:msg] %>"</p>
```

**PostgreSQL: Linux downloads (Ubuntu)**

PostgreSQL is available in all Ubuntu versions by default. However, Ubuntu "snapshots" a specific...

www.postgresql.org

**Apt - PostgreSQL wiki**

The PostgreSQL Global Development Group (PGDG) maintains an APT repository of PostgreSQL packages for Debian and Ubuntu...

wiki.postgresql.org

**Ruby PostgreSQL tutorials**

I am trying to write a ruby script that interacts with a PostgreSQL database. I am trying to piece...

stackoverflow.com

. . .

# Enhancing the Hello World example

Photo by rawpixel on Unsplash

Now we will enhance the Hello World example. The following shows using the `yield` statement.

```ruby
require 'date'

#############################################################
# Returns the name of the day of the time provided
# -> Time
# <- String (name of the day)
#############################################################
def day_of_the_week(time)
    return Date::DAYNAMES[time.wday]
end
```

day.rb hosted with ❤ by GitHub view raw

```ruby
require 'sinatra'
require './day'
require './tag'

# Listen on all interfaces in the development environment
# This is needed when we run from Cloud 9 environment
```

```ruby
# source: https://gist.github.com/jhabdas/5945768
set :bind, '0.0.0.0'
set :port, 8080

get '/' do
  tag("h1", "Hello, world! Happy #{day_of_the_week(Time.now)}.") do |markup|
      "#{markup}"
  end
end
```

ruby-hello-world-sinatra.rb hosted with ❤ by GitHub view raw

```ruby
def tag(tagname, text)
  html = "<#{tagname}>#{text}</#{tagname}>"
  yield html
end
```

tag.rb hosted with ❤ by GitHub view raw

The `tag` function will wrap the provided text with the provided HTML tag name. With Sinatra, we provided this markup text to the output when the root of the website is accessed by the client.

We created another function, `day_of_the_week` , to get the name of the day of the week. For this we use the built-in `date` library of Ruby.

. . .

## Closing words

I hope you have learned something new today. Thank you for reading this article.

The Message Board App can be accessed in my Github repo.

**nlharri/RubyMessageBoard**

Ruby Message Board Example. Contribute to nlharri/RubyMessageBoard development by...

github.com

. . .

# References


**Learn Enough to Be Dangerous**

Learn Enough to Be Dangerous is designed to unleash your technical genius by teaching you...

www.learnenough.com

**Ruby Programming Language**

A dynamic, open source programming language with a focus on simplicity and productivity. It has an elegant syntax that...

www.ruby-lang.org

**PostgreSQL: The world's most advanced open source database**

The official site for PostgreSQL, the world's most advanced open source database

www.postgresql.org

**sinatra/sinatra**

Classy web-development dressed in a DSL (official / canonical repo) - sinatra/sinatra

github.com


)

Ruby Css3 Postgresql Sinatra Linux

About Help Legal