CS 4990: Ruby on Rails Web Application Development

Assignment: TDD

Requirements

- Implement a new resource in a Ruby on Rails application while following the test-driven development (TDD) process.

- Strictly follow the TDD mantra of “red, green, refactor.” Before writing code to implement your resource, first write a failing test for the code you wish you had. Then, write just enough code to make the test pass. Finally, refactor your code as needed and repeat the process by writing another failing test.

- For the purposes of this assignment, and to facilitate learning, use Git to demonstrate and record your progress in following the TDD process. After completing a phase in the process (red, green, or refactor), use Git to commit your changes with a message that communicates the phase you just completed. You must have a commit for every red and green phase, as these are vital to the TDD process. However, the refactor phase is optional and should be performed only as needed throughout the process.

- For full credit, you are required to write at least 5 tests to test your model, and at least 5 tests to test your controller. Each test must be accompanied by a pair of Git commits (one red followed by one green). Any tests written without the corresponding Git commits will receive no credit.

- Use the RSpec testing framework to write the tests (called specifications, or specs, in RSpec) for this assignment.

Steps

Setup

1. Start by adding RSpec to your project. Add the following to your Gemfile and then run bundle in Terminal to install the required gems.

   ```ruby
   group :development, :test do
     gem 'rspec-rails', '~> 3.0.0'
   end
   ```

2. Tell RSpec to install itself into your project by running the following Rails generate command (provided by RSpec).

   ```bash
   $ bin/rails generate rspec:install
   ```

3. Run a Rails generate command similar to the following to generate a new resource within your project. Notice that RSpec automatically generates spec files for your new controller and model. You’ll add your specs to these two files. Don’t forget to apply the new database migration.

   ```bash
   $ bin/rails generate resource resident nickname:string date_of_undeath:date recruitment_tally:integer on_diet:boolean
   $ bin/rake db:migrate
   ```

4. Before proceeding, commit all changes to your project’s Git repository. Be sure to initialize a Git repository for your project first (if you haven’t already).

   ```bash
   $ git init
   $ git add .
   $ git commit -m "Initial commit"
   ```

Red, Green, Refactor

1. Now write your first failing spec. Open the spec file for your model or controller (located within spec/models/ or spec/controllers/) and add your spec to the describe block therein. For help writing your first specs, refer to the class example, and also consult the RSpec documentation for a list of
matchers that you may use to write the expectations for your specs.

2. Once you’ve written your first failing spec, run the \texttt{spec} rake task to run all specs for your application. If your spec is indeed failing, you may continue. But, if your spec passes, fix it so that it fails like a good little spec.

\begin{verbatim}
$ bin/rake spec
\end{verbatim}

3. Commit your spec changes, and be sure to prefix your commit message with \texttt{[RED]}. For example:

\begin{verbatim}
$ git add spec/models/resident_spec.rb
$ git commit -m "[RED] Add failing spec for Resident#some_method"
\end{verbatim}

4. Now that you’ve got your first failing spec, add just enough code to make your spec pass. Don’t add any additional code until you add another failing spec. Once you’ve added the code, run your specs again to see everything pass. If your specs still don’t pass, fix your code until they do.

5. Commit your code changes, and be sure to prefix your commit message with \texttt{[GREEN]}. For example:

\begin{verbatim}
$ git add app/models/resident.rb
$ git commit -m "[GREEN] Add initial implementation for Resident#some_method"
\end{verbatim}

6. If you feel it’s necessary, take a moment to refactor your existing code to clean it up, but be sure you don’t break any specs in the process (always run specs to be sure). If you make any refactor changes, commit them, and be sure to prefix your commit message with \texttt{[REFACTOR]}. For example:

\begin{verbatim}
$ git add app/models/resident.rb
$ git commit -m "[REFACTOR] Clean up Resident model"
\end{verbatim}

7. Simple, right? Start again at step 1 and write another failing spec!

\section*{Resources}
\begin{itemize}
  \item Refer to the example from class when writing your own specs.
  \item RSpec documentation
  \item Test-driven development on Wikipedia
\end{itemize}

\section*{Continued Learning}
\begin{itemize}
  \item The RSpec Book
\end{itemize}

\section*{Submission}
\begin{itemize}
  \item Show your completed assignment to the instructor during class or office hours to receive credit.