SE31520 Developing Internet-Based Applications

Assignment Report

Assignment 1 Part 2

Forum for the CSA

Version 1.0

Dimitar Velikov

dpv@aber.ac.uk

Word count: 2179

10.12.2017

Table of Contents

1 Introduction

This report discusses the process of solving the "Developing Internet-Based Applications" Assignment 1 Part 2 - "Forum for the CSA".

Every section of this report describes a part of the assignment. Section 2 describes the implementation and the architecture of the forum feature. It is split in several subsections – each of them is corresponding to a task given in the assignment diary. Section 3 describes the implementation of the RESTful client. Section 4 describes the Cucumber testing task. It also includes a description of the gem files that were used in order to implement it. Section 5 lists and describes all the features that are considered to be flair. Section 6 includes critical evaluation, weak points and self-evaluation.

2 Architecture of the CSA application

In order to support a Forum feature, the architecture of the basic CSA application has been extended.

The Forum feature is using Thread and Reply scaffolds. A scaffold is Rails build-in generator of model, view, controller and database migration [1]. Initially the Thread scaffold has been named comments, but some bugs occurred after an attempt of refactoring. **For the sake of readability of this report "Thread" will be used instead of "Comment".**

2.1 Forum index page

The pages that a user can see are the pages stored in app/views folder. By default, the index page has a table listing the details (details such as author, title, date etc.) of each of the threads. The order of listing the threads has been specified in the index function in threads controller, so could the most recently updated post be at the top. Initially the "created_at" and "updated_at" attribute dates are equal, but after replying to a thread, the thread's "updated_at" attribute is updated and this is why the order is set by most resent update time.

```
.order('updated_at DESC')
```

2.2 Creating a thread

Initially each scaffold generated index page has a **New button.** The button redirects the user to views/new which renders "views/_form". After fulfilling the form with the correct data and submitting it the create function in the controller is executed and the thread post is stored into the database. The user is then redirected to the new thread post "show" page. All the thread post replies are listed under the thread using **breadth first search**.

2.3 Displaying thread details in the index page

Each thread post in the index page displays the:

- Author's "firstname" and "surname" if the thread is not anonymous.
- Title – links to the threads/show page of the corresponding thread

- Number of replies – Saved as a column in table “threads” – incremented by 1 when a new reply is creation.
- Number of unread replies. Each user has a table “read_post_ids”. When user visits a thread post page all the reply ids are added to that table. From the moment, the user reads a reply the reply is no longer counted as unread. This can be considered as a not very efficient implementation since it is necessary to loop through all the replies and all the read_post_ids for every thread on the index page.
- Thread post date which is explained in section 2.1.

2.4 Adding pagination to the threads

Thread pagination has been included for the following actions:

- Show thread.
- Edit thread.
- Delete thread.
- New thread.

The necessary steps were to modify each linking part of the forum feature. This includes the index, create, update and destroy functions in the threads controller.

It also includes passing current page, current thread or both as parameters into each of the buttons and actions that link to another page in the views. This includes the images – edit and delete, the back buttons in new thread, show thread and update thread and the update and create thread buttons.

2.5 Thread deletion

Every thread can be deleted by admin user or by the user who has created the thread. This is accomplished by using a conditional statement and `is_admin?` (function which returns true if the current user is admin.). If the user is not admin there is another conditional statement which checks whether the id of the user who has created the post is equal to the id of the current user. The id of the user who has created the post is stored as a database column in table “threads”.

When the thread is deleted it deletes all its replies, even the nested replies. This is accomplished by specifying **`dependent: :destroy`** in the model of Threads and Replies.

2.6 Creating thread replies

In a case when the user tries to submit the form without fulfilling the title and the body of the thread, the thread will not be stored and the user will get an error message displayed. This is accomplished by specifying which are the compulsory fields in the model of the threads via `validates_presence_of`.

The user may post an anonymous thread by specifying it via checkbox which is interpreted as a Boolean.

The user is also able to cancel the new reply clicking the back button which will redirect him back to the thread “show” page.

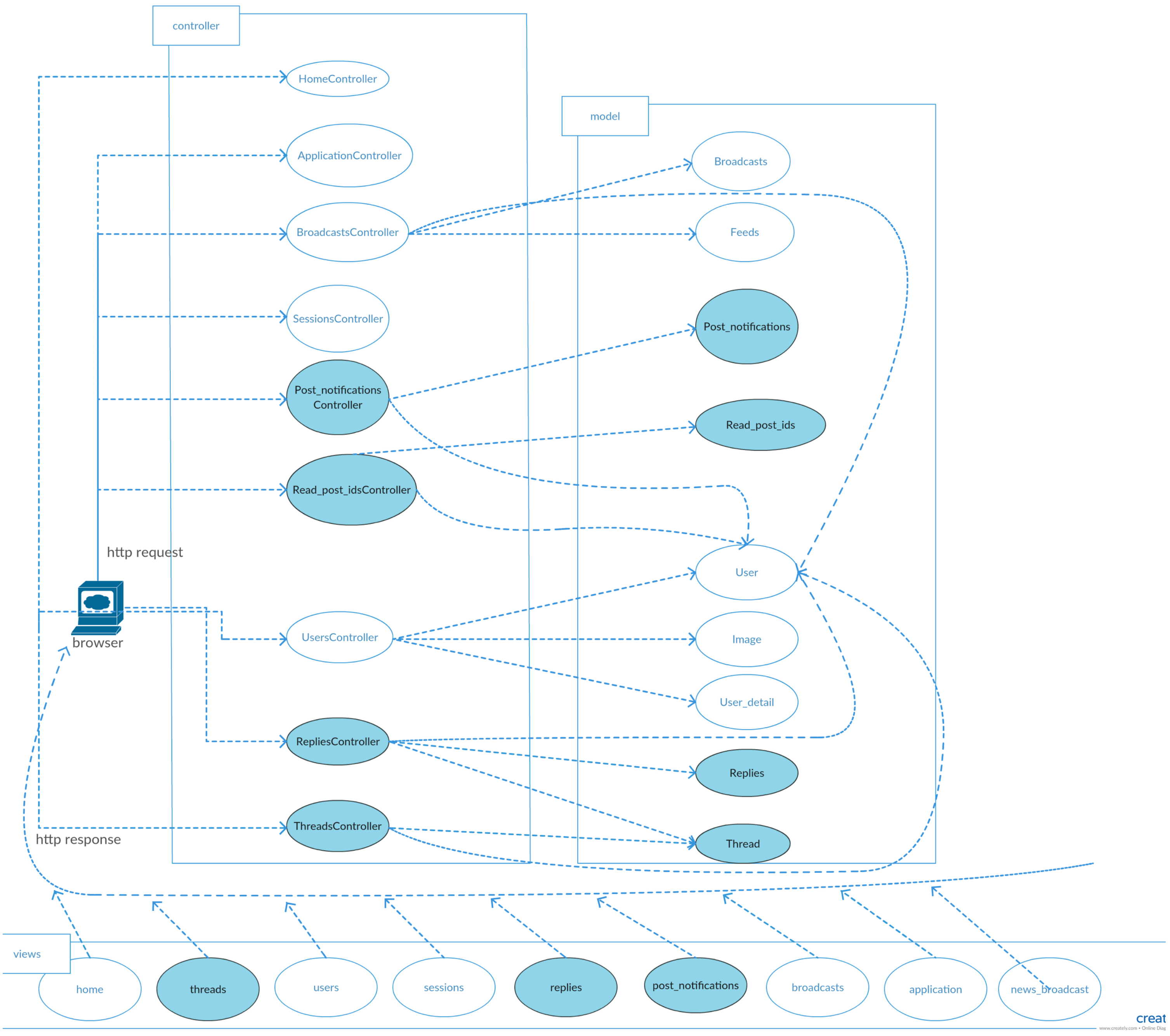
2.7 Nested replies

Implementing nested replies is considered to be one of the difficult parts of this assignment. It is represented as replies having 0..* replies and takes several steps to be implemented:

- Modifying replies model – adding “has_many” replies and “dependent: :destroy”, so could the nested replies be destroyed at Thread deletion.
- Generating a rails migration to add “parent_id” and foreign key column in the replies table.
- Modifying the “routes.rb”.
- Modifying the create function in reply controller. An important step is to check whether “parent_id” is null. If it is not null the reply Is nested and belongs to another reply. Otherwise it is first level reply and belongs to the comments.
- The threads/show page was modified in order to add left margin to the nested replies. The size of the margin depends on the nesting level i.e. margin = nesting level * 50px

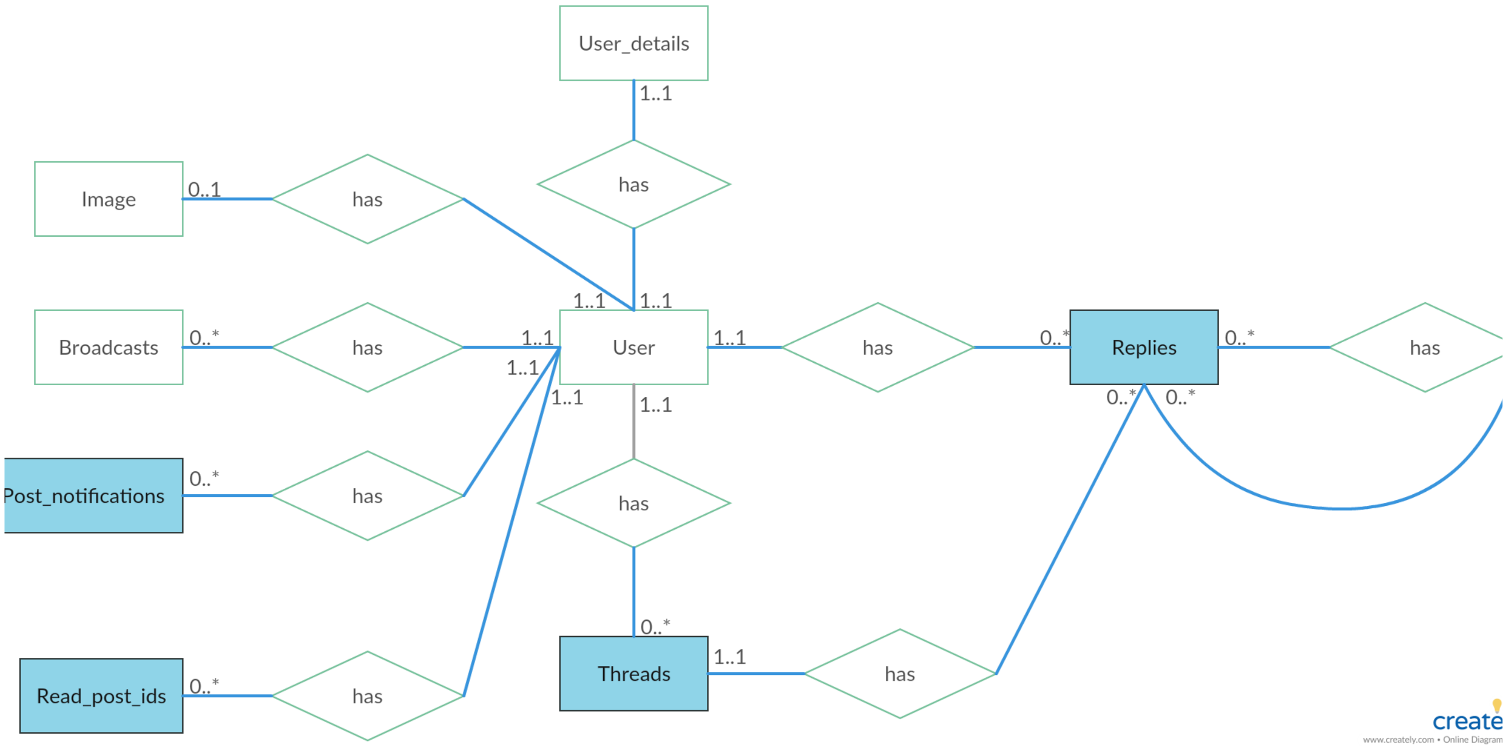
2.8 UML Diagram

The parts of the diagram, highlighted in blue are the features added to the csa application for the purpose of this assignment.



2.9 Entity Relationship Diagram

“The entity relationship diagram the shows the relationships of entity sets stored in a database.” [2] The highlighted entities coloured in blue are the features added to the csa application.



3 Architecture of the RESTful client

The main parts of extending the RESTful client are separating the .json part of the code and files from the rails code and files, modifying the “routes.rb” and extending the provided with the csa client – “csa_rest_client.rb”.

The first step is to set up a namespace for threads in “routes.rb” and set up a folder that that will be expected from rails. This folder has been named “api”. There are also other options that can be set up such as the formatting. The used format is json, which is also a default format.

The next step is to create a copy of the “threads_controller” and paste it into “controllers/api/” folder. Then all the .json references from “controllers/threads_controller.rb” have to be removed as well as all rails references from “controller/api/threads_controller.rb”.

It is also important to move the .json.jbuilder files from the views folder to “views/api” folder. There are hardcoded paths inside the moved files, so they should be changed with adding “api/” at the beginning of each path

The last step is to extend the “csa_rest_client.rb” application. This is a simple console application which as any other application operates as a result of executing a sequence of functions. At the start of the application there is a menu is getting printed in the console. The user can easily navigate through the menu and select the desired options. As required in task 3 from the assignment diary, the menu has been extended to support thread creation and displaying of all threads.

Create thread function works as saving the compulsory fields from the console's user input into variables using "STDIN.gets.chomp". The thread is then created using post method and passing a thread with all the variables into the "domain/api/threads.json", which is basically where all the threads are stored. The domain in this case is localhost:3000, since the "csa-application" is hosted locally.

An important step for each operation is to be authorized otherwise it will not be successful. The authorisation method came up with the basic "csa_rest_client". It works on the principle of hashing. The hash in this case has the admin username and password hardcoded and encoded using Base64 encoding.

4 Cucumber testing

The second task of this assignment is running Cucumber tests. The necessary gems are included in **development** and **test group** in the **Gemfile**. The **key gems** are:

- cucumber-rails – Tool for running automated acceptance testing. Rails version.
- capybara – Testing helper. Simulates the user interaction.
- rspec-rails – Testing framework. Rails version.
- database_cleaner – comes with cucumber-rails. It **deletes** all the **records** from the **test database** before and after each scenario [3]
- selenium-webdriver

This task consists of including the "forum-post.feature" file provided with the assignment into the features folder, creating a step definition file "forum_post_steps.rb" and placing it into "features/step_definiton" folder, defining each step listed in the Forum post feature and implementing it using capybara element matching.

Cucumber testing using capybara is easy to implement because it comes with very helpful debugging console output. It prints statements such as element is not found or if a step is not defined or not defined properly.

Capybara can also check if a page contains some element or has some text displayed on it. This have helped understanding whether a user has logged in or for example if a post has been successfully created. There are also code examples "Capybara cheat sheet" that helped out with figuring out what the matching capabilities of capybara are [4,5,6].

5 Flair section

All of the listed subsections are considered to be flair, because they were not included in the assignment diary and might be looked as extra effort.

5.1 CSA Application

5.1.1 Notifications

Post notifications had been added to the CSA application. In a case when user replies to a thread, the user who has posted that thread gets a notification. When the notification is created there is a check whether the user already has a notification for that post. If it is true the notification gets its "updated_at" "dateTime" updated. A user does not get more than one notification for one thread.

5.1.2 Design

The website design has been modified. It does include a background picture, changing the colour scheme and adding a div sections in threads/show page.

5.1.2 Broadcasting

An option for disabling the broadcasting panel has been added for each user. It can be enabled in the edit page of each user profile. This feature is implemented by adding a "boolean-value" column in the user table.

5.2 Cucumber testing

Additional testing steps have been implemented. They were added to the scenario that has been provided with the assignment diary, because it has been found that after each scenario the database cleaner deletes all the records from the test database.

The additional testing steps are creating a second user which replies to the thread created by the admin user. It shows that the number of posts and number of unread posts are counted correctly. It also shows that there is a notification (**5.1.1**) appearing in the admin account.

6 Critical evaluation

I have found some difficulties when I started working on the Forum feature. After spending a while playing around with Ruby on Rails I've learned how the MVC model operates and how the project files are related to each other. After gaining the understanding the work on the Assignment became easier. The most difficult part from the Forum feature was to implement the nested replies. It also took me a while to understand that the cucumber testing is using separate database, because I had difficulties in my attempts to sign in with an account that was not present.

There are parts of the assignment that present vulnerabilities or that can be implemented in a more efficient way. One of the main vulnerabilities is the threads/index page. The author of each thread is not stored in a column in the threads table, but it is displayed using `User.find(user_id).` In a case when the user account is no longer present (it is deleted for some reason) this will lead to an error. This can be prevented by adding extra column to the Threads table that will store the user who has posted the Thread name, or by including a conditional statement that checks whether the user account is present. The same vulnerability counts for displaying the thread replies. Another way to solve this problem is to implement a ban feature. If the admin decides that the user should no longer have access to the forum he can restrict him for a certain amount of time or forever instead of deleting the user account.

Considering that I have implemented all the requirements, documented them in a well formatted and readable report and added extra features to the given tasks, I do evaluate my attempt for solving this assignment with a high first mark.

Bibliography

[1] "The Rails Command Line — Ruby on Rails Guides", Guides.rubyonrails.org, 2017. [Online]. Available: http://guides.rubyonrails.org/command_line.html. [Accessed: 10- Dec- 2017]
[2] "Entity Relationship Diagram - Everything You Need to Know About ER Diagrams", Smartdraw.com, 2017. [Online]. Available: https://www.smartdraw.com/entity-relationship-diagram/. [Accessed: 10- Dec- 2017]
[3] "cucumber/cucumber-rails", GitHub, 2017. [Online]. Available: https://github.com/cucumber/cucumber-rails. [Accessed: 10- Dec- 2017]
[4] "capybara cheat sheet", Gist, 2017. [Online]. Available: https://gist.github.com/zhengjia/428105. [Accessed: 10- Dec- 2017]
[5] "Capybara cheatsheet", Gist, 2017. [Online]. Available: https://gist.github.com/tomas-stefano/6652111. [Accessed: 10- Dec- 2017]
[6] "capybara cheat sheet", Gist, 2017. [Online]. Available: https://gist.github.com/dakull/085ec87b6a3bf7ffcbf480c46e20b802. [Accessed: 10- Dec- 2017]
[7] "Creately - Online Diagram Editor - Try it Free", Creately.com, 2017. [Online]. Available: https://creately.com/app/. [Accessed: 10- Dec- 2017]
[8] "Ruby on Rails Guides", Guides.rubyonrails.org, 2017. [Online]. Available: http://guides.rubyonrails.org/index.html. [Accessed: 10- Dec- 2017]