SE31520 Assignment 1 - Part 2: Forum for the CSA - version 1

Michal Wojciech Goly [mwg2]

30th November 2017

Contents

1 Introduction **2**

2 The CSA application **2**
2.1 Entity Relationship Diagram 2
2.2 The forum feature 3
2.3 Controllers Diagram 4
2.4 Models Diagram 5
2.5 Redesign for the REST API 6

3 The REST client **6**

4 Cucumber testing **6**

5 Flair **6**
5.1 The overall look and feel 6
5.2 Docker 6
5.3 Build and extra testing 6
5.4 Heroku 6
5.5 Feature branches and Kanban 6

6 Critical evaluation **6**

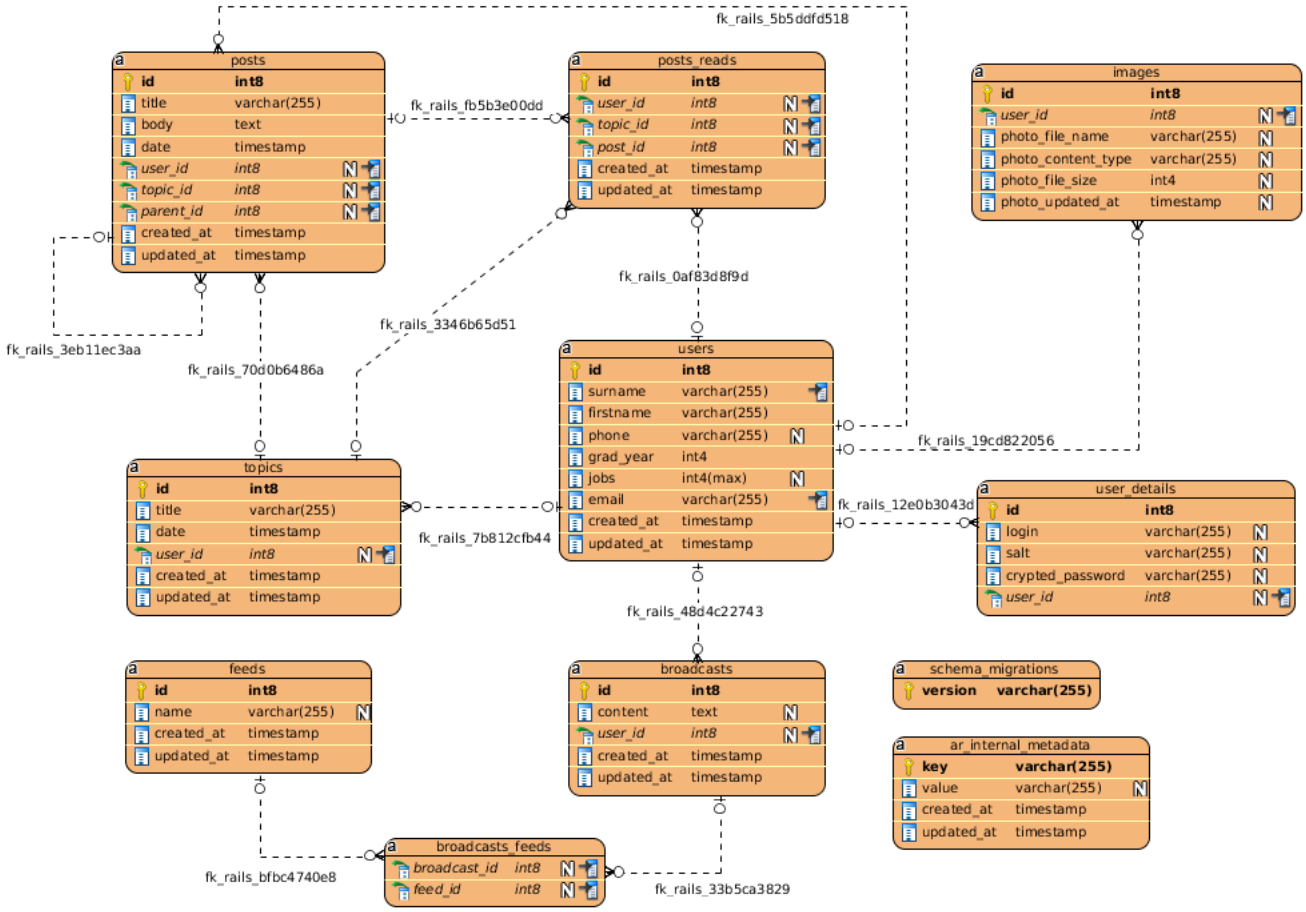
1 Introduction

This document describes the architecture of the forum feature added to the CSA application as part of the assignment submission for the SE31520 module at Aberystwyth University. The following sections describe the design and implementation choices made during the development, alongside testing performed and the implementation of the REST API. The report finishes with the summary of the activities performed to achieve "flair" marks and sums up the work with a critical evaluation.

2 The CSA application

I started the development by familiarising myself with the initial CSA architecture and thinking about different ways I could add the forum functionality required. The Entity Relationship Diagram below depicts the overall database schema of the final system, but my initial low fidelity sketch included only the `Topics`, `Posts` and `Users` tables.

2.1 Entity Relationship Diagram



2.2 The forum feature

Once the initial planning phase has finished, I have created a GitHub repository for the project, created work items, Dockerised the project, integrated it with Circle CI and continuously deployed the `master` branch to Heroku (I have covered the details in the *Flair* section of this document).

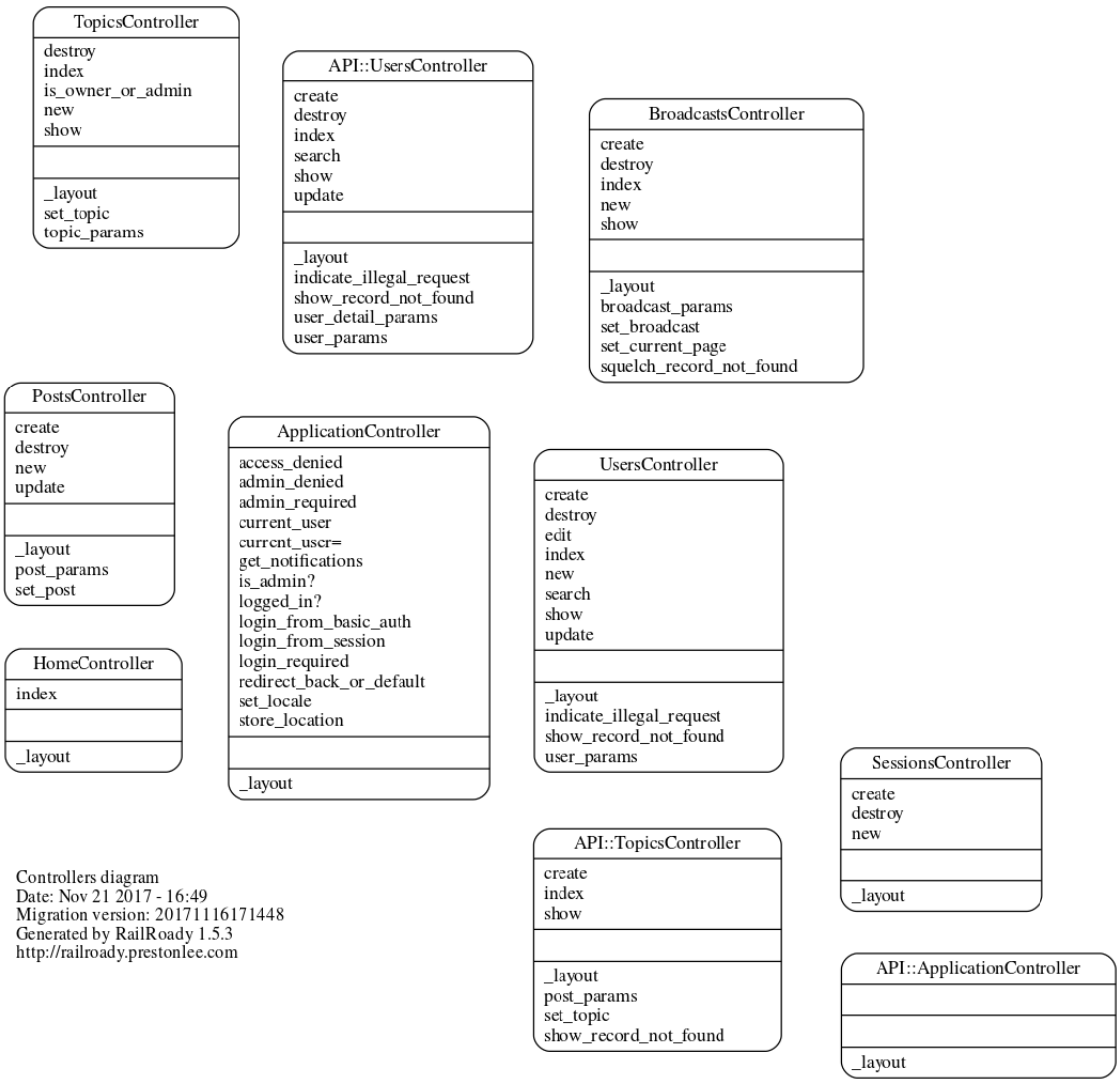
Having cleaned up the CSS and fixed the broken migrations provided (timestamps were in a wrong order), I have scaffolded the `Topic` migration, model, controller and views. I then added a *Forum* tab in the navbar and scaffolded the `Post` in a similar fashion. I wanted to generate as much as possible early on and then remove "dead code" once the forum functionality was implemented. The scaffolded code was not ideal and I had to manually define the `belongs_to` and `has_many` relations between the models. I also used self joins to make sure each `Post` could have a parent to represent posts' replies.

New topics can be created by calling the `/topics/new` route which will trigger the `TopicsController#new` action. Now, instead of creating a new `Topic` model and passing it to the view, I decided to create a `Post`, pass it to the view and use partial rendering to reuse the `/views/posts/_form.html.erb` view for both `Topic` creation and `Post` replies later on. Logically, `Topic` cannot exist on its own therefore such approach seemed optimal to minimise the amount of code written.

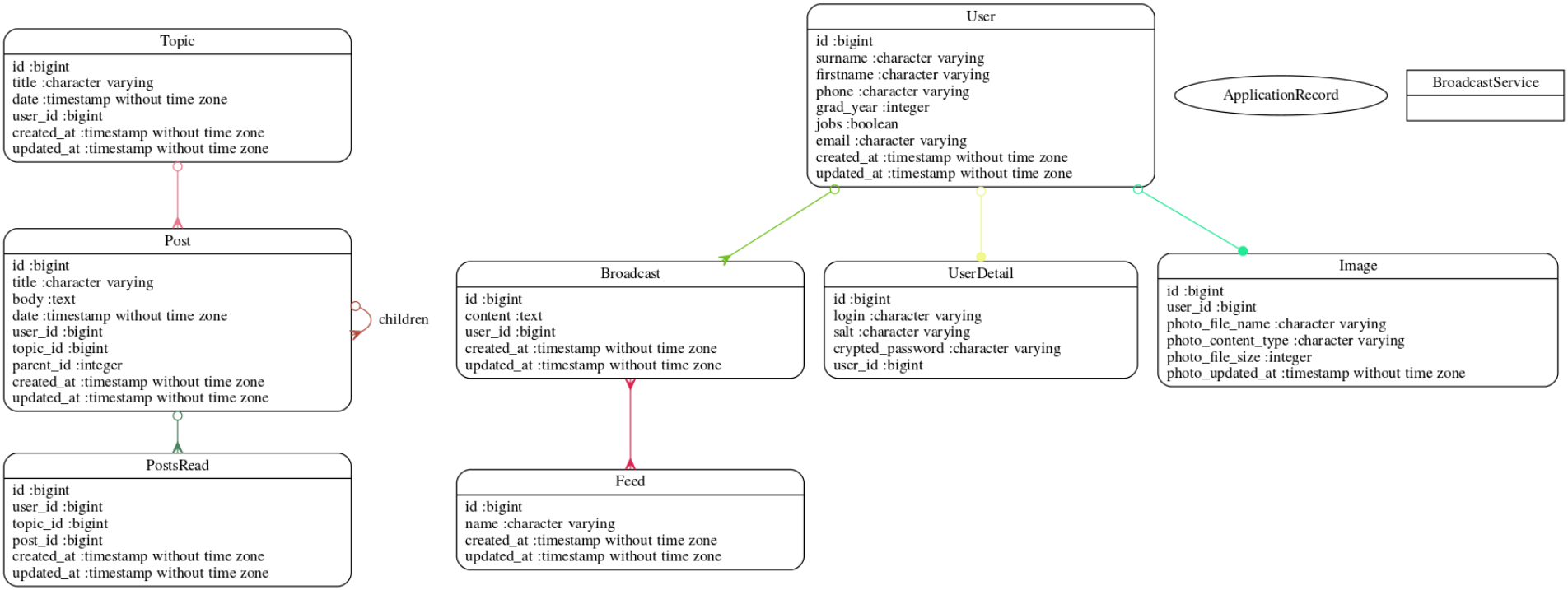
Implementation of the post replies and the associated indenting was argubly the most interesting part of the assignment. The `/views/topics/show.html.erb` view uses material design cards to render each post. The indentation was achieved using the Materialize CSS[1] grid system. The `Topic#post_wrappers` method iterates over the posts of a topic, recursively sorts them and calculates the offsets based on the amount of parents above a specific node. These wrappers are then returned as a simple array back to the view and rendered appropriately.

Finally, I have added the anonymous posting by treating `nil` valued `User` of a `Topic` or `Post` appropriately, the total count of posts for each `Topic` and the unread posts counter. This required the creation of a new table in the database with the associated model `PostsRead`. I took the simple approach of storing 3 foreign keys for a `User`, `Topic` and `Post`. This allowed me to add new records to this table each time user accessed the `/views/topics/show.html.erb` view and render the counter in the topics list.

2.3 Controllers Diagram



2.4 Models Diagram



2.5 Redesign for the REST API

3 The REST client

4 Cucumber testing

5 Flair

5.1 The overall look and feel

Materialize CSS

5.2 Docker

Docker and Docker Compose of test and dev environments

5.3 Build and extra testing

Circle CI, running integration, cucumber tests, deployment of master to a production environment

5.4 Heroku

link to the deployment env

5.5 Feature branches and Kanban

the development process, pull requests, screenshot of the kanban board

6 Critical evaluation

- Could have added controller tests, but did not know how to mock sessions in rails 5 - Improve the look and feel of the rest of the app

References

[1] *Materialize CSS*, materializecss.com [Online], Available: http://materializecss.com/grid.html, [Accessed: Nov. 22, 2017].