Game Contest Server

David Nicholls
Taylor University Department Computer Science and Engineering

Introduction

The Game Contest Server project is a continuation of work that students began in the fall semester of 2013. Our senior project this J-term was to continue work on the server so that professors could utilize it for their classes in the spring. The scope of the project was to develop server that hosts contests and tournaments between players that are uploaded by students in the COS120 and 121 classes.
The students who worked on the project this J-term were: Phil Brocker, Doug Brown, Juan Comboni, Nathan Lundell, David Nicholls, Alex Sjoberg and Justin Southworth.

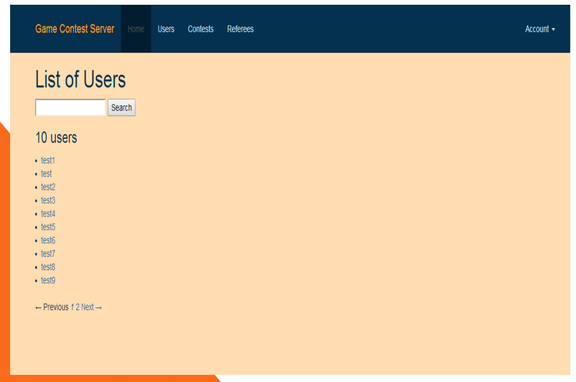


Figure 01: Landing page after the user logs in

Goals

The items that we decided to make priorities for inclusion in the website by the end of our project were the following:
- User Interface Redesign
- Admin Interface Functionality
- Continuous Test Integration
- Back-end Development
- Pagination and Search Functionality

Technical Information

This project was developed in Ruby on Rails. We used Trello, a project organization tool, to track team members time investment and to assist with task allotment. Half of our group developed using the Nitrous online code development environment while the other half of the group used a virtual server that we made at the beginning of J-term specifically for this project. The advantage using Nitrous was that we could access the code from anywhere, this allowed those seniors who lived off campus to develop code from their homes if they chose to. The main advantage for using the virtual server was the longevity it provided. It was a server dedicated to this project and it was hosted on Taylor's computer science server which is very reliable. This meant that we did not have to rely on the nitrous servers being up and running and any downtime could be scheduled for when it was most convenient for us.

Pagination

My main focus for this project was implementing the pagination on all of the necessary pages on our game server. We knew that there were quite a few pages on our website that will grow to have long lists of games, users, tournament etc. and we wanted this to be as user friendly as possible. This meant that a user should not have to look a page with 60 u

```
file | 60 lines (48 sloc) | 1.333 kb   Open Edit Raw Blame History Delete
class ContestsController < ApplicationController
  before_action :ensure_user_logged_in, except: [:index, :show]
  before_action :ensure_contest_creator, except: [:index, :show]
  before_action :ensure_contest_owner, only: [:edit, :update, :destroy]


  def index
    @contests = Contest.search(params[:search]).paginate(:per_page => 10, :page => params[:page])
    if @contests.length == 0
      flash.now[:info] = "There were no contests that matched your search. Please try again!"
    end
  end

  def new
    @contest = Contest.new
  end

  def create
    @contest = current_user.contests.build(acceptable_params)
    if @contest.save
      flash[:success] = 'Contest created.'
      redirect_to @contest
    else
      render 'new'
    end
  end

  def edit
  end

  def update
    if @contest.update(acceptable_params)
      flash[:success] = 'Contest updated.'
      redirect_to @contest
```

Figure 02: Contest Controller Code

```
file | 27 lines (24 sloc) | 0.835 kb   Open Edit Raw Blame History Delete
<h1>List of Contests</h1>
<%= form_tag contests_path, :method => 'get', :id => "contest_search" do %>
  <%= hidden_field_tag :direction, params[:direction] %>
  <%= hidden_field_tag :sort, params[:sort] %>
  <p>
    <%= text_field_tag :search, params[:search] %>
    <%= submit_tag "Search", :description => nil %>
  </p>
<%end%>

<h3><%= pluralize(@contests.length, 'contest', 'contests') %></h3>

<div id="pagination">
  <%if @contests.blank?%>
    <h3><%="There are no contests available!"%></h3>
  <% else %>
    <% @contests.each do |contest| %>
      <li><%= link_to contest.name, contest %>
        <% if current_user==contest.user and current_user.contest_creator %>
          | <%= link_to 'edit', edit_contest_path(contest) %>
        <% end %>
      </li>
    <% end %>
  <% end %>
  <%= will_paginate %>
</div>
```

Figure 03: Contest Index View Code

The two figures above display some the code for the pagination of the lists of contests. Fg.02 shows the contest controller where you specify how many items are should be displayed on each page. Fg.03 shows the code that grabs the lists of contests and at the end of the document that is a call to the will_paginate function which implements that pagination that I specified in the controller.
It was very interesting to see how our code worked together. This was the first time I had worked with pagination and I had never realized how intertwined it would be with the search functionality that we also wanted to include in the website. I enjoyed learning how to work in a relatively large team because I rarely have the opportunity to code with more that one people in computer science.

ERD

Below in fg.04 there is an example of the entity relationship diagram that we developed at the beginning of this project. We made a few changes since its initial development 4 months before in the COS243 class. This is because our understanding of the project and how ERD were to be used had improved due to this class.

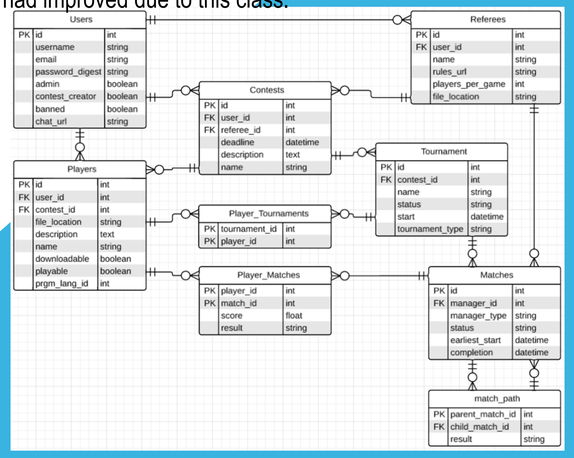
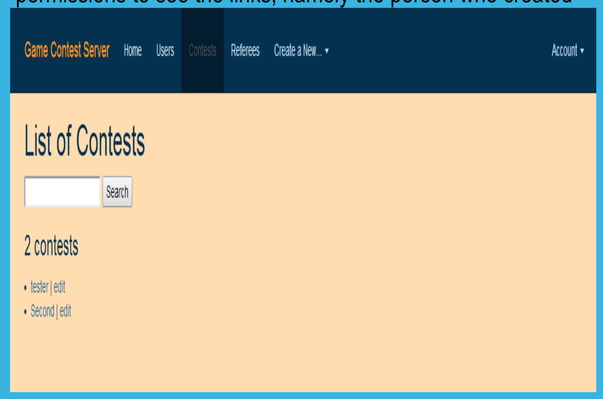


Figure 04: Entity Relationship Diagram of the project.

Pagination on its own does not appear in the ERD and this is because it is an addition to improve the aesthetic appeal of the page and to improve the user experience throughout the website. But when you look at the ERD you can get an idea of how for the reach of pagination is with regard to the project. The same can be said of the edit links that were added after the completion of the pagination addition.

Edit Links

Another aspect of the project that I worked on was the addition of links to the edit pages. These are just links to pages that we created but setup no way for users to navigate to in the early stages of the project. Though this may seem like a simple addition, what makes this complicated is specifying the links to only show up for those who have permissions to see the links, namely the person who created



Future Progress

A recurring theme in our field is the fact that projects can never be developed in a way that they will never have to be updated. This is because our technology is always changing and being improved. That being said, there are still quite a few improved that can be implemented into this project in order to maximize its functionality and the user experience.

Some of the tasks that we need to complete in the future include:
- Individual challenge both frontend and backend
- Allowing games to be played that are more than 2 players
- Allow for multiple rounds per match (currently this must be done in the referee)
- Allow brackets to be customized from the frontend (currently randomly created)
- Compile players when uploaded (currently they must be executable)
- Test uploaded players and referees to ensure they interact with match_wrapper correctly
- Logging and recording games for playback
- Visually displaying games as they are played
- Add more tournament types
- Improve error checking, such as if the player or referee dies
- Much more robust unit testing of the backend.
- Create Matches page with pagination and search
- Add download player functionality

Acknowledgements

I want to express my gratitude to all everyone who had a hand in the making of this project. But I would like to make a few specific mentions:
Dr. Geisler – for his guidance during this project and assistance when we needed encouragement
Dr. Brandle and Dr. White - for their submissions of their referees and players so that we could thoroughly test our system.
Students of 243 - for the initial work on this project and their contribution to the general development of the frontend of the project.
Nate White and Nathan Lickey - for help with our server and virtual machine where some of the team members did their development.

What I learned

This project has been a great learning experience for me. It has serve to solidify many of the concepts that I was taught in COS243. I now understand how beneficial test driven development can be. This is because it forces you consider the features that are essential for the job you are about to complete and it also helps you to know when you have met the requirements of the job you are working on. I also learned the importance of consistently merging with main dev branch. Quite a few time I broke my repository because certain files were changed that caused my tests to fail regardless of whether I had changed them or not.