

Game Contest Server

Justin Southworth

Department of Computer Science and Engineering, Taylor University, Upland, Indiana 46989

Introduction

The Game Contest Server is a project run by the Computer Science and Engineering Department at Taylor University. It began in the fall semester of 2013 in COS 243 – Multi-Tier Web Application Development, taught by Dr. Geisler. The long-term goal of the project is to create and maintain a web site where professors or other authorized users can create contests of certain game types into which users (mainly students) can upload AIs in order to have a fun and easy way to see which program is the best. More general information can be found on our group poster.

Fall 2013

Work done in the fall semester focused on creating the front-end of the website through test-driven development. Test-driven development is a coding process which starts by determining short-term goals, then writing tests for those goals. Each test checks if a certain piece of functionality is present, and together check if all goals are currently met. The tests should initially fail, as no code has been written yet, but as correct code is added, the tests will begin passing, which shows what functionality is working and what still needs to be created or corrected. Together the tests give a clear assessment of the project's progress .

In the fall, Dr. Geisler wrote the tests for the front-end after which the students would add content to their websites to pass those tests. They built the project using Ruby on Rails, which is an open source web application framework based off of the Ruby programming language. They also used Bootstrap, a front-end toolkit, to easily increase the visual appeal of their websites. Dr. Geisler kept a branch of his own personal implementation on GitHub, which is the code we started the project with.

Goals

We had all had at least some exposure to the project, with most of us completing COS 243 in the fall. Partially because of this, Dr. Geisler gave us some freedom in what features we wanted to add to the website, while specifying some must-have deliverables. All together, our main goals for the J-term were:

- Front-end redesign
- Create an interface for the administrator
- Back-end development
- Implementation of pagination and search functionality
- Continuous test integration

Of these goals, I initially focused on redesigning the front-end interface before moving on to creating tests for new functionality we had added to the system.

User Interface

Before this project, I had no experience designing websites outside of class assignments, so working on the front-end of the site was intimidating.

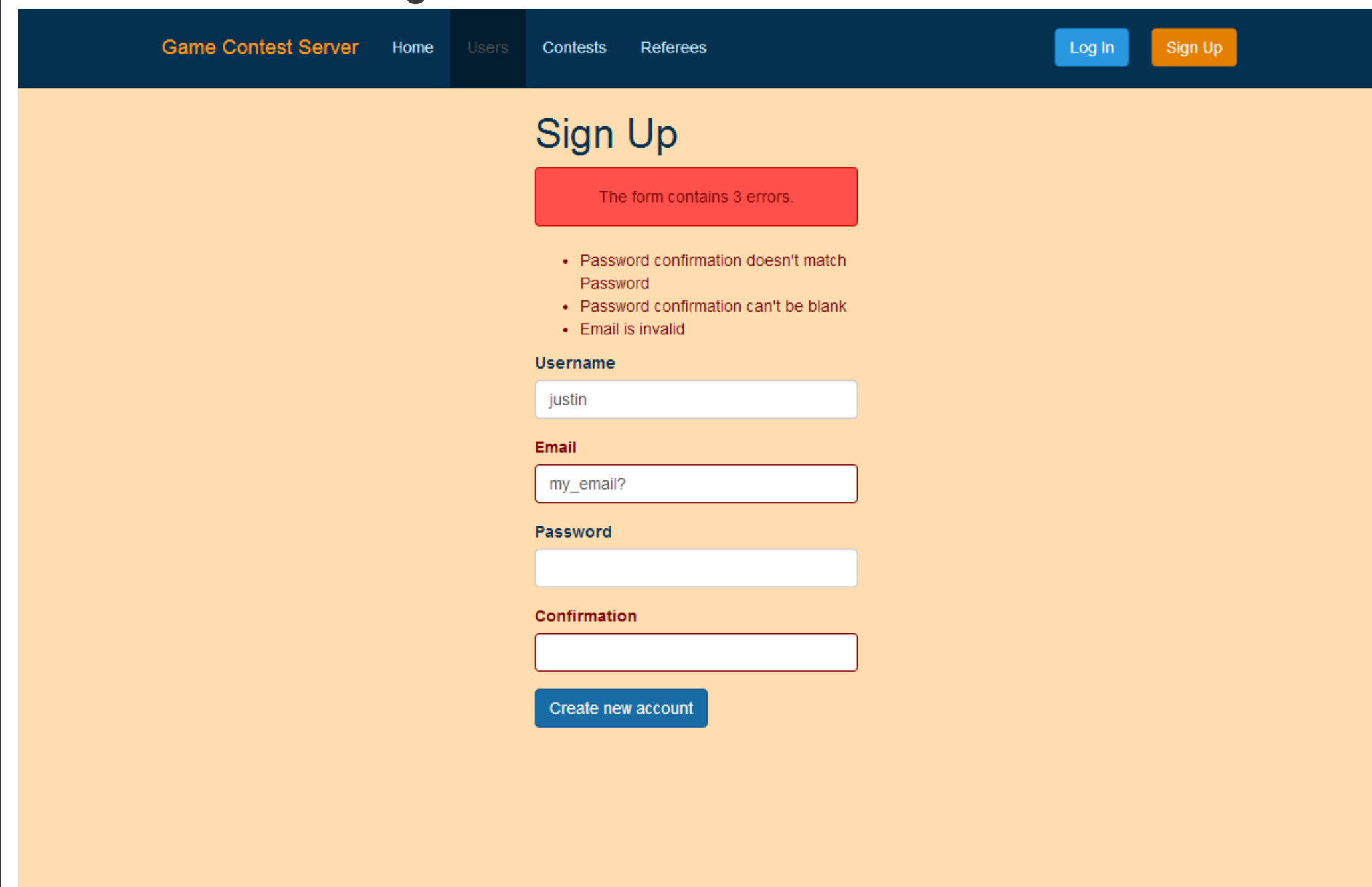


Figure 1: Sign up page with errors from previously submitted illegal inputs. The red box is a called a flash, which is a Rails way of notifying users of the status of their actions, such as informing them they have submitted an invalid form, as seen above.

My main contributions to the front-end included a new color scheme, which is illustrated in the screenshots taken from the website. I used Adobe Kuler and colorschemedesigner.com to create a simple color scheme that included a variety of colors and was easy on the eyes.

I also worked on quite a few small improvements to the user interface. For instance, I removed rounded corners from the navigation bar and any flashes that were the entire width of the web page, while retaining rounded corners on anything that was not the full width, since it is Bootstrap's default behavior and standard for many modern websites. I also added links and buttons to the navigation bar as needed for the updated system.

I completed one final large piece of functionality for the front-end side of the project, which was changing the way the URLs worked. By default, if I wanted to navigate directly to the “show” page for a certain tournament, I would have to type “www.futureurl.edu/tournaments/365” into the address bar of my Web browser if the ID for that tournament was 365. There are a few problems with this. First, the tournament's ID is auto-generated by the database when the tournament is first created. This means that the number is meaningless to the creator of the tournament or any other user of the system. In addition, it means that a user must memorize the number in order to navigate directly to the page. URLs should be meaningful and help the user understand where they are in the website's hierarchy rather than confuse them with meaningless numbers. Having the ID in the URL makes for a bad user experience, so I tried to find a way for the URL to be understandable for a user.

User Interface (cont'd.)

The first option was to define overwrite the default to_param method in each model. Overwriting it would allow us to control what is in the URL rather than the default ID; however, removing the ID from the URL breaks Rails' ability to search the database through the ID, since the URL doesn't show what it is. We could overwrite the find method to search by whatever we put in the to_param method (username, contest name, etc.), but then the find method is unable to search by an ID. Perhaps this would not have been a problem if we were developing from scratch, but we already had code that assumed we could search our database through the ID of an object. Also, any future developer of the system would assume the same thing, adding to their learning curve and giving them unnecessary frustration. It also violates the “don't repeat yourself” (DRY) coding practice, since the value in the URL for one model must be maintained in two separate places. Beyond that, it must be maintained for every model we have a page for, which is a horrendous violation of good coding practice.

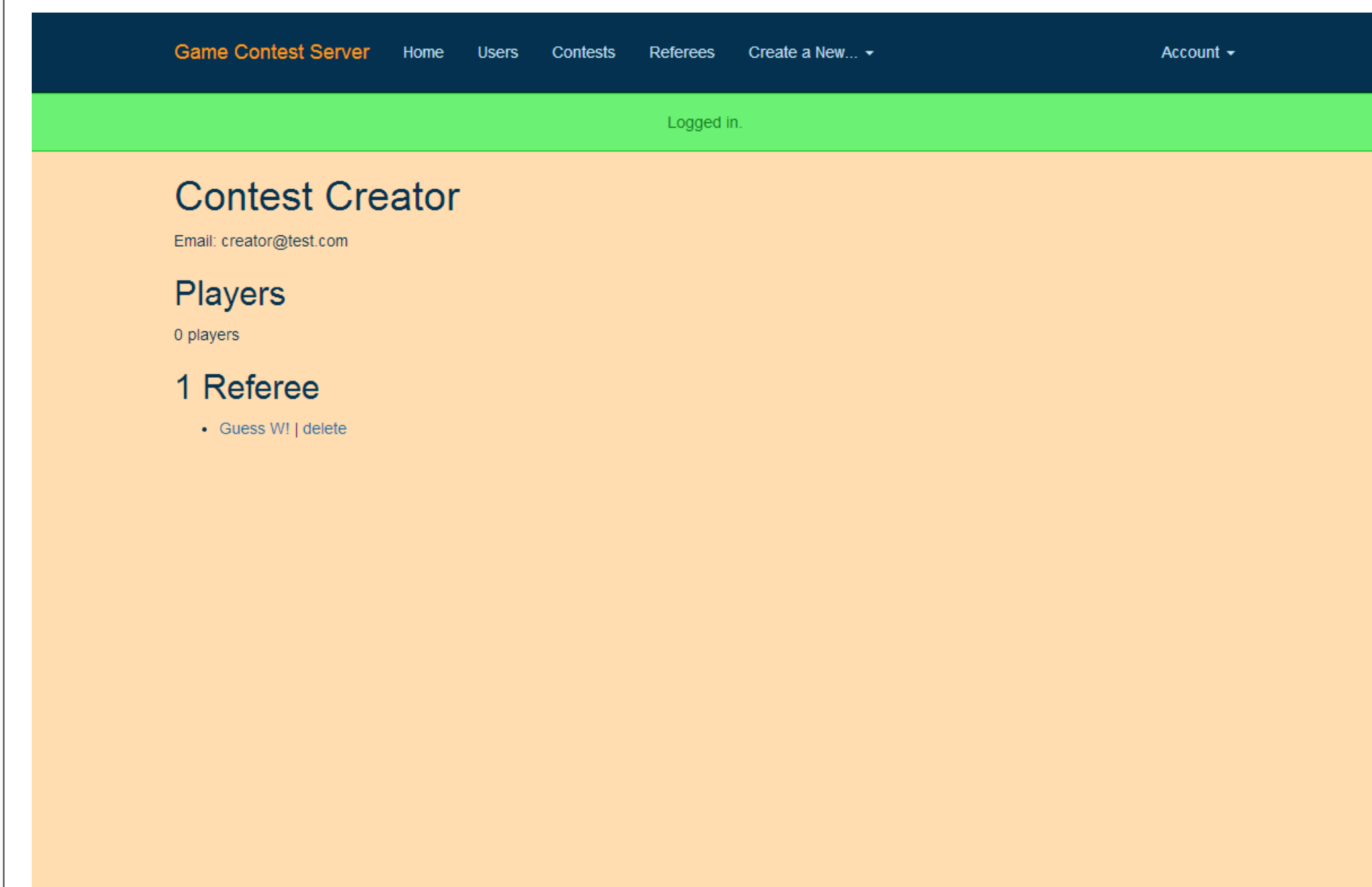


Figure 2: “Show” page for a user who is a contest creator. This user has just logged in, so there is a full-width flash informing them that their log in attempt was successful.

Fortunately, I found a Ruby gem called FriendlyID, which solved the problem. It allowed me to customize the URLs, while extending the Rails find method to be able to search the database by an ID or the value given to it through the URL, all while allowing the code to remain DRY. The installation was a bit tricky, though. It caused some database-migration errors that were disguised as undefined-variable errors, along with some other errors. In the end, though, the hassle was worth the added benefit of URLs that make sense to users.

Testing

The majority of my time was spent editing and writing tests in the RSpec testing framework for front-end improvements that were made. For instance, we added a new table to our database that needed a front-end view, so I wrote all the tests for the model

Testing (cont'd)

and views for the table. This was my first time writing tests of this scope, but I was able to learn RSpec's syntax and formatting through reviewing Dr. Geisler's tests, sometimes only copying and editing them if they were close to the test I needed to write.

What I Learned

Don't work 70 hours in 6 days. I enjoy helping other's solve their issues and bugs, which is a good thing, but I spent so much time working on other's code it forced me to either not complete my own code, or work a ridiculous number of hours. I chose to do the latter because of our time constraint. In nearly all situations, (including mine) there's a better solution.

It is worth learning a powerful text editor. I don't think I would be exaggerating to say that knowing how to use Vim well saved me hours even though the project only lasted for about 2 weeks. I'm very grateful Dr. Brandle brought me out of the world of gedit in COS 121.

In addition to continuing to learn how to efficiently use Vim, I have gained valuable experience using Rails, Bootstrap, RSpec, FactoryGirl, Git, a VM, and Trello. But even more important are the concepts behind each of the frameworks or applications that persist regardless of what specific framework or application I use in the future.

Future Work

The UI is functional, but not pretty. Since we had such a short amount of time to complete the project and I have very little experience designing websites, I stopped short of trying to create an ideal UI. Instead I settled for a decent setup that someone with more design experience can turn into a professional-looking website.

As I wrote the tests for the new tournaments table, I found some errors and weaknesses in some of the preexisting tests, some of which I had time to rewrite, while others I had to leave be. Those still need to be fixed, and others need to be checked.

Acknowledgments

Thanks to:

- Dr. Geisler for being readily available and willing to give valuable insight into Ruby on Rails, FactoryGirl, and RSpec, along with guiding us in COS 243 and during J-term.
- Dr. White and Dr. Brandle for their interest and assistance in the development of this project
- Nathan Lickey and Nate White for helping set up or VM and for printing our posters
- Phil Bocker for being an awesome team lead
- Alex Sjoberg for helping me solve many difficult problems