Horse Show Administration Program

Shannon Darroch
University of Virginia
sed5qg@virginia.edu

Lalita Mallapragada
University of Virginia
lm3gc@virginia.edu

Shivani Nathan
University of Virginia
sn3nu@virginia.edu

Gina Pappagallo
University of Virginia
gmp5vb@virginia.edu

Rhea Prahlad
University of Virginia
rp8jd@virginia.edu

Tarun Saharya
University of Virginia
ts4pe@virginia.edu

Anna Wu
University of Virginia
yw6fy@virginia.edu

ABSTRACT

The Horse Show Administration App is a locally-run application that allows the Hoof-N-Woof 4-H officials to administer their horse shows. Currently, not much technology exists that aids in the administration of a horse show. Our clients, Bertha Durbin and Rebecca Adams, wanted to escape their labor-intensive, pen-and-paper methods. The Horse Show Administration App handles rider and horse registrations, show organization, scoring, billing, and several document generation functionalities. The application successfully speeds up the entire process of running a horse show, as well as the administrative tasks associated with creating a show.

Our team researched building a program that allowed data storage to be done more efficiently. The final product replaced the manual methods and should be maintained and debugged easily through thorough testing and documentation. This transition to a data management system will save our clients hours of time, and allow them to more easily transfer data from one show to the next. This paper reviews the original system, explains the background and design of the new system, and analyzes the resulting improvements. Understanding these components will allow the reader to appreciate the transformation from a completely manual data management system to a self-sustaining software program that works in environments without internet access.

1 INTRODUCTION

Hoof 'N' Woof is a community outreach program that organizes horse shows and other knowledge-based competitions that stem from the educational mission of 4-H. 4-H is a larger national organization that supports "learning by doing" and organizes projects in areas like health, science, agriculture, and civic engagement. Hoof 'N' Woof extends this mission through volunteering in general and also in equine fields. Hoof 'N' Woof is run by its youth members, so most of the administration is handled by students younger than 18 with the exception of finances and show organization.

Bertha and Rebecca handle any responsibilities that require adult supervision. Bertha Durbin is considered the main client because she organizes the shows primarily, while Rebecca has mainly acted as an adviser and interpreter for Bertha throughout the year due to having more experience working with technology. Volunteering in the community is required in order to go to some of the equine competitions available to the members. Community support includes going to senior centers, highway cleanup, food pantry volunteering, and more. The club is completely youth run, but adults assist only when they need an adult to step forward for special permissions and financial and data management practices. Youth members aren't equipped to handle certain aspects of business management, and also sometimes are unable to perform certain business tasks without an adult's permission.

The main problems faced by Hoof 'N' Woof include high rates of club management turnover, community recruitment, event management, and data management. There is a changeover in club management almost every year, and the next leaders are groomed each year they are involved to take on more responsibility. This constant change in leadership makes it difficult to maintain continuity regarding how to manage data and other procedural aspects of the club. It's also difficult to get the word out about the club and encourage students to join; there is a low turnover rate, and these members are necessary for managing the activities, making sure everyone knows where they need to be, and other administrative problems.

The organization's administrators aren't very tech savvy and also rely on the technical ability of members' parents and volunteers to help manage the system. One example of previous technology use was that a volunteer parent handled score sheets with nice spreadsheets and he wanted to step down, so someone needed to take over and maintain continuity. Continuity is improved with a technical program due to the job being streamlined and the ability to have software to manage the horse shows year to year.

The way the client currently fix problems like this is by grooming leadership and maintaining a strong volunteer base so they can continue to maintain programs annually. They handle information management by storing and writing everything manually. They have a giant grid for every show that handles maintaining how everyone did and how much to charge each rider. To put all of the information together, they manually score, send reports, and handle grids and billing. They work all day at the horse show, and prep is at least 2 weeks of input, making grids, creating horse-rider combinations, printing labels for back of a combo number, organizing the notebook of everyone's registration, and more. The current way to solve the problem is to use some database management, which fixes the tedious way of setting up the show as it is done now, eliminates the day of the show billing issues, gate entry admission, and removing horse-rider combinations from classes as needed. Physical file systems typically get the job done, but they are inefficient and less secure than a software program with a database system.

Our system will be beneficial because it will eliminate all of the current tedious manners of managing data, and will efficiently store information that can be used for present and future shows. We will make almost all of the functionality available online so that there is no need to keep paper records. The information for reports will be streamlined so that the administrators can click a button in order to fill out the form. The models we have created will allow all of the relationships they currently draw out by hand to exist in the database.

2 BACKGROUND

We used Python 3.7.x with the Django 2.1.x front-end and back-end web framework to develop the Horse Show Administration application. Django is a useful framework to quickly develop applications and provide built-in security features, such as CSRF protection. Django provides an abstraction layer to the SQLite database, which makes database manipulation easy and SQL-command free. [1]. The front-end framework Bootstrap was also used to add styling to the various HTML elements in the templates of the application and provide dynamic behavior to the app. The Pylabel and ReportLab packages were used to generate the riders' labels. The Pdfrw package was used to generate PDFs for the show score report.

The Hoof-N-Woof clients specifically wanted to run the app locally rather than deploying it with a hosting service provider because the internet connection at many horse show locations are not available, which would prevent the clients from accessing the application. To simplify the process of starting, stopping, or restarting the local server or loading or backing up the database file for the clients, a Windows desktop application was created using the Windows Form Application project template and batch scripts.

3 RELATED WORK

The Hoof-N-Woof horse shows, like many other horse shows, are usually held in locations in which internet access is limited or non-existent. Another problem is that in rural communities, there are not as many software development companies available to construct these types of programs. Due to these issues, there have not been many innovative software solutions developed for the purpose of managing horse shows. Most automated apps that could meet our clients' requirements require wireless connection to the internet, which they often don't have onsite for a show.

This project is also tailored to the specific functionalities that the Hoof-N-Woof 4-H clients have requested. This system is designed to fit the custom structure of the Hoof-N-Woof horse shows and includes features such as the ability to automatically generate labels and populate PDF files with show scores in a manner that accommodates the specific needs of the clients. Many horse show organizations are run differently on a local level, and uniformity is only found on a national level.

4 SYSTEM DESIGN

4.1 Users and UI

The main goal of this system is to handle all the functionality required for organizing a horse show. The main user of the system will be an adult administrator who has been trained to navigate the system and has created an account through Django's User Creation Form middleware[2]. There are Django template tags that block content and prevent users that have not logged into the system from viewing any pages except the login and signup page. The main user organizes and employs all of the system's functionality to create a show, so there is only one type of user, and they have access to every view once they are logged in. There is no need for a superuser in this case because any necessary database changes can be made through the program and creating a superuser would only cause potential problems if the users don't understand how to properly use this access.

Bootstrap helps employ user permissions and maintains the user interface to be responsive and simple to navigate.

Bootstrap allowed for consistency in the types of buttons, navigation, and fonts. The client is not used to using software programs, so the navigation and ease of the UI is extremely important. Bootstrap also made it simpler to utilize JQuery for better response time. Autocomplete functionality, step-by-step navigation, tables for visualization, and AJAX all helped the client have an easier time navigating the system and inputting data.

Bootstrap was used to style UI elements such as buttons and the side navigation bar. This side navigation bar provides the client with quick and easy access to important pages of the app. Also, breadcrumb navigation was implemented to help the user understand their location and move between views quickly. Additionally, the system utilizes modal dialogs (popups), which reduced the number of redirections to other pages.

4.2 **Show Organization (Models and Relationships)**

The design of the system considered the use of Django and SQLite when developing the relationships between different models, and implementing functionality. The models included are necessary for set up, registration, and entering information on the day of the show. The relationships established between these models allowed the program to set up multiple shows, score classes and divisions, and adding or removing horse-rider combinations from a class. There is also a complex hierarchy between Shows, Divisions, and Classes that is identified through model relationships. Shows contain both divisions and classes, and divisions contain classes, but divisions are only necessary to define for organization and scoring the division champion and reserve champion.

Forming the system models was easy because they were one-to-one representations of the system's high-level objects (Rider, Horse, Show, Division, Class, etc.) However, many-to-many relationships between certain models posed challenges for structuring information. For instance, there is a many-to-many relationship between Horse and Rider, since multiple riders could be associated with one horse, and vice versa. The challenge was determining in which model to place fields containing information about their pairing. This was resolved by creating an intermediate model (HorseRiderCombo) that would represent the pairing between Horse and Rider and hence would contain information about their relationship. Both Horse and Rider contain a field to the other model marked with the code "through = HorseRiderCombo" to indicate that HorseRiderCombo was their intermediate model. The same was done for the pairing of HorseRiderCombo and Class: ClassParticipation.

4.3 **VHSA Reports and Labels**

The clients required the generation of score reports that adhere to the standards of the VHSA and 4H club. A combo is made up of a horse and a rider that are uniquely paired to compete in the show together, and there is a many-to-many relationship between horses and riders. Horse-rider combinations are added to classes, billed for each class they participate in, and receive a score depending on how well they performed in each class. Shows require information to be reported to the Virginia Horse Show Association (VHSA) in order for participants to attend higher-level shows in the future, so that information is included for each horse-rider combination. Once all of the necessary data inputs are made for the VHSA report, a pdf containing the required information was filled out using a package called pdfrw, which is a python library for reading and writing pdf files [3].

Pdfrw functionality is also used to help print labels for horse-rider combinations on the day of the show. The labels can be printed so that each horse-rider combination can receive their label when the check in on the day of the show. The label sheets are generated using PyLabels, are three columns by ten rows, and able to be downloaded or printed after generation like any other pdf. Labels are adhered onto the backs of the riders' number which are worn during competition. Once the label is filled out, the user can either download or directly print the information from the browser. The system's design allows ranking horse-rider combinations and combination information to be inserted into pdfs much more efficiently so that the client doesn't have to manually enter this data after the show is over. Printing is not required to be directly connected to our system because our program is run on a local browser, but if no internet is available, the printer will need to be manually connected instead of through a wireless connection.

4.4 **Local Server Implementation**

The system is run by a local Windows desktop application due to the need to run the application locally. The desktop application consists of buttons which call upon batch script commands to make it easier for the user to perform operations. It was determined that this would be the most suitable option (as opposed to using a virtual machine) when considering ease of use for the client as well as her previous experience with Windows applications.

The client would make use of the Windows application in order to start the Horse Show application. The Windows application has a button which runs the server and then launches the application in the browser automatically. The desktop application also has buttons to both stop and restart the application as well. In addition, the desktop application provides functionality to both load and backup the database file. The ability to update the code in the local repo from the GitHub repository is also included as a functionality as well. The desktop app serves as an effective way to make it easier for the client to run the project locally without having to manually type shell commands.

4.5 System Design Challenges

4.5.1 Models

One challenge we had was improperly assigning primary keys to each model. For example, we assigned "class_num" or the class number as the primary key of Class under the false assumption that the class number would uniquely identify a class. However, both the foreign key to the show containing the class and the class number are required to uniquely identify the class, since other shows could contain a class with the same class number. This incorrect primary key problem also applied to every other model.

Since creating composite primary keys was impossible, the solution was to remove the primary keys from affected models and let Django generate the primary key (id) for each model. To enforce uniqueness constraints, i.e. constraints that specify which fields any two model instances cannot have in common, the "unique_together" tuple metafield was added to each affected model. For example, the Rider model contains "unique_together = ('last_name', 'first_name', 'email')" because no two riders can have the same first name, last name, and email. This field ensures that the model's forms validate against uniqueness-violating input.

The validation did not always work correctly. For example, in some creation forms in which a model's field is excluded, the form could not correctly validate since it did not have access to the excluded field. As a result, the form falsely became valid when the uniqueness constraint was violated. Since the unique constraint is enforced at the database level, violating it would incur a "UniqueConstraint" exception, crashing the entire page. To fix this, either the exception was handled in the form of a try-catch block as soon the form was saved or the "validation_unique" function in the model's form was manually overridden to exclude the problematic field.

4.5.2 Static and Dynamic Interface

Originally, the app's design was poor. There was a poor use of space, a substantial lack of color-coding, UI elements were misaligned, and it was hard to navigate. These issues were fixed by incorporating the front-end framework Bootstrap and a form-revamping module called "crispy forms". Into the app. The combination resulted in a much more usable and aesthetically-pleasing interface. Bootstrap UI elements such as the sidebar enhanced the navigation of the website through several buttons that linked to other parts of the website. Autocomplete dropdown menus were added to quickly search and filter for options, which sped up UI performance. Lists were converted into more organized, easier to read tables.

Nevertheless, our app suffered from being static. Most user interactions, especially involving form submissions, reloaded the page instead of re-rendering only parts of the page or dynamically altering UI elements. This was inefficient, unmaintainable, made it difficult to handle data passing and persistence between UI elements, and sometimes confused the client, ultimately delivering a poor user experience. Our app attempted to work around these issues. For example, session variables were created to store data between pages so that pages could retain and display crucial information. However, this led to a stateful system, which introduced such new issues as different pages being shown for the same URL requests, making debugging harder and messing with user expectations. Also, certain pages could not be reused for other workflows because they were designed to redirect to a particular page based on the state of the system. To fix this, we incorporated AJAX using JQuery into our pages. Now, UI elements could be re-rendered without having to reload the page, due to the asynchronous requests to the server.

Additionally, we implemented Bootstrap's dynamic UI elements such as the modal popup, which allowed model creation through a form, precluding the need to visit another page and create the model there. The "data" attribute of HTML elements was used to pass arbitrary data from UI elements to javascript handlers. All this ultimately resulted in a much more fluid, comfortable, and dynamic interface.

4.5.3 Refactoring

Over the course of the project, much of the code needed refactoring. Much redundant code was cut. Since the naming scheme for most functions, variables, and other modules was inconsistent and prone to cause programmer confusion and frustration, functions were renamed to verbal phrases instead of nouns, as is convention. Then each HTML file was renamed to the same name of its corresponding function for consistency's sake. Overall, this led to a consistent naming scheme that reduced programming errors and frustration with using wrong names.

Meanwhile, the URL dispatcher (the handler that calls appropriate server functions in response to requested URLs) was too large and disorganized; it contained every URL pattern, e.g. "/rider/edit" ,"/show/2019-05-21/edit", making it difficult to traverse the file and modify certain URL patterns. Since the URL configuration scheme is hierarchical, the main dispatcher was split up into smaller dispatchers that handled only one type of request, e.g. a dispatcher was formed for requests involving just the Rider object and another was formed for Show objects. The result was an organized, easy to read and modify URL dispatcher file.

5 RESULTS

The password-protected web application uses an SQLite database for all show information, e.g. division, class, horse, rider, combination, billing, and so on. Through the app, the

user can create and manipulate shows, divisions, classes, riders, horses, horse-rider combinations, and keep records such as the bills of riders, class rankings, and other important horse-show related information.

Our app solved many of the problems of the current paper-based system. Its authentication system prevents unauthorized people from manipulating show information and accessing sensitive data such as street addresses, emails, phone numbers, and minors' information. It removes the tediousness of separately typing up or writing down information for the annual report, labels, the 4-H county form, and the results grid. The paper results grid shows the number of points that a horse-rider combination earned for each class, with the combinations as row headers and the classes as column headers.

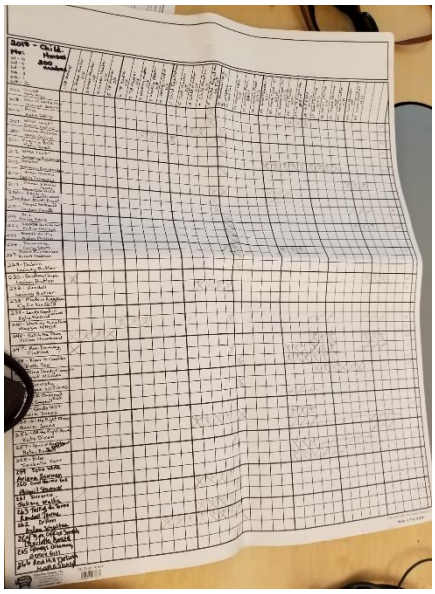

Figure 1: **The image above shows how show data used to be manually entered and organized on a large, complex grid with limited space.**

This grid was a straightforward tool to record results but is highly prone to human error. Our system reduces human error by replacing this grid with a proper ranking form for each class with fields for horse-rider combination numbers. This approach prevents the client from being overloaded with information. The client can enter in a combination number for champion and reserve champion for a division based of the scoring calculation completed by the application, which removes the risk of incorrect calculations. Our application also computes the billing for each combination instead of letting the clients do it by hand.

Furthermore, all web pages that prompted the user to enter information to create a show, division, class, rider, horse, or combination numbers have input validations. The input validations guarantee that the correct data types are entered for each field and that required fields are filled in. This prevents the client from making errors at the earliest stage, by presenting warning messages to the user. The system allows users to backup and load databases, so users can save copies of most up-to-data database elsewhere, such as in a hard disk or flash drive, and easily restore information. This resolved the problem of losing paperwork in a paper based system.

4H horse show administrators - both as the clients and as the only stakeholders - used the system to organize a show by setting up and presenting the show in terms of its divisions and respective classes. The users navigated the web application to view and input various information. Upon clicking a 'submit' button, if an error was found, the user would be notified with an error message instantly (within 1 second). The user can add new riders and horses to the database. The user can assign a combination number to a rider and horse in 9 clicks. The user can generate labels for all the horse-rider combinations in a show in five seconds. The user can register a horse-rider combination to a class in less than a second. Bills for riders are automatically calculated depending on whether they pre-registered or registered the day of the show. The user can access the billing page for each horse-rider combination in 2 clicks from the show home page. The user can rank riders in class. The user can input the final champion and reserve champion for each division referencing provided scoring. The user can obtain a filled out annual VHSA pdf file based off information available in the database within 10 seconds. The user can obtain a complete excel file for the 4-H county within 5 seconds.

6 CONCLUSION

Previously, there has not been much software involved in the area of conducting horse shows due to limited internet access. However, through using Python 3.7.x and the Django 2.1.x framework, we designed a system that fulfilled the needs of the Hoof-N-Woof 4-H Club administrators. The application is fully capable of designing and running a horse show. Administrators can easily create a show, add participating riders and horses, and calculate show results and riders' bills such that mistakes are less likely to occur. Error messages that appear in the application serve as timely alerts to the user that unintentional or invalid operations are being performed.

In order to accommodate the client's request to use the web application locally, the Windows desktop application provides an easy way to locally run the server as well as extra features to load and backup the database file. This application also provides an extra layer of protection through security and backup features such as login authentication and database file backups. The system also significantly reduces the amount of time that is required to manage a horse show with features such as the automatic PDF generation of

scores, automatic generation of rider labels, billing calculations, and the ability to quickly select riders and horses that are present in the database from previous shows.

This application reduces the workload of setting up a horse show and allows for a more organized and user-friendly experience by eliminating the need for both physical documents as well as manual entries and calculations. This system provides a convenient and time-efficient way for the Hoof-N-Woof 4-H club administrators to run and manage horse shows. We designed this program in order to simplify the work for Hoof 'N' Woof's administrative team by replacing their physical filing system with an online database administration system.

7 FUTURE WORK

Future work would most likely involve extending the scoring capabilities, allowing participants in a show to register online, hosting other types of shows on the service, and generating the official show pamphlet. The scoring currently includes ranking classes and inputting the division champion and the reserve division champion based on the total points accumulated. The scoring calculations to determine the division champion and the reserve division champion for each division is complex and unique to each division, so it would be a great improvement to design a system to handle the various complexities of automatically calculating these final scores so that the client don't have to do so manually. There could also be functionality added to view horse-rider combinations that participated in a show. Additionally, the client could have the ability to see how they did in previous shows in specific classes they competed in by creating a viewing page and search option for that combo.

Currently, instructors or show participants send in their registration forms so that the client can input their information manually for the show. Our system makes it simpler for the client to organize this information by allowing her to input the form's information into our program so that all she has to do is type up their information and add the combo to different classes they want to participate in. One way to improve our system's design would be to allow users to remotely register and add the information required for the show and add classes they want to register for. This way the burden would no longer be placed on the client to type up each participant's information, and instead the participants would be able to confirm that their information was added. This would be more complex to implement due to the need for the service to be accessed remotely by various riders.

Another potential future update is to extend our application to host other types of shows. The 4-H organization holds various different kinds of animal shows, and though our application only allows for data input from horse shows, this can be updated to allow for data input from something such as a dog show, with all the same attributes (registration, billing, scoring, ranking) as the horse show. This would allow our application to be used by the entire Hoof-N-Woof team, not just Bertha and Rebecca.

It would also help our clients to have a way to generate the official show pamphlet. This would be based on the show date created by our clients, and with all the information about 4H and the Hoof-N-Woof program, and any additional information based on the data entered into the system. This would be a good way to keep track of the horse-rider combinations participating in each class and will save time from manually designing a pamphlet for every show.

ACKNOWLEDGMENTS

We would like to thank our client, the Hoof-N-Woof 4-H administration, particularly Bertha Durbin and Rebecca Adams who met with us on a bi-weekly basis to work with us on refining the requirements, clarifying horse show terminologies and rules, testing the application features and providing valuable feedbacks during the design process. Their cooperation and support along the way was crucial to the final outcome of our horse show administration application.

REFERENCES

[1] Anon. Django Documentation. Retrieved March 31, 2019 from: https://www.djangoproject.com/

[2] Anon. Django Creating Users Using User Creation Form. Retrieved March 31, 2019 from https://overiq.com/django-1-10/django-creating-users-using-usercreationform/

[3] Mike. Creating and Manipulating PDFs with Pdfrw. Retrieved March 31, 2019 from https://www.blog.pythonlibrary.org/2018/06/06/creating-and-manipulating-pdfs-with-pdfrw/