Horse Show Administration Program Improvements

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science
University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science, School of Engineering

Jacob Fullerton
Fall, 2019

Technical Project Team Members
Draden Gaffney
Jack Schumann
Andrew Yim
Alvin Yuan

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Signature _____ Date _____

Approved _____ Date _____
Dr. Ahmed Ibrahim, Department of Computer Science

Table of Contents

Abstract

In any software engineering project, maintenance is often the most time consuming portion of the software development process. This semester, we primarily performed maintenance on a previous team's Django application. Additionally, our team completed smaller tasks such as writing documentation on branching strategy, JIRA usage, and installation instructions for all products used. Development and maintenance of this software was guided by a specific customer from the Charlottesville area. This customer runs a horse show for which the software was originally built for, but the final product did not meet all of their needs. At the beginning of the semester, there were five specific bugs that were requested to be fixed, but more bugs and potential improvements were recognized throughout the semester. Our team would meet bi-weekly with this customer to showcase our changes and discuss further improvements. Throughout the semester, our team was able to complete all of the bug fixes and improvements that the customer desired.

One of the major takeaways of this product was the importance of following procedures in both what to work on and how it was added to the master branch of the GitHub repository. As our team consisted of five members, we needed to utilize extra tools to ensure that the work was distributed efficiently. We used Jira to track the various features to be implemented or fixed, and we used Git to create a branch for every user story, which was then merged into the master branch using a peer-reviewed pull request. This project improved upon an existing piece of software, but another result of our work is that a future team will be able to utilize our team methods and documentation to perform maintenance on other software products.

1. Introduction

When working as a full time developer, the majority of the workload consists of adding new features and fixing bugs. A recent survey found that developers spend around 30% of their time performing maintenance (Grams. 2019). This project follows the workflow of a team that was assigned an existing, incomplete project. As this application was not initially developed by our team, this brings the challenge of learning the codebase before being able to make meaningful improvements. Additionally, this challenge limits the design of the application as the majority of design decisions have already been made.

For our technical project, our team took a Django application developed in a past capstone project to track data in horse shows and implemented improvements and bugfixes. This application allowed the user to enter riders, horses, and combinations of horses and riders to track the points of each competitor throughout a horse show. Then, when the show is over, the application calculates the winners of each category throughout the entire show. The desired changes were elicited through the customer's original request and through bi-weekly customer meetings.

1.1 Problem Statement

The customers for this project are the organizers of the Hoof-n-Woof horse show. This show is comprised of many different events and throughout each competition there must be a system in place to track all entrants and their scores. The original system that was used by the organizers was a poster board chart where all riders, horses and combos would be written on the border and their scores would be recorded.

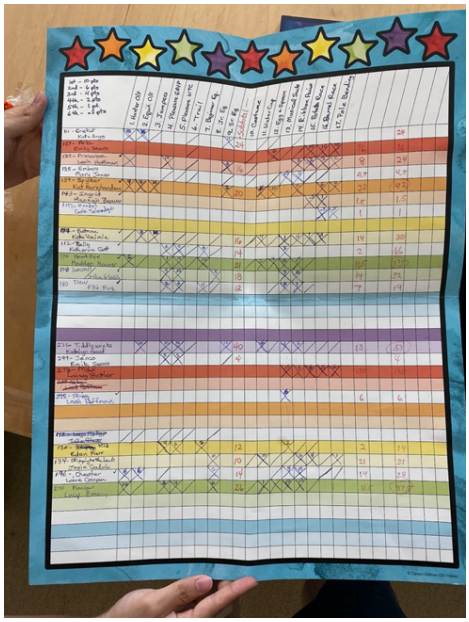


Figure 1: Previous system of scorekeeping for Hoof-n-Woof

This system was not ideal as scores has to be stored physically in one place, and if mistakes were made, it would be very messy to correct them. Adding up all scores had to be done manually, which is time consuming and prone to error. Additionally, in order to find a specific result for a past show the organizers would have to find the specific board for that show. Instead, a web app would allow them to quickly navigate to any previous show to check scores. Previous students working on this project stated that high turnover in event management also led them to pursue an online approach to score tracking (Darroch et al. 2019). In a previous year, students developed a Django application that would allow the organizers to track all of the information from the chart above. Unfortunately, there were bugs in this implementation that prevented the application from being used to its full potential. These bugs included things such as a search function not working, the application crashing if a user is already logged in when visiting, and tables not being able to be properly sorted. Our goal for the semester was to fix these bugs and work with the customer to find additional improvements that would help them optimize the application.

1.2 Contributions

As our team was working on improving an existing application, the changes that our team implemented mainly focused on the application's efficiency and user experience. For example, in the original application, the table's data on entrants could not be sorted, so we implemented a new table that could be sorted by any column. Additionally, there were many other fixes such as being able to create new riders and horses when making a combo, whereas in the original version, the user would have to navigate to a new page to do this. There were also various bugs in the application that were fixed, such as visiting the site while already logged in would return an error and the search function not working properly. Since the users of the application are not very tech savvy, we also installed the new software on their computer using GitHub and updated a local program so that the software could be updated without meeting in person.

2. Related Work

Few systems exist that work for horse show administration, especially given the use case for the customers. Horse shows are already a niche field, and much of the software that does exist only allows for online data entry. The Hoof-n-Woof horse shows are run completely offline, so any system that requires internet usage cannot be used. Some systems are also too bloated in their feature list, adding many unnecessary and confusing features to the system since they are generalized applications. The goal for this project was to create a very simple and easy to use system that worked specifically for how the customer runs their shows. That way the system can be clear and easy to navigate, and every feature works according to the customer's needs.

The previous system for running horse shows was with a poster board chart, and all results and combinations were handwritten. This resulted in a number of problems, as it was time inefficient for recording and looking up horse information, and difficult to edit and reuse the information across shows. The customers had to manually calculate scores and costs for every entry, and they couldn't sort any of the horses or riders to easily find them. With the new Django application, horse shows can be better managed by the customers, and many of the steps from before are automated. It is run locally on the client's machine, so it works even without internet access. Horses and riders can be easily added and edited, and all of their information is quickly accessible through searching and sorting. Horse shows and their classes are automatically ranked based on scores, and the costs for all entrants are calculated across shows. Since this software is custom tailored for the customer, they can easily record and track their horse shows much quicker than before.

3. System Design

The high level goal of this system was to create a web-based approach to track all entrants and their scores when competing in different events throughout each horse show. This system was originally developed using Python and Django, which was chosen by the previous team. Our team continued to use Python and Django and built upon the previous team's work. The license for this system is the same one that the previous team used.

3.1 System Requirements

Gathering system requirements allows the developers to know what exactly they will be developing. This ensures that the product created by the developers is what the client is requesting. If done incorrectly, the developers will waste time working on features that are unnecessary and undesired. By communicating thoroughly and frequently with the client, the developers can ensure that the product is being developed correctly.

Minimum Requirements

- As a user, I want horses/riders alphabetically sorted by default instead of by time added.
- As a user, I want to be able to add classes while creating or editing combos.
- As a user, I want to be able to search for riders/horses while creating a new combo.
- As a user, I want valid combo numbers to be all numbers from 0-999.
- As a user, I would like to be able to create horses/riders while creating combos without losing entered combo data.
- As a user, I would like to be able to enter accession numbers for horses that contain hyphens.
- As a user, I would like to be able to search for horses by any part of their name.

Desired Requirements

- As a user, I would like to see the horse, rider, and owner after a combo number is ranked on the class result page.
- As a user, I should be able to access class pages by clicking on the table entrees on the division scores page.
- As a user, I want to be able to sort tables by each column by clicking on the column title.
- As a user, I would like to add an existing combination number to a class by entering the combination number on the rank class page.
- As a user, I would like to restrict who can sign up for the website.
- As a user, I would like to label a class with numbers plus an optional letter (e.g. class 123 or 123A).
- As a user, I would like to calculate first and second total points across all classes.

Optional Requirements

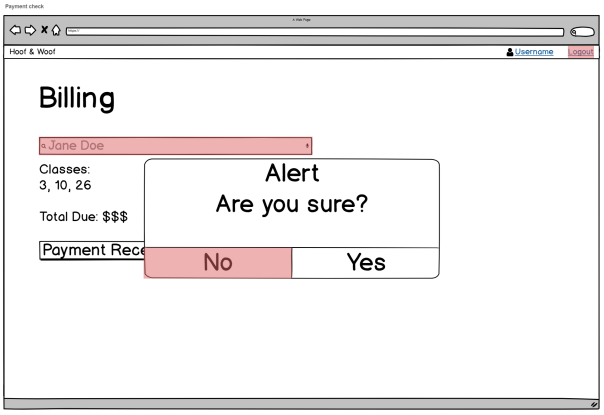
- As a user, I would like a report detailing rankings/results for the entire show by class with class number, class name, ranking combo #, rider name, horse name, and owner.
- As a user, I would like to see a notification or alert when I add or remove items.
- As a user, I would like the messages alerting me of a new/updated horse, rider, or combo to include the name of the horse, rider or combo(id).
- As a user I would like a confirmation before a rider or horse is deleted.

3.2 Wireframes

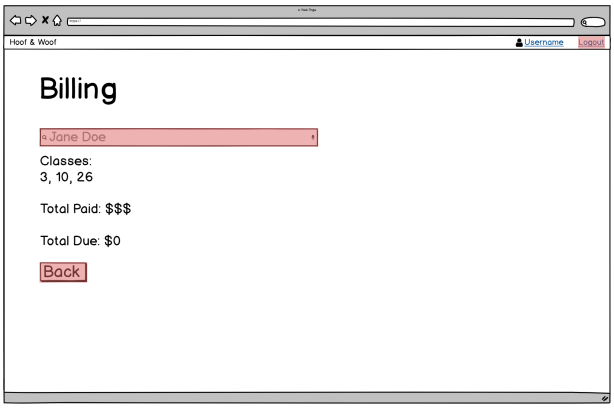
Wireframes are used early in the development process to outline the basic structure of pages of the web application. They allow the developers to hash out the necessary pages and their

corresponding purposes without having to worry about design elements. The developers can also walk through the wireframes to ensure that they all make sense and work together cohesively. Our team did not develop this system from scratch, so we did not have to make our own wireframes; however, we included the previous team's wireframes below.

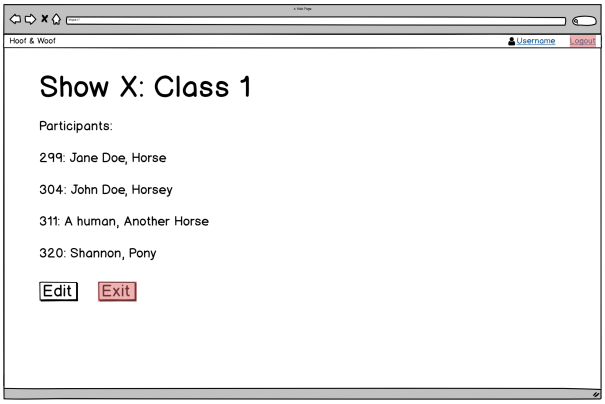
Payment check



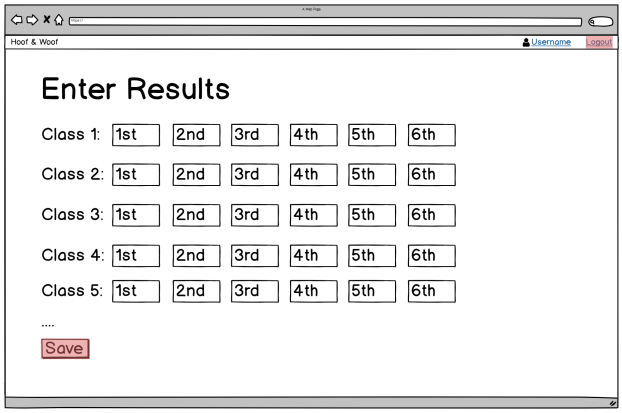
Bill Paid



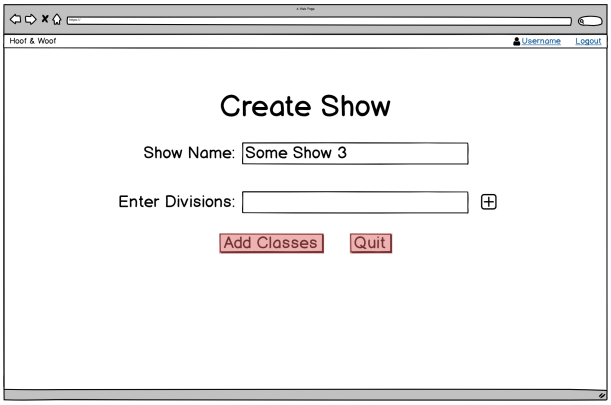
Class Info



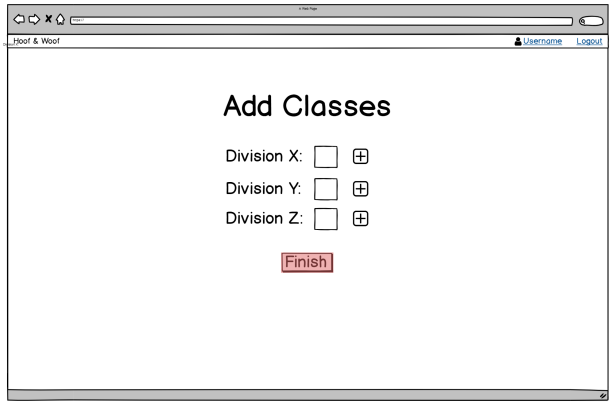
Enter Results



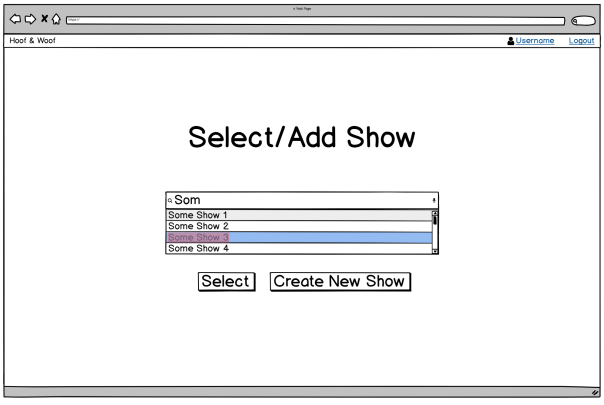
Create New Show



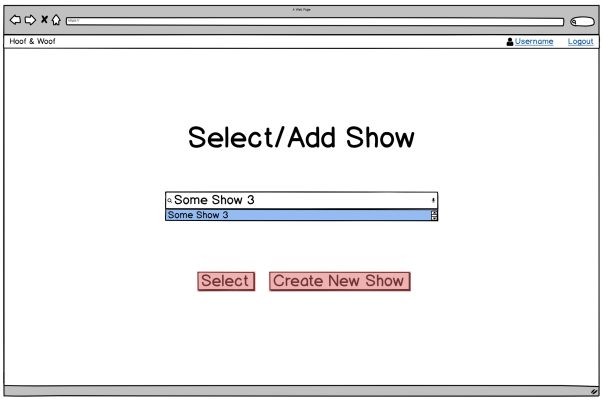
Add Classes to Show

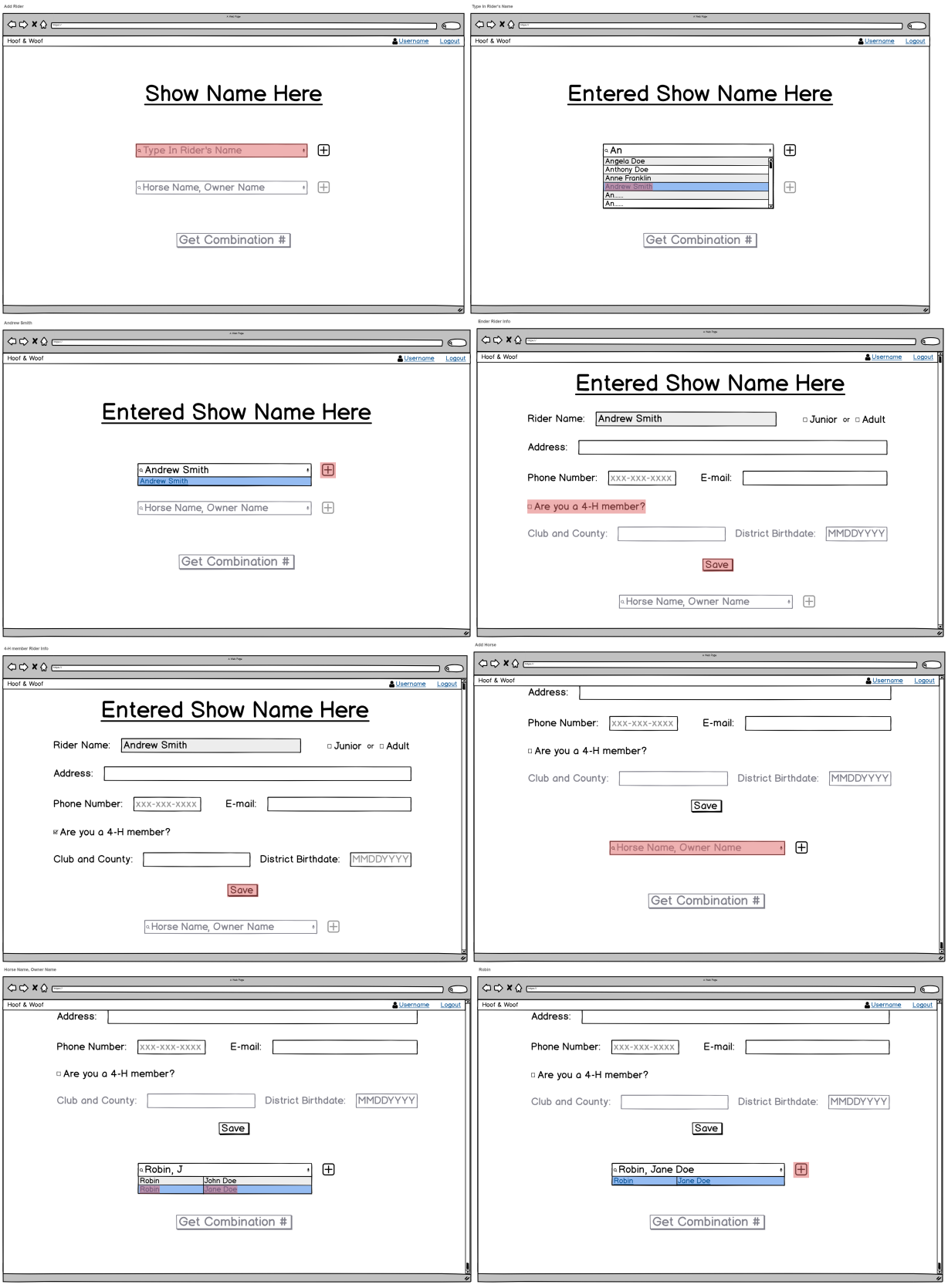


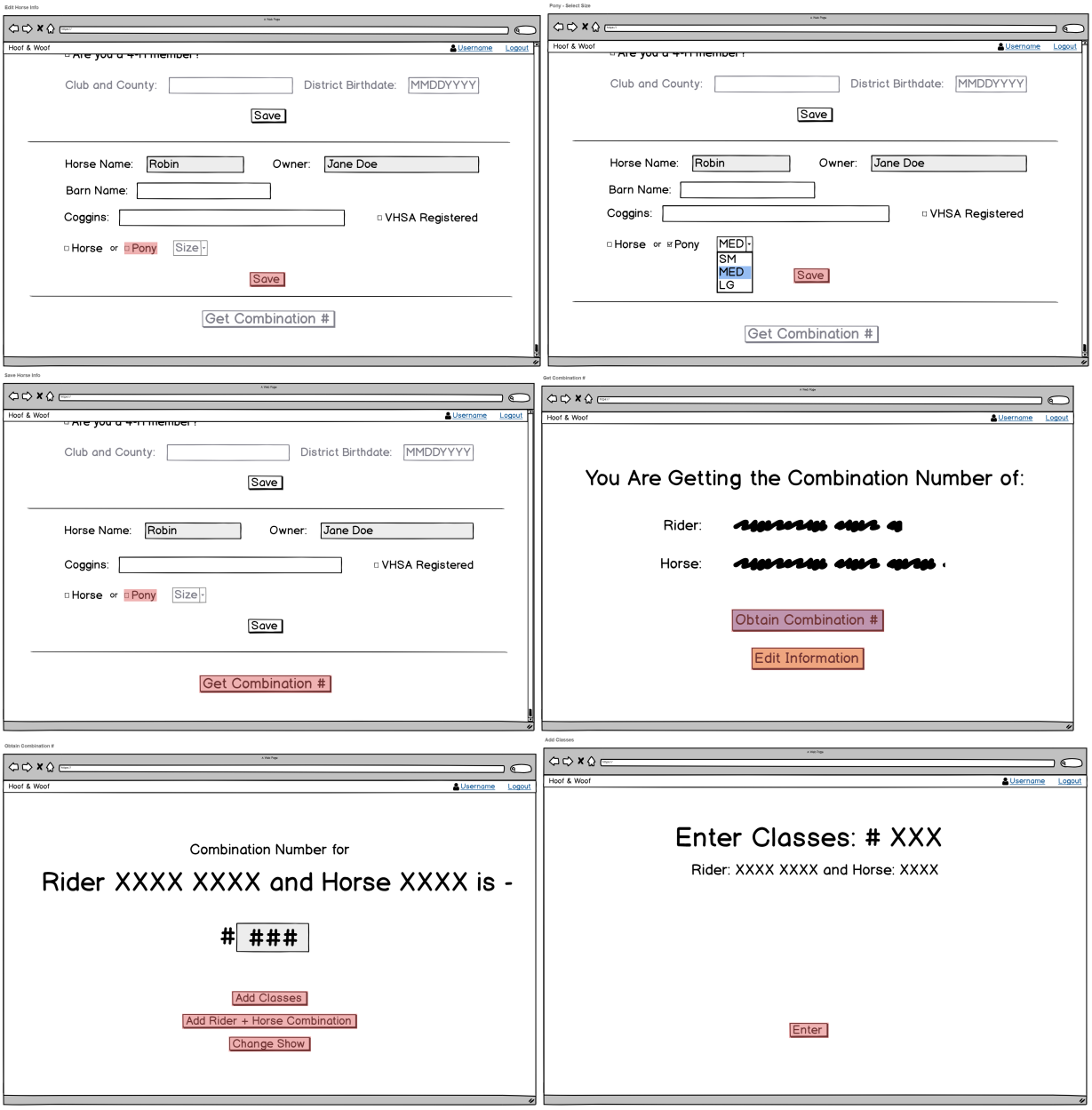
Type Show Name Here



Choose to Auto-fill




Add Rider
Hoof & Woof
Username
Logout
Show Name Here
Type In Rider's Name
Horse Name, Owner Name
Get Combination #
Type In Rider's Name
Hoof & Woof
Username
Logout
Entered Show Name Here
An
Angela Doe
Anthony Doe
Anne Franklin
Andrew Smith
An.....
An.....
Get Combination #
Andrew Smith
Hoof & Woof
Username
Logout
Entered Show Name Here
Andrew Smith
Andrew Smith
Horse Name, Owner Name
Get Combination #
Rider Info
Hoof & Woof
Username
Logout
Entered Show Name Here
Rider Name:
Andrew Smith
Junior or Adult
Address:
Phone Number:
xxx-xxx-xxxx
E-mail:
Are you a 4-H member?
Club and County:
District Birthdate:
MMDDYYYY
Save
Horse Name, Owner Name
4-H member Rider Info
Hoof & Woof
Username
Logout
Entered Show Name Here
Rider Name:
Andrew Smith
Junior or Adult
Address:
Phone Number:
xxx-xxx-xxxx
E-mail:
Are you a 4-H member?
Club and County:
District Birthdate:
MMDDYYYY
Save
Horse Name, Owner Name
Add Horse
Hoof & Woof
Username
Logout
Address:
Phone Number:
xxx-xxx-xxxx
E-mail:
Are you a 4-H member?
Club and County:
District Birthdate:
MMDDYYYY
Save
Horse Name, Owner Name
Get Combination #
Horse Name, Owner Name
Hoof & Woof
Username
Logout
Address:
Phone Number:
xxx-xxx-xxxx
E-mail:
Are you a 4-H member?
Club and County:
District Birthdate:
MMDDYYYY
Save
Robin, J
Robin
John Doe
Robin
Jane Doe
Get Combination #
Robin
Hoof & Woof
Username
Logout
Address:
Phone Number:
xxx-xxx-xxxx
E-mail:
Are you a 4-H member?
Club and County:
District Birthdate:
MMDDYYYY
Save
Robin, Jane Doe
Robin
Jane Doe
Get Combination #



3.3 Sample Code

Models

```python
class Show(models.Model):
    """
    Model for a Show, includes basic information such as name/date/location,
    with prices for day of registration and early registration
    """

    date = models.CharField(primary_key=True, max_length=100)
    name = models.CharField(max_length=100)
    location = models.CharField(max_length=100)
    day_of_price = models.IntegerField(
        blank=True, null=True, default=0, verbose_name="Day-of Price"
    )
    pre_reg_price = models.IntegerField(
        blank=True, null=True, default=0, verbose_name="Preregistration Price"
    )

    def __str__(self):
        return str(self.date)
```

```python
class Division(models.Model):
    """
    Model for a single division. Includes a name, number for the division,
    and a champion and champion reserve for the division as well as the points
    they earned in that division
    """

    class Meta:
        unique_together = ("show", "name")

    id = models.AutoField(primary_key=True)
    first_class_num = models.IntegerField(default=1000)
    name = models.CharField(max_length=100, default="")
    champion = models.IntegerField(default=0)
    champion_pts = models.IntegerField(default=0)
    champion_reserve = models.IntegerField(default=0)
    champion_reserve_pts = models.IntegerField(default=0)
    show = models.ForeignKey(
        Show, on_delete=models.CASCADE, related_name="divisions", null=True
    )

    def __str__(self):
        return f"Show: {self.show.date}, Division: {self.name}"
```

```python
class Horse(models.Model):
    """
    Model for a horse, includes possible sizes of the horse and an option to
    select horse or pony. Coggins date is important for health consideration,
    and the owner is not necessarily the riders
    """

    class Meta:
        unique_together = ("name", "owner")

    alphanumeric_validator = RegexValidator(
        r"^[0-9a-zA-Z-]*$", "Only alphanumeric characters and hyphens are allowed."
    )

    size_choices = (("N/A", "N/A"), ("SM", "SM"), ("MED", "MED"), ("LG", "LG"))
    type_choices = (("Horse", "Horse"), ("Pony", "Pony"))
    name = models.CharField(max_length=200, verbose_name="Name (Barn Name)")
    accession_num = models.CharField(
        max_length=20,
        verbose_name="Accession Number",
        validators=[alphanumeric_validator],
    )
    coggins_date = models.DateField(
        default=datetime.date.today, verbose_name="Coggins Date"
    )
    owner = models.CharField(max_length=200, verbose_name="Owner")
    type = models.CharField(
        max_length=200, choices=type_choices, default="Horse", verbose_name="Type"
    )
    size = models.CharField(
        max_length=200,
        choices=size_choices,
        default="N/A",
        verbose_name="Size (if pony)",
    )
```

Views

```
def view_show(request, show_date):
    """ used as the home page for a selected show """
    show = Show.objects.get(date=show_date)
    request.session["show_date"] = show_date
    context = {
        "request": request,
        "show_name": show.name,
        "date": show_date,
        "date_obj": datetime.datetime.strptime(show_date, "%Y-%m-%d"),
        "location": show.location,
        "divisions": show.divisions.all().order_by("first_class_num"),
        "show": show,
    }
    return render(request, "view_show.html", context)
```

```
def sign_up(request):
    """ creates a new user account """
    if request.method == "POST":
        form = UserCreationForm(request.POST)

        if form.is_valid():
            form.save()
            username = form.cleaned_data.get("username")
            raw_password = form.cleaned_data.get("password1")
            user = authenticate(username=username, password=raw_password)
            login(request, user)
            return redirect("select_show")
    else:
        form = UserCreationForm()
    return render(request, "sign_up.html", {"form": form})
```

```
def view_division_classes(request, show_date, division_id):
    """ lists the classes in a division """
    show = Show.objects.get(date=show_date)
    division = show.divisions.get(id=division_id)
    context = {"request": request, "classes": division.classes.all(),
               "id": division_id}
    # passes the division's name/classes to "division_classes.html" to render
    return render(request, "view_division_classes.html", context)
```

Forms

```python
class ShowForm(forms.Form):
    """
    Form for creating a Show and saving its information
    """

    name = forms.CharField(
        max_length=100, widget=forms.TextInput(attrs={"autocomplete": "off"})
    )
    date = forms.DateField(widget=forms.SelectDateWidget(), initial=timezone.now())
    location = forms.CharField(
        max_length=100, widget=forms.TextInput(attrs={"autocomplete": "off"})
    )
    day_of_price = forms.IntegerField(label="Day-of Price")
    pre_reg_price = forms.IntegerField(label="Preregistration Price")
```

```python
class RiderForm(forms.ModelForm):
    """
    Form for an individual rider, including name, address, email, and birth date.
    Birth date is only necessary for people who are 18 or younger.
    """

    year_range = list(reversed(range(1920, datetime.date.today().year + 1)))

    birth_date = forms.DateField(
        help_text="Only enter if you are 18 or younger",
        widget=forms.SelectDateWidget(years=year_range),
    )

    state = USStateField(widget=USStateSelect(), initial="VA", required=False)

    zip_code = USZipCodeField(required=False)

    class Meta:

        model = Rider
        fields = "__all__"
        exclude = ["horses"]

    def clean(self):
        cleaned_data = super().clean()
        if cleaned_data["member_4H"] and not cleaned_data["county"]:
            raise ValidationError(
                "You must specify a county if the rider is a member of 4H."
            )

        return cleaned_data
```

```python
class HorseForm(forms.ModelForm):
    """
    Form for a horse including its name, owner, type, size, accession number,
    and coggins date
    """

    year_range = list(reversed(range(1920, datetime.date.today().year + 1)))

    coggins_date = forms.DateField(widget=forms.SelectDateWidget(years=year_range))

    accession_num = forms.CharField(
        widget=forms.TextInput(attrs={"autocomplete": "off"})
    )

    class Meta:
        model = Horse
        fields = "__all__"

    def clean(self):
        cleaned_data = super().clean()
        horse_type = cleaned_data["type"]
        size = cleaned_data["size"]
        if horse_type == "Pony" and size == "N/A":
            raise ValidationError("A pony must have its size specified.")
        return cleaned_data
```

3.4 Sample Tests

Testing is important because it verifies that the code works as intended. It allows the developer to move onto the next task with confidence, knowing that the code just written is correct. If a bug is not caught early on, it could cause significant problems down the road and waste a lot of time. Tests will also immediately catch new changes that break previous features.

```python
class Add_Combo_Classes(TestCase):
    def add_classes_to_combo(self):
        horse1 = Horse.objects.create(
            name="Lollipop",
            coggins_date="2011-10-11",
            accession_num=48,
            owner="John",
            size="medium",
            type="horse",
        )
        rider1 = Rider.objects.create(
            name="Bob",
            address="555 ct",
            birth_date="1990-09-25",
            email="55@s.edu",
            member_VHSA=True,
            county="fairfax",
        )
        c1 = Class.objects.create(name="Test", num="1")
        combo = HorseRiderCombo.objects.create(num=555, rider=rider1, horse=horse1)
        class_participation = ClassParticipation(participated_class=c1, combo=combo)
        self.assertEqual(c1, class_participation.participated_class)
```

This test is responsible for testing the ClassParticipation and HorseRiderCombo models. It creates a ClassParticipation object using the Class and HorseRiderCombo objects. Then, it verifies that the ClassParticipation's class field is accurate using an assert statement.

3.5 Code Coverage

The code coverage package used for this project is called Coverage, which uses tools provided in the Python standard library to track code coverage.

Setting up Coverage

1. Install Coverage using https://pypi.org/project/coverage/ or `pip install coverage`
2. Create a file called `.coveragerc` in the same directory as your manage.py file. There are a lot of different options in this optional configuration file for running Coverage which can be used to personalize Coverage. Add the following into the `.coveragerc` file:

   ```
   [run]
   source=.
   ```

```
[report]
show_missing=True
```

The `source` option is the root directory of the files for Coverage to check, and `show_missing` displays the line numbers of code that has not been covered by tests.

3. Add the following into your `.gitignore`

```
htmlcov/
.coverage*
coverage.xml
```

Running Coverage

1. Run `coverage run manage.py test your-app` to run Coverage on your Django project. This will run the entire test suite with Coverage.
2. Run `coverage report` to view all the files Coverage was run on and the line numbers for code that was not covered. The coverage percentage for each file will be displayed.
3. Run `coverage html` to generate html files in the `htmlcov` directory, which can then be opened to view detailed coverage details in a browser.
4. Run `coverage erase` to remove Coverage data from previous runs. This should ideally be done before every time Coverage is run.

For this project, Coverage reports that 92% of our code is covered by the tests.

3.6 Installation Instructions

The customer is running the system on a laptop running Windows 10, so these instructions are for installation on a Windows machine. This instructions are made to host the software locally since that is the customer's use case. To obtain this software, the user must have a GitHub account with access to the code repository.

There are three main parts to running this software: setting up the computer with required dependencies, cloning the code repository onto the computer, and creating the database/running the server.

Create your Club Windows User Account

> **This step is optional and is only required if your user does not have administrator privileges. If this step is skipped, replace "Club" with your Windows account name in the following steps.**

1. On your machine, go to the search tab on the start menu, and type in "Settings"
2. Click on "Account"
3. Under "Family & Other People", click "Add Someone Else to this PC"
4. Click "I don't have this person's sign-in information"
5. Click "Add a user without a Microsoft account"
6. Create the new account while making sure that the username is "Club"
7. Click on the Club account from the "Family & other people" screen and change the account type to Administrator
8. Log into the new "Club" account

Set up the computer with the required dependencies (NOTE: This step requires internet connection)

1. Manually install dependencies
    - Download Python from here: https://www.python.org/downloads/
- On the main page, under "Download the latest version for Windows", click "Download Python 3.7.2", open the file once it's fully downloaded, and follow the instructions by accepting the default options and proceeding through the screens to install
    - Download Git from here: https://git-scm.com/downloads

- Click the Windows button and once the package is fully downloaded, open it and follow the instructions by accepting the default options and proceeding through the screens to install

2. Add the dependencies to the environment variables (python, git)
    - Go to the search tab on the start menu and then type "Edit the system environment variables"
    - Click "Environment Variables"
    - Click on the "Path" row under user variables and select "Edit"
    - Click "New" and type the path name where Python is installed on your computer to add it. It can usually be found at the following path: "C:\Users\Club\AppData\Local\Programs\Python\Python37-32"
    - Click "New" and type the path name where Python Scripts is installed on your computer to add it. It can usually be found at the following path:"C:\Users\Club\AppData\Local\Programs\Python\Python37-32\Scripts"
    - Click "New" and type the path name where the Git Bin is installed on your computer to add it. It can usually be found at the following path: "C:\Program Files\Git\bin"
    - Click "New" and type the path name where the Git Cmd is installed on your computer to add it. It can usually be found at the following path: "C:\Program Files\Git\cmd"
    - Then press "OK" to save

Clone the repo onto the computer (NOTE: This step requires internet connection)

1. Press the Windows Start. Type "Command Prompt" and then click on it.

2. On the command line, type `cd C:\Users\Club\Documents` and run `git clone https://github.com/uva-cp-1920/horse_show.git` to pull the code
    - If a folder does not exist, cd into the existing folders and type "mkdir " + folder name or use the "File Explorer" to manually create the new folder
3. Type `dir` into the command line, and if you see a folder called "horse_show", cloning the repo was successful.
4. To install the project dependencies, type `cd horse_show` in the command line to enter the project folder. Then type `pip3 install -r requirements.txt` to install the dependencies.

Create the database

This step is only required the first time the project is installed.

1. Press the Windows Start. Type "Command Prompt" and then click on it.
2. On the command line, type `cd C:\Users\Club\Documents\horse_show\src\newenv\horseshow-proj`
3. Type `python manage.py makemigrations show`
4. Type `python manage.py migrate`

Run the server

1. Press the Windows Start. Type "Command Prompt" and then click on it.
2. On the command line, type `cd C:\Users\Club\Documents\horse_show\src\newenv\horseshow-proj`
3. Type `python manage.py runserver`

4. The server is now running locally. To access it, go to localhost:8000/show/login within your browser. To stop the server, close the command prompt window from step 1.

4. Results

The goal of the system was to act as an offline registration platform for the client's horse show competitions. The clients operated the system while the other stakeholders, competitors, simply conveyed information to the clients behind the register. As a result of the updated design the system was able to address all features requested by the client. This included an improved search which filters across multiple fields and various workflow improvements. For example, the central flow of adding a horse-rider combination was reduced to 7 clicks from the previous 9 clicks. This flow was also modified to handle previously registered horses and riders, reducing the required clicks to 3. Another example was the class ranking flow, where each competitor is assigned a rank, was updated to handle the case where a competitor was accidentally not registered to the class it was being ranked in. In the previous system, this error would require 4 more clicks; however, the updated system can handle this error in one click.

This updated system allowed the horse show administration system to be used effectively in a real horse show for the first time. The client tried the software in a smaller show on October 27th with approximately 20 horse-rider combinations. The system is used to register competitors to the horse show, calculate billing, and track the score of contestants. Compared to the previous version of the software, the clients estimated that they were 3-4 times more efficient during registration. The clients also called out an improvement in the system which added additional information to the results page. The page now includes the ranks of riders and horses by name rather than simply the three-digit combination number. The clients noted that these improvements significantly improved the quality of result ceremonies as having Horse and Rider names "made the award announcement seem more polished". The clients' primary show is in

Spring 2020 with over 60 horse-rider combinations and as a result of the trial run the clients plan to use the new system in the Spring show.

Throughout the update of the administration website many bugs were unearthed and patched. This included an edge-case where if the same combination id was registered to multiple shows, the ranking system could incorrectly include a contestant from a previous show in the current show. Bugs like these were revealed by maintaining 100% code-coverage in the team's test cases and demoing the products bi-weekly with the clients and at the client' smaller horse show. As a result the clients reported an improved feeling of polish and found the software to be much more stable than the previous version.

5. Conclusions

We accomplished the needs of our client by fixing the bugs of the code stack left behind by the previous team, and we added newly implemented features that the client requested. We were able to add new features while not introducing any new bugs due to our 100% code coverage. Meeting with the clients every other week allowed for us to update them with our progress and communicate effectively the project's roadmap. Since we finished the needs of the client early, we suggested new ideas to add that the client might like. With the finished project, the client will no longer have to rely on pen and paper or a buggy platform to keep track of horse shows. Now the client can use our system in a more efficient manner since we reduced the number of clicks required for functions, implemented easier search, and allowed sorting and ranking of horses, riders, and combinations. During this process, we learned how to function and communicate as a team. Each member focused on improving different aspects of the project, and weekly meetings allowed us to bring our ideas together to create one cohesive product. Our clients can now use the software without any issues, and the documentation we've created allows future teams to build off of our methodologies in their projects. As a result, our clients are excited to be using the software for the first time in a full-scale horse show in Spring 2020.

6. Future Work

Even after improving upon the original software, there are still some changes that could greatly benefit horse show administration in the future. One of the features present in the original project was a function that automatically filled out a Virginia Horse Shows Association (VHSA) PDF with show results, ranking horses along with their owners and riders. This feature no longer works because the VHSA has changed their horse show system from the time that this feature was implemented, making it unusable for current shows. As a future goal, the original functions to generate this PDF can be modified to fit the current system and make it functional again. Further research should be done on which aspects of horse show administration have changed since this feature was originally made; more information on the rules and regulations for these shows can be found on the VHSA handbook on their website. With this additional information, the PDF generating function can be fixed and used in future shows.

7. References

[1] Grams, C. (2019, March 14). Developers spend 30% of their time on code maintenance: our latest survey results https://blog.tidelift.com/developers-spend-30-of-their-time-on-code-maintenance-our-latest-survey-results-part-3.

[2] Darroch, Mallapragada, Nathan, Pappagallo, Prahlad, Saharya, Wu (2019) Horse Show Administration Program Technical Thesis