List Eaters

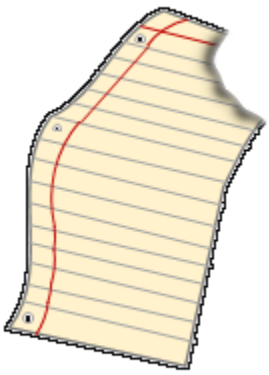
Final Report

Authors:

Baldwin A. Bakkal, Kareem A. Hirani, Reid G. Neason, and Jon R. Waterman

Final Project Demo Date:

Tuesday, May 10, 2022



CSCE-431-550
Department of Computer Science & Engineering
Texas A&M University

Contents

1 Getting Started

1.1 Configuring the Development Environment

The given configuration set-up is designed with Linux Ubuntu V.20.04 in mind. If a different operating system is to be used, the following steps may not suffice. Keep that in mind before continuing without Ubuntu V.20.04.

1.1.1 Step 1 - Install Rbenv and its Dependencies

Ruby is dependent upon several packages that can be installed through your package manager. Once these are setup, you can install *Rbenv* which will allow you to install versions of Ruby and will manage those versions.

First, update your system's package list:

```
sudo apt update
```

Now, install the dependencies required for Ruby:

```
sudo apt install git curl libssl-dev libreadline-dev zlib1g-dev
autoconf bison build-essential libyaml-dev libreadline-dev
libncurses5-dev libffi-dev libgdbm-dev
```

After installing the dependencies, install *rbenv*. Use `curl` to transfer the contents of the GitHub repository into the `~/.rbenv` directory:

```
curl -fsSL https://github.com/rbenv/rbenv-installer/raw/HEAD/bin/rbenv-installer | bash
```

Add `~/.rbenv/bin` to your `$PATH` to allow `rbenv` to be used in the command line. This can be done by writing to the `~/.bashrc` file:

```
echo 'export PATH="$HOME/.rbenv/bin:$PATH"' >> ~/.bashrc
```

Add the following line to the `~/.bashrc` file to load `rbenv` each time the terminal starts-up:

```
echo 'eval "$(rbenv init -)"' >> ~/.bashrc
```

Apply the changes made to the file above in the current shell session:

```
source ~/.bashrc
```

Assuming `rbenv` was properly set-up, install the `ruby-build` plug-in to allow use of the `rbenv install` command:

```
git clone https://github.com/rbenv/ruby-build.git
```

After cloning the repository, there will be a directory called `ruby-build` in the working directory. Within this directory is a script named `install.sh` that will install the ruby-build plug-in once executed. To complete setting up rbenv and prepare you to install the needed ruby versions, execute the following command:

```
PREFIX=/usr/local sudo ./ruby-build/install.sh
```

Now, rbenv and ruby-build are both installed. Now you can move on to installing ruby versions.

1.1.2 Step 2 - Installing Ruby

The version of Ruby needed for *List Eaters* is v3.0.0:

```
rbenv install 3.0.0
```

Once the install is complete, set the current version of Ruby being used by:

```
rbenv global 3.0.0
```

Verify that we installed the correct version.

```
ruby -v
```

If Ruby v.3.0.0 was installed and set correctly, then you are ready to install Bundler, which will be used to install Rails and the other gems specified in the `Gemfile`.

1.1.3 Step 3 - Installing Gems and Using Bundler

Bundler is a tool that manages gem dependencies for projects. Install the bundler V.2.3.6 gem:

```
gem install bundler:2.3.6
```

1.1.4 Step 4 - Installing NodeJS Using the Node Version Manager (NVM)

Get raw contents from GitHub repository:

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.0/install.sh | bash
```

Next, execute the commands just stored from the curl:

```
source ~/.bashrc
```

Now, NVM is installed on the system. Next, we will continue on to install NPM to manage different nodeJS versions:

```
nvm install node
```

The above command installs the latest stable nodeJS version.

1.1.5 Step 5 - Installing Yarn for Webpacker

Now that we have nodeJS installed, we must install Yarn to be used for Webpacker:

```
npm install --global yarn
```

Next, we will clone the List Eater application repository and finish installing the remaining dependencies.

1.1.6 Step 6 - Installing PostgreSQL

First, update your system's package list:

```
sudo apt update
```

Then, install the postgresql package:

```
sudo apt install postgresql
```

1.1.7 Step 7 - Cloning the List Eaters Repository

Now that we have our Ruby package manager, Bundler, NVM/NPM, Yarn, and PostgreSQL installed, we need to clone the List Eaters repository:

```
git clone https://github.com/Aggie-Pull-Day/Aggie-Pull-Day.git
```

Once the List Eaters repository has been cloned, change to the root directory of the application

```
cd Aggie-Pull-Day/
```

and verify that the already installed files in `node_modules/` did not get removed by executing:

```
yarn install --check-files
```

Then, use the following command to install Rails and the remaining gems needed for the project:

```
bundle install
```

This will create a `Gemfile.lock` file in the app's root directory.

1.1.8 Step 8 - Setting up the Local Development Database

Check the status of the postgreSQL service running on the system:

```
sudo systemctl status postgresql
```

There should be output to the terminal indicating `active(exited)` highlighted in green. Next, we're going to establish a connection with the postgres database and log into the postgres account using:

```
sudo -i -u postgres
```

From here, open a `psql` prompt using:

```
psql
```

Now, you should be connected to PostgresSQL. To log into the development database `aggiepullday_development`, we need to create the role `aggiepullday` as a superuser by typing the following queries into the `psql` prompt:

```
CREATE ROLE aggiepullday WITH LOGIN PASSWORD 'GigEm2022';
ALTER USER aggiepullday SUPERUSER;
```

Once the `aggiepullday` account has been created we need to ensure that the *local* Unix Domain Socket connection method is changed from *PEER* to *MD5*. If it's already set to MD5, then no change to the `pg_hba.conf` file is needed:

```
sudo nano /etc/postgresql/12/main/pg_hba.conf
```

Once the file has been updated, or even if it hasn't, we want to restart the PostgreSQL service:

```
sudo service postgresql restart
```

Now, while still in the applications directory, run the commands:

```
rake db:create
rake db:migrate
```

This will create the `aggiepullday_development` database and run the migrations to configure its schema. Next, run

```
rake db:seed
```

to seed the database with initial data. Finally, the database should be set-up and running. To verify the above steps in creating the database were done correctly, log into the `aggiepullday_development` database with the `aggiepullday` user and `GigEm2022` password to verify the connection and database tables:

```
sudo psql -d aggiepullday_development -U aggiepullday
```

The above command will pull up the `aggiepullday_development` prompt for the postgresql database. Here, type the following into the prompt:

```
\conninfo
```

If the output is similar to the following,

> You are connected to database "aggiepullday_development" as user "aggiepullday" via socket in "/var/run/postgresql" at port "5432".

then you were successful in connecting to the development database. You can further verify the database tables by executing `\dt` in the `aggiepullday_development` prompt.

1.1.9 Step 9 - Running the List Eaters App Locally

Now that we have completed setting up the application and its development environment, List Eaters can be run locally by executing:

```
rails server
```

1.2 Production Deployment

1.2.1 Step 1 - Pushing to Heroku

Given that this is an ongoing product, and the production deployment is live, there is no need to create a new Heroku deployment. Therefore, any changes to the production branch just needs to have its changes pushed to Heroku:

```
git push heroku main
```

1.2.2 Step 2 - Set-Up the Database for Heroku Between Changes

To update the production database between changes on Heroku, execute the following commands. This should be done each time the production branch is updated.

```
heroku pg:reset DATABASE
heroku rake db:schema:load
heroku rake db:seed
```

1.2.3 Step 3 - Using Heroku Scheduler

For setting up Heroku Scheduler and to run the `app/lib/tasks/scheduler.rake` file for every Sunday at 12:00am, you must first install the Scheduler add-on either by using the CLI or GUI. We choose to use the CLI:

```
heroku addons:create scheduler:standard
```

Now that Heroku Scheduler is installed, test the rake task `weekly_clear_groups`:

Given how the one-off dynos work, if it doesn't have issues with `heroku run`, it will work with Heroku scheduler. Now to configure the schedule for tasks, open the scheduler dashboard using the CLI:

```
heroku addons:open scheduler
```

On the Scheduler Dashboard, click "Add Job...", enter a task `rake weekly_clear_groups`, select a frequency (daily and 00:00), dyno size (free), and the next run time.

NOTE: The next run time for daily jobs is in UTC. If you want to schedule the job at a certain local time, add the proper UTC offset. Lastly, for inspecting output logs for scheduled jobs go into your `logs` as process `scheduler.X`.

2 Introduction

2.1 Project Introduction

The current state of Aggie football ticket-pull is riddled with inefficiencies and bottle-necks. First, a student must present and scan their sports pass (and any additional guest ticket payment) by a Kyle Field box office clerk to be claimed for that week's game. This transaction grows increasingly troublesome as the size of a group grows; for which, then each group member's pass must be scanned to pull a ticket. With this in mind, student ticket pull begins at 08:00am on a Monday and continues throughout the week with over tens of thousands of students preparing to pull tickets. Not only does this waste an unnecessary amount of time for students, but it also wastes employee resources for Texas A&M University and the 12^{th}-Man Foundation.

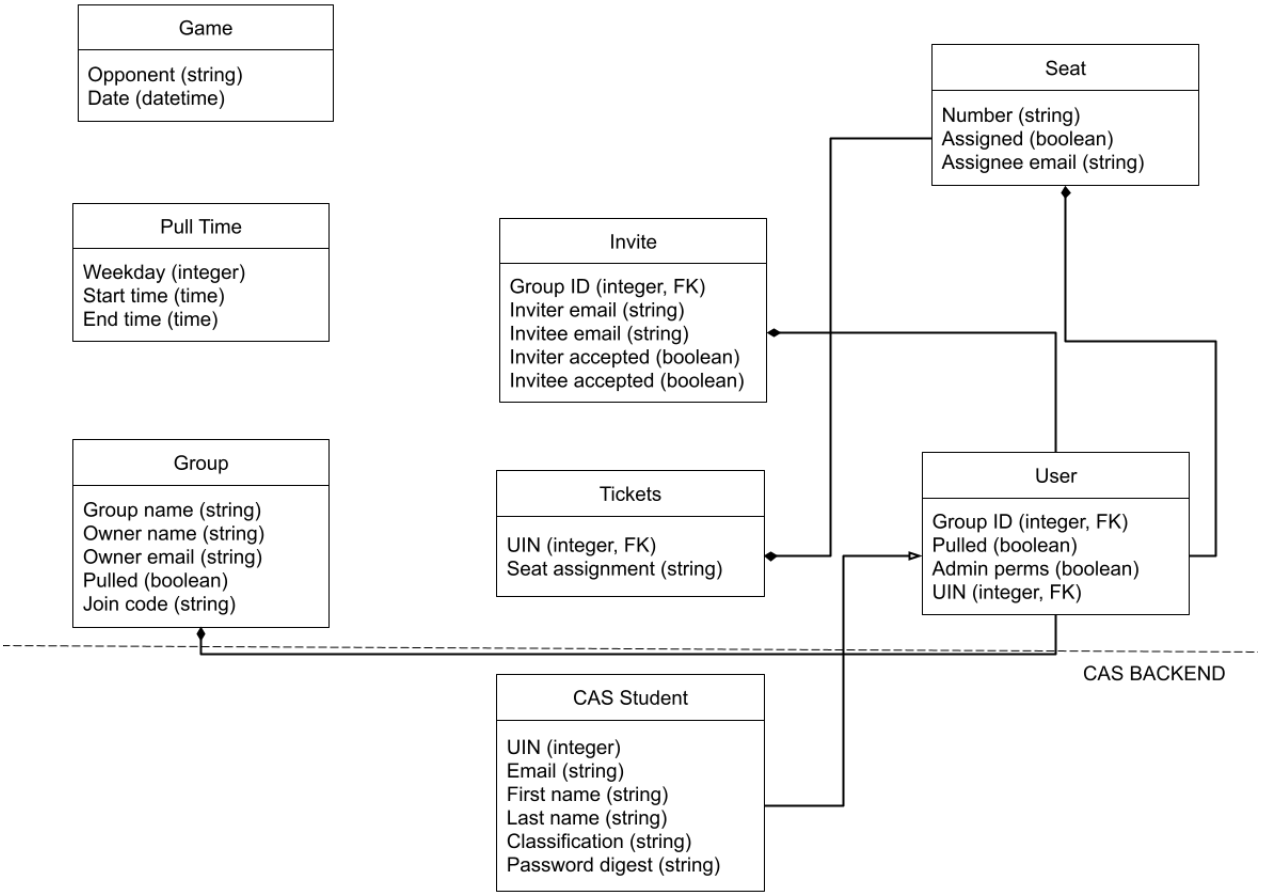
Students and the 12^{th}-Man Foundation are in need of a system to streamline the process of pulling student tickets for football games. This is where our product, List Eaters, comes into action. List Eaters does not eliminate the tradition of camping out for student football tickets. Instead, it provides students with the opportunity to organize their groups ahead of their designated ticket pull day; thus, when it comes time to pull tickets, there are no delays in scanning passes or making payments. The outdated system of human-mechanical processing that reinforces the act of collecting physical sports passes; scanning those passes; and manually determining the ratio of classifications on sports passes is undercut by presenting students and 12^{th}-Man Foundation staff members with a web service that handles the mechanics of ticket pull and management of ticket-pull through digital means.

2.2 Implementation Summary

In our implementation, our sign in system mocks the CAS authentication system. Because of this, students do not need to "Create an Account" as long as they are Texas AM University students. Instead, if it is their first time signing in with their AM credentials, we add them as a new user to our system automatically. After signing in, a student can choose to create a new group or join an existing group. When a student creates a new group, they become the owner of that group and have permission to invite other users, remove members currently in their group, and promote other members to the owner position. When a group member is invited, they are emailed and sent a random code they must enter in the "Join Group" section of their dashboard to be added to that group. A group code will not work for a student if that student has not been invited by the group owner. Once a student joins a group, they are emailed the QR code that can be used to pull their tickets when "pull day" arrives. All students within a group are able to leave that group and join another at any time until tickets are pulled, but only the group owner can invite or remove other members.

At the top of the dashboard page, statistics calculate the relative classification of the group using the classification of each member within it and displays the day that group can pull tickets and the number of days until they can pull. When it is time to pull tickets, any group member can find their group QR code in their email or click the Display Group Pull Code button, which will display the QR code in their browser. That code is then scanned by an administrator account – the 12th Man Foundation. When the 12th Man Foundation scans the code, they are shown a list of raw UINs that represents the members of the group and a "Pull Tickets" button. When the button is pressed, it sends an API call to our mocked Tickets model, which assigns seats within the Tickets database, then responds to our application with a list of UINs that were successfully pulled. From there, our dashboard page updates to say that tickets have been pulled, and removes functionality to leave the group or add any additional users. Once a week on Sunday at 12:01 am, those groups will be cleared from the system to allow new groups to be formed.

2.3 Models Summary



User - The User model contains a number of helper methods as well as the general function to send an email containing the qr code to a user.

Group - This model has the "hasmany" relationship with the User model allowing it to use it for queries. This model contains the method to determine the group's classification and is recalculated every time a member is added or removed. Each group contains a unique code by which users may join.

Student - the student model extrapolates off the User model to provide more information to a user, specifically the uniqueness of their uin so there are no duplicate users. This is modeled after the TAMU CAS System to allow for interchangeability with the software used by the university.

Invite - This models the relationship between invitee and inviter and is used to determine if a user can join a group, which only occurs once the inviter and invitee both accept.

Ticket - this is used in the mockup of the ticket pull system to assign a ticket with a unique seat number to a unique uin. The software mockup is completely independent and unreliant on the model and can be replaced with ease with the university-approved software.

2.4 Account Creation Testing

This codebase is set up to mock the Central Authentication System and the internal Texas A&M database of student information. In our implementation, the User model represents people who have an account in the List Eaters system, and the Student model represents an external service through the university that provides information on every student. Because every user for our application must have a student profile

through the University, a User account for List Eaters can only be created if that user has a matching email within our Student model. Whether or not a student has used our system before, they log in to List Eaters using their Texas A&M email and password. From there, if they are a new user, our system will automatically create a User account for them in our system.

Because the Student model is supposed to represent a third party service like CAS, our current implementation does not allow a new user or admin for List Eaters to create a new Student profile. Anybody can create a User by logging into the system with their same credentials stored in the Student model. So, in order to test the application, review the seed data for the Student model and use any of the user/password combinations to create/utilize a User account.

2.5 Team Roles

- Product Owner: Jon Waterman
- Scrum Master: Kareem Hirani
- Development: Reid Neason
- Development: Baldwin Bakkal

3 User Stories

1. As a user I'd like to have a login page for students to view and access the application so that each user may have their personal information secure.
   (a) Points: 3. This is a pivotal feature of the app, as it is basically rendered useless without defined users.
   (b) This item was implemented by adding a simple "Log In" button to the app's navbar, which would turn into a "Sign Out" button when the user is logged in.
   (c) Originally, in a mock of our system, we had planned to make this feature a dropdown, with options to log in as a student or admin. However, we eventually realized that it would be better served with one login page, with the credentials being used to determine on the backed whether the user was a student or admin. As such, our tests had to be refactored to reflect this.
2. As a user, I would like to receive an email once my group has pulled so that I am aware that we are ready for gameday.
   (a) Points: 1. This is a light task that also isn't necessary for the core functionalities of the application.
   (b) This item was implemented by creating a new Mailer object, which serves the sole purpose of notifying group members when their ticket has been pulled.
   (c) No major changes were made throughout the progress of this user story.
3. As a user I'd like to have systems in place to test the integrity and reliability of the site so that all features may be working as intended.
   (a) Points: 3. TDD and BDD are vital parts of the development process and creating a functional application.
   (b) This item was accomplished via RSpec and Cucumber tests, which are discussed at greater length in other sections of this report.
   (c) Some tests had to be changed throughout the project in order to reflect changes to be made in other user stories.
4. As an Admin I'd like to have a database to store student, football game, and pull date information so that it is easy to display for both admins and users.
   (a) Points: 3. The database is the life force of the application, which would bring down the app if it isn't functioning as intended.
   (b) This item was implemented via Rails models, such as Game and Group.
   (c) There were numerous changes made to the database throughout the development process. The implementation of other stories often required the creation of new models or columns, while old columns would sometimes be rendered useless and get discarded.
5. As a user I'd like to...Have an Admin view: buttons to select, add, and edit the selected season's football games so that they may be changed if necessary.
   (a) Points: 2. While the administrator view may be useful in some scenarios, the app is able to function with the data we provide it when the app deploys.
   (b) This item was created via views through which the administrator can make use of their full CRUD capabilities on all models in the app.
   (c) There were some slight changes made to this app in the development process, mainly revolving around the user interface. We had the advantage of creating CRUD capabilities early on, which could then be seamlessly integrated into an administrator dashboard.
6. As a user I'd like to have a Student ticket pull view: given the ratio of sports passes, display the date when a group can begin pulling so that users may know when their accurate pull date is.

(a) Points: 2. This information may be vital for some scenarios, but they mostly fall outside the scope of the app's use, such as actually showing up for ticket pull.

(b) This item was simply done by pulling data from the various models and controllers and rendering them in the pull group view.

(c) The main changes made to this story were the implementation of links to various functionalities that were implemented in other stories and then placed on the same page.

7. As an admin, I would like the ability to edit the game schedule so that it can be changed if there are delays.

   (a) Points: 2. See User Story 5 for reasoning.

   (b) This item was accomplished by creating functions on the backend which allowed the administrator's CRUD privileges to take effect.

   (c) There were no major changes made to this story.

8. As an admin, I would like the ability to edit the users on the application so that users may be removed or added.

   (a) Points: 1. See User Story 5 for reasoning.

   (b) This item was done by editing the User model controller to add functions to create, update, or delete users.

   (c) There were no major changes made to this story.

9. As an admin, I would like the ability to edit the ticket pull groups so that groups may be removed or added.

   (a) Points: 1. See User Story 5 for reasoning.

   (b) This item was done by editing the Group model controller to add functions to create, update, or delete users.

   (c) There were no major changes made to this story.

10. As an admin, I would like a view to show all the users on a ticket pull team so that the pull system may easily read the group members' names.

    (a) Points: 2. This is one endpoint of the app, where our functionality would end and another connected application would pick up.

    (b) This story involved creating a view which displayed all relevant information, as noted in the user story, which was displayed when a QR code is scanned.

    (c) Once we settled on what data to display and to show it when a QR code is scanned, no major changes were made.

11. As a user, I would like a QR code that links to a view with all members of my pull group so that it may be easily shown at the ticket kiosk on pull day.

    (a) Points: 1. This story was simply porting Story 10 to the user side.

    (b) This story involved creating a view which displayed all relevant information, as noted in the user story, which was displayed when a QR code is scanned.

    (c) Once we settled on what data to display and to show it when a QR code is scanned, no major changes were made.

12. As an admin, I would like a system to pull tickets for students so that it may emulate the 12^{th} Man Foundation's software.

    (a) Points: 3. This is how the students would actually receive their tickets in a real-life situation, so it was vital for this functionality to be implemented.

(b) This story involved creating a function in the User model which processed all the relevant information and assigned all group members a seat.

(c) We initially had created a Seat model consisting of randomly generated length-2 strings to emulate seat numbers, but upon further consultation, we decided to focus our efforts elsewhere.

13. As a user, I would like to be able to join a group by inputting a unique code so that users may be easily added.

    (a) Points: 2. This story allows for increased mobility between groups.

    (b) This story was accomplished by adding a code attribute to the Group model, which was then checked in a form submitted by a user when attempting to join a new group.

    (c) This solution provides a temporary fix to a group security problem. Upon consultation with our client, we decided to pivot to an invite-based system, with the current application being a combination of the two. However, if another team were to pick up development, we expect this user story to be phased out.

14. As a user, I would like my QR code to be displayed as a resizable PNG so that is easily displayed on a phone.

    (a) Points: 3. This allows the ticket pulling process to begin, which then kicks off to Story 12.

    (b) This item was accomplished using another Mailer object, which takes a QR code created in a User model function and emails it as an attachment to all group members. A user can then scan it with a camera to pull up the aforementioned information.

    (c) The one major change made to this story was that the domain was eventually changed to be different depending on whether the email was sent form a development/test environment or a production one.

15. As a user I would like to receive my group's QR code as an email once I join it so that I have access to it on pull day if necessary.

    (a) Points: 2. Story 14 is more vital to the pulling process, but this does increase the convenience of the app.

    (b) This item was accomplished using a Mailer similar to Story 14.

    (c) There were no major changes, as this was an addition to Story 14.

16. As a group owner, I would like to be able to leave a group and automatically reassign group owner so that I may join another group if I choose to.

    (a) Points: 2. This helps increase the mobility between groups that is possible in the app.

    (b) This item was accomplished by taking a previously existing "Leave Group" function and allowing the group owner to use it.

    (c) This story was eventually discarded in favor of forcing the group owner to transfer ownership to another member before being allowed to leave.

17. As a group owner, I would like to be able to invite a user to join my group so that I may have people I would like to pull with in my group.

    (a) Points: 2. This was implemented late in development, but it will be a key component of being able to move between groups.

    (b) This story involved creating an Invite model, which contained information on both the inviter and the invitee. We then created MVC functions and displays that revolved around the model.

    (c) We expect the invite system to fully take over for the aforementioned code system if development is taken over by another team.

18. As a group owner, I would like to be able to remove a member from the group so that members who will not be pulling with us are not registered in our group.

    (a) Points: 2. Another story that increases the mobility factor.

    (b) This item was done by creating a "Remove From Group" button, which linked to a simple controller function that eliminates that person's group association.

    (c) There were no major changes made to this story.

19. As a user, I would like a navbar to easily navigate the site so that I may find what I need quickly.

    (a) Points: 3. The navbar is how a user is able to reach pages they might not otherwise be able to, including login.

    (b) The navbar was made in a simple HTML layout, with some conditional logic being implemented determining what is displayed in the bar and when.

    (c) Once the team settled on what to put in the navbar and when to show it, no major changes were made.

20. As an admin, I would like a dashboard with basic stats so that I may have an overview of information from the models.

    (a) Points: 1. Most of the work was done in various user stories.

    (b) This story was used to connect the frontend and backend into one continuous system.

    (c) No major changes were made to this user story.

21. As a user, I would like a landing page to show all upcoming games and pull dates/times so that I am aware of the next game.

    (a) Points: 2. This is the first impression users get of the application, and the information it shows is important to a smooth pulling process for all involved.

    (b) This story was a simple view, with a few models being called to pull the appropriate data.

    (c) At first, we had static data for pull times, but once we realized that the administrators may need to change these for special conditions, the data shown was gathered from a Pull Time model.

22. As a group owner, I would like to be able to transfer group ownership so that there will always be a group owner.

    (a) Points: 2. This story, while not vital, is important to the functionality of other user stories, especially Story 16.

    (b) This story was created with a simple controller function, which was then called on the pull group page if the user is the owner.

    (c) No major changes were made to this story.

23. As a user, I would like a static page to display all information on schedule and pull dates by classification.

    (a) Points: 2. The importance of this story was addressed in Story 21.

    (b) This was accomplished in a simple view with no data being pulled into it.

    (c) This story was a temporary fix, eventually discarded for Story 21.

24. As an admin, I would like to be able to delete groups and reset the pull status of all users.

    (a) Points: 2. This was implemented late in development, but it is a key component of the application.

    (b) This story is accomplished in a controller function, which then gets called from the admin dashboard.

(c) We originally only planned for this functionality to take place asynchronously on a regular basis, but we decided to give administrators the power to execute it at will as well.

25. As an admin, I would like the deletion of groups and reset of pull status to be asynchronous and automated.

    (a) Points: 2. This allows users to form different groups with ease from week to week.

    (b) This story combines the controller function from Story 24 with a scheduler that allows it to run on its own each week.

    (c) As mentioned previously, this function was only planned to run automatically, but we then implemented the admin privilege discussed in Story 24.

4 Scrum Iterations

4.1 Iteration 0

Iteration-0 began with the proposal of Aggie Pull-Day (which later became List Eaters) to our client, The 12^{th}-Man Foundation (Dr. Ritchey).

1. Client Meeting:
   (a) Date: December 29, 2021
   (b) Location: Thessaloniki, Greece
2. Summary of Accomplishments:
   - Establishment of team roles.
   - Conceptualization and mock-ups of an initial UI:
     – Feature: View informative landing page.
     – Feature: Login for users.
     – Feature: View current pull group.
     – Feature: Add current season's home football games to landing page.
   - Approval of List Eaters

4.2 Iteration 1

Iteration-1 consisted of introducing the early CRUD features of the project; generation of a simple web app layout; provisioning of databases; and the introduction of unit testing for the application. Lastly, the basic CRUD features for students to interact with groups was demonstrated; as well as admins using CRUD features to manage football games and seasons.

1. Client Meeting:
   (a) Date: January 4, 2022
   (b) Location: Thessaloniki, Greece
2. Points Completed:
   - 17
3. Summary of Accomplishments:
   - Site-wide toolbar to navigate between login and landing page (not functional at the time).
   - Landing page with static season football schedule and classification pull days/times.
   - CRUD for admins to select, add, and edit football games and seasons.
   - CRUD for students to create, join, edit, or leave groups.
   - CRUD for students to add or remove users from a group.
   - Provision of databases to store student, group, and game information.
   - Addition of unit tests to verify CRUD actions.

4.3 Iteration 2

Iteration-2 consisted of revamping the database's entity-relationships between students and groups; cleaning up current CRUD for student and admin pages; adding more coverage for unit testing; and provide a pull day and time for a group given the ratio of students' classifications.

1. Client Meeting:
   (a) Date: March 3, 2022
   (b) Location: College Station, Texas, United States
2. Points Completed:
   - 11
3. Summary of Accomplishments:
   - For a given group, the ratio of student classifications is calculated and a group is given when they are eligible to pull tickets during the week.
   - Cleaned up the CRUD for students and admins to be tied to a bare, but structured, interface.
   - Improved the database entity relations between students and groups. Decreased the redundancy and implemented more DRY (Don't Repeat Yourself) principles.
4. Client Remarks and Requests:
   - Introduce group owners that are able to invite others, lock the group, and determine when the group is ready to pull.
   - Don't focus on assigning seats. That should be an external service from your own.
   - When a new game week begins, all pull-statuses need to default back to not-pulled.

4.4 Iteration 3

Iteration-3 consisted of implementing more CRUD features that interfaced with the user experience (*i.e.,*email confirmations, ticket-pull QR codes, and adding users to a group). In addition, Iteration-3 introduced clock-based automated database maintenance that would clean the *Groups* table records and reset a students pull status on a weekly basis. The QR code generation for a ready-to-pull group and the countdown timer were new features demonstrated during the meeting.

1. Client Meeting:
   (a) Date: April 19, 2022
   (b) Location: College Station, Texas, United States
2. Points Completed:
   - 6
3. Summary of Accomplishments:
   - When groups are ready to pull, a QR code is generated to pull all of the tickets at once.
   - This code is emailed to users.
   - Count down timer to indicate how much time till a group is eligible to pull tickets.
   - Began development on a scheduler to automate tasks.
   - Added functionality for buttons to appear on a student's account page depending on whether a student's group has or has not pulled tickets yet.
4. Client Remarks and Requests:

- Heroku Scheduler is a free add-on and can be good for automating the production-level tasks; though, it isn't always reliable. Maybe consider another approach.
- Make a callback service. This would be external to your List Eaters application. Think of it as a Kyle Field API. Input to this 3rd-party service would be a group ID and a list of students associated with that group. Keep this in CSV or JSON format to consume in a flexible manner.
- With the countdown timer, do not allow the disabling of JavaScript to break your page.
- Provide a more readable site URL. The current *frozen-inlet-69932.herokuapp.com* is not informative or easy to remember.
- Model and abstract the pull policy.
- Load the QR code in the web page in addition to email.

4.5 Iteration 4

Iteration-4 consisted of providing more features that users can utilize to better interact with fellow students in the process of creating groups, removing users, and adding users. Additionally, this iteration featured work on the addition of a mock application to represent how the 12th-Man Foundation handles ticket and seat assignment for Kyle Field. Lastly, continued work on the scheduler database tasks was done. The new functional admin dashboard, removing of users from groups, and sending of usable codes to join groups was demonstrated during the meeting.

1. Client Meeting:
   (a) Date: April 26, 2022
   (b) Location: College Station, Texas, United States
2. Points Completed:
   - 16
3. Summary of Accomplishments:
   - Incorporated a fully-functional admin dashboard.
   - Implemented a CAS-linked login system.
   - Reconfigured the QR code formatting in email messages.
   - Display the QR code on a student's page.
   - Fix the functionality for clearing the database entries associated with users' group information on a weekly basis.
   - Introduced a functional method for removing users from a group.
   - Send a usable code that can add users to a group when accepted.
4. Client Remarks and Requests:
   - Role-based access control for students and admins.
   - Use the CAS model you build because you likely will not have API access to A&M's CAS system.
   - For account creation and sign-up, consider what information CAS pulls.
   - Don't think of it as registering for an account. Instead, it's more like claiming your UIN or linking a TAMU account.
   - The only field that should be user-changeable for students is the *group* field.
   - Cache the whole table view for football games to be static.

4.6 Iteration 5

Iteration-5 consisted of finalizing and concluding the remaining user stories that were still in development. Once those stories were completed, then the GUI for List Eaters was polished. The demonstration from this client meeting included more features added to the admin dashboard and connectivity across pages. Iteration-5 features consisted of finalizing features for the final demo.

1. Client Meeting:
   (a) Date: May 6, 2022
   (b) Location: College Station, Texas, United States
2. Points Completed:
   - 25
3. Summary of Accomplishments:
   - Completed mock 3rd-party service to model the 12^{th}-Man Foundation's Kyle Field ticket and seating assignment service.
   - Completed the automated clearing of groups in the database; resetting pull statuses to *not pulled* on Sunday at midnight.
   - Added button to admin dashboard for groups CRUD to delete all groups manually.
   - Revamped the GUI for the entire web application.
   - Greatly increased overall test coverage.
   - Upgraded the mailer.
   - Get the landing page to show dynamic game data.
   - Restructure the database to create a CAS-like form.
   - Increased test coverage.
   - Fix QR code in email.
   - Fix displaying group QR code on website.
   - Fix emails with mock-CAS system.
   - Allow user to leave group they created by reassigning ownership.
   - Update Heroku link.
   - Finalized 12^{th}-Man mock app to pull tickets.
   - Send group code as an invite and implement an invitation system.
   - Make ticket-pull information on the landing page in the database and editable.
   - Edit user page to provide more information about group.
   - Revised code, commented functions, and improved naming conventions.
   - Allow user to reassign ownership without leaving the group.
4. Client Remarks and Requests:
   - Replace the white space on pages; avoid the glaring white background. This is a lot of real-estate. You could condense the content and make the background translucent with some image; or, you could darken the background color.
   - Make sure the site works even if JavaScript is disabled.
   - Have a HUD for group info.
     – The day they pull.
     – Group name.
     – Collective classification.
   - You don't want anyone to enumerate all group IDs. Group IDs need to be unpredictable.

4.7 Icebox Features for the Future

- Handshake protocol for invites between a group owner and potential new group member: Both a group owner can invite a new group member, and new members can request to join a group. In either situation, both parties must provide consent before being added to a group. If a user requests to join a group and the group owner invites them simultaneously, that invite is accepted and the new user is added.
- Show all invites view: Show all received invites for all users and also show all sent invites for group owners. Allow invitations and requests to join to be removed after they are sent.
- Role-based access control: Group owners can provide addition functionality to some group members like the ability to add/remove members, or restrict who is able to pull tickets for the group.
- Delete invites/requests that are no longer pending: After an invite has been accepted, delete it from the table so that a user must get permission to join the group again if they leave.
- Associate groups with games: Allow for users to plan multiple games in advance by making every game a different row in the group table. This will also allow users to view past groups, and offer the functionality to recreate a group from one week to the next.
- Preference seating choices: Allow group owners to select their preference for seating sections. This can then be sent to the Ticket model (the mock 12th Man Foundation) and be used for assigning seats to further speed up the ticket pull process by eliminating another question that needs to be asked at the ticketing window.
- Build to work with the 12th Man Foundation's API and CAS: Reach out to the foundation and to the TAMU IT department to see how this could be implemented using AM's current sign on and ticket pull system.

5 Engineering Design Process

5.1 Behavior-Driven Design/Test-Driven Development

Our team followed a standard Behavior-Driven Design process, in which we created Cucumber user stories that determined how we created our views. We enjoyed the typical benefits of Cucumber-based BDD, such as the ability to make tests out of our verbatim user stories. However, one drawback we sometimes faced was that it was sometimes hard to determine how to structure our steps when there wasn't an existing code base for a certain feature.

We also followed the standard process of Test-Driven Development by basing our model controller functionalities on RSpec tests. This was even less disadvantageous for us, as RSpec does not require us to interact with the intricacies of a model or controller the way Cucumber/Capybara did with views.

5.2 Configuration Management

Our configuration management process would begin with the creation of a new branch for a feature by the person in charge of developing that feature. Once the feature was developed and ready to be merged into the app, as product owner, Jon would review and test the branch. After any potential bugs were found and fixed, the work would be merged into the main branch.

Over the course of this project, we created almost 60 branches. However, not all of these were merged. Often times, if one team member was assigned multiple tasks in an iteration, they would merge the work from all of their branches into one pull request so as not to make the review process tedious and repetitive.

We followed a protocol of releasing at least once every iteration to our external Heroku app. However, we tended to release more than that so as to constantly check for potential Heroku-related bugs or errors that would arise.

5.3 Production Release

Our first issues we faced with deploying our app to production was dealing with back-end issues revolving around Heroku's database. Furthermore, when using Heroku Scheduler, we ran into issues with the tasks not running as intended based on the time intervals set. After reading through more documentation on Heroku Scheduler it became apparent that we were not considering the time for Heroku Scheduler is based in the UTC timezone and not the CST. Once this discrepancy was acknowledged, changes to the scheduler rake task was made to account for the offset of UTC and CST time.

5.4 Other Tools

Our team made use of the gem SimpleCov, which compiled a report of our test coverage between RSpec and Cucumber tests each time the tests were run. This was a helpful tool for us by showing us areas in which we could further verify the integrity of our MVC system.

For maintaining secure passwords, we chose the gem BCrypt. This allows us to store hashes of users' passwords in our database rather than raw passwords, which would be much less secure.

We luckily found a gem that handled the creation of QR codes for URLs, called RQRCode. This handled all aspects related to creation and styling of QR codes, which only left to us the controller functions that get called when the QR is scanned.

In order to generate random seat numbers and other seed data, the Faker gem was brought into the project. This allowed us to seamlessly generate lists of fake data that we could easily use to test MVC code with random data that we had never seen before.

In order to schedule the regular flushing of the database, we used the gem Whenever, which abstracts the writing and execution of cron jobs. While this handles automated execution on the development side, we had to learn to use Heroku Scheduler to run the same job in production. However, we ran into issues with using dynos, which are designed to be only executed once.

6 Project Links

- https://github.com/Aggie-Pull-Day/Aggie-Pull-Day
- https://www.pivotaltracker.com/n/projects/2547057
- https://list-eaters.herokuapp.com/
- Presentation Video:
  - https://drive.google.com/file/d/14eq0h744467WEhh8lys1sYpQ5Pw88K3x/view?usp=sharing
- Combinbed Presentation and Demo Video:
  - https://drive.google.com/file/d/1fDKmRUl4_whe6vdRjag3cXzlkFlA_G8i/view?usp=sharing

7 Repository Contents

The project repository contains all files needed to run the application on a local machine, after following the steps outlined in Section 1 of this report. All MVC, configuration, routing, database, and testing code is contained in the repo, with the only files not being uploaded to the repo regularly being caches, temporary files, and Yarn modules.

The application is a fairly standard Ruby on Rails application, so no special processes or scripts are needed to run it apart from what is listen in step 1 in the normal setup.