CSCE 606
Spring 2022

EventNXT Final Report

Summary of Project

The customer of the project, Mr. Tito Chowdhury, needed an automated system to manage the guest lists for events an event owner creates for invited guests, as well as guests who decide to attend the events on their own. The proposed system combines the total count of confirmed guests (VIP and box office guests), giving a clear picture to the event owner about the number of occupied seats and vacant seats. The exclusivity of this system lies with its ability to combine both sales into one portal, as previously designed systems managed only one at once. The system also tracks referrals and incentivizes appropriately with a reward system. A user can manage event tickets, promotion strategies, and guest referrals. The stakeholders in EventNXT are event organizers and managers who are responsible for ticket sales and seating arrangements, thus ensuring their events are successful.

The application meets the customer need by managing both RSVPs from invited guests and box office sales data via a spreadsheet imported into the application, and provides a ticket commitment inventory. Most of the changes to the application during this semester, as meeting the needs of the customer, include a rework of the UI to make the process of event management more intuitive, as well as implementing main features of the proposed system after fixing some bugs in the legacy code. Future work into the project can perfect the UI design, as well as keep the UI in mind while implementing relevant user stories (e.g. separate pages in event creation and management) to reduce refactoring in the future.

Team Roles

The team consisted of five members: Edmund Do (product owner), Apurva Shinde (scrum master), Apoorva Sathe, Joseph Muir, and Leon Liu. The customer was Mr. Tito Chowdhury.

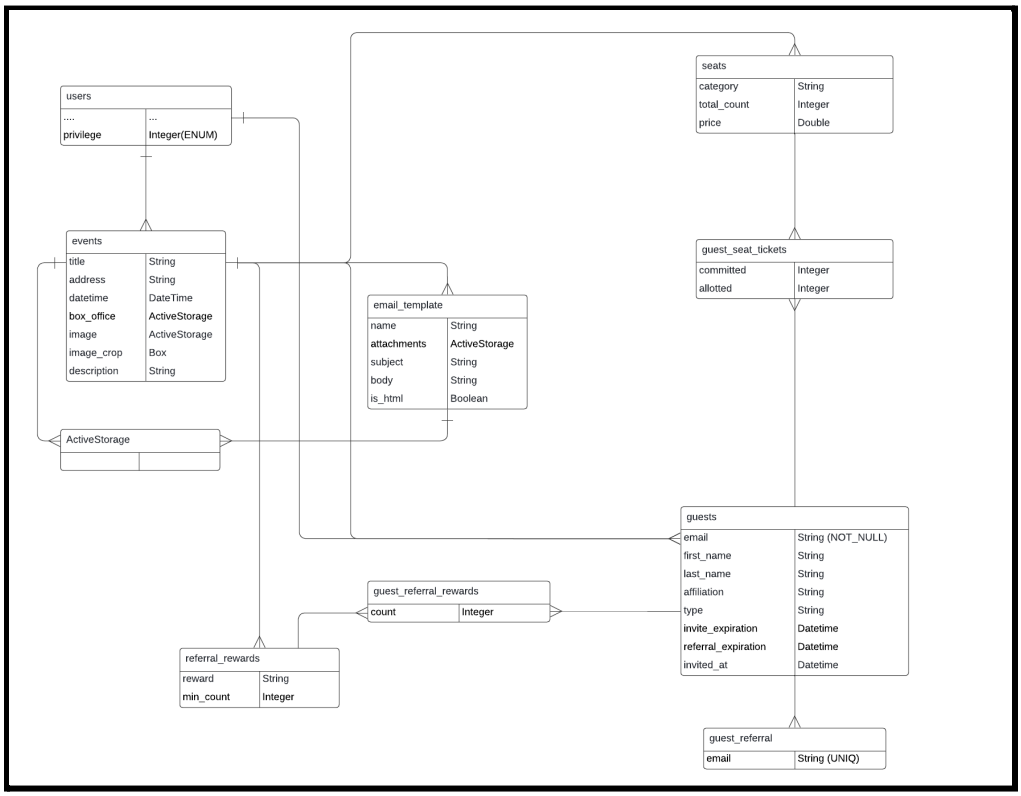
During the beginning of the project, Edmund, Apurva, and Apoorva focused on the backend side of the project (implementing the main user stories), while Joseph and Leon focused on the frontend and refined UI design (including lofi sketches and implementation of functionalities including managing multiple events and a refined process of event creation). As the semester progressed, Apoorva moved to the frontend team, since more work needed to be done in that territory once user stories in the backend team were in the process of being completed.

User Stories

Database redesign and migration of seating, guests, and events controllers - total of 8 points - finished

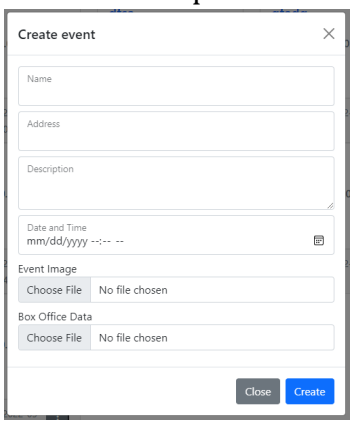
Many unnecessary fields were stored in the database that would frequently change based on the user input. The redesign of the database only stored essential information and properly modeled entity relationships corresponding to the client's specification. The storage system uses Rails's ActiveStorage to store images and uploaded data. We created new tables to store referral rewards and email templates. Nearly all models in Rails had to be changed and the code utilizing them. The join tables modeling the many-to-many relationships between entities provide essential information for the booking summary information and keep track of guest referrals to properly reward guests that refer more people to an event. The events, seating, and guest controllers were

migrated to utilize the database redesign by returning and updating the models through a restful API.

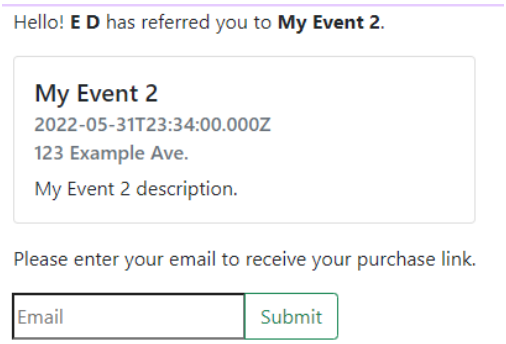


Upload image to invitation card via input field- 1 point - finished
The user should be able to upload an image from the event creation form. This was done via an input field, although further work would be required to integrate the user interface for image uploading to the backend.

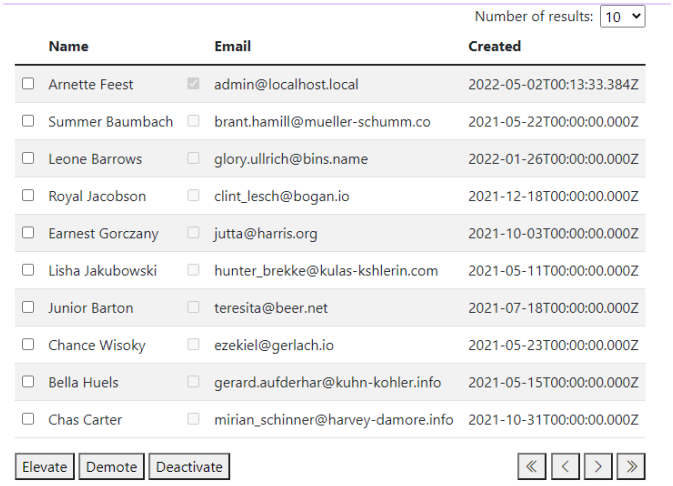
Box office data upload - 1 point - in progress
Box office data should be associated with an event so that functionality utilizing box office data will work as expected and users do not have to reupload box office data.



Referrals System - 2 points - finished
The user should be able to send and create referral links and rewards for guests. When a referral link is clicked, it is recorded and attributed to the referring guest (the main hook being */api/v1/events/:id/rewards*). The RESTful API was added to allow guests to share a referral link to new customers in exchange for rewards when they meet a certain threshold. All features can be utilized by sending the appropriate request (get, patch, post, delete) to the respective url (i.e. *'/api/v1/events/refer/:guest_id'*). Referrals that click on a guest's referral link will be presented with a page that displays a text box to input their email so that when the event organizer receives updated box office data, the guest will be attributed with a referral.



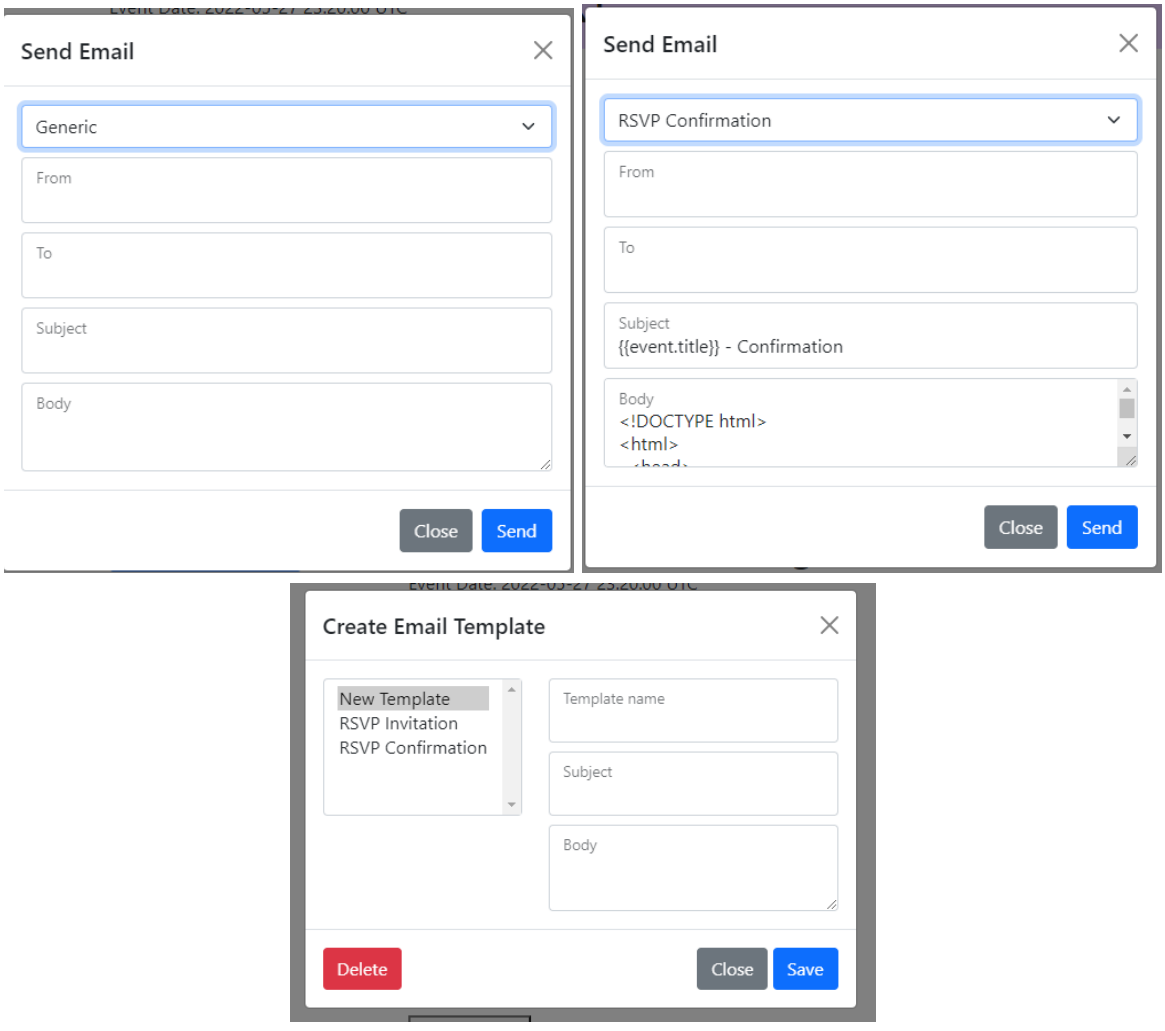
Download event guest list page as csv - 1 point - finished
The user should be able to export the guest list as it is seen on the event portal as a spreadsheet in csv format.



Users List - 1 point - finished
The user should be able to get a list of other event organizers (the main hook being */api/v1/users*). The RESTful API was added for querying and deleting users. The features could be utilized by sending the appropriate request (get, delete) to the respective url (i.e. *'/api/v1/user/:id'*). The post and patch requests are handled by the registration system. A basic interface was added for managing users by elevating/demoting privileges or deactivating other users.

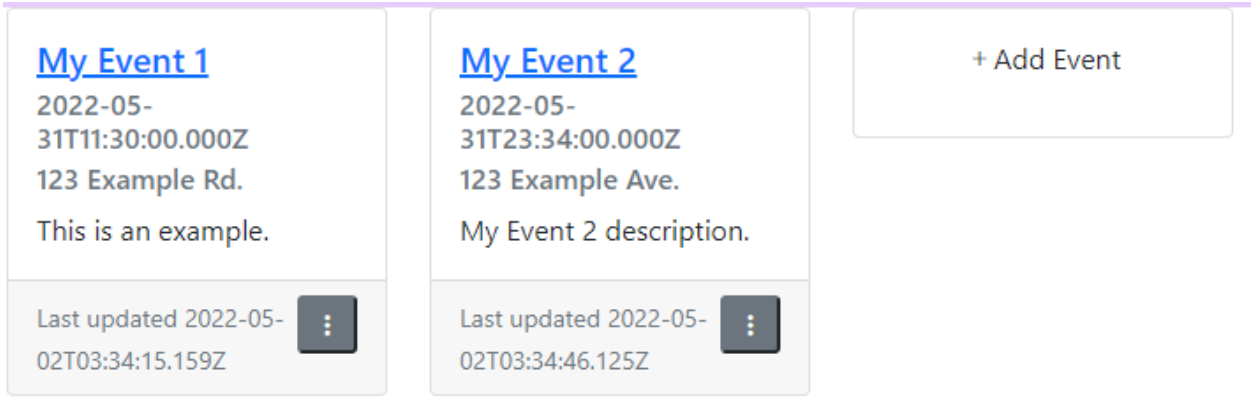
Guest check-in - 1 point - finished
The event owner should know which guests have arrived at their event. A hook was added for updating the check-in status of a guest for when a guest has arrived at an event.

Email templating and emails - total of 4 points - finished
The user should be able to send a template email or custom email to a specific selection of guests with their login email as the sender email (the main hook being */api/v1/event/:id/email* and the individual guest hook being */api/v1/event/:event_id/guest/:guest_id/email*). The RESTful API was added to eventually allow users to send regular emails and manage email templates for their events. Previously, the templates had to be changed by adding a view. Now, the template is saved to the database as an html string. We opted to use the mustache library to allow for filling in predefined variables (i.e. event details or guest details) due to its logicless design. Additionally, default attachments can be sent along with the template. The event organizer should be able to send a custom email to their guests, using the template mentioned in the user story above.



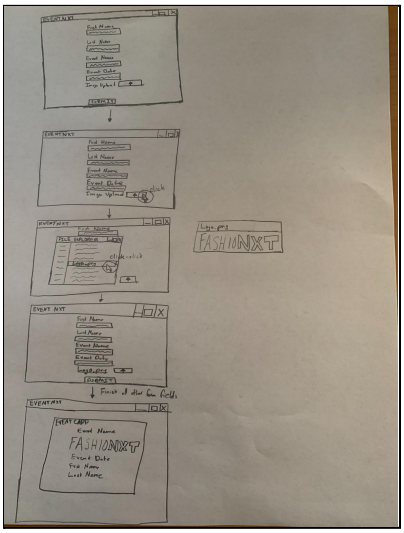
Event page cards - 2 points - done

The event organizer should see event information displayed in card-based format on the dashboard.



Ticket inventory - 2 points - in progress
The user should be able to get an inventory count on the box office commitments, guest list commitments, and remaining seats available based on their seating configuration. For example, a GET request on an */api/v1/events/:id/summary* hook would get the ticket summary.

Event creation UI redesign - 3 points - in progress
The user should be able to easily make a full event with all aspects of the event. For example, this could go from clicking the "Create Event" button, to filling out the event invitation information correctly, to managing various seating tiers for the event, to adding appropriate information to add guests, and displaying all this information for the event page.



Validation for user inputs - 1 point - in progress

The user should be able to input appropriate information for an invitation card so that there is validation for their information user inputs. An unfinished controller has been added to handle form validation on the front end.

User picks relevant columns from box office spreadsheet - 2 points - unstarted
The user should be able to select the column from the spreadsheet that pertains to seating information to compute a basic summary of seating information using any box office data format.

Address field for invitation card - 2 points - unstarted
The user should be able to fill in an address for the invitation card. For example, when an user hits submit for an address of the invitation card, the address should be on the invitation card preview.

Analytics - 3 points - unstarted
An event organizer should be able to have data on referrals sent out, conversions from referrals, and etc.

Legacy Code and Modifications

We initially looked through the controllers, and then the tests to understand the expected behavior if the code in the controllers were not clear. The first thing we opted to change in the legacy code was the database. It was mostly used as a data store for frequently changing information rather than modeling the relationships between data tables. As a result, many of the controllers had to change as well to query the right information from the database. We additionally used Rail's ActiveStorage to store the spreadsheet data and images, rather than saving the string representation into the database. The customer also wanted to change the event creation process, so the event creation UI and order had to change as well.

We also changed the structure of the application. The code was broken into two parts: backend and frontend. The benefit to this design is that each member can be more familiar with either the database-side or UI-side, and each member can work independently on a very specific feature. The backend currently consists of an API that simply queries the database and returns the result in JSON format. In this case, the "view" consists of a JSON format of the database result. Most of the current work on the backend simply returns the database without any additional processing. Future work can provide options for specifically choosing what is needed and providing features like ordering. The backend was initially intended to work with a JS framework (e.g. React), but we opted for Stimulus.js instead since it was simpler for those that did not know React, but in this case, we still had to have frontend controllers in Rails. As a result, the design is rather clunky and the UI is unpolished. Future work into the project can perfect the UI design, as well as keep the UI in mind while implementing relevant user stories (e.g. separate pages in event creation and management) to reduce refactoring in the future.

Scrum Iterations

Iteration 0:
Our project group met up with the customer for an initial meeting to ask about the user stories we should focus on and about how we could improve the legacy code. The customer required that the

automated system for an event guest list had the functionality to manage guests attending the events. The key points of improvement of the system were discussed from the standpoint of the legacy code of previous software engineering groups for this project. The system needed to reflect the total confirmed guests attending the event so the event owner can strategize publicizing the event accordingly. The system should also include a reward system depending on the referrals to award the guests with incentives.

The application manages a spreadsheet document which includes RSVPs from the invited guests, as well as the ticket sales at the box office that provides a ticket inventory for the client to keep track of. Previously, the application was unable to manage the RSVP and box office ticket sales data at once, which was reflected in our new design of the application.

Iteration 1 (2 points):
In this iteration, we focused on improving the user interface of the application which would be more intuitive and easy for the client to navigate all through creating a new event as well as making changes like editing the guest tickets as per the requirements of the client. The UI was redesigned in a way that accommodated multiple events created by the client at a time which was displayed in a card format on the main hub page. Furthermore, a user story for image uploading was implemented partially and tested with Rspec.

This iteration also included redesigning of the database tables to build a better relationship model from the previous legacy code. The database was redesigned in a manner that only essential information and modeled entity relationships corresponding to client's specifications were stored. This included Rails ActiveStorage as a storage system to store the images and upload the data. Additionally, for the referral system and email templates new tables were created. Some basic features which were overlooked in the legacy code like creating new events but not being able to access them or the changes could only be made to the default event created were looked upon to improve along with some additional functionalities like backend of referral system.

Iteration 2 (8 points):
This iteration includes querying and updating guests and the corresponding seats assigned to them using RESTful API. All the features were utilized by send appropriate request to the respective url. Furthermore, RESTful API was added to query the events which allows the user to update the events created by the user and access them. A function to return the ticket summary was implemented which included total number of seats committed, seating level for the guests, total seats left in a particular row etc for the client to better understand about a particular event he is organizing and take action accordingly.

Iteration 3 (4 points):
This iteration included a lot of work in progress from the previous iterations like ticket summary query, integrating frontend and backend for uploading the image functionality. Furthermore, a RESTful API was added for the referral system for the guests to share the referral link to the new customers in exchange for the rewards when they meet a certain threshold. These features can be utilized by sending the appropriate request to the URL.

Iteration 4 (6 points):
For this iteration, an addition of RESTful API for tickets allocation for the guests to commit the number of tickets was implemented along with updating the referral system which includes tracking of the email conversations so that the box office data is updated according to the latest user sales. An email system was set up that allows the user to send regular emails to guests and manage the email templates for the events. Particular focus was given on the UI so as to not fall behind.

Customer Meeting Dates
The following is a list of customer meeting dates and a brief description of what they pertained of (*note that the list below does not include meetings our project group made separately from the customer to discuss implementation and splitting tasks accordingly*):

1. *February 22, 2022 (02/22/2022)* - Initial meeting to discuss user stories, legacy code, and expectations
2. *March 1, 2022 (03/01/2022)* - Iteration 1 tasks including image uploading for invitation card, form validation on user input, and address field for invitation card
3. *March 8, 2022 (03/08/2022)* - Iteration 1 tasks progress; additional tasks such as a UI rework of separating pages and RSVP emails sending from the event account holder to guests' inboxes
4. *March 22, 2022 (03/22/2022)* - Demoed for Iteration 1; Iteration 2 tasks progress as no meeting during Spring Break; kept most of the Iteration 1 tasks although new tasks brought up by customer included timeout control by the owner to update event details and a repository of emails sent to guests for the event owner's inbox
5. *March 29, 2022 (03/29/2022)* - Demoed for Iteration 2; Iteration 3 tasks including referral links, downloading the event guest list page as a spreadsheet, and having a list of events for users
6. *April 5, 2022 (04/05/2022)* - Iteration 3 tasks progress
7. *April 12, 2022 (04/12/2022)* - Demoed for Iteration 3; Iteration 4 tasks including user permissions and guest check-in
8. *April 19, 2022 (04/19/2022)* - Iteration 4 tasks progress
9. *April 26, 2022 (04/26/2022)* - Demoed for Iteration 4 and discussing final tasks to to wrap up the project

BDD & TDD Process
For behavioral-driven development (BDD), firstly our project group adhered to the general guidelines of generating user stories for Iteration 0 documentation. Since BDD involves developing user stories (i.e. the features the customers desired) to describe how the application would work, we first began by compiling and describing a general list of user stories as defined by the customer from our initial meeting. As we continued on throughout the project, the customer proposed additional user stories or features that we added to the Pivotal Tracker after unanimous agreement by members. After the ideation process, we also proposed changes to the user interface using lofi diagrams and basic sketches representing the culmination of different user stories. Based on requests from the customer, as well as the hopes of our project group, we wished to improve the user interface to make it easier for users to create and manage events, guest lists,

ticket sales, and the referral system, by making the process more intuitive and spread across separate pages.

For test-driven development (TDD), which involves writing unit and functional tests for code before the code is written itself (due to the step definitions for a new story), we utilized Rspec, which is a unit test framework for Ruby. For each user story defined, we created a comprehensive set of test cases using Rspec reflecting the UI sketches we made in Iteration 0. Each unit test had both successful and exceptional cases (happy or sad baths) in order to account for both the general and exceptional situations.

In terms of project management, we utilized the Agile software development methodology where each iteration we focused on a subset of tasks as described in the product backlog (which can be seen as the list of user stories from Iteration 0). Some if not all tasks were tested for functionality and integration into the application at the end of the iteration to create a working prototype, which was demoed to the customer for feedback and suggestions. We kept track of tasks using Pivotal Tracker, an Agile project management tool, where we assigned tasks to each project member with a respective number of points and a general description. At the end of each iteration, we could tally up the points and calculate the velocity (or the number of points finished) per iteration to gather expectations on how many user stories our project group could develop by the end of the semester.

In terms of the BDD and TDD process in using Rspec, Cucumber, and the Agile software development methodology, the main benefits were that it kept each group member accountable and understanding of the general tasks needed to be accomplished. Also BDD and TDD inherently made our iterations include user stories that were relevant and reflective of what the users actually wanted to see in the application, as well as integrating testing functionality with development.

Configuration Management Approach

In terms of the configuration management approach, we utilized spikes when needed to either familiarize the team with a new software, or analyze a problem (or user story) thoroughly and assist in dividing up work equally among respective members. For example, a couple of times throughout the project we needed to investigate how much work was needed to implement a specific feature (i.e. a functional spike), or allow all members an opportunity to familiarize themselves with a specific API (i.e. a technical spike).

We utilized branching in our GitHub in order to develop user stories independently to prevent integration problems on the main branch. Branching in Git allowed for easy and cheap merging capabilities, an isolated environment for changes involved with a specific user story, and facilitating Agile development among different team members. Over the course of the project, we created a separate branch for each user story named with the corresponding ID from Pivotal Tracker.

Our project group produced releases at the end of each iteration (out of four iterations) tagged with *"Iteration #"* respectively. Releasing code at each iteration allowed for future software engineering groups to view the progress of code development over the course of the semester.

Production Release Process
We deployed our code to Heroku at the end of each iteration, which is a cloud platform that lets our project group build, deliver, monitor, and scale the application. Although our code was tested implicitly via the test-development development process, deploying our code to Heroku and hosting on a server allowed our changes to be tested by the customer when we demoed it to him at the beginning of the next iteration. There were minor issues deploying to Heroku relating to configuration files used in development. There were also minor issues with fixing email functionality for the website on Heroku, which prevented us from testing the referral system on the server.

Other Tools
[any issues with AWS Cloud9 and GitHub and other tools; describe other tools/GEMs used and their benefits.]

We used Docker to manage a consistent development environment between developers that is also fast to get started without each developer configuring their computers to use the same Ruby/Rails version and database. We used some additional gems that we thought were useful.

- Faker — Generates realistic, fake data including emails addresses, names, lorem, etc.
- FactoryBot — The tool provides a way to automatically create or generate attributes for models. Combined with the Faker gem, it provides a very easy to use interface for generating a lot of fake data.
- Devise — A gem providing secure authentication controllers that can be extended to get needed functionality.
- Doorkeeper — Gem providing oauth tokens for determining authorization to access and use certain pages.
- Roo & Rover-df — Used to parse and quickly query spreadsheet data. Rover-df is akin to Pandas used in Python.

Repository Contents

Aside from standard Rails content, the repository contains a Readme for getting started with development. The script folder has a custom script for generating random images of various sizes, but these images have already been committed into the spec/fixtures folder for testing purposes. The public folder also contains sample box office ticket data.

Important Links
Pivotal Tracker: https://www.pivotaltracker.com/n/projects/2554870
GitHub repository: https://github.com/jgmuir/EventNXT-606
Heroku deployment: https://eventnxt-sp22.herokuapp.com/
Link to video: