Introduction
By Jane Li

Lineup is an event and queue (aka “line”) management application that users can use to create events. An event can have multiple lines so this is perfect for a class with multiple sections or an event such as a job fair with multiple companies. I worked on the project planning, UI design, user manual, and testing documentation. Tong worked on database schemas and UI testing. Xinhao and Yitao worked on the front and backend. Yitao, Tong, and I worked on testing.

Planning
By Jane Li

Xinhao and I both took Software Design and Documentation so we had the idea to first create some planning documentation to help us draft and organize the project’s features before working on the project. I created use cases, user stories, and user scenarios to plan out what the data flow would look like. The use cases helped me think about how the pages could be set up for more efficient flow to reduce the number of user actions. The user scenarios and user stories helped identify the classes we would need to create, the features that would be absolutely necessary vs ones that could be done if we had time, and the permissions each user would need. This also helped plan the user permissions for the database schema.

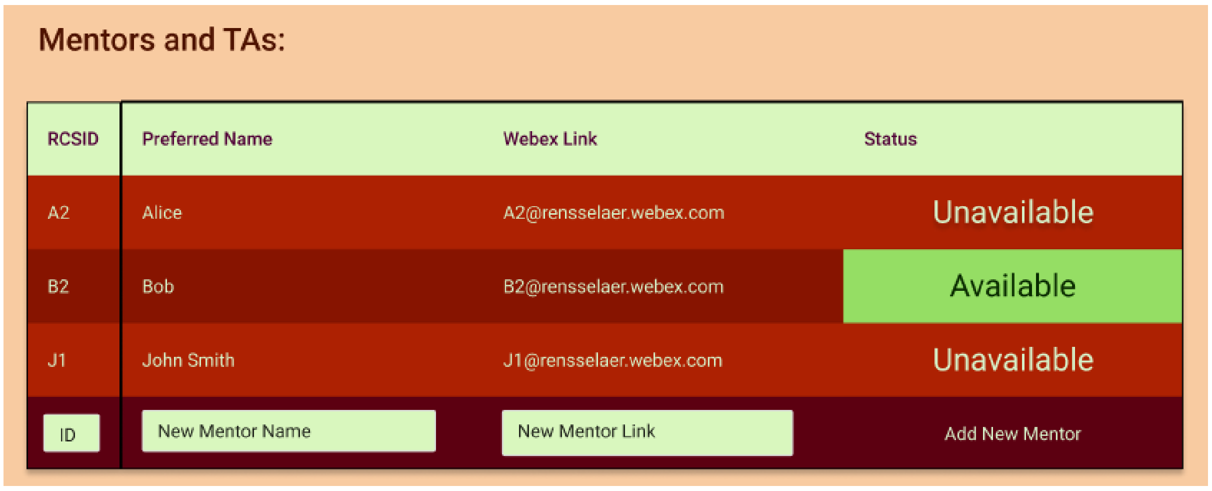
User Experience Design
By Jane Li

Using the use cases and user scenarios, Tong and I created a mockup to help plan out what the app would look like. Tong would be working on the database schema so we worked on it together since I wanted to make sure the mockup would be consistent with the database design. We created the mockup using Figma. While creating the mockup, I tried following the four golden rules of UI design which are: place users in control of the interface; make the UI comfortable to interact with; reduce cognitive load; and make user interfaces consistent.

We made a page for each important part of the user scenarios such as the Login/ Registration page, the Event creation page, and the queue page with Teacher, Mentor, and Student views. I took some inspiration from the existing submitty queue page layout but also I added some features that would be nice to have.

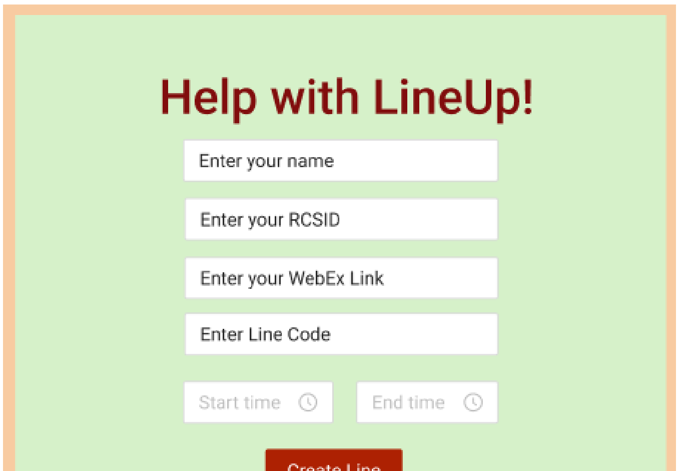
Some features from submitty I used was the student view of only seeing their current position in line instead of seeing all the students. This feature would make it so that we would not need to disable any buttons per permission while also protecting other student's data. This promotes visual clarity for the student by hiding all the features they would not be able to use.

Some features from the mockup were not added to the final product due to lack of time so I will talk about those in this section. One feature I added was the "Teacher Status" panel at the top of the queue page which would have the list of all the Teacher and Mentors for that event's queue. It would list their status' next to their names to show that they were "Available to Help" or "Unavailable to Help". This panel would be visible to all the people in the event so a student could see if there was anyone available to help and not be left waiting and worrying. The panel would also help out Teachers manage their mentors better. This panel helps with showing the visibility of system status which is a facet of the principle of placing users in control of the interface. I added a picture of the feature from the mock up. The panel would also be where the Teachers could add their mentors to help the queue but that idea was later removed during production. Mentors would be added by way of a Teacher user promoting a Student user in the event to a Mentor allowing the Student to have the permission to help other students.


Mentors and TAs:

RCSID	Preferred Name	Webex Link	Status
A2	Alice	A2@rensselaer.webex.com	Unavailable
B2	Bob	B2@rensselaer.webex.com	Available
J1	John Smith	J1@rensselaer.webex.com	Unavailable
ID	New Mentor Name	New Mentor Link	Add New Mentor

Another feature I added was a way to set the time the queue would close when the queue was created. The teacher would be able to just set the times of the specific duration of the event and the line would automatically delete itself. This would help with cleaning up the database of old queues and teachers would not need to worry about managing the queue after office hours had closed or students accidently going to the unmanned queue. I added a picture of this feature from our



mockup. The two input boxes would have let the Teacher type in a starting and ending time for the event. If no inputs were given the event would automatically delete itself after 24 hours.

These two features were not added to the final product because we lacked the time to actually finish them. However they would have been a great addition to a future iteration of Submitty. The features that we did keep were the registration and logging in, users editing their preferred names and statuses, creating and joining events and queues. Features that were added post-mockup were the “Records” feature which would hold the requests the users make to the Teachers and Mentors in the event’s queue and the “Manage” feature which would let users check on their mentors and existing queues.

Mockup and flow of data

By Tong Wu

First all users of LineUp will need to register an account. Registering will add them to the User table in the database. The username and password of the user are defined when registering and will be used to login in the future. A unique user id is also automatically assigned to the user so the application can distinguish the users easier.

After logging in, the user can change their display name and status in the Navigate tag on the top right of the page. The changes will be directly saved to the database and their display name can be seen by other users.

All users will have the option to create an event or join an event from the “Event” tag. The user will be able to enter the name of the event, select the number of queues and set the names of queues before the event starts. Then a token will be automatically generated in the database and sent back to the user so that the user can share the token with other users for them to join the event. The token is found under the Manage tab in the Event option. The users will be assigned different roles in the EventUser table, which grants different permissions to the user. By default, the event creator will have the highest permission. Users’ id in the database will also be linked to the event id if the user is attending the event.

Different queues are also available for the user to join as well so a user can join multiple queues. The application would link the queue’s id with the event’s id. The queue should also have a status that shows if it is joinable.

After joining the event, the users would have queues and helpers (mentors/TAs) presented to them with the helpers’ states (available/unavailable/offline). Therefore, the user’s id, the assistant’s id and the queue's id will need to be linked. However this feature wasn't actually fully implemented in the final product due to lack of time. The time that users entered the queue would be recorded so their position in the queue can be determined.

Database schema

By Tong Wu

The database contains multiple tables that assists different features of the LineUp application. The first table, User, is used to store basic information of all the users using the system, including teachers/TAs, mentors and students. The *display_name* attribute enables the student to choose their preferred name that will be displayed to the users using this system. Since a mentor's online/offline status might be available to students, the user table has an attribute of status.

Another essential functionality is event creation. The teachers/TAs using this system have the permission to create events. Each event will have a token. The token can be sent out by teachers so that the mentors and students will be able to join the event.

The users of LineUp are expected to be assigned different roles that would grant them with different permissions during an event. Therefore, to make sure the users are assigned proper roles and to resolve conflicts, an EventUser table is created. The major attributes are linked to the User table *id* and Event table *id*. This link is essential for the functionality of assigning mentors/students to different events by adding new rows to the table. Creators of the events will get an admin role and they will be able to promote others who joined the event as regular people so they become mentors. The role attribute makes it easier to present different UI with different functionalities to specific groups of users. For instance, the feature of removing a user from a queue should not be available to students, and event creation is reserved only for teachers/TAs but not for mentors.

The Log table is used to store logs of events. It stores the events with event id and associated users in the events with *user_id*. A user specified event description is also an attribute. The *created_at* timestamp helps keep track of time events that are being created.

During the event, the Queue table is deployed to keep track of users lining up in the queue. The name attribute serves as the displayed name that other users will see. Each queue is assigned an event with *event_id* and has a status, which is either open or closed.

Associated with the Queue table is the QueueUser table. This table stores the users waiting in the queue. Each user in the queue is assigned a new QueueUser id to distinguish them from each other. There is also an *assistant_id* attribute which is the mentor's id that helps the student. The status attribute represents if the student is waiting, being assisted, has left or has finished being assisted. The *joined_at* and *ended_at* timestamp attributes are used to track the position of the student in the queue. The application will scan the rows in this QueueUser table to determine which users will be helped first.

Frontend

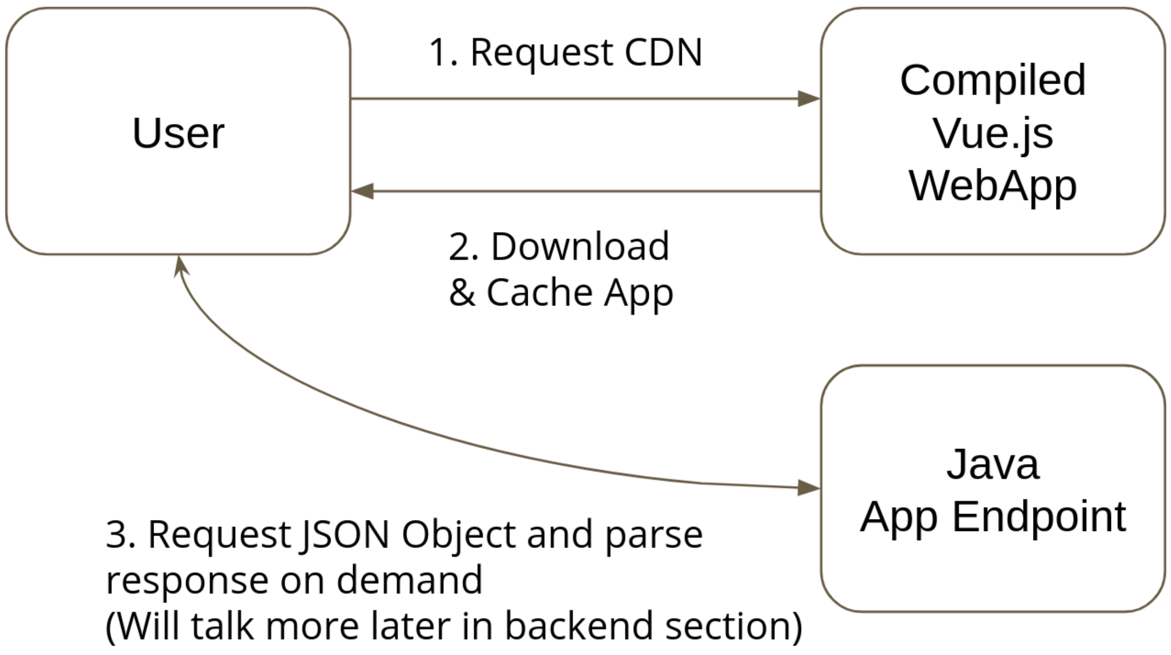
By Xinhao Luo

We have our project's structure separated into the frontend and backend part. This provides the following advantages:

1. Lift the pressure from render templates/save mobile device's cellular network off the server
2. WebApp can be cached and reused without loading the static resources again
3. Decouple frontend HTML code with Java backend logic
4. Potential to turn into native mobile App/embed into other system (Follow the API standard)

Here we choose Vue.js as our main JS framework to build the WebApp, and Bootstrap UI library for the user interface appearance. Vue is an easy to use frontend framework and is actively maintained by the community. It comes with powerful add-ons such as vuex, a state management plugin that eases the pressure to cache and organize API data. Bootstrap is a modern UI framework that is built with accessibility in mind, so our application can better serve people in need e.g. using screen readers, out of box. Bootstrap also makes it easy to make a responsive website so we would not need to do any fancy css to make pages look good.

As the frontend is not the focus of the course, I will briefly talk about the workflow as well as some core concepts that help understanding the structures.


User
1. Request CDN
Compiled Vue.js WebApp
2. Download & Cache App
Java App Endpoint
3. Request JSON Object and parse response on demand (Will talk more later in backend section)

In production, our WebApp will be first compiled and served via the content distributed network(CDN) as these are static HTML, JavaScript and CSS resources. Users' browsers will download and cache the WebApp so that no further static resources need to be downloaded when the user reopens the page, etc. The Webapp will perform corresponding requests, such as login, join event, etc. on demand. These requests will be in JSON format via HTTP/HTTPS to the Java Backend then processed and sent back to the clients. Using JSON format will reduce unnecessary transferring of rendered HTML code and ease the pressure of the network, especially in events where lots of people need to perform action at the same time.

The Vue.js App also helps reuse HTML codes and provides a stronger connection recovery logic since it is actually an standalone app, instead of a single rendered page, more logic could be delivered and encoded at once. For further details related to this part, feel free to visit Vue.js website and explore its community.

Here I would like to mention that as I was too optimistic about the time and underestimated the frontend work. In the end the frontend took more time than I expected, and I left the layout and color to the default style provided by Bootstrap. However, this could be changed via lines of CSS adjustment in the future, as Bootstrap provides the ability to replace its own style user provided via SASS, a rule based CSS generator.

To sum up, I implemented all the frontend sections of the application, but due to the lack of time, I mostly used the default theme color and deviated from the UI plan talked about above.

Backend Database, Data Representation and APIs

By Xinhao Luo

We choose to use the relational database for our application. To be specific, we used the embedded H2 database for our application. The reason for using an embedded database is we wanted to give administrators the ability to spin up an instance when necessary (portable). Therefore there is no need to rely on any external services. With this method there would only be a few system requirements and there is no need to set up a complicated configuration for database systems like MySQL, etc. H2 also supports working in memory which makes it earlier to write unit tests.

In order to map data relationships and persist Java objects to the database, we use Spring Data, which is a standard for manipulating relational databases for Java objects. We also use Hibernate, an implementation of Java Persistence API, to build the schemas and persistence from Java annotations.

I have studied the Spring JPA and set up hibernate to automatically mapping the schemas on each start up of the applications. Our application is separated by the features, e.g. User, Event, Queue, so I built the mapping for each of these Java objects. Also, since there are composite keys, I have also implemented some custom compositions (e.g. EventUser for the Event ManytoMany relationship with User class), as well as the role mapping (e.g. ADMIN, ASSISTANT, and USER). I have also implemented some custom repositories that help cover frequently used client use cases such as finding whether a user is in the event and have the permission to assist other users, or whether a user has already waited in the given queue to prevent joining the same queue twice. JPA doesn’t cover these situations so custom repositories are necessary.

Since I have also been working on the frontend part, I design the API to communicate between the frontend and the backend. Most of the controllers’ are implemented by me and I have adopted some Restful standard, while most of the APIs are remote procedure calls since it is more flexible and allows me to finish prototyping a useful demo for our projects.

To sum up, I implemented all the schemas transformation to the JPA forms, as well as the relations, useful queries, updates, insertions, primary key mappings, etc.; designed the APIs of the application to communicate with the frontend and potential other types of clients, and implemented them in the controllers accordingly.

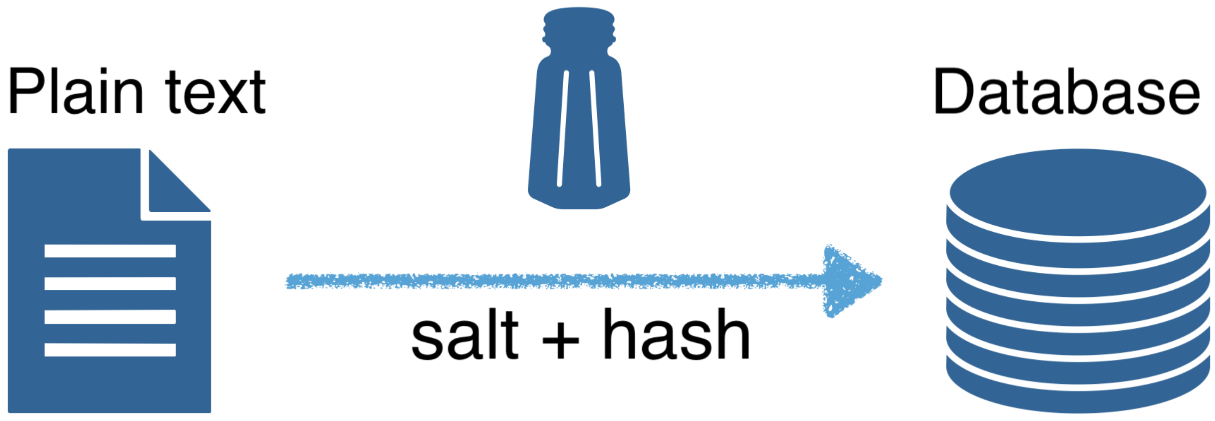
Backend Security

By Yitao Shen

We used Spring Security as the framework for user authentication, authorization and protection against common attacks. Since we don’t need to place any special configuration files for the JRE (Java Runtime Environment), it provides us much deployment flexibility, which means we can easily copy the target artifact from one system to another without any special configuration.

To implement the authentication module, we defined a class called `CustomUserDetails` to simply implement the interface (e.g. getUsername, getPassword, getAuthorities) in Spring Security.

As for password security, it is risky to directly store the plain-text password. Considering hashes are impossible to convert back into plain text, and we also don’t need to convert them back for authentication, we can just store the hash value of the user password in case the database can be leaked.



A rainbow table is a precomputed table for caching the output of cryptographic hash functions, usually for cracking password hashes. To mitigate the effectiveness of Rainbow Tables, instead of using just the password as input to the hash function, random bytes, also known as “salt”, would be generated for each password.

The plain-text password and the salt would be run through the hash function to produce a unique hash. The salt would be stored as plain text alongside the user password input. Then every time a user is trying to authenticate, the hashed password would be compared with the hash of the stored salt and the password the user just input.

With modern hardware, it has become easier to crack a password since more iterations of hash calculations can be performed simultaneously. Consequently, we used the leveraged adaptive one-way functions to store passwords. As a trade-off, the adaptive one-way function takes more time for validation (usually around one second) due to its intent, allowing the “work factor” of the functions to grow as the hardware process.

There are many examples of adaptive one-way functions including bcrypt, PBKDF2, scrypt, argon2 and etc. To implement the authentication module, we stored the hash value of the password via the hash function `BCryptPasswordEncoder`, which is offered by Spring Security using bcrypt algorithm. In the class `com.example.demo.user.UserController`, we will encode (add salt and hash) the plain-text password fetched from the frontend and store that hash value into the `UserRepo` which is a JPA (Java Persistence API) repository class offered by the Spring Framework.

Documentation

User Manual

By Jane Li

For the User Manuals, I structured it around flow of the typical user’s action and data. I started with the Login and Registration pages, then moved on to the Create an Event and Join an Event pages, and finally discussed managing the Event features. I added screenshots to show the pages and buttons for the features. I added descriptions to show what the user’s actions would achieve and examples of invalid actions.

README

By Yitao Shen

In README, we listed the reference link of the framework we used in the project (e.g. Spring-Boot, Yarn etc.). For a user who wants to get use of the project, we listed all prerequisites for building and running the project with automation scripts in the section “Quick Start”. For a user who wants to utilize the project partially, either the frontend or backend, we provide the tutorial to build and launch it manually. We also provided the tutorial about the usage of the in-memory database. For a user who only wants to test our API, the test case usage is also listed in README.

Test Manual

Test scenarios and Test plan

By Jane Li

As the code for the backend and frontend was created, Yitao, Tong, and I tested and recorded our findings. We based our test scenarios on the use cases and user scenarios I wrote from the beginning of the assignment. I wrote the testing manual to include our testing strategy, testing scenario, and the test cases.

We tested all the API requests using Hoppscotch.io since it’s completely free, online, and easier to use than Postman. To store our test cases, we added them to the Hoppscotch Collection for our project which was exported to a file called “hoppscotch-collection.json”. This file can be imported into Hoppscotch.io for us to all test with and is included in our project’s directories. We tested the frontend by manually interacting with the frontend of the application.

Testing the backend with unit tests
By Yitao Shen

The objective of the unit test is to verify that the functionality of different modules works according to the requirements. Although we already have manual tests at the frontend, it can still be hard to locate the bug without any functional unit test. Using our experience from previous homework, we found that unit testing is pretty helpful for excluding the potential bugs one by one. It can also be helpful for us to make sure critical defects are removed before the next levels of testing can start.

In the test manual, we list many essential testing scenarios, and the functional unit testing will be performed to check the functions of application. The functional unit testing will be carried out by feeding the test case input and validates the output from the application.

For instance, if we found the user cannot successfully join the queue, we can check whether the module `auth` is working, and then check if the event module is working. and so on. Only after verifying all functions implemented inside (e.g. reset, userJoin and etc.), can we move forward to the next level of the testing plan (manual testing).

Test cases
By Tong Wu

The test cases in general fall under two categories. The API test and functionality test. As mentioned before, API tests are performed using the Hoppscotch.io website and functionality tests are done manually.

The API test is designed around combining different possibilities when sending the HTTP requests. For example, when sending requests to register, cases such as null username, null password and duplicate username need to be considered. The website performs fairly well during the API testing. The returned json responses are as expected with the test cases.

Performing functionality testing is more intuitive. The functionality test follows the workflow of normal users of LineUp application. The users are all the same upon registering. After logged in, the users will either create a new event (as admin/teachers) or joining events (as students/mentors). Joined users can be promoted to mentors. Therefore, the test would focus on the interactions between these roles and check if the functionalities guaranteed by the application works well during these interactions. For instance, if a mentor is not in the event yet, the admin of the event cannot promote the user to be a mentor. Similarly, if a student joined the queue, only those who are assigned as assistant of this event can see and assist the students. Even the admin cannot help if he or she is not assigned as an assistant. The functionality tests went very well as most of the functionalities we planned in the proposal are successfully implemented.