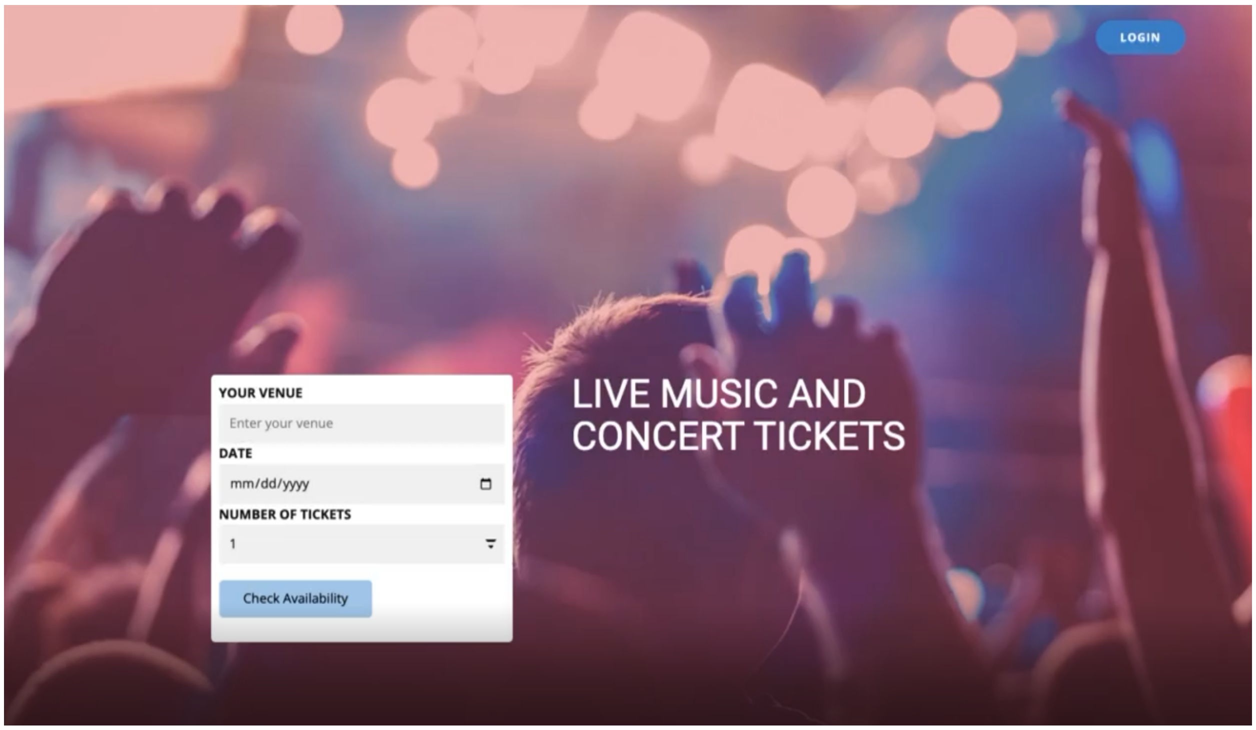
Live Music and Concert Tickets

Kyle Louderback 015331667 kyle.louderback@sjsu.edu

Tanya Gupta 014263158 tanya.gupta01@sjsu.edu



https://github.com/contactatfp/school.git (Main repo where Kyle Louderback and Tanya Gupta contributed and committed changes)
https://github.com/tgupta43/157aProj.git (copy of repo, each of us required to submit repo)

Goals:

The application will be a website that sells tickets for various venues. There will be three main user stories, the consumer, the hosts, and the performers. The web application will aim to help users find fun events and concerts, and also allow them to plan to stay at hotels near the event. Additionally, users will be able to see where their friends are going, allowing them to plan events and seats with friends.

Description:

Consumer:
The first will be from the view of the consumer. A consumer will be able to access the site and search for performers that are playing in the near future. The consumer can search for performers and venues without being logged in. They can search by area, date, and performer. Consumers can also have “friends” on the platform. All their friends will be listed on the platform. If they are friends, then they will be able to see any upcoming concerts their friends are attending (currently stored in DB, not displayed). Feature to be added for improvement: Consumers can have favorite performers, which will display all the upcoming events and venues.

Hosts/Venue:
The hosts of the venue will be the next user story. The application will display specific venues, their address, and date availability. This can be accessed by consumers, hosts, and performers. All hosts are the arenas and concert halls where the performers will play. These are a different type of user from the consumer and will have to be logged in as such.They can list the dates and performers that will be playing at their venue. They can set the consumer capacity, ticket price and hours for the event (Currently stored in DB). Host users will have a dashboard for total ticket sales, revenue, and tickets remaining. Feature to be added for improvement: All hosts receive a percentage of the ticket sales, and this will also be displayed on their dashboard.

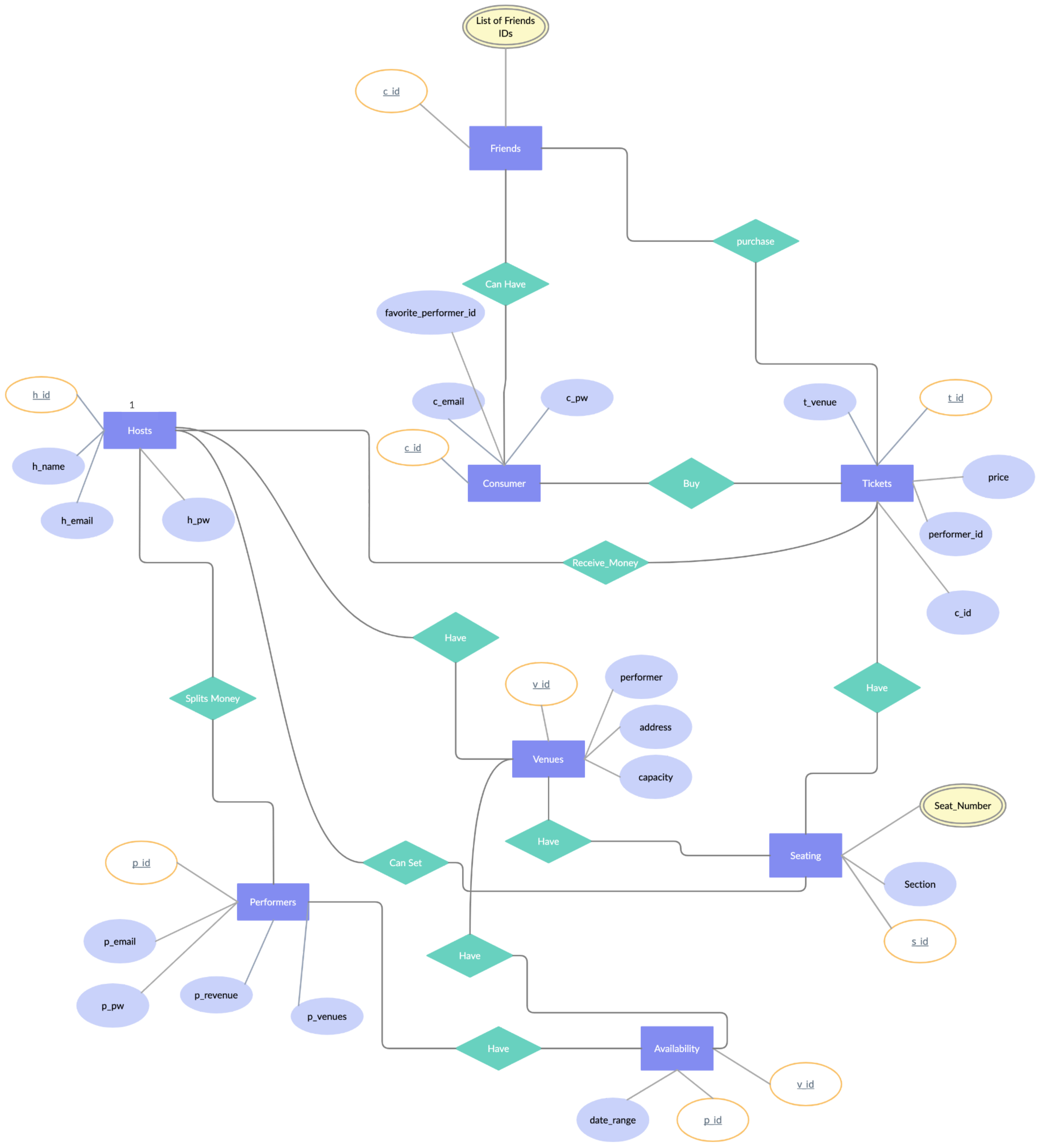
Performers:
Performers at the venue will be the next user story. All performers are a different type of user from consumers and hosts. The performers can see where they are booked and when they are performing. Performer details are displayed on the app. Feature to be added for improvement: When performers log in to the platform, they are directed to a different page where they can manage their events at various venues.

Tickets:
To purchase tickets, the user must be logged in and have an account setup with a credit card on file. They will also be able to buy tickets for the concert. Hosts can set ticket prices and see ticket revenue. Performers can see ticket revenue. Tickets will include prices and code numbers, and will use consumer information to confirm the order (id information in DB). Feature to be added for improvement: The application will have a section where consumers can buy tickets, and hosts and performers can see which/how many tickets have been sold.

Application/Functional Requirements:

1. The home page will allow users to choose a venue, date, and the number of tickets consumers want to purchase. A log-in/sign-up window will show up where the email and password will be entered and stored in the database.
2. The user will then be taken to a page that displays different events that fit the description the user typed in. The screen will show details about each event including ticket prices, and the user can click on the venue or directly buy tickets for the event. This screen represents the Ticket/Availability entity.
3. If the user clicks on the venue, another page will be displayed, where venue details are shown, representing the hosts and venue entity. In a future implementation, these will be separated so one venue can show which hosts have been there.
4. Another button will then lead us to the screen that displays information about performers, representing the performer entity.
5. The top right corner of the screen has a profile button, which will show user information. Consumers can see their friend list here (friend entity). Consumers can see their own account information (consumer entity).
6. The home screen is in charge of taking in data and storing it.

ER Data Model



DB Design:

Overall summary:
3 kinds of users: Consumers (Friends are a kind of consumer), Hosts, Performers
Entities needed for app: Venues, Tickets, Seating
Underlined attributes: Primary Key
Color codes: Foreign Keys

Consumers:

id	c_email	c_pw	first_name	performer_id

Hosts:

id	first_name	h_email	h_pw	performer_id

Performers:

id	band_name	p_email	p_pw	p_revenue	date_available

Venues:

id	performer_id	address	capacity	date_available

Tickets:

id	performer_id	venue_id	consumer_id	the_date	price

Seating:

id	ticket_id	section

Friends:

id	consumer_id

Architecture:

For the backend side of things, we used Java to write the functionalities of the entities. We used JDBC and Spring Framework to interact with the database. We developed an SQL script to create database tables with foreign key relationships, and to capture user data. We worked with Oracle DB.
For the middleware, we used Tomcat.
For the frontend side of things, we decided to use js, html, and css. Our HTML pages include the About, Home, Tickets, index etc. This helped us develop the application's screen's templates and layouts. CSS helped with displaying the styles of the app
css helped display the style of the app, which means it organized features like font size and spacing. JS handles display features like pop-up windows upon user interaction.

Design decisions:

For handling user data, or in other words, database management, we used Oracle DB and SQL. We used Java Database Connectivity (JDBC) to actually connect to the information in the database. JDBC is the driver manager, and packages like javax and java allow us to do this. The database interaction can be seen in the files like Consumer, Friends, Tickets, etc. In these files, we read the database table and use its data to make the java classes, which will then be used to create objects to further advance the application. JPA API, which is a part of the Spring Framework, maps, stores, updates and retrieves data from Oracle DB to Java objects and vice versa. This assisted us with taking in user inputs and displaying the data on the application for the user to see. These actions can be seen in the Service and Controller files with packages like org.springframework.data.

Implementation details:

We used Intellij and SQL Developer. We coded the js, java, html, and css files in Intellij, and our SQL script in the developer and ran the script there. Once we ran the script, we connected to Intellij. We used JDBC, Spring Framework, and repository/service/model structure to implement the application and be successful with running the app. We used the CrudRepository interface, which provided us with methods to read/write to a database. In the service files, we actually used the methods and defined functions. The controller files then use the repository/service model and html files to get the information and interact with data on the app.

Demonstration of example system run:

1. Run the SQL script named ticketDBScript using SQL Developer or Docker and connect to Oracle DB using Docker.
2. Using Intellij, connect to the Database.
3. Then download the project from github. MyWebApplication2 is the root directory. Run on an IDE (we used Intellij).
4. Then run project MyWebApplication2 in Intellij and run on your browser localhost:8080 to access the app.

Demo (Can find video on github → databaseDemo.v3.mp4):
At the home page, you can type in the choice of venue, the date, and number of tickets you would like to buy to explore availability of certain events. Log in or sign up to make a user account and it will lead you to screen with choices of venue, performers, and ticket prices. Upon clicking on the venue name, it will give you venue information. Another button on this screen will lead you to performer information. Clicking on a button at the top right will lead you to your profile, where you can see your friends, bought tickets and details, and your profile page with a picture and account information.

Possible improvements:

There are many possible improvements that we would like to implement in the future.
Future goals:

1. We want to make the friend list cleaner. Currently, the database has the friend's information and events they are booked for. However, we have yet to display that in the friend list screen.
2. We also have a seating table in the Database, and we want to add a page for seating so users can decide specific seats.
3. The database also takes in the price information, and we want to use that data to allow hosts and performers to split up revenue on the application itself.
4. On the home page, we will be adding a place for the consumer to type in a budget range.
5. Lastly, we will allow users to pick favorite artists so the application can save it and show consumer suggestions.

Challenges/ Lessons Learned/ Conclusion:

We faced many challenges while doing this project. It was hard to schedule meetings due to clashing schedules. Splitting the tasks helped. We encountered database connectivity issues in the beginning, which delayed our testing and coding process. Additionally, both of us were new to coding this kind of application. Learning Spring Framework, and having js, java, html, and css to work together with the DB took extra research. We learned that we could make the code cleaner, and with more time, add a lot more functionalities. We also learned that we may have been too ambitious with a lot of our ideas within a short time period, as although our database script does contain all the relationships, our UI does not display everything yet, like seating. We gained new skills within a short amount of time through these challenges, and were able to come up with a web application that takes in user data as well as displays data that is already in the database tables that we created.

Tasks:

Kyle was in charge of writing up the code (java, css, and html) for the tickets, consumers, and friends entities. He also researched Spring Framework for us to implement into our app. Tanya was in charge of writing code (java, css, and html) for the host, performers, and venue entities. Tanya was also in charge of working on the ER Diagram. Both of us worked on the final report, presentation, and the SQL script for the software.