

1. Technologies Used

- 1) Spring framework for Java – backend
- 2) JavaScript with Sass– frontend
- 3) MySQL – DBMS system (MySQL Community Server Distribution)

2. Functionalities supported

- 1) **Customers can sign in to an existing account or register a new account**
- 2) **Customers can view their booking history**
- 3) **Customers can book new events**
- 4) **Customers can add new payment methods**
- 5) **Employees can add new suppliers, clients, events, venues, suppliers, and items (any type)**
- 6) **Employees can view the list of all suppliers, venues, events, and/or items in the database**

3. General front-end explanation

The front-end was written with vanilla HTML and JavaScript and styled with Sass stylesheets which were compiled down to CSS with NodeJS. When pages are loaded, relevant tables and dropdowns are filled with data from the respective tables or queries via post-requests to back-end endpoints. When forms are submitted, they are checked for validation of format, and then once valid the data is read from the forms and sent via post requests to the back-end for insertion into the relevant tables. The front-end takes care of automatic fields like the currently logged-in user and item types on similarly templated pages, and the back-end takes care of automatic fields that don't need to be entered otherwise.

4. Back-end explanation

1) /userLogin

The backend queries the database using statement “SELECT * FROM CustomerAccount WHERE Username = ? AND UserPassword = ?;” If a user with the specified credentials exists, the backend returns their info, and an empty response otherwise. The frontend then either takes the user to their account page or displays an error message, respectively.

2) /registerCustomer

Upon clicking the register button on user registration page, the frontend sends a request with the specified info to the “/registerCustomer” endpoint. The backend utilizes statement “INSERT INTO CustomerAccount (Username, UserPassword, FirstName, LastName, PhoneNumber, EmailAddress) VALUES (?, ?, ?, ?, ?, ?);” to insert a new tuple into the CustomerAccount table. It also returns the newly created CustomerAccount object to the frontend.

3) /registerEmployee

Adds a tuple to EmployeeAccount table with the provided information, and returns the newly created tuple as a JSON object

4) /addPaymentMethod

Once confirm button is clicked on the screen to add a payment method, the frontend sends the provided info to this endpoint. The backend uses "INSERT INTO PaymentMethod (Username, CardNumber, FirstName, LastName, ExpiryDate, CVV, PhoneNumber, AddressLineOne, AddressLineTwo, PostalCode, Province, Country) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?);" to add a new tuple to PaymentMethod table. The newly created PaymentMethod object is then returned to the frontend.

5) /addMenuItem

Utilizes "INSERT INTO Item (SupplierId, ItemType, ItemName, ItemDescription, Price) VALUES (?, ?, ?, ?, ?);" to add an Item tuple into the corresponding table. Due to use of specialization, it then proceeds to use "INSERT INTO MenuItem (ItemId) VALUES (?);" command to add a tuple to MenuItem that refers to the newly created entry in the Item table. Returns a MenuItem JSON object that inherits attributes from Item object.

6) /addFlowerArrangement

Works similarly to /addMenuItem, but with specialization towards FlowerArrangement table

7) /addDecorItem

Similar to /addMenuItem, but with specialization towards DecorItem table

8) /addMusicEntertainmentOption

Similar to /addMenuItem, but with specialization towards MusicEntertainmentOption table

9) /recordPurchase

Represents a transaction between the user and the store. Utilizes "INSERT INTO Purchase (PurchaseDate, Username, CardNumber, BookingId, Amount) VALUES (?, ?, ?, ?, ?);" to add a new tuple to Purchase table.

10) /getDateRangePurchaseSum

Returns the sum of all the transactions that occurred within a specified date. Utilizes "SELECT SUM(Amount) FROM Purchase WHERE PurchaseDate BETWEEN ? AND ?;" statement.

11) /getVenueTable

Returns a list of all Venue tuples using "SELECT * FROM Venue;" command

12) /getBookingTable

Returns a list of all Booking tuples using "SELECT * FROM Booking;" command

13) /getItemTable

Returns a list of all Item tuples using "SELECT * FROM Item;" command

14) /getSupplierTable

Returns a list of all Supplier tuples using "SELECT * FROM Supplier;" command

15) /addSupplier

Utilizes command "INSERT INTO Supplier (SupplierName) VALUES (?);" to insert a new tuple into the Supplier table with the specified SupplierName. Returns a Supplier object with the auto-generated ID number.

15) /addVenue

Utilizes command "INSERT INTO Venue (VenueName, Address, BuildingType, Capacity, OperationStartTime, OperationEndTime) VALUES (?, ?, ?, ?, ?, ?);" to insert a new tuple into the Venue table with the specified info. Returns a Venue object with the auto-generated ID number.

16) /addBooking

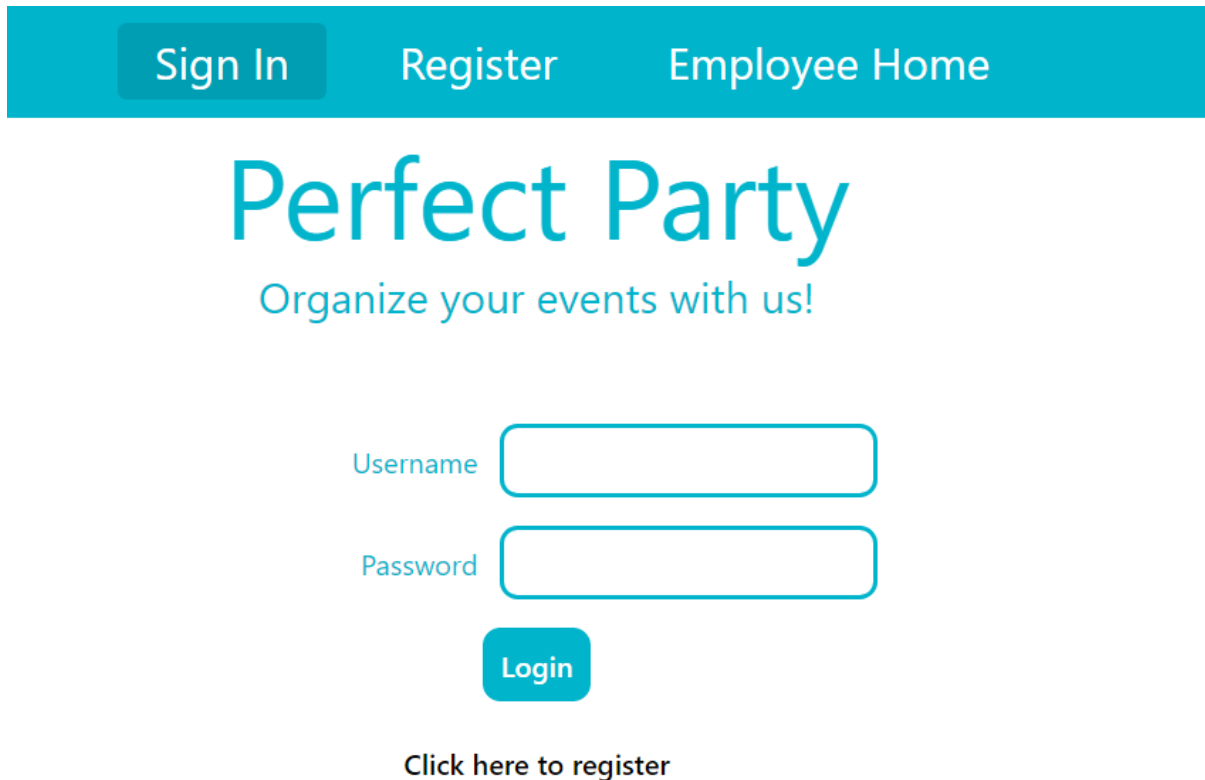
Utilizes command "INSERT INTO booking (BookingType, Username, StartDateTime, EndDateTime, VenueId, AttendeesNumber)" +

" VALUES (?, ?, ?, ?, ?, ?);" to insert a new tuple into the Booking table with the specified info. Returns a Booking object with the auto-generated ID number.

17) /getUserBookingHistory

Utilizes command "SELECT * FROM Booking LEFT OUTER JOIN Venue ON Booking.Venueld = Venue.Venueld WHERE Username = ?;" to return a list of all the Booking tuples associated with a user who has the specified Username.

5. Main features of the application (screenshots here)



The screenshot shows the homepage of the 'Perfect Party' application. At the top, there is a teal navigation bar with three buttons: 'Sign In' (highlighted with a darker teal background), 'Register', and 'Employee Home'. Below the navigation bar, the text 'Perfect Party' is displayed in a large, teal, sans-serif font. Underneath this, the tagline 'Organize your events with us!' is written in a smaller, teal, sans-serif font. The main content area features a login form with two input fields: 'Username' and 'Password', both with teal borders. Below these fields is a teal 'Login' button. At the bottom of the form, there is a link that says 'Click here to register'.

This is the homepage. Users can log in from here, and it checks if the entry for the username and password combination exists in the database, if it does, the website redirects to the user portal. The user can also click on the "Click here to register" link or the "Register" button in the navbar to get to the register screen (seen next).

[Sign In](#)[Register](#)

Perfect Party

Organize your events with us!

Username

Password

Confirm Password

First Name

Last Name

Email

Phone Number

[Register](#)

This is the register page. Passwords matching between the two fields is checked in the front-end, and then the form can be submitted, and a new user will be added to the CustomerAccount table. After registering, the user will be directed to the Add Card page so they can add a payment method to their account.

[Portal](#)[Booking](#)[History](#)[Add Card](#)[Sign Out](#)

USERNAME Customer Portal

Organize your events, check your past events, or change your account settings.

This is the user portal and navbar. This is the screen a customer sees after logging in.

[Portal](#)[Booking](#)[History](#)[Add Card](#)[Sign Out](#)

Add Credit Card

Card Number	<input type="text"/>
First Name	<input type="text"/>
Last Name	<input type="text"/>
Expiry Date	<input type="text"/>
CVV	<input type="text"/>
Phone Number	<input type="text"/>
Address Line 1	<input type="text"/>
Address Line 2	<input type="text"/>
Postal Code	<input type="text"/>
Province	<input type="text"/>
Country	<input type="text"/>
<input type="button" value="Confirm"/>	

This is the Add Card screen. Expiry Date, CVV, and Postal Code format are verified on the front-end, and then once a valid form is filled out, clicking confirm will add a new payment method, using the username of the currently logged in user. The user is redirected back to the portal.

[Portal](#) [Booking](#) [History](#) [Add Card](#) [Sign Out](#)

Book a new event

Event Type

Start Date

End Date

Venue

Brown's Concerts and Events ▾

Attendees

Book

This is the Add Booking screen. The Venue dropdown is auto-populated from the Venue table. The Attendees field can only take numbers, and the username is taken from whatever user is currently logged in when submitted, after which the new event is added to the Event table.

[Portal](#) [Booking](#) [History](#) [Add Card](#) [Sign Out](#)

Event History

7/1/2019 1:00 PM to 9:00 PM	Wedding for 100	Hacienda Sarria 1254 Union St, Kitchener
6/26/2019 2:00 AM to 6:00 AM	Party for 20	Starlight Social Club 47 King St N, Waterloo

This is the History tab. The user can see their booked event history here. The booked event history is compiled from a join of the Event and Venue tables, filtered on the currently logged in user's username.

On any of the above portal pages, the user can click on "Sign Out", where they'll be redirected back to the home page.

- Add new client
- Add new event
- Add new supplier
- Add new venue
- Add new menu item
- Add new flower item
- Add new decor item

- View all suppliers
- View all venues
- View all events
- View all items

This is the Employee home page. Clicking on any of the buttons takes you to the relevant form or view. The forms and views are pictured and described below.

Client Home

Employee Home

Add new client:

Username

Password

Confirm Password

First Name

Last Name

Email

Phone Number

Add Client

This is the Add Client screen. It works exactly the same way as the register page, except it's for employees' usage.

Add new event booking:

Event Type

Client Username

Start Date

End Date

Venue

Brown's Concerts and Events ▼

Attendees

Add Event

This is the Add Event (Employee) screen. It works the same way as the user-side Add Booking screen, except that it also takes a Client Username field. This field is checked against the CustomerAccount table. The Venue dropdown is auto-populated from the Venue table.

Add new supplier:

Supplier Name

Add Supplier

This is the Add Supplier screen. It adds a new entry to the Supplier table.

Client Home

Employee Home

Add new venue:

Venue Name

Venue Address

Venue Type

Capacity

Start Time

End Time

Add Venue

This is the Add Venue screen. It adds a new entry to the Venue table. Capacity is checked to make sure it's ≥ 0 .

[Client Home](#)[Employee Home](#)

Add new menu item:

Supplier

Party City ▼

Item Name

Item Description

Price

Add Menu Item

This is the Add Menu Item Screen. The Supplier dropdown is auto-populated from the Supplier table. Submitting the form adds a new item to the Item table with a “MenuItem” item type, and then a new entry to the MenuItem table with the foreign key of the new item. For conciseness, I’ll omit the Add Decor Item and Add Flower Arrangement screens, as they function exactly the same but with different item types (and their respective tables).

Client Home

Employee Home

Add new music or entertainment item:

Supplier

Party City ▼

Item Name

Item Description

Price

Availability Start Date & Time

yyyy-mm-dd --:-- --

Availability End Date & Time

yyyy-mm-dd --:-- --

Add Music or Entertainment Option

This is the Add Music or Entertainment Option. It adds a new item to the item table the same way as the previous add item screens, as well as the availability start and end datetimes to its own table.

[Client Home](#)[Employee Home](#)

All Suppliers

1	Party City
2	Micheals
3	The Cake Box
4	OldMill
5	Supersonic Hearts

This is the Suppliers List view screen. It displays the contents of the Supplier table.

[Client Home](#)[Employee Home](#)

All Venues

1	1000	Brown's Concerts and Events	35 University Avenue East, Waterloo	Arena	06:00:00	23:00:00
2	200	Hacienda Sarria	1254 Union St, Kitchener	Church	09:00:00	18:00:00
3	1000	Moore Event Centre	151 Charles St W, Kitchener	Conference Centre	09:00:00	21:00:00
4	500	Starlight Social Club	47 King St N, Waterloo	NightClub	22:00:00	02:00:00
5	2000	Columbus Conference Centre	145 Dearborn Pl, Waterloo	Conference Centre	09:00:00	17:00:00
6	39	Pizza Mania	200 University Ave W	Restaurant	05:00:00	21:00:00

This is the Venues List view screen. It displays the contents of the Venues table.

[Client Home](#)[Employee Home](#)

All Events

1	2	100	Wedding	ThomasSmith	2019-07-01 13:00:00	2019-07-01 21:00:00
2	4	20	Party	ThomasSmith	2019-06-26 02:00:00	2019-06-26 06:00:00

This is the Events List view page. It displays the contents of the Events table.

[Client Home](#)[Employee Home](#)

All Items

1	1	DecorItem	Helium Balloon	Various Helium Balloons including animal, number, etc	5
2	4	FlowerArrangement	Wedding Flower	Beautiful flower designed by Old Mill Flower Shop	20
3	3	MenuItem	Wedding Cake	Custmized Wedding Cake from The Cake Box	200
4	5	MusicEntertainmentOption	Live Band	High energy live music for parties	800

This is the Items List view page. It displays the contents of the Items table.

6. Additional work

The front-end has been styled with CSS (using Sass) to make it a bit more colourful and approachable

7. Appendix: Installation

In order to see the web app in action, the environment has to be configured. First step is to install a MySQL distribution on the machine. While the specifics of the installation differ depending on the operating system used, there will be a prompt to select a password for the root account, and the port, through which the database can be connected to on localhost. To avoid the testers having to make changes to the backend code, we recommend setting the password to “CS348Spring2019”, and the port to 3306.

Next step is to run the backend, found in the code folder. This requires JDK 8 and a Gradle distribution installed on the system, or a Java IDE with a Gradle plugin (e.g. IntelliJ IDEA). The important settings that tell Spring how to connect to the database are stored in the application.properties file (located in code\src\main\resources folder):

```
spring.datasource.url=jdbc:mysql://localhost:3306/store
spring.datasource.username=root
spring.datasource.password=CS348Spring2019
```

These settings specify the network location, and the login credentials for the database, and should be configured accordingly.

The backend can be run either from the IDE, or from a command prompt by using the command “gradle build” and then “gradle bootrun” within the \code folder. The backend accepts REST API requests to query and modify the database, examples of

which can be found in `Sample_Commands.postman_collection` file in `\templates` folder. Note that Postman application is required to use this file.

To get the front-end working, you must use a browser without cross-origin security, or otherwise disable security options that intercept cross-origin requests. Then, run the front-end on a locally hosted live server (we used the LiveServer extension for VSCode). This last part is necessary, as the front-end will attempt to use cookies to support the ability to log in.