CS F212 – DATABASE SYSTEMS

Fest Management System [Documentation]

By –

Akshat Kumar ***2020B4A71976P***

Jay Prakash Mudhra ***2020B3A70799P***

Project No.: 4

System Requirements Specifications:

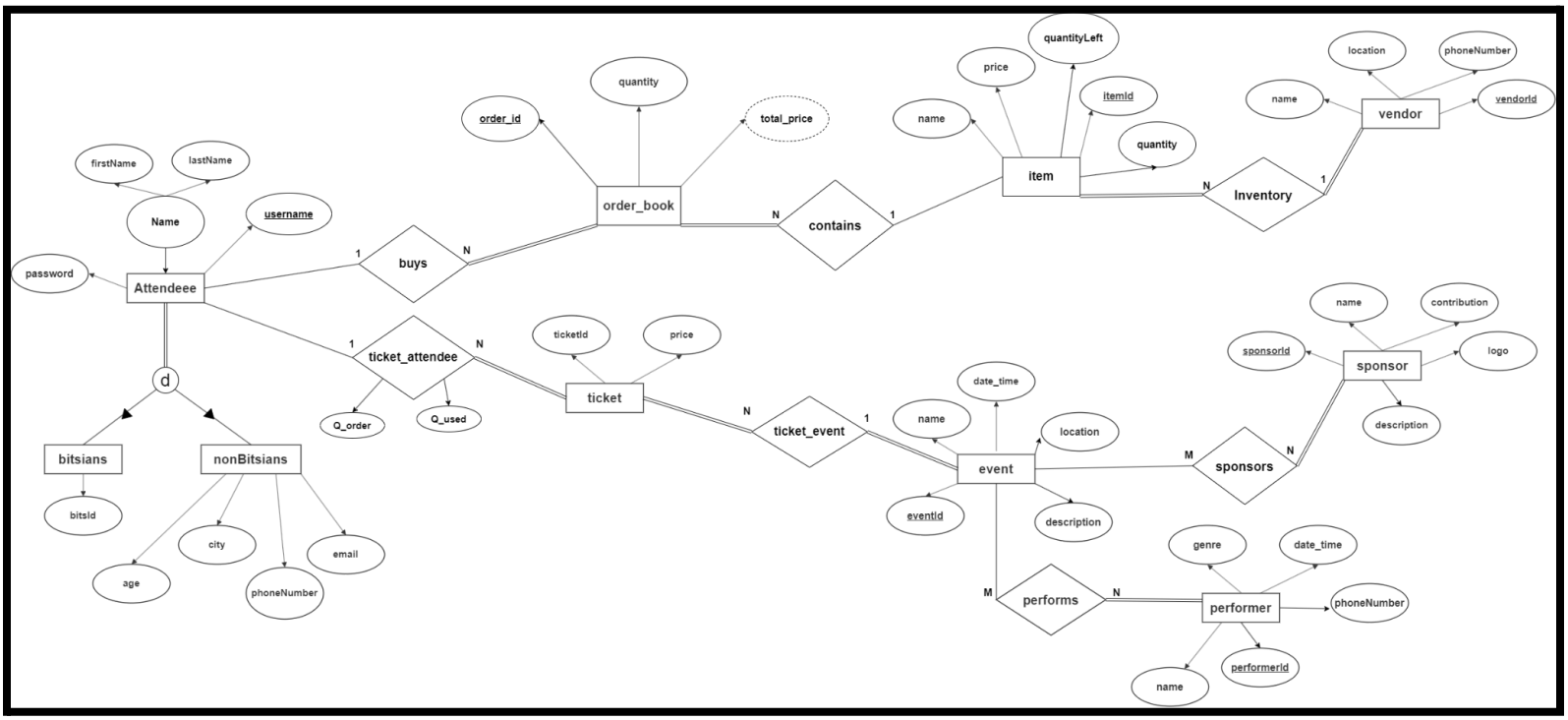
- *Software Configuration*:(Tech Stack)
  - This Gate Management System is developed using MySQL 8.0.32 by Oracle Corporation, as the database
  - Node.js (v18.15.0) for making APIs, backend and connecting MySQL
  - HandleBars, HTML, CSS, JS for the frontend
  - Operating System: Linux, macOS, Windows
  -

VIDEO - https://drive.google.com/drive/folders/1X5m5zRaE37zW2ClCzUHgj1r2o_sRp61X?usp=share_link

Steps to Setup this Project

- Download and install NodeJS on your operating system from here
- To check if it is installed successfully type node —version in your terminal.
- Make sure that MySQL workbench CE is installed on your system. If not, download from here
- Open festManagement.sql in MySQL workbench and execute all the commands.
- questionQuery.sql contains all the queries related to the questions.
- Now, all the Tables, Functions and Procedures will be imported.
- Extract all the contents of festManagement.zip in a folder and cd into that folder.
- Open terminal in that directory
- Run npm install to install the dependencies of the project.
- Change the MySQL username and password in the mysql.js file depending on your system.
- Run npm run start to start the project's frontend.
- Open localhost:3000 on your favorite browser.
- To Quit the project press cmd/ctrl + c in your terminal.

Entity Relationship Diagram



Entity Relationship description:

1. The **<u>Attendee</u>** entity is used to store the details of the people who are going to attend the fest. It has name as a composite attribute with firstName and lastName as its sub-attributes, username and password attributes. **The primary key for this entity is the username attribute**.
2. **<u>Bitsians</u> and <u>nonBitsians</u>** are the **disjoint and total specialization of the Attendee entity**, where both have **username as a primary key**. Attributes for the bitsian entity are bitsId, and the entity for nonBitsians is age, city, phoneNumber and email.
3. The **<u>vendor</u>** is an entity which is used to store vendors that are selling items in the fest. It has name, location, phoneNumber and vendorId attributes and **vendorId as a primary key**.
4. **<u>Item</u>** is an entity used to store the items available for sale during the fest. It has name, price, quantityLeft, itemId and quantity attributes, and **itemId is the primary key**. Quantity shows the total number of items available for sale, and quantityLeft indicates the number of items remaining for purchase during the fest.
5. **<u>Item</u> is in a many-to-one relationship with the <u>vendor</u> entity** where the relationship name is **inventory**. **Item has a total participation in this**

relationship and depicts that each item is being sold by a vendor and it might happen that a vendor does not have any item to sell.

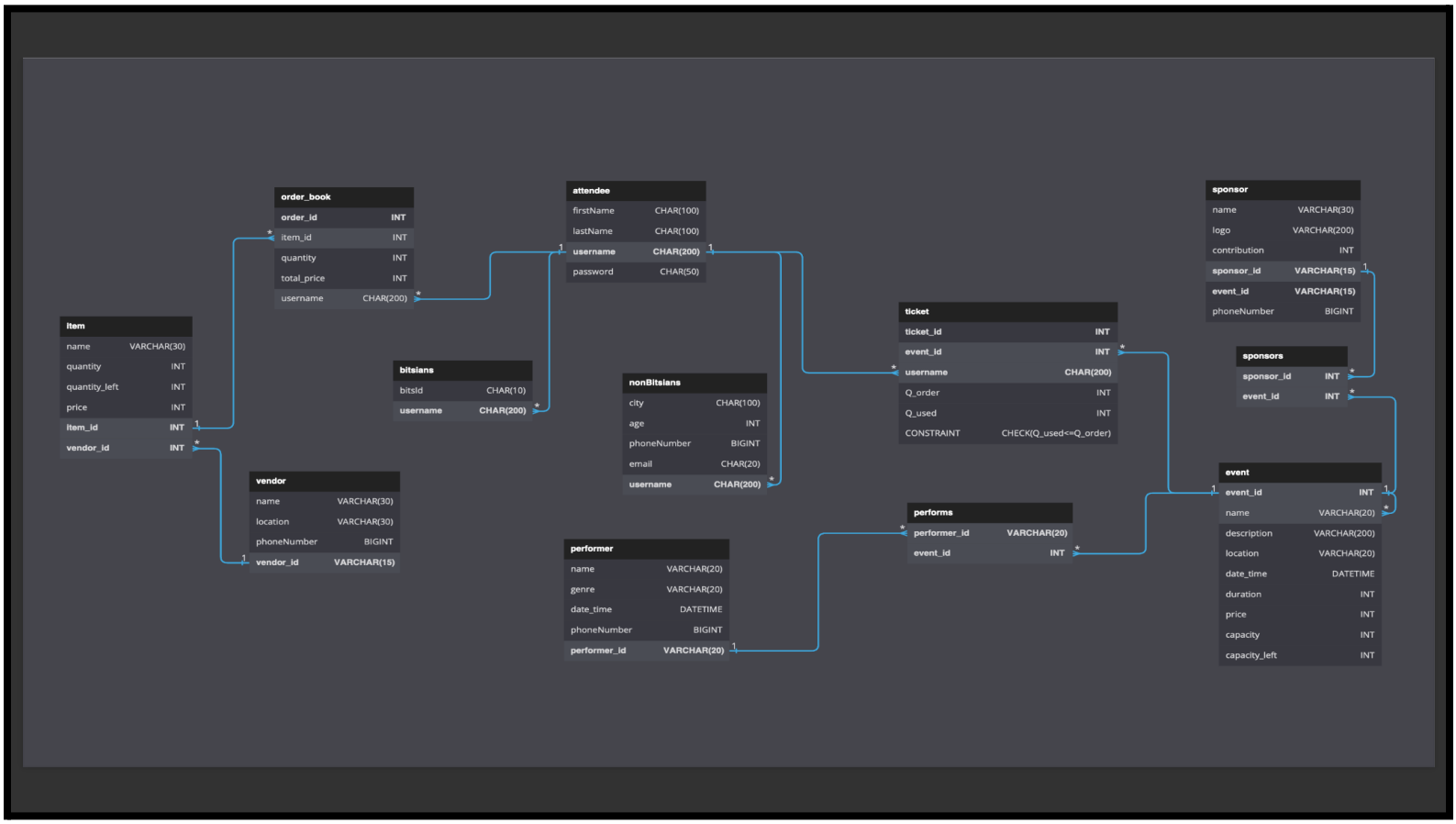
6. **<u>Order_book</u> is an entity used to store orders created by attendees**. It has orderId, quantity and tota_price attributes, and **orderId is the primary key. It is in a many-to-one relationship with the item entity**, **the <u>item</u> entity has total participation in <u>contains</u> relationship** i.e. whenever an order is placed, there must be an element.
7. **<u>Order_book</u>** entity is also in **a many-to-one relationship with the <u>attendee</u> with <u>buys</u> as the relationship name in which it has total participation**.
8. **<u>Sponsor</u>** is an entity to store the sponsors of the fest. It has name, sponsor_id, logo, contribution and description attributes with **sponsor_id as the primary key**.
9. **<u>Event</u>** is an entity to store the events that are in the fest. Name, price, capacity_left,event_id, capacity attribute and **event_id is the primary key**. Capacity is the total number of seats available for sale during the fest, and capacity_left is the number of remaining seats available for sale. Price tells the cost of attending the event.

10. **Event is in many-to-many relationship with the sponsor with sponsors as the relationship name.** In this relationship, **sponsor has total participation** i.e. a sponsor must at least sponsor an event, whereas an event might remain unsponsored.
11. **Performer is an entity to store the performers that will be performing in an event**. It has genre, date_time, phoneNumber, performerId and name attributes, and **performerId is the primary key.**
12. **The performer is in many-to-many relationship with event where performs is relationship name**. **Performer has total participation** in this relationship i.e. performer must at least perform in an event, whereas there might exist an event in which there is no need for performer.
13. **Ticket stores tickets purchased during the fest** and have ticket_id Q_order and Q_used attributes and **ticket_id as the primary key**. Q_order stores the number of tickets purchased by the attendee for a particular event, and Q_used stores the number of tickets used by the user for that specific event.
14. 14. **Ticket is in a many-to-one relationship with Event, with ticket_event as the relationship name**. Ticket has total participation in this relationship. **Ticket is also in many to one relationship with Attendee with relationship name as ticket_attendee** with **total participation of ticket**, since, when a ticket is purchased, it must always have an event and must always be purchased by an attendee.

Assumption: 1. We have assumed that all the events are ticketed.

2. It is assumed that all the data of Bitsians will already be present for the proper functioning of the application.

Relational Schema



1. **attendee**(**<u>username</u>**,password,firstname,lastname)
   1.1. **bitsian**(**<u>username</u>**,password,firstname,lastname,bitsId)
   1.2. **nonBitsian**(**<u>username</u>**,password,firstname,lastname,age,city,phone,email)
2. **sponsor**(**<u>sponsor_id</u>**,name,contribution,logo,phoneNumber)
3. **vendor**(**<u>vendor_id</u>**,name,location,phoneNumber)
4. **item**(**<u>item_id</u>**,vendor_id,name,price,quantity,quantity_left)
5. **event**(**<u>event_id</u>**,name,date_time,duration,description,location,price,capacity,capacity_left)
6. **performer**(**<u>performer_id</u>**,name,genre,date_time,phoneNumber)
7. **order_book**(**<u>order_id</u>**,username,item_id,quantity,total_price)
8. **ticket**(**<u>ticket_id,</u>**event_id,username,Q_order,Q_used)
9. **performs**(**<u>performer_id,event_id</u>**)
10. **sponsors**(**<u>sponsor_id,event_id</u>**)

Redundant Relations -

1.**buys**(**<u>username,order_id</u>**) - As this relation was one to many relationship with total participation on the side of **order_book**, hence the relation was merged with the **order_book** relation while the **order_book** relation maintained its primary key.

2.**contains**(**<u>order_id,item_id</u>**) - As this relation was one to many relationship with total participation on the side of **order_book**, hence the relation was merged with the **order_book** relation while the **order_book** relation maintained its primary key.

3.**inventory**(**<u>vendor_id,item_id</u>**) - As this relation was one to many relationship with total participation on the side of **item**, hence the relation was merged with the **item** relation while the **item** relation maintained its primary key.

4.**ticket_attendee**(**<u>username,ticket_id</u>**,Q_order,Q_used) - As this relation was one to many relationship with total participation on the side of **ticket**, hence the relation was merged with the **ticket** relation while the **ticket** relation maintained its primary key.

Functional Dependencies

- **Bitsian** relation:
  - *{username} -> {password,firstname,lastname,bitsID}*
- **NonBitsian** relation:
  - *{username} -> {password,firstname,lastname,age,city,phone,email}*
- **Sponsor** relation:
  - *{sponsor_id} -> {name,contribution,logo,phoneNumber}*
- **Vendor** relation:
  - *{vendor_id} -> {name,location,phoneNumber}*
- **Item** relation:
  - *{item_id} -> {vendor_id,name,price,quantity,quantity_left}*
- **Event** relation:
  - *{event_id} -> {vendor_id,name,date_time,duration,description,location,price,capacity,capacity_left}*
- **Ticket** relation:
  - *{ticket_id} -> {event_id,username,Q_order,Q_used}*
- **Performs** relation:
  - *{performer_id,vendor_id}*
- **Sponsors** relation:
  - *{sponsor_id,event_id}*

Normalization -

1NF -

1. Each attribute in a table must be atomic
2. Each attribute in a table must contain only a single value
3. Each row in a table must be unique.

Since all of the above conditions satisfy all the relations in the relational schema, the database is already in 1NF.

2NF -

- The table must be in 1NF at this point.
- There must be a primary key for the table.
- The primary key must be a functional dependency of all non-key properties. Any non-key attributes must therefore be dependent on the complete main key, not simply a subset of it.

Since all the above conditions satisfy all the relations in the relational schema,the database is already in 2NF.

3NF -

- The table must be in 2NF at this point.
- No transitive dependencies should exist. In other words, none of the non-key properties in the table should be dependent on any other non-key attributes other than the main key.

Since all the above conditions are satisfied by the relations in the relational schema, the database is already in 3NF.

Concurrency and Consistency

By ensuring that all queries are atomic in nature and so operate similarly to transactions in that they are either fully executed or not at all, the database accomplishes concurrency on a fundamental level. The issue of consistency is eliminated because the queries are processed serially. All internal updates and insertions, such as when a customer orders a ticket or an item, happen simultaneously and aid in keeping the entire database in a consistent state. When and when necessary, the consistency of the data was ensured using internal Update/Delete triggers like Cascading and Restricting.

SQL Queries -

1. **How many tickets were sold for a particular festival event?**

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost`
PROCEDURE TicketsSold(
IN event_id INT UNSIGNED)
DETERMINISTIC
BEGIN
SELECT SUM(Q_order) as Sum_total FROM ticket
WHERE ticket.event_id = event_id;
END$$
DELIMITER ;
call TicketsSold(13001);
```

Sum_total
10

2. **Which festival events are sold out?**

```
SELECT * FROM event
WHERE capacity_left = 0;
```

event_id	name	description	location	date_time	duration	price	capacity	capacity_left
13012	Concert	Music Night	cnot	2023-03-10 15:30:00	2	500	1000	0
13013	Concert	Music Night	cnot	2023-03-10 15:30:00	2	500	1000	0
130014	Concert	Music Night	cnot	2023-03-10 15:30:00	2	500	1000	0
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

3. **What is the total revenue generated from ticket sales for a particular festival event?**

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost`
PROCEDURE RevenueFromAnEvent(
IN event_id INT UNSIGNED)
DETERMINISTIC
```

```
BEGIN
SELECT SUM(event.price*Q_order) as tot_rev FROM ticket
   INNER JOIN event ON ticket.event_id = event.event_id
WHERE ticket.event_id = event_id;
END$$
DELIMITER ;
call RevenueFromAnEvent(13001);
```

tot_rev
5000

4. **Which vendors have applied to participate in a festival?**

```
SELECT * from vendor;
```

name	location	phoneNumber	vendor_id
MomoMia	FD II	7877886622	12001
Biryani By Kilo	FD I	7877833622	12002
Domino's	FD III	7877886644	12003
Kapde	Rotunda	7977886622	12004
Thode Aur Kapde	FD II	7877899622	12005
Icecream	NAB	7877811622	12006
Gaming VR	FD II	7877822622	12007
Keventers	South Park	7877832622	12008
Vada Pav	Lawns	7877867622	12009
Burger King	Clock Tower	7877891622	12010
NULL	NULL	NULL	NULL

5. **How many attendees have purchased tickets for a specific festival event?**

```
DELIMITER $
CREATE DEFINER=`root`@`localhost`
PROCEDURE NumAttendeesEvent(
IN event_id INT UNSIGNED)
```

```
DETERMINISTIC
BEGIN
SELECT SUM(Q_order) AS count FROM ticket
WHERE ticket.event_id = event_id;
END$$
DELIMITER ;
CALL NumAttendeesEvent(13001);
```

count
10

6. **Who are the sponsors for a particular event if there are any?**

```
DELIMITER $
CREATE DEFINER=`root`@`localhost`
PROCEDURE sponsorsEvent(
IN event_id INT UNSIGNED)
DETERMINISTIC
BEGIN
SELECT * FROM sponsors NATURAL JOIN sponsor
WHERE sponsors.event_id = event_id;
END$$
DELIMITER ;
CALL sponsorsEvent(13001);
```

sponsor_id	event_id	name	logo	contributi...	phoneNumber
11001	13001	Red Bull	/random file name	1000000	9173689112

7. **Which festival events have the highest attendance?**

```
WITH T(event_id,name,tot) AS
(SELECT event.event_id,event.name,SUM(Q_order)
 FROM event NATURAL JOIN ticket
 GROUP BY event.event_id,event.name),
```

```
M(max_value) AS
(SELECT MAX(tot) FROM T)
SELECT T.event_id,T.name,T.tot
FROM T,M
WHERE T.tot = M.max_value;
```

event_id	name	tot
13001	Concert	10
13006	Robo Wars	10
13007	Drone Show	10
13008	Drone Race	10
13009	One more Dance Show	10
13010	One more concert	10

8. **What is the average ticket price for a specific event?**

```
SELECT price FROM ticket NATURAL JOIN event
WHERE event.event_id = "13001";
```

price
500

9. **Which performers are scheduled to perform at a particular event?**

```
DELIMITER $
CREATE DEFINER=`root`@`localhost`
PROCEDURE performersInEvent(
IN event_id INT UNSIGNED)
DETERMINISTIC
BEGIN
SELECT event.event_id,event.name,performer.performer_id,performer.name,performer.genre
FROM performs,performer,event
WHERE performs.performer_id = performer.performer_id
AND event.event_id = performs.event_id
AND event.event_id = event_id;
END$$
```

DELIMITER ;
CALL performersInEvent(13001);

event_id	name	performer_...	name	genre
13001	Concert	14001	Kalesh...	Fighting
13001	Concert	14002	Amit Ter...	Singing

10. What is the total number of tickets sold for all festival events combined?

SELECT SUM(Q_order) FROM ticket;

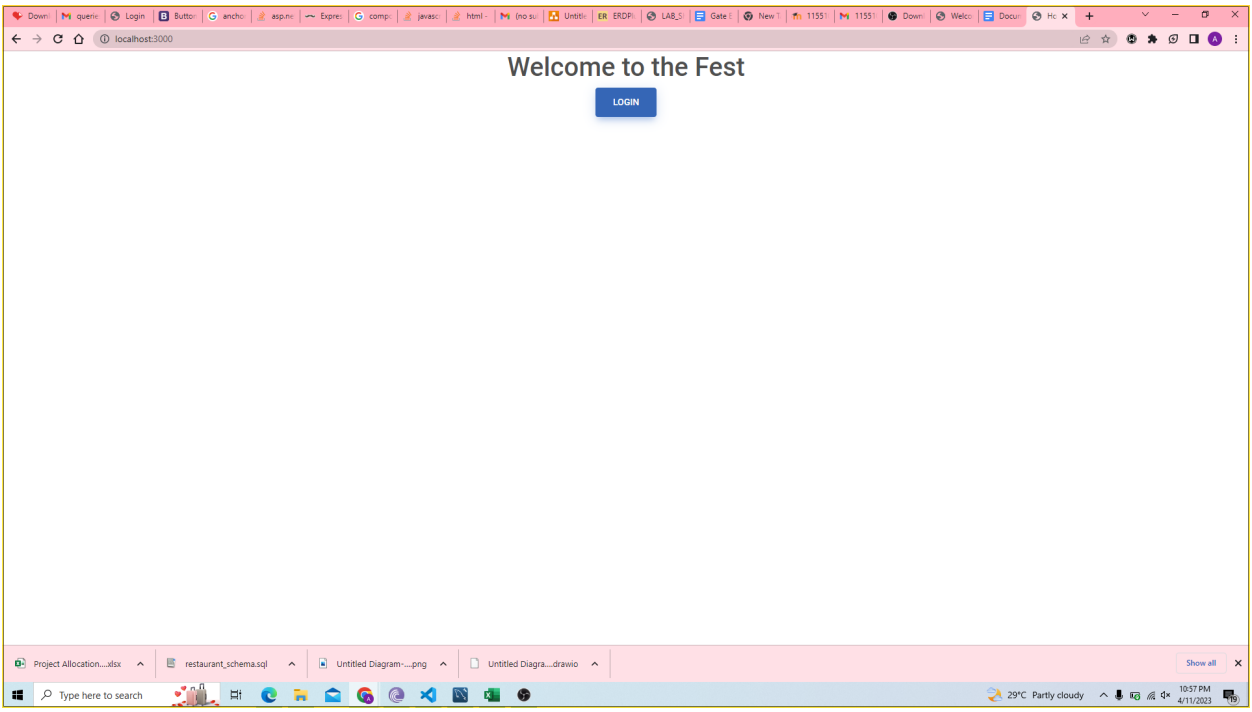
SUM(Q_order)
72

11. All the events on a particular date?

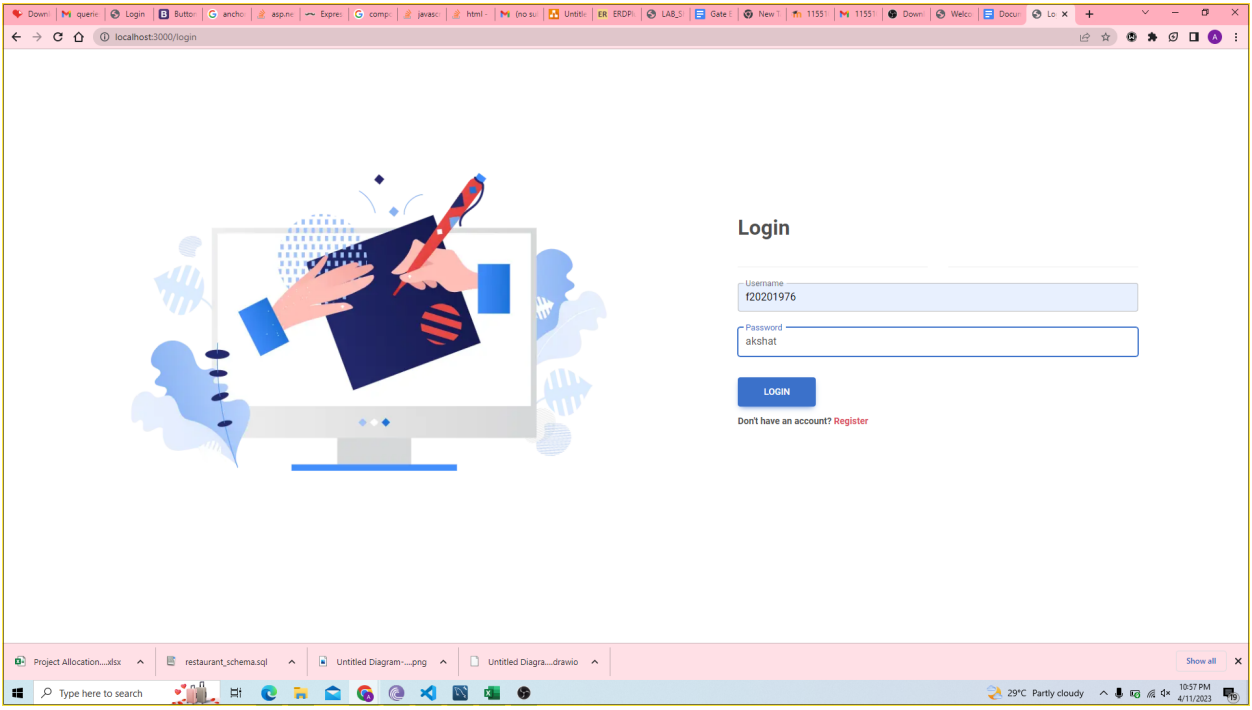
SELECT * FROM event
WHERE CAST(event.date_time as DATE) = "2023-03-10";

event_id	name	description	location	date_time	duration	...	capacity	capacity_left
13001	Concert	Music Night	cnot	2023-03-10 15:30:00	2	...	1000	990
13004	Dance Show	Dance Show	cnot	2023-03-10 15:30:00	2	...	1000	998
13005	Rap Battle	Rap Battle	cnot	2023-03-10 15:30:00	3	...	1000	996
13006	Robo Wars	Robo Wars	cnot	2023-03-10 15:30:00	1	...	1000	990
13007	Drone Show	Drone Show	cnot	2023-03-10 15:30:00	2	...	1000	990
13008	Drone Race	Drone Race	cnot	2023-03-10 15:30:00	3	...	1000	990
13009	One more Dance Show	Dance Show	cnot	2023-03-10 15:30:00	2	...	1000	990
13010	One more concert	Music Night	cnot	2023-03-10 15:30:00	1	...	1000	990
13011	Test No performer	Music Night NO performer	budh	2023-03-10 15:30:00	1	...	1000	1000
13012	Concert	Music Night	cnot	2023-03-10 15:30:00	2	...	1000	0
13013	Concert	Music Night	cnot	2023-03-10 15:30:00	2	...	1000	0
130014	Concert	Music Night	cnot	2023-03-10 15:30:00	2	...	1000	0
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

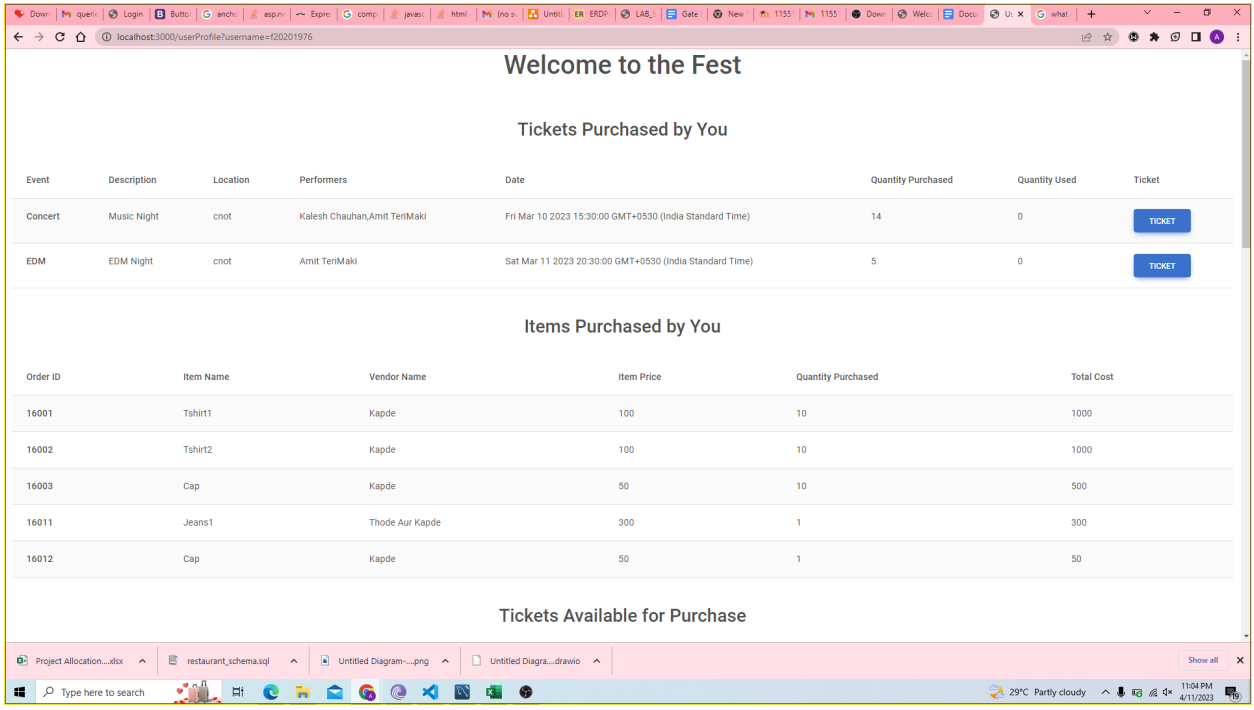
Frontend Documentation



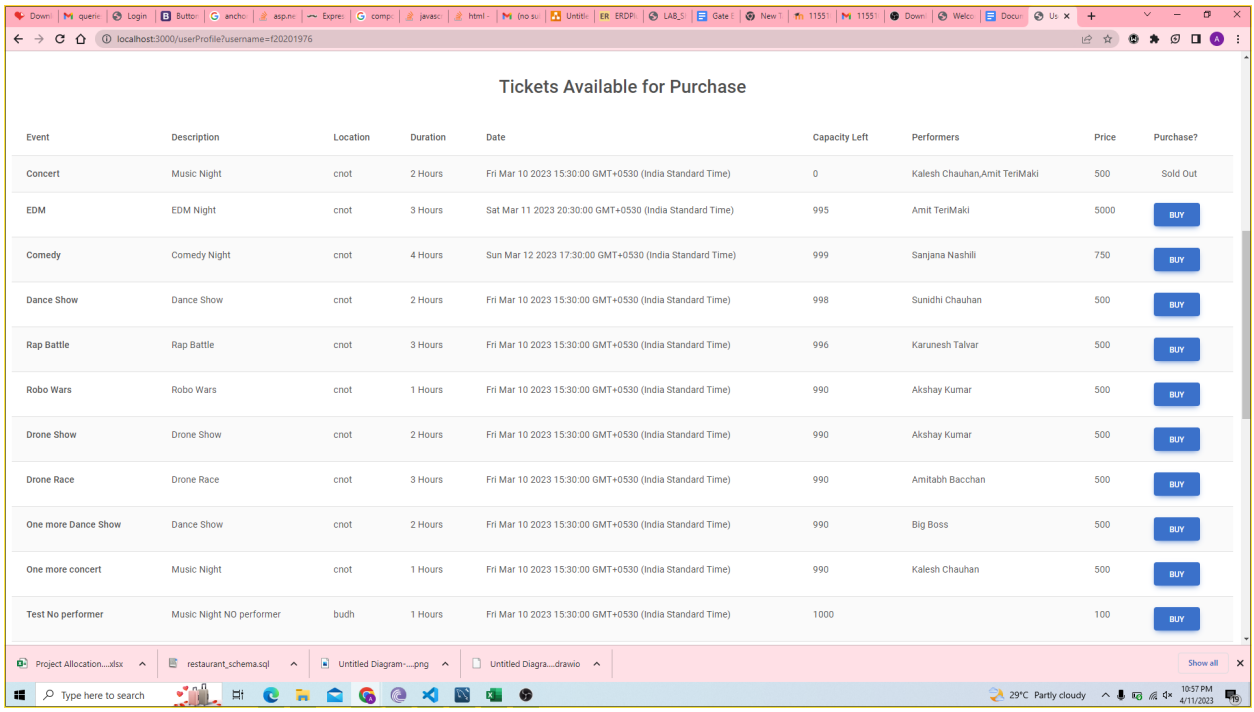
Img 1. Home page of the application contains the link to open the login page for both Bitsians and Nonbitsians.



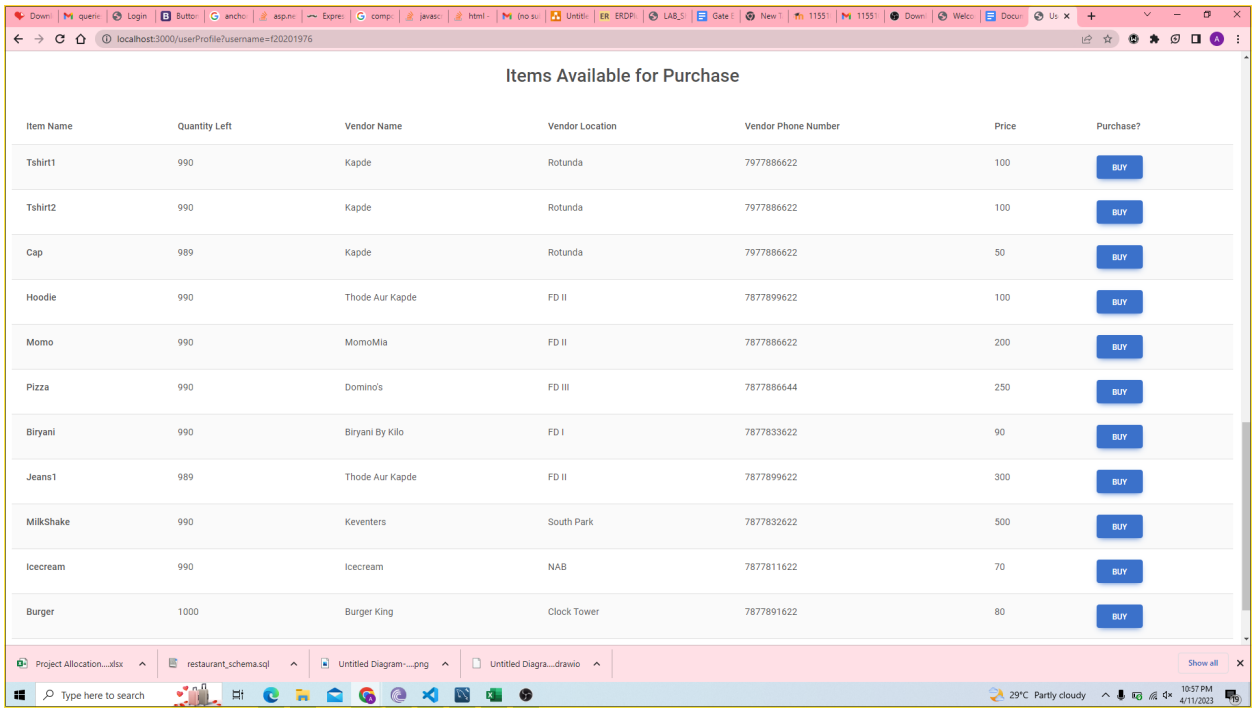
Img 2. Login page where user can enter username password to login to the profile page.



Img 3. Profile page contains all the Tickets purchased by the user and all the orders he made



Img 4. Profile page shows the list of events that are taking place in the fest and a link to purchase tickets for the same. If there is no capacity left the sold out is shown instead of the buy button. Purchasing the tickets update the purchased tickets' information above.



Items Available for Purchase

Item Name	Quantity Left	Vendor Name	Vendor Location	Vendor Phone Number	Price	Purchase?
Tshirt1	990	Kapde	Rotunda	7977886622	100	BUY
Tshirt2	990	Kapde	Rotunda	7977886622	100	BUY
Cap	989	Kapde	Rotunda	7977886622	50	BUY
Hoodie	990	Thode Aur Kapde	FD II	7877899622	100	BUY
Momo	990	MomoMia	FD II	7877886622	200	BUY
Pizza	990	Domino's	FD III	7877886644	250	BUY
Biryani	990	Biryani By Kilo	FD I	7877833622	90	BUY
Jeans1	989	Thode Aur Kapde	FD II	7877899622	300	BUY
MilkShake	990	Keventers	South Park	7877832622	500	BUY
Icecream	990	Icecream	NAB	7877811622	70	BUY
Burger	1000	Burger King	Clock Tower	7877891622	80	BUY

Img 5. Profile page shows the list of items that are available to purchase during the fest and a link to purchase the same. If there is no quantity left the sold out is shown instead of the buy button. Purchasing the item updates the orders' information above.

Backend Integration

1. To integrate the backend with the database we used the mysql2 library of nodejs to connect to the database and to execute the queries.
2. We created stored functions and procedures for all the queries and tasks which we needed to on the database and we are calling those functions in our backend through nodejs library.
3. We have implemented transactions in the purchase ticket and purchase item function since there are multiple update statements which need to be executed all or none.

Handled Edge Cases

We have tried to handle as many edge cases as possible few of which are:

1. We have written an authentication function in MySQL which returns false boolean when credentials are wrong. The same has been handled in the frontend where a red error banner appears when entered credentials are wrong.
2. purchase_ item and purchase_ticket procedures are used to purchase ticket and items in our application which implement transaction and check for the case if the items or tickets are remaining before purchasing or making changes to the database and stores the result of the same in out variable status, which is 1 is the purchase was successful(i.e. Transaction committed) and 0 if purchase was unsuccessful (i.e. transaction roll backed).
3. All the DDL statements were written carefully following all the foreign key constraints and relationships.

Future Improvements

- Registration page will be added for the registration of non Bitsian students.
- Admin portal will be added for organizers to manage all the aspects of the festival.
- Currently there is no provision to use the purchased tickets, which will be added in the future iterations.
- Application can be made more attractive.