SECD2523 - 03

DATABASE

PROJECT PHASE 3

GROUP NAME : SPICE GIRLS

NO	NAME	MATRIC NUMBER
1	NOOR ALIYAH BINTI AHMAD SAUFI	A22EC0234
2	NUHA BINTI MOHAMED	A22EC0238
3	SERI NUR AYUNI BINTI SALIMI	A22EC0268
4	PUTERI NURIN I'RDINA BINTI FADZIL	A22EC0261
5	LUVLYNN JULIEAN DE JONES	A22EC0190

TABLE OF CONTENTS

1.0 Introduction

NexScholar is an app working for a better social networking platform and is designed for groups of people targeting students, academics, researchers and industry professionals. Created and curated by Dr. Ahmad Najmi and Dr. Muhammad Aliff from the Faculty of Computing at Universiti Teknologi Malaysia, in collaboration with Dr. Seah Choon Sen from Universiti Tun Razak, NexScholar provides a range of modules, such as events, scholarships and expert listings in one platform to ease and meet the needs of users within the academic and professional scope.

As NexScholar is striving to be a better app for all, they face difficulties and lacking in their features, and the solution to the problems surfacing is to create a better system – an integrated booking system as well as a centralized ticketing system. Prior to this, the users of this system need to manually go through processes of booking an event, hence this project focuses on developing a seamless and user-friendly booking system with the aim of making the current system more convenient and smooth.

In addition to that, the absence of a centralized ticketing system has led the system to have inefficient tracking of attendee reservations, availability and payments. Manual processes contribute to delays and potential errors in collecting and storing data of attendees. Furthermore, the platform lacks an automated communication system, as the users need to manually exert the effort to find information and confirm their choices. All in all, NexScholar does not have a feedback mechanism that could be useful to collect user's satisfaction evaluation and to identify areas of enhancement within the system.

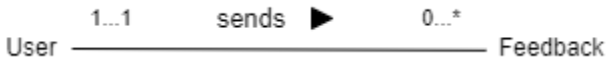
By addressing these issues and current drawbacks, this project aims to revolutionize the event registration process within NexScholar. This implementation will assist administrators in managing registrations, monitoring payments and analyzing attendee data as well as to allow users to have a better experience in using this platform.

2.0 Overview of project

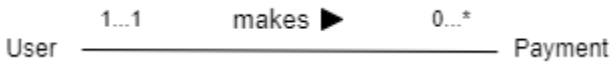
As part of this project, the conceptual Entity-Relationship Diagram (ERD) that was created in phase 3 will be transformed into a logical Entity-Relationship Diagram (ERD). The team intends to eliminate non-relational characteristics like many-to-many and complex relationships in order to achieve this goal. Following the completion of the logical ERD development, the relation schemas will be extracted and normalization will be performed using the Boyce-Codd Normal Form (BCNF) standard. Following the completion of this procedure, the final logical ERD will be generated, which will provide an accurate representation of the BCNF relations and will be ready to serve as the basis for the subsequent steps. Furthermore, the group will assess the logical ERD in relation to the transaction requirements of the system and revise the data dictionary based on the generated normalized relations. This will guarantee that the design satisfies all requirements and is suitable for the system. This method aids in the development of a reliable, accurate, and effective database design.

3.0 Database Conceptual Design

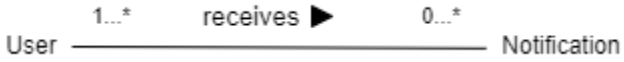
3.1 Updated Business Rules



- Each user can send zero or many feedback.
- Each feedback can be sent by a user.



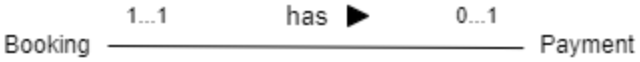
- Each user can make zero or many payments.
- Each payment can be made by only one user.



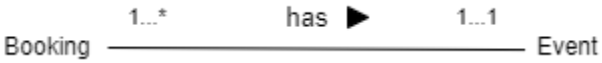
- Each user receives zero or many notifications.
- Each notification is received by one or more users.

- Each user attends one or many events.
- Each event is attended by one or many users.

- Each user makes one or many bookings.
- Each booking is made by one user.



- Each booking has zero or many payments.
- Each payment is made by one booking.



- Each booking has one event.
- Each event can have one or many bookings.

3.2 Conceptual ERD

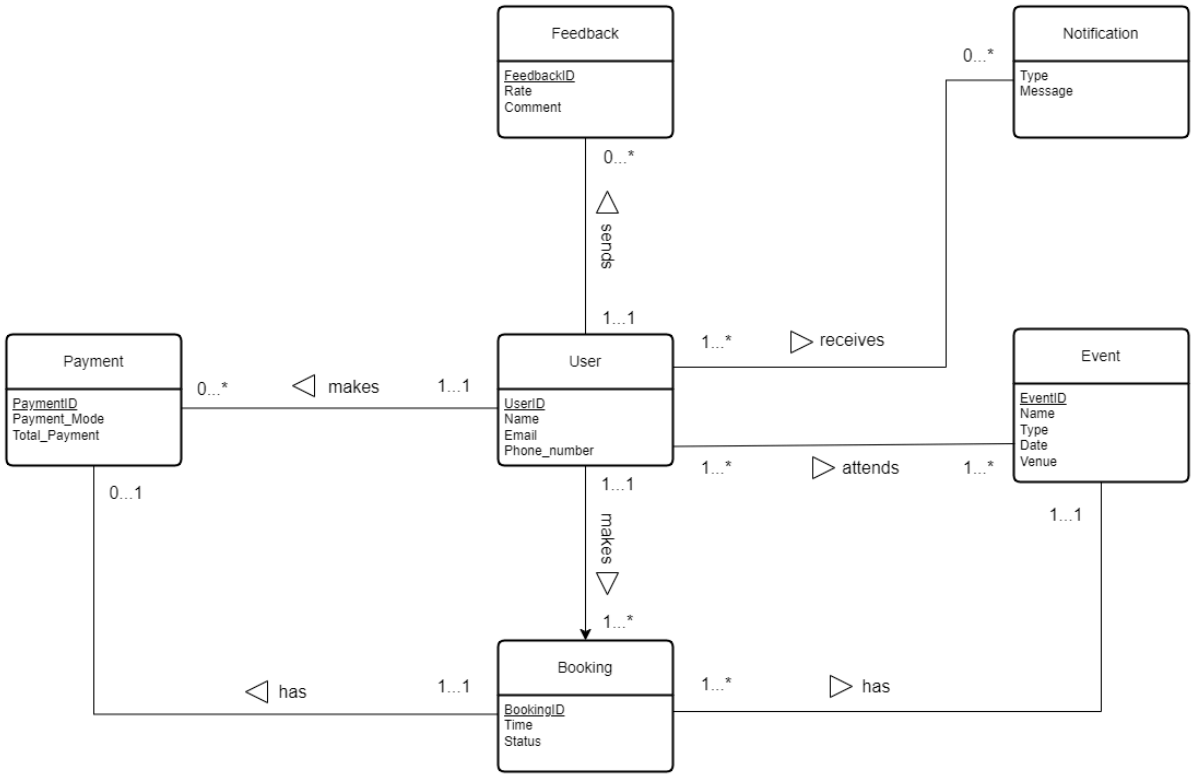


Figure 4.2: ERD of the Proposed System

The figure above illustrates the ERD for our proposed systems. It is crucial to implement an ERD in order to illustrate and identify the entities, relationships, primary keys, and myriad other information. For example, the instances of the said entities as shown in the ERD are User, Event, Booking, Payment, Feedback, and Notification.

3.2.1 Enhanced ERD (EERD)

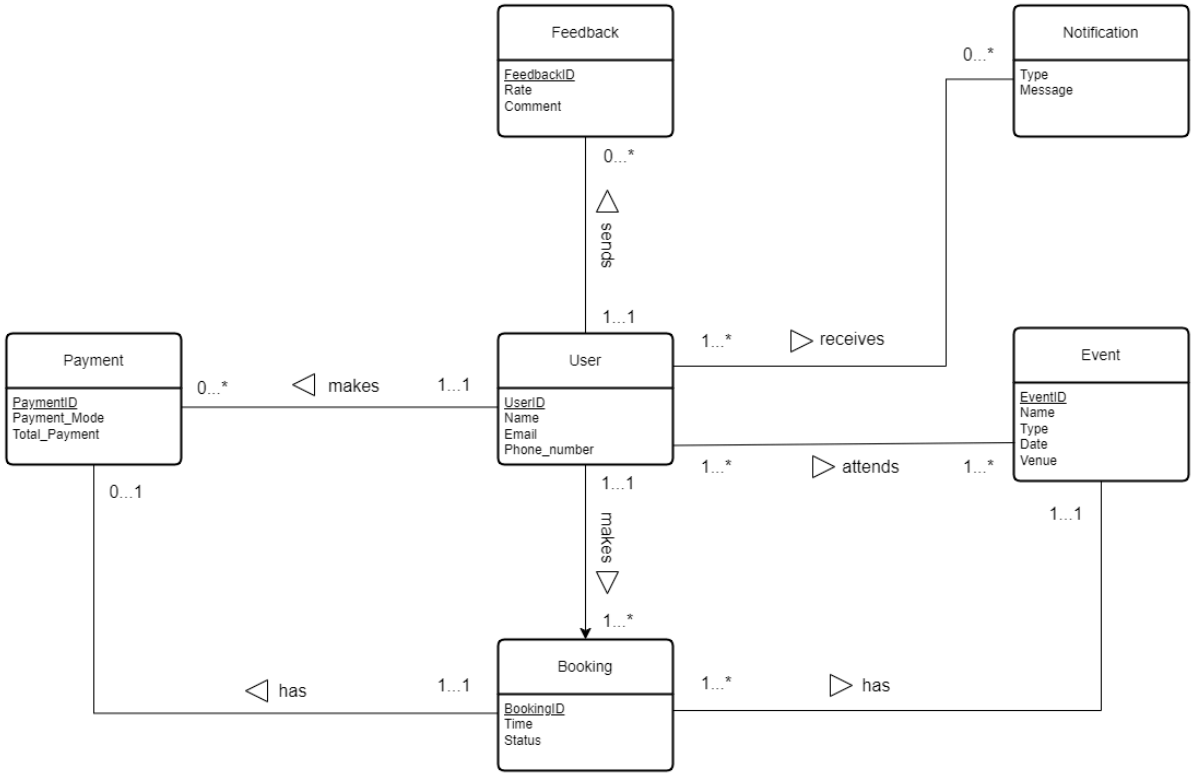


Figure 4.2.1: Enhanced ERD (EERD) of the Proposed System

The figure above illustrates the EERD for our proposed systems. It guides us to provide an enhanced visualization of the relationship between different entities in the system, by further incorporating additional concepts that are able to represent requirements of the system. However, because our system does not employ the specialization/generalization concept, the ERD and EERD are technically the same.

4.0 Database Logical Design

4.1 Logical ERD

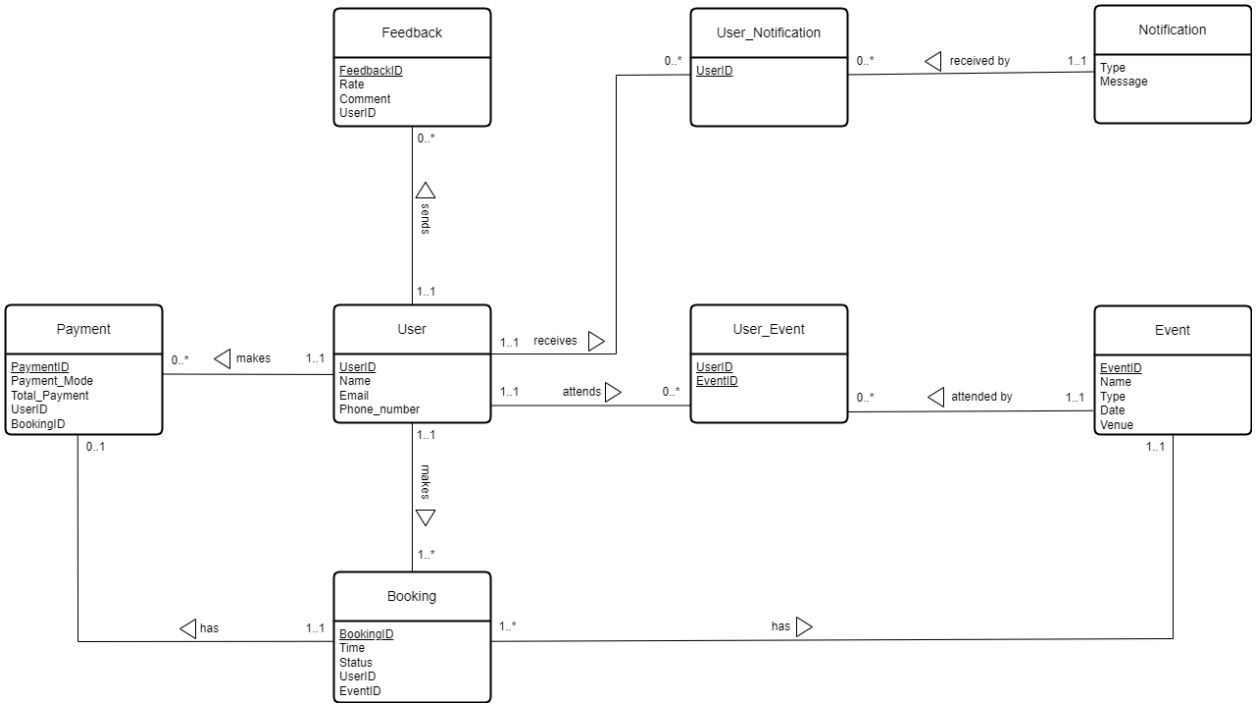


Figure 4.1 Logical ERD of the Proposed System

The figure above illustrates the logical ERD for our proposed systems. The ERD has been altered in order to provide a logical data model that represents the entities, relationships, and attributes identified in the previous conceptual ERD.

4.2 Updated Data Dictionary

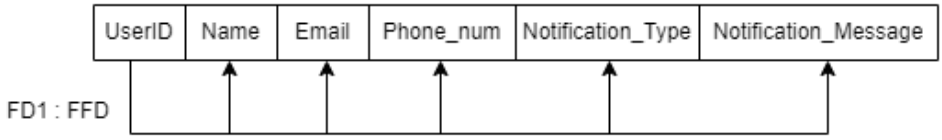
Entity name	Attributes	Data type & length	Nulls	Multi-valued	Multiplicity	Relationship	Entity name	Multiplicity
User	UserID	5 variable characters	No	No	1..1	sends	Feedback	0..*
	Name	30 variable characters	No	No	1..1	receives	User_Notification	0..*
	Email	30 variable characters	No	No	1..1	attends	User_Event	1..*
	Phone_number	11 numbers	No	No	1..1	makes	Payment	1..*
Event	EventID	5 variable characters	No	No	1..1	attended by	User_Event	0..*
	Name	20 variable characters	No	No	1..1	owned by	Booking	1..*
	Type	15 variable characters	No	No				
	Date	Date	No	No				
	Venue	30 variable characters	No	No				
Booking	BookingID	5 variable characters	No	No	1..*	made by	User	1..1
	Time	Date time	No	No	1..1	has	Payment	0..1
	Status	10 variable characters	No	No	1..*	has	Event	1..1
	User ID	5 variable	No	No	1..1	owned by	User	

		characters						
	EventID	5 variable characters	No	No	1..*	owned by	Event	
Payment	PaymentID	5 variable characters	No	No	0..*	made by	User	1..1
	Payment_mode	10 variable characters	No	No	0..1	owned by	Booking	1..1
	Total_Payment	5 number (2 digits for scale)	Yes	No				
	UserID	5 variable characters	No	No	0..*	owned by	User	1..1
	BookingID	5 variable characters	No	No	0..1	owned by	Booking	1..1
Feedback	FeedbackID	5 variable characters	No	No	0..*	sent by	User	1..1
	Rate	5 numbers	No	No				
	Comment	100 variable characters	Yes	No				
	UserID	5 variable characters	No	No	0..*	owned by	User	1..1
Notification	Type	15 variable characters	No	No	1..1	received by	User_Notification	0..*
	Message	500 variable characters	No	No				
User_Notification	UserID	5 variable characters	No	No	0..*	received by	User	1..1

Table 4.2 Updated Data Dictionary of the Proposed System

4.3 Normalization

4.3.1 Normalization for UserNotification



1NF:

Functional Dependency 1: Fully Functional Dependency

UserID → Name, Email, Phone_num, Notification_Type, Notification_Message

UserNotification(UserID, Name, Email, Phone_num, Notification_Type, Notification_Message)
Primary Key: UserID

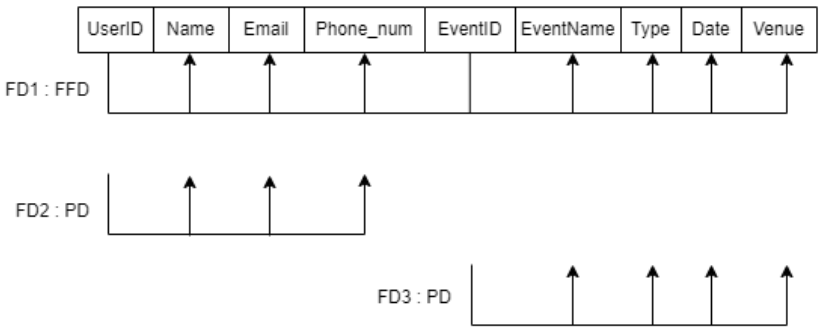
2NF:

There is no partial dependency so the second normalization form is the same as the first normalization form.

3NF:

There is no transitive dependency so the third normalization form is the same as the second normalization form.

4.3.2 Normalization for UserEvent



1NF:

Functional Dependency 1: Fully Functional Dependency

UserID, EventID → Name, Email, Phone_num, EventName, EventType, Date, Venue

UserEvent(UserID, EventID, Name, Email, Phone_num, EventName, EventType, Date, Venue)
Primary Key: UserID, EventID

2NF:

Functional Dependency 2: Partial Dependency

UserID → Name, Email, Phone_num

Functional Dependency 3: Partial Dependency

EventID → EventName, EventType, Date, Venue

User(UserID, Name, Email, Phone_num)
Primary Key: UserID

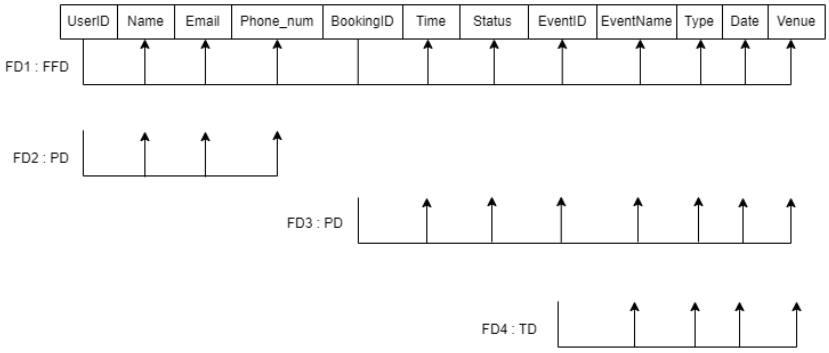
Event(EventID, EventName, EventType, Date, Venue)
Primary Key: EventID

UserEvent(UserID, EventID)
Primary Key: UserID, EventID
Foreign Key: UserID references User(UserID)
Foreign Key: EventID references Event(EventID)

3NF:

There is no transitive dependency so the third normalization form is the same as the second normalization form.

4.3.3 Normalization for UserBookingEvent



1NF:

Functional Dependency 1: Fully Functional Dependency

UserID, BookingID → Name, Email, Phone_num, Time, Status, EventID, EventName, EventType, Date, Venue

UserBookingEvent(UserID, BookingID, Name, Email, Phone_num, Time, Status, EventID, EventName, EventType, Date, Venue)
Primary Key: UserID, BookingID

2NF:

Functional Dependency 2: Partial Dependency

UserID → Name, Email, Phone_num

Functional Dependency 3: Partial Dependency

BookingID → Time, Status, EventID, EventName, EventType, Date, Venue

User(UserID, Name, Email, Phone_num)
Primary Key: UserID

BookingEvent(BookingID, Time, Status, EventID, EventName, EventType, Date, Venue)
Primary Key: BookingID

UserBookingEvent(UserID, BookingID)

Primary Key: UserID, BookingID

Foreign Key: UserID references User(UserID)

Foreign Key: BookingID references BookingEvent(BookingID)

3NF:

Functional Dependency 4: Transitive Dependency

EventID → EventName, EventType, Date, Venue

User(UserID, Name, Email, Phone_num)

Primary Key: UserID

Event(EventID, EventName, EventType, Date, Venue)

Primary Key: EventID

BookingEvent(BookingID, Time, Status, EventID)

Primary Key: BookingID

Foreign Key: EventID

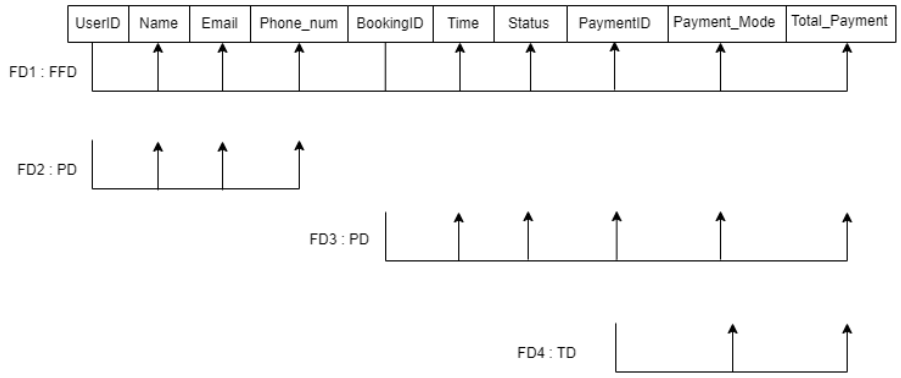
UserBookingEvent(UserID, BookingID)

Primary Key: UserID, BookingID

Foreign Key: UserID references User(UserID)

Foreign Key: BookingID references BookingEvent(BookingID)

4.3.4 Normalization for UserBookingPayment



1NF:

Functional Dependency 1: Fully Functional Dependency

UserID, BookingID → Name, Email, Phone_num, Time, Status, PaymentID, Payment_Mode, Total_Payment

UserBookingPayment(UserID, PaymentID, Name, Email, Phone_num, Time, Status, PaymentID, Payment_Mode, Total_Payment)
Primary Key: UserID, BookingID

2NF:

Functional Dependency 2: Partial Dependency

UserID → Name, Email, Phone_num

Functional Dependency 3: Partial Dependency

BookingID → Time, Status, PaymentID, Payment_Mode, Total_Payment

User(UserID, Name, Email, Phone_num)
Primary Key: UserID

BookingPayment(BookingID, Time, Status, PaymentID, Payment_Mode, Total_Payment)
Primary Key: BookingID

UserBookingPayment(UserID, BookingID)

Primary Key: UserID, BookingID

Foreign Key: UserID references User(UserID)

Foreign Key: BookingID references BookingPayment(BookingID)

3NF:

Functional Dependency 4: Transitive Dependency

PaymentID → PaymentID, Payment_Mode, Total_Payment

User(UserID, Name, Email, Phone_num)

Primary Key: UserID

Payment(PaymentID, Payment_Mode, Total_Payment)

Primary Key: PaymentID

BookingPayment(BookingID, Time, Status, PaymentID)

Primary Key: BookingID

Foreign Key: PaymentID

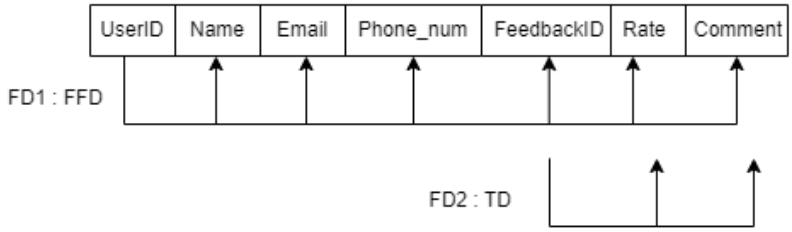
UserBookingPayment(UserID, BookingID)

Primary Key: UserID, BookingID

Foreign Key: UserID references User(UserID)

Foreign Key: BookingID references BookingPayment(BookingID)

4.3.5 Normalization for UserFeedback



1NF:

Functional Dependency 1: Fully Functional Dependency

UserID → Name, Email, Phone_num, FeedbackID, Rate, Comment

UserFeedback(UserID, Name, Email, Phone_num, FeedbackID, Rate, Comment)
Primary Key: UserID

2NF:

There is no partial dependency so the third normalization form is the same as the second normalization form.

3NF:
Functional Dependency 2: Transitive Dependency

FeedbackID → Rate, Comment

Feedback(FeedbackID, Rate, Comment)
Primary Key: FeedbackID

UserFeedback(UserID, Name, Email, Phone_num, FeedbackID)
Primary Key: UserID
Foreign Key: FeedbackID references Feedback(FeedbackID)

5.0 Relational DB Schemas (after normalization)

User(UserID, Name, Email, Phone_num)

PK: UserID

Event(EventID, EventName, EventType, Date, Venue)

PK: EventID

Feedback(FeedbackID, Rate, Comment)

PK: FeedbackID

Payment(PaymentID, Payment_Mode, Total_Payment)

PK: PaymentID

UserNotification(UserID, Name, Email, Phone_num, Notification_Type, Notification_Message)

PK: UserID

UserEvent(UserID, EventID)

PK: UserID, EventID

FK: UserID references User(UserID)

FK: EventID references Event(EventID)

UserFeedback(UserID, Name, Email, Phone_num, FeedbackID)

PK: UserID

FK: FeedbackID references Feedback(FeedbackID)

BookingEvent(BookingID, Time, Status, EventID)

PK: BookingID

FK: EventID references Event(EventID)

BookingPayment(<u>BookingID</u>, Time, Status, PaymentID)

PK: BookingID

FK: PaymentID references Payment(PaymentID)

UserBookingEvent(<u>UserID</u>, <u>BookingID</u>)

PK: UserID, BookingID

FK: UserID references User(UserID)

FK: BookingID references BookingEvent(BookingID)

UserBookingPayment(<u>UserID</u>, <u>BookingID</u>)

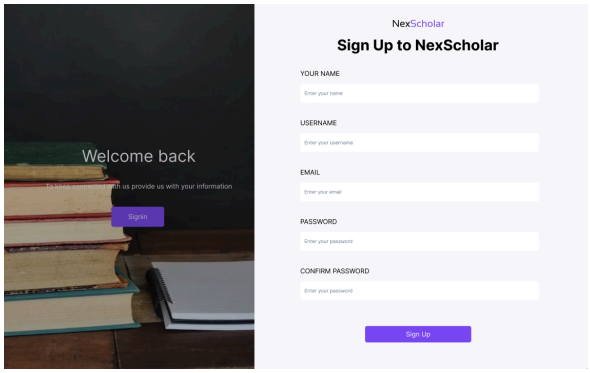
PK: UserID, BookingID

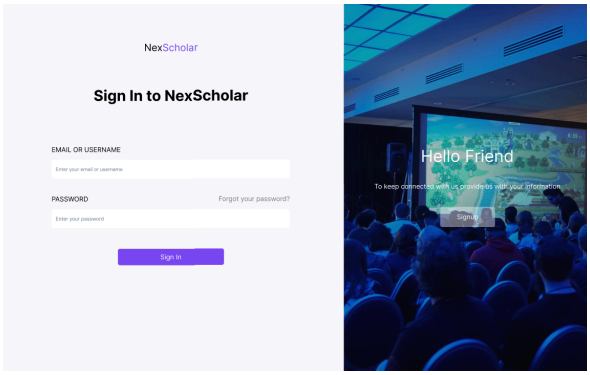
FK: UserID references User(UserID)

FK: BookingID references BookingPayment(BookingID)

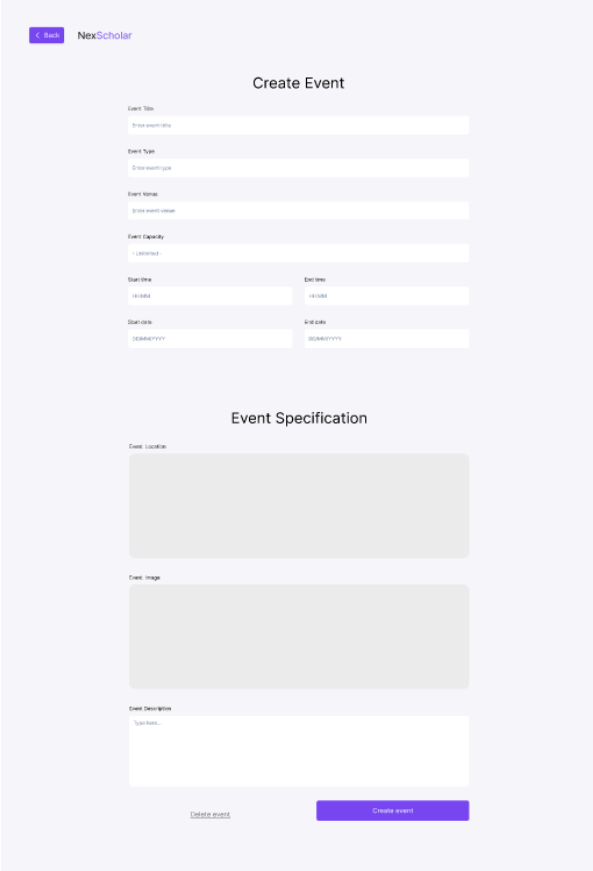
6.0 Interfaces

6.1 Login

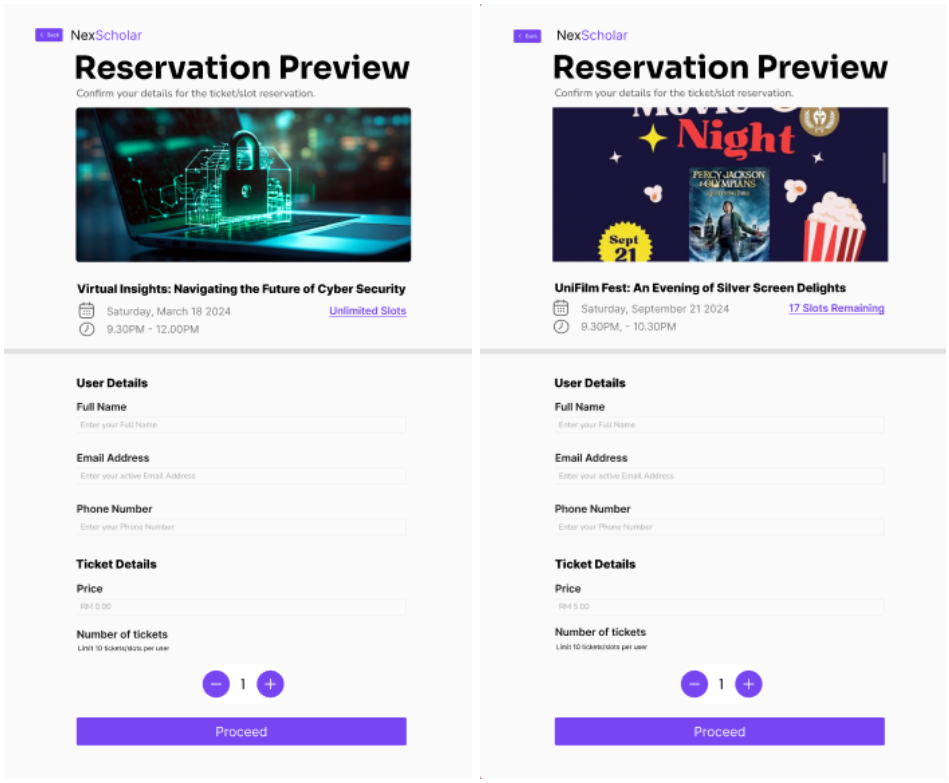


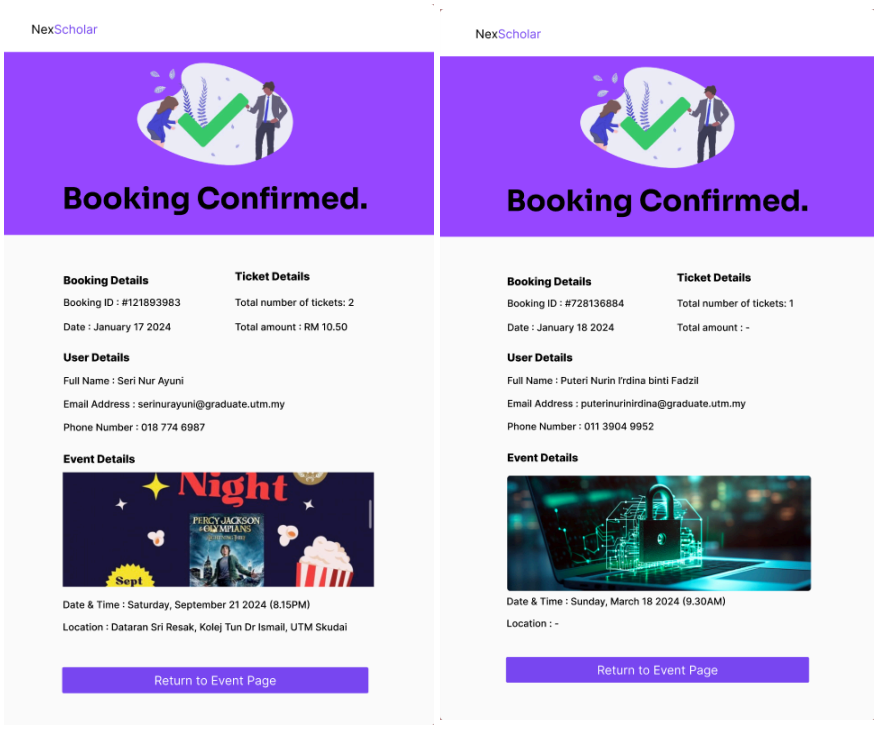


6.2 Create Event

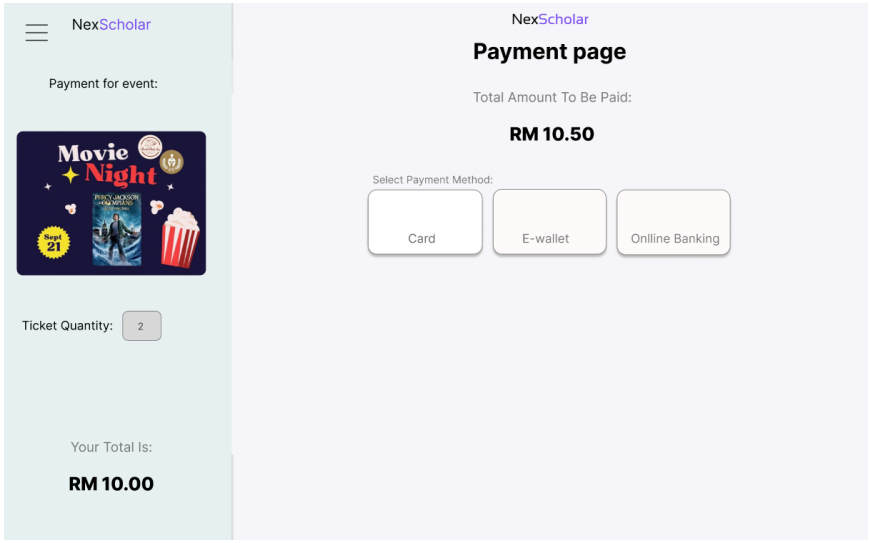


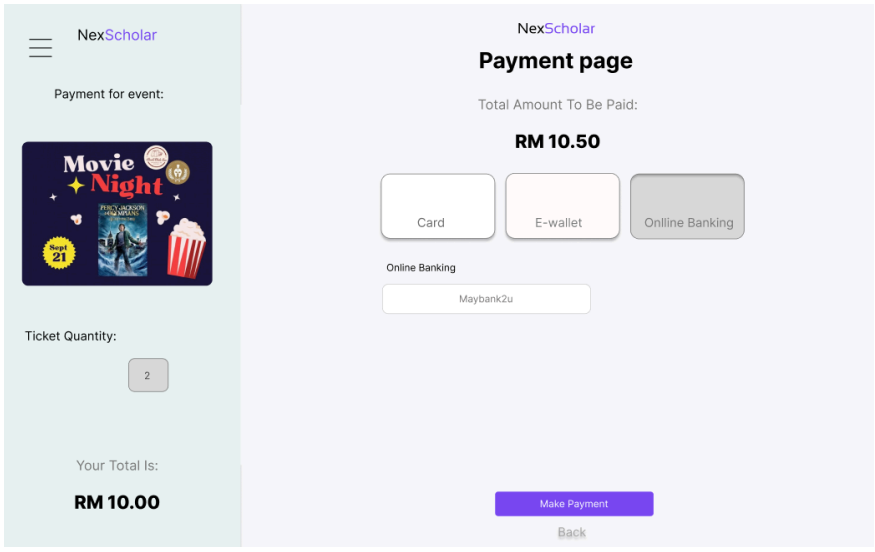
6.3 Make Booking

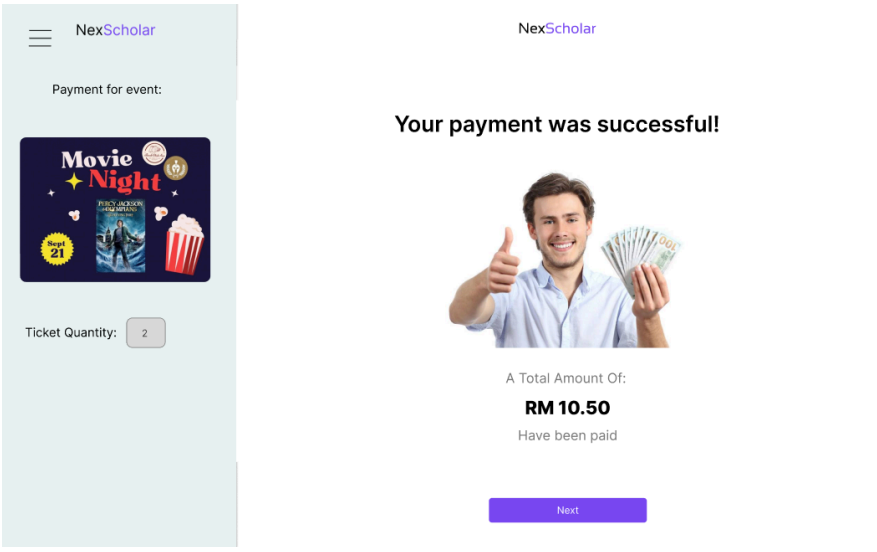




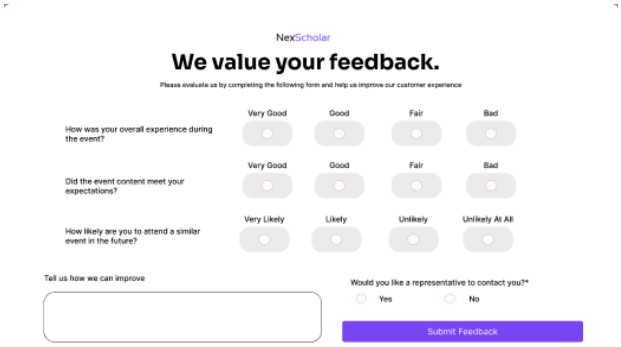
6.4 Make Payment

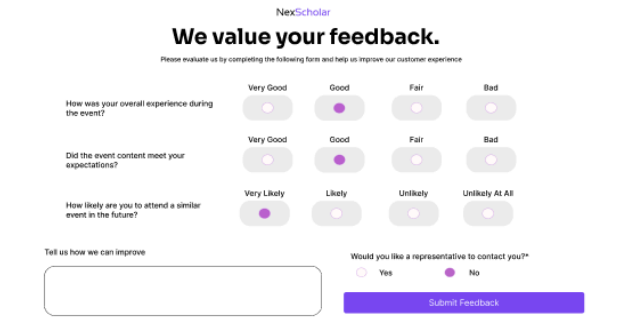


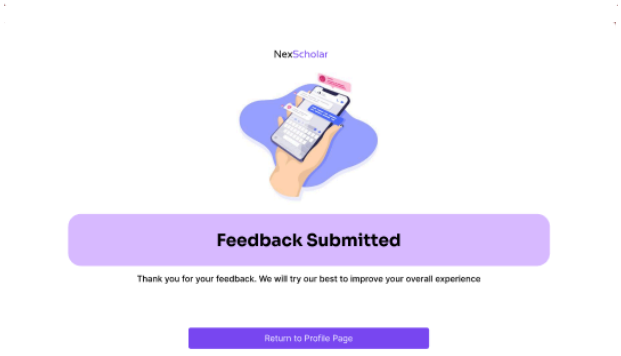




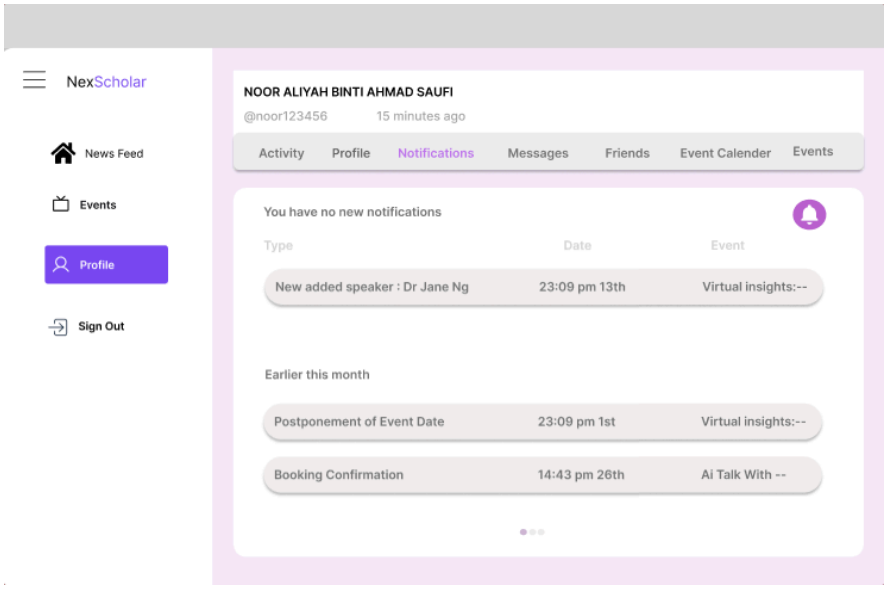
6.5 Leave a Feedback







6.6 Receive Notification



7.0 SQL Statements (DDL & DML)

DDL:

CREATE TABLE Users

```
CREATE TABLE Users (
    UserID VARCHAR2 (5) CONSTRAINT UserID_pk PRIMARY KEY,
    Name VARCHAR2 (30) NOT NULL,
    Email VARCHAR2 (30) NOT NULL,
    Phone_number NUMBER(11) NOT NULL
);
```

Create table is used to create a new table titled Users. UserID has a data type of VARCHAR2 for strings that can contain 5 variable characters and is a primary key. Variable Name and Email are strings that can contain 30 variable characters and can not be null. Phone_number has a data type int number that can contain 11 numeric characters and can not be null.

CREATE TABLE Event

```
CREATE TABLE Event (

    EventID VARCHAR2 (5) CONSTRAINT EventID_pk PRIMARY KEY,

    EventName VARCHAR2 (20) NOT NULL,

    Types VARCHAR2 (15) NOT NULL,

    Dates DATE NOT NULL,

    Venue VARCHAR2 (30) NOT NULL

);
```

Create table statement is used to create a new table titled Event. EventID has a datatype of varchar2 and can contain 5 variable characters as well as the primary key. Variable EventName and Types can hold 20 and 15 string characters respectively and are not null. Dates hold date datatype and can not be a null, variable venue has a datatype of strings and can contain 30 string characters as well as can not be null.

CREATE TABLE Booking

```
CREATE TABLE Booking (
      BookingID VARCHAR2 (5) CONSTRAINT BookingID_pk PRIMARY
KEY,
    bookingTime TIMESTAMP NOT NULL,
    status VARCHAR2 (10) NOT NULL,
    userID VARCHAR2 (5),
    eventID VARCHAR2(5),
    FOREIGN KEY (userID) REFERENCES Users(UserID),
    FOREIGN KEY (eventID) REFERENCES Event(EventID)
);
```

Create table statement is used to create a new table titled Booking. Booking table has 4 variables namely BookingID, userBookedID, bookingTime and status. Only xsbookingTime has timestamp as the data type as it contains date of the booking, meanwhile other variables have varchar2 data type as they contain string characters. BookingID is the primary key. Both BookingID and UserBookedID can hold 5 strings while status can hold 10 characters. All variables can not be null. userId and EventID are both foreign keys referencing to Users and Event respectively.

CREATE TABLE Payment

```
CREATE TABLE Payment(
    PaymentID VARCHAR2(5) CONSTRAINT payment_pk PRIMARY KEY,
    Payment_mode VARCHAR2(10) NOT NULL,
    Total_Payment NUMBER(5,2),
    UserID VARCHAR2(5),
    BookingID VARCHAR2(5),
    FOREIGN KEY (UserID) REFERENCES Users(UserID),
    FOREIGN KEY (BookingID) REFERENCES Booking(BookingID)
);
```

Create table statement is used to create a new table titled Payment. PaymentID is a primary key that has a datatype of varchar2 that can contain 5 variable characters. Variable Payment_modehas a data type of varchar2 that can contain 10 variable characters and is set to not null. Total_Payment has a number data type that can hold 5 numeric characters with 2 decimal places. UserID and BookingID are both foreign keys referencing to Users and Booking tables respectively.

CREATE TABLE Feedback

```
CREATE TABLE Feedback(
      FeedbackID VARCHAR2(5) CONSTRAINT feedback_pk PRIMARY
KEY,
    Rate NUMBER(5) NOT NULL,
    Comment_ VARCHAR2(100),
    UserID VARCHAR2(5),
    FOREIGN KEY (UserID) REFERENCES Users(UserID)
);
```

Create table statement is used to create a new table titled Feedback. FeedbackID is a primary key that has a datatype of varchar2 that can contain 5 variable characters. Next, the variable rate has a datatype of number that can contain 5 numerical characters and is set to not null. The variable Comment_ has a datatype of varchar2 that can hold up to 100 characters. Lastly, UserID is a foreign key referencing to Booking table.

CREATE TABLE Notification

```
CREATE TABLE Notification(
    Type_ VARCHAR2(15) NOT NULL,
    Message VARCHAR2(500) NOT NULL
);
```

Create table statement is used to create a new table titled Notification that contains only two variables which are Type_ that has a datatype of varchar2 that can contain up to 15 variable characters. Next is variable Message that can contain up to 500 variable characters.

DML:

INSERT DATA INTO TABLE Users

```
INSERT INTO Users VALUES ('A001', 'Selena Gomez',
'rareBeauty@gmail.com', '0116734219');
INSERT INTO Users VALUES ('A002', 'Taylor Swift',
'red1989@gmail.com', '0195578920');
INSERT INTO Users VALUES ('A003', 'Justin Bieber',
'jb@yahoo.com', '0134695046');
INSERT INTO Users VALUES ('A004', 'Hailey Bieber',
'rhode@gmail.com', '0147780239');
INSERT INTO Users VALUES ('A005', 'Shawn Mendes',
'sm@hotmail.com', '0123456789');
INSERT INTO Users VALUES ('A006', 'Ariana Grande',
'rem@gmail.com', '0146759903');
INSERT INTO Users VALUES ('A007', 'Bruno Mars',
'bruno@hotmail.com', '0176899912');
INSERT INTO Users VALUES ('A008', 'Lady Gaga',
'ladygg@gmail.com', '0141233342');
INSERT INTO Users VALUES ('A009', 'Drake',
'anitamaxwynn@gmail.com', '0192564458');
INSERT INTO Users VALUES ('A010', 'Mariah Carey',
'aiwfciy@hotmail.com', '0150203443');
```

```
INSERT INTO Users VALUES ('A011', 'Britney Spears',
'itsbritneyb@gmail.com', '0116768901');
INSERT INTO Users VALUES ('A012', 'Rihanna',
'badgalriri@yahoo.com', '0196721198');
INSERT INTO Users VALUES ('A013', 'Zayn Malik',
'1d@yahoo.com', '0187724561');
INSERT INTO Users VALUES ('A014', 'Steve Lacy',
'stevelacy@gmail.com', '0139982043');
INSERT INTO Users VALUES ('A015', 'Boy Pablo',
'roypablo@gmail.com', '0186923301');
INSERT INTO Users VALUES ('A016', 'Daniel Caesar',
'daniel@hotmail.com', '0199912765');
INSERT INTO Users VALUES ('A017', 'Frank Ocean',
'ocean@yahoo.com', '0172433391');
INSERT INTO Users VALUES ('A018', 'Freddie Mercury',
'queen@hotmail.com', '0185920504');
INSERT INTO Users VALUES ('A019', 'Olivia Rodrigo',
'guts@gmail.com', '0148935512');
INSERT INTO Users VALUES ('A020', 'Jennie Kim',
'blackpink@gmail.com', '0116921912');
```

INSERT INTO statement is used to insert values into the table Users. For each INSERT INTO statement, there are new values for each Users variable to be inserted.

INSERT DATA INTO TABLE Event

```
INSERT INTO Event VALUES ('E056', 'CPP Basics Clinic',
'Education', '29-DEC-2023', 'MPK3');
INSERT INTO Event VALUES ('E057', 'Cookie Night',
'Entertainment', '16-APR-2024', 'N24');
INSERT INTO Event VALUES ('E058', 'SUSKOM', 'Sports',
'23-NOV-2023', 'Sports Hall');
INSERT INTO Event VALUES ('E059', 'StartUp', 'Seminar',
'9-JAN-2024', 'N28');
INSERT INTO Event VALUES ('E060', 'Hackathon', 'Competition',
'20-MAY-2024', 'L50');
```

INSERT INTO statement is used to insert values into the table Event. For each INSERT INTO statement, there are new values for each Event variable to be inserted.

INSERT DATA INTO TABLE Booking

```
INSERT INTO Booking VALUES ('B0101', '27-DEC-2023 9:00:02',
'BOOKED', 'A001', 'E056');
INSERT INTO Booking VALUES ('B0102', '10-APR-2024 9:30:50',
'BOOKED', 'A002', 'E057');
INSERT INTO Booking VALUES ('B0103', '18-NOV-2023 5:25:20',
'PAID', 'A003', 'E058');
INSERT INTO Booking VALUES ('B0104', '29-DEC-2023 11:45:01',
'PENDING', 'A004', 'E058');
INSERT INTO Booking VALUES ('B0105',  '1-NOV-2023 12:59:40',
'CANCELED', 'A005', 'E058');
INSERT INTO Booking VALUES ('B0106',  '20-DEC-2023 12:59:40',
'BOOKED', 'A006', 'E056');
INSERT INTO Booking VALUES ('B0107',  '15-MAY-2024 5:09:56',
'PAID', 'A007', 'E060');
INSERT INTO Booking VALUES ('B0108',  '15-DEC-2023 2:32:23',
'CANCELED', 'A008', 'E056');
INSERT INTO Booking VALUES ('B0109',  '16-MAY-2024 10:19:46',
'PENDING', 'A008', 'E060');
INSERT INTO Booking VALUES ('B0110',  '22-DEC-2023 7:34:57',
'BOOKED', 'A010', 'E056');
```

```
INSERT INTO Booking VALUES ('B0111',  '15-DEC-2023 9:57:12',
'CANCELED', 'A011', 'E056');
INSERT INTO Booking VALUES ('B0112',  '27-DEC-2023 4:12:00',
'BOOKED', 'A012', 'E059');
INSERT INTO Booking VALUES ('B0113',  '18-MAY-2024 12:34:42',
'CANCELED', 'A013', 'E060');
INSERT INTO Booking VALUES ('B0114',  '17-NOV-2023 11:56:21',
'BOOKED', 'A002', 'E058');
INSERT INTO Booking VALUES ('B0115',  '11-MAY-2024 6:13:08',
'PAID', 'A002', 'E060');
INSERT INTO Booking VALUES ('B0116',  '11-APR-2024 2:16:32',
'BOOKED', 'A008', 'E057');
INSERT INTO Booking VALUES ('B0117',  '15-NOV-2023 7:46:57',
'PAID', 'A017', 'E058');
INSERT INTO Booking VALUES ('B0118',  '17-DEC-2023 2:25:12',
'PENDING', 'A018', 'E056');
INSERT INTO Booking VALUES ('B0119',  '12-APR-2024 10:36:07',
'BOOKED', 'A019', 'E057');
INSERT INTO Booking VALUES ('B0120',  '15-APR-2024 12:45:13',
'BOOKED', 'A020', 'E057');
```

INSERT INTO statement is used to insert values into the table Booking. For each INSERT INTO statement, there are new values for each Booking variable to be inserted.

INSERT DATA INTO TABLE Payment

```
INSERT INTO Payment VALUES('RX001', 'FPX',
8.00,'A001','B0101');
INSERT INTO Payment VALUES('RX012', 'CREDIT',
5.00,'A002','B0102');
INSERT INTO Payment VALUES('RX003', 'FPX',
11.00,'A003','B0103');
INSERT INTO Payment VALUES('RX004', 'DEBIT',
50.00,'A004','B0104');
INSERT INTO Payment VALUES('RX005', 'DEBIT', 8.00,
'A004','B0107');
INSERT INTO Payment VALUES('RX016', 'CREDIT',
2.00,'A006','B0106');
INSERT INTO Payment VALUES('RX007', 'CREDIT', 8.00,
'A007','B0107');
INSERT INTO Payment VALUES('RX008', 'E-WALLET',
10.00,'A008','B0108');
INSERT INTO Payment VALUES('RX009', 'FPX',
5.00,'A009','B0109');
INSERT INTO Payment VALUES('RX010', 'FPX',
10.00,'A010','B0110');
```

```
INSERT INTO Payment VALUES('RX011', 'DEBIT',
20.00,'A011','B0111');
INSERT INTO Payment VALUES('RX002', 'FPX',
15.00,'A012','B0112');
INSERT INTO Payment VALUES('RX013', 'FPX',
10.00,'A013','B0113');
INSERT INTO Payment VALUES('RX014', 'DEBIT',
5.00,'A014','B0114');
INSERT INTO Payment VALUES('RX015', 'E-WALLET',
12.00,'A015','B0115');
INSERT INTO Payment VALUES('RX006', 'E-WALLET',
2.00,'A016','B0116');
INSERT INTO Payment VALUES('RX017', 'CREDIT', 2.00,
'A017','B0117');
INSERT INTO Payment VALUES('RX018', 'E-WALLET',
3.00,'A018','B0118');
INSERT INTO Payment VALUES('RX019', 'E-WALLET',
5.00,'A019','B0119');
INSERT INTO Payment VALUES('RX020', 'FPX',
8.00,'A020','B0120');
```

INSERT INTO statement is used to insert values into the table Payment. For each INSERT INTO statement, new data is inserted into each Payment variable.

INSERT DATA INTO TABLE Feedback

```
INSERT INTO Feedback VALUES('F0011', 5, 'This event very educational', 'A001');
INSERT INTO Feedback VALUES('F0014', 2, 'system crash when I try to make my booking the first time.', 'A014');
INSERT INTO Feedback VALUES('F0012', 5, 'This event shined bright like a diamond.', 'A012');
INSERT INTO Feedback VALUES('F0013', 3, 'The MC light up my night like nobody else.', 'A013');
INSERT INTO Feedback VALUES('F0015', 1, 'REFUND MY MONEY!!!!', 'A003');
INSERT INTO Feedback VALUES('F0016', 1, 'This event is very boring…', 'A016');
INSERT INTO Feedback VALUES('F0017', 2, 'The event seems very unorganized', 'A002');
INSERT INTO Feedback VALUES('F0018', 3, 'No vegetarian food option!', 'A018');
INSERT INTO Feedback VALUES('F0019', 4, 'This event is very fun and engaging.', 'A002');
INSERT INTO Feedback VALUES('F0020', 5, 'Will definitely go to this event again!', 'A008');
```

```
INSERT INTO Feedback VALUES('F0021', 5, 'The event was a blast! Great entertainment and atmosphere.', 'A001');
INSERT INTO Feedback VALUES('F0022', 1, 'I knew this event was trouble when I walked in, shame on the organizer!', 'A002');
INSERT INTO Feedback VALUES('F0023', 2, 'The venue were hard to find and very uncomfortable in my opinion.', 'A003');
INSERT INTO Feedback VALUES('F0024', 5, 'I hope UTM makes more educational events like this!', 'A004');
INSERT INTO Feedback VALUES('F0025', 3, 'The event was ok.', 'A003');
INSERT INTO Feedback VALUES('F0026', 4, 'I learned a lot from this event.', 'A006');
INSERT INTO Feedback VALUES('F0027', 2, 'The event is not engaging', 'A003');
INSERT INTO Feedback VALUES('F0028', 1, 'Oh baby this event was toxic.', 'A011');
INSERT INTO Feedback VALUES('F0029', 4, 'This event is very entertaining.', 'A009');
INSERT INTO Feedback VALUES('F0030', 5, 'Love the fun activities in this event.', 'A010');
```

INSERT INTO statement is used to insert values into the table Feedback. For each INSERT INTO statement, there are new values for each Feedback variable to be inserted.

INSERT DATA INTO TABLE Notification

```
INSERT INTO Notification
VALUES('reminder', 'REMINDER: C++ BASICS CLINIC' ||chr(10)||
'DATE: 29 DECEMBER 2023' ||chr(10)|| 'TIME: 9.30 AM'
||chr(10)|| 'VENUE: MPK3');
INSERT INTO Notification
VALUES('reminder', 'REMINDER: COOKIE NIGHT' ||chr(10)||
'DATE: 16 APRIL 2024' ||chr(10)|| 'TIME: 8.30 PM' ||chr(10)||
'VENUE: N24');
INSERT INTO Notification
VALUES('confirmation', 'SUSKOM BOOKING CONFIRMED');
INSERT INTO Notification
VALUES('announcement', 'STARTUP BOOKING FAILED');
INSERT INTO Notification
VALUES('cancellation', 'HACKATHON BOOKING CANCELLATION');
```

INSERT INTO statement is used to insert values into the table Notification. For each INSERT INTO statement, new values are inserted for each Notification variable.

QUERIES

1) **Display the booking made by each user**

```
SELECT b.userID, b.bookingID, b.eventID, u.name, u.email,
b.status
FROM Users u
JOIN Booking b
    ON u.UserID = b.UserID;
```

USERID	BOOKINGID	EVENTID	NAME	EMAIL	STATUS
A001	B0101	E056	Selena Gomez	rareBeauty@gmail.com	BOOKED
A002	B0102	E057	Taylor Swift	red1989@gmail.com	BOOKED
A003	B0103	E058	Justin Bieber	jb@yahoo.com	PAID
A004	B0104	E058	Hailey Bieber	rhode@gmail.com	PENDING
A005	B0105	E058	Shawn Mendes	sm@hotmail.com	CANCELED
A006	B0106	E056	Ariana Grande	rem@gmail.com	BOOKED
A007	B0107	E060	Bruno Mars	bruno@hotmail.com	PAID
A008	B0108	E056	Lady Gaga	ladygg@gmail.com	CANCELED
A008	B0109	E060	Lady Gaga	ladygg@gmail.com	PENDING
A010	B0110	E056	Mariah Carey	aiwfciy@hotmail.com	BOOKED
A011	B0111	E056	Britney Spears	itsbritneyb@gmail.com	CANCELED
A012	B0112	E059	Rihanna	badgalriri@yahoo.com	BOOKED
A013	B0113	E060	Zayn Malik	1d@yahoo.com	CANCELED
A002	B0114	E058	Taylor Swift	red1989@gmail.com	BOOKED
A002	B0115	E060	Taylor Swift	red1989@gmail.com	PAID
A008	B0116	E057	Lady Gaga	ladygg@gmail.com	BOOKED
A017	B0117	E058	Frank Ocean	ocean@yahoo.com	PAID
A018	B0118	E056	Freddie Mercury	queen@hotmail.com	PENDING
A019	B0119	E057	Olivia Rodrigo	guts@gmail.com	BOOKED
A020	B0120	E057	Jennie Kim	blackpink@gmail.com	BOOKED

To select important data to be displayed, the SELECT b.userID, b.bookingID, b.eventID, u.name, u.email, b.status FROM Users statement is used. JOIN statement is used to join UserID on User table with UserBookedID from Booking table. This is to sort the data by the common columns of both Users and Booking tables.

2) **Sort the Event table by their event date**

```
SELECT *
FROM Event
ORDER BY dates;
```

EVENTID	EVENTNAME	TYPES	DATES	VENUE
E058	SUSKOM	Sports	23-NOV-23	Sports Hall
E056	CPP Basics Clinic	Education	29-DEC-23	MPK3
E059	StartUp	Seminar	09-JAN-24	N28
E057	Cookie Night	Entertainment	16-APR-24	N24
E060	Hackathon	Competition	20-MAY-24	L50

SELECT* FROM Event statement is used to select all from Event table. To sort them according to the dates of the events, ORDER BY statement is used.

3) **Show the payment made that is above RM5**

```
SELECT * FROM Payment

WHERE Total_payment > 5

ORDER BY paymentID;
```

PAYMENTID	PAYMENT_MODE	TOTAL_PAYMENT	USERID	BOOKINGID
RX001	FPX	8	A001	B0101
RX002	FPX	15	A012	B0112
RX003	FPX	11	A003	B0103
RX004	DEBIT	50	A004	B0104
RX005	E-WALLET	15	A005	B0105
RX007	CREDIT	8	A007	B0107
RX008	E-WALLET	10	A008	B0108
RX010	FPX	10	A010	B0110
RX011	DEBIT	20	A011	B0111
RX013	FPX	10	A013	B0113
RX015	E-WALLET	12	A015	B0115
RX020	FPX	8	A020	B0120

To select all the variables from Payment table that satisfy the condition of Total_payment more than RM5, the SELECT * FROM Payment statement is used with WHERE Total_payment > 5 statement to prompt the system to display Total_payment that is more than 5 only. To sort the data according to the PaymentID, ORDER BY statement is used. Only payment with Total_payment that is more than 5 are displayed.

4) **Display feedback details based on rating in descending order**

```
SELECT * FROM Feedback

ORDER BY Rate DESC;
```

FEEDBACKID	RATE	COMMENT_	USERID
F0030	5	Love the fun activities in this event.	A010
F0020	5	Will definitely go to this event again!	A008
F0012	5	This event shined bright like a diamond.	A012
F0024	5	I hope UTM makes more educational events like this!	A004
F0011	5	This event very educational	A001
F0021	5	The event was a blast! Great entertainment and atmosphere.	A001
F0026	4	I learned a lot from this event.	A006
F0019	4	This event is very fun and engaging.	A002
F0029	4	This event is very entertaining.	A009
F0018	3	No vegetarian food option!	A018
F0013	3	The MC light up my night like nobody else.	A013
F0025	3	The event was ok.	A003
F0014	2	system crash when I try to make my booking the first time.	A014
F0017	2	The event seems very unorganized	A002
F0023	2	The venue were hard to find and very uncomfortable in my opinion.	A003
F0027	2	The event is not engaging	A003
F0022	1	I knew this event was trouble when I walked in, shame on the organizer!	A002
F0016	1	This event is very boring…	A016
F0028	1	Oh baby this event was toxic.	A011
F0015	1	REFUND MY MONEY!!!!	A003

SELECT* FROM Feedback statement is used to select all data from Feedback table. To sort them according to the rate in descending order, ORDER BY Rate DESC statement is used.

5) **Display the user's name in alphabetical order.**

```
SELECT *
FROM Users
ORDER BY Name;
```

USERID	NAME	EMAIL	PHONE_NUMBER
A006	Ariana Grande	rem@gmail.com	146759903
A015	Boy Pablo	roypablo@gmail.com	186923301
A011	Britney Spears	itsbritneyb@gmail.com	116768901
A007	Bruno Mars	bruno@hotmail.com	176899912
A016	Daniel Caesar	daniel@hotmail.com	199912765
A009	Drake	anitamaxwynn@gmail.com	192564458
A017	Frank Ocean	ocean@yahoo.com	172433391
A018	Freddie Mercury	queen@hotmail.com	185920504
A004	Hailey Bieber	rhode@gmail.com	147780239
A020	Jennie Kim	blackpink@gmail.com	116921912
A003	Justin Bieber	jb@yahoo.com	134695046
A008	Lady Gaga	ladygg@gmail.com	141233342
A010	Mariah Carey	aiwfciy@hotmail.com	150203443
A019	Olivia Rodrigo	guts@gmail.com	148935512
A012	Rihanna	badgalriri@yahoo.com	196721198
A001	Selena Gomez	rareBeauty@gmail.com	116734219
A005	Shawn Mendes	sm@hotmail.com	123456789
A014	Steve Lacy	stevelacy@gmail.com	139982043
A002	Taylor Swift	red1989@gmail.com	195578920
A013	Zayn Malik	1d@yahoo.com	187724561

SELECT* FROM Users statement is used to select all data from Users table. To sort them according to alphabetical order, ORDER BY Name statement is used.

6) **Display the booking made along with its payment details by increasing order in total payment.**

```
SELECT p.paymentID, p.userID, u.name, u.email,
p.payment_mode, p.total_payment
FROM Users u
JOIN Payment p
ON u.UserID = p.UserID
ORDER BY p.Total_payment;
```

PAYMENTID	USERID	NAME	EMAIL	PAYMENT_MODE	TOTAL_PAYMENT
RX006	A016	Daniel Caesar	daniel@hotmail.com	E-WALLET	2
RX016	A006	Ariana Grande	rem@gmail.com	CREDIT	2
RX017	A017	Frank Ocean	ocean@yahoo.com	CREDIT	2
RX018	A018	Freddie Mercury	queen@hotmail.com	E-WALLET	3
RX014	A014	Steve Lacy	stevelacy@gmail.com	DEBIT	5
RX019	A019	Olivia Rodrigo	guts@gmail.com	E-WALLET	5
RX012	A002	Taylor Swift	red1989@gmail.com	CREDIT	5
RX009	A009	Drake	anitamaxwynn@gmail.com	FPX	5
RX007	A007	Bruno Mars	bruno@hotmail.com	CREDIT	8
RX001	A001	Selena Gomez	rareBeauty@gmail.com	FPX	8
RX020	A020	Jennie Kim	blackpink@gmail.com	FPX	8
RX005	A004	Hailey Bieber	rhode@gmail.com	DEBIT	8
RX010	A010	Mariah Carey	aiwfciy@hotmail.com	FPX	10
RX008	A008	Lady Gaga	ladygg@gmail.com	E-WALLET	10
RX013	A013	Zayn Malik	1d@yahoo.com	FPX	10
RX003	A003	Justin Bieber	jb@yahoo.com	FPX	11
RX015	A015	Boy Pablo	roypablo@gmail.com	E-WALLET	12
RX002	A012	Rihanna	badgalriri@yahoo.com	FPX	15
RX011	A011	Britney Spears	itsbritneyb@gmail.com	DEBIT	20
RX004	A004	Hailey Bieber	rhode@gmail.com	DEBIT	50

To select important data from Users and Payment table, the SELECT p.paymentID, p.userID, u.name, u.email, p.payment_mode, p.total_payment FROM Users u statement is used along with JOIN Payment p statement to join UserID on User table with UseID from Payment table. This data is sort by increasing total payment by using ORDER BY p.Total_payment statement.

7) Display all the current event attendees that have made bookings.

```
SELECT b.EventID, e.EventNAme, b.UserID, u.Name
From Booking b
JOIN Event e
    ON b.EventID = e.EventID
JOIN Users u
     ON b.UserID = u.USerID
ORDER BY b.EventID;
```

EVENTID	EVENTNAME	USERID	NAME
E056	CPP Basics Clinic	A006	Ariana Grande
E056	CPP Basics Clinic	A001	Selena Gomez
E056	CPP Basics Clinic	A018	Freddie Mercury
E056	CPP Basics Clinic	A011	Britney Spears
E056	CPP Basics Clinic	A010	Mariah Carey
E056	CPP Basics Clinic	A008	Lady Gaga
E057	Cookie Night	A020	Jennie Kim
E057	Cookie Night	A002	Taylor Swift
E057	Cookie Night	A019	Olivia Rodrigo
E057	Cookie Night	A008	Lady Gaga
E058	SUSKOM	A004	Hailey Bieber
E058	SUSKOM	A002	Taylor Swift
E058	SUSKOM	A003	Justin Bieber
E058	SUSKOM	A017	Frank Ocean
E058	SUSKOM	A005	Shawn Mendes
E059	StartUp	A012	Rihanna
E060	Hackathon	A002	Taylor Swift
E060	Hackathon	A008	Lady Gaga
E060	Hackathon	A007	Bruno Mars
E060	Hackathon	A013	Zayn Malik

To select important data from Booking, Event and Users table, SELECT b.EventID, e.EventNAme, b.UserID, u.Name From Booking b statement is used along with JOIN Event e statement to join EventID from Event table with EventID from Booking table. JOIN Users u statement is used to join UserID from User table with UserID from Booking table.

8.0 Summary

The proposed system aims to enhance the current system by adding new features that could accelerate the process and limit the shortcomings, such as current manual processes, unreliable payment system and fragmented data handling. The proposal introduces a centralized ticketing system and automated systems for booking and feedback mechanism. It includes an event reservation process, a dedicated payment gateway and refund process, together with feedback from users. Conceptual ERD as well as logical ERD have been discussed to further show the design and flow of the new system. With these proposed features, the goal is to ease the administrators, making sure that the management of users' data will enhance more efficiently and effectively. All the new proposed features also intend to address the issues faced by the current system, ensuring a better user experience and overall a better system for NexScholar.