

VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY

Ho Chi Minh City University of Technology

Faculty of Computer Science and Engineering



DATABASE SYSTEMS (CO2013)

Assignment 2

BKinema

A Movie Ticket Booking System

Instructor: Le Duc Hoang Nam

Semester: 251
Class: CC06
Group: BKinema
Students: Nguyen Trung Nhan – 2352852
Le Pham Tien Long – 2352688
Ho Minh Nhat – 2352858
Nguyen Vinh Phu – 2352919
Pham Nam An – 2353016



Contents

1	Introduction	4
2	Database Schema Implementation	4
2.1	Core Entity Tables	4
2.1.1	User Table	4
2.1.2	Membership Table	5
2.1.3	Customer Table	5
2.1.4	Staff Table	6
2.2	Theater Infrastructure Tables	6
2.2.1	Theater Table	6
2.2.2	Auditorium Table	7
2.2.3	Seat Table	7
2.3	Movie and Showtime Tables	8
2.3.1	Movie Table	8
2.3.2	Movie Multivalued Attributes	8
2.3.3	Showtime Table	9
2.4	Booking and Transaction Tables	10
2.4.1	Booking Table	10
2.4.2	Showtime_seat Table	11
2.4.3	Payment Table	11
2.4.4	Coupon Table	12
2.4.5	Refund Table	12
2.5	Food and Beverage Tables	13
2.5.1	FWB_Menu Table	13
2.5.2	FWB and Contains_item Tables	13
2.6	Gift Relationship Tables	14
2.7	Indexing Strategy	14
3	Triggers	15
3.1	User Management Triggers	15
3.1.1	Trigger: Auto-Create Customer Record	15
3.1.2	Trigger: Validate Customer Updates	16
3.2	Seat Capacity Management Triggers	16
3.2.1	Trigger: Update Capacity on Seat Insert	16
3.2.2	Trigger: Update Capacity on Seat Delete	17
3.2.3	Trigger: Update Capacity on Seat Move	17
3.3	Showtime Scheduling Triggers	18
3.3.1	Trigger: Prevent Overlapping Showtimes on Insert	18
3.3.2	Trigger: Prevent Overlapping Showtimes on Update	19
3.4	Booking and Payment Triggers	20
3.4.1	Trigger: Validate Booking Time and Seat Availability	20
3.4.2	Trigger: Set Booking Gift Flag	21
4	Stored Procedures	22
4.1	Authentication and User Management Procedures	22
4.1.1	Procedure: sp_register_user	22
4.1.2	Procedure: sp_login_user	23
4.1.3	Procedure: sp_update_user_profile	23
4.2	Showtime Management Procedures	23
4.2.1	Procedure: sp_delete_showtime	23
4.3	Booking Management Procedures	25
4.3.1	Procedure: sp_start_booking	25
4.3.2	Procedure: sp_confirm_payment	26
4.3.3	Procedure: sp_cancel_expired_bookings	27
4.4	Payment and Refund Procedures	28



4.4.1	Procedure: sp_apply_coupon	28
4.4.2	Procedure: sp_apply_points	28
4.4.3	Procedure: sp_create_refund_coupon	28
4.5	Query and Reporting Procedures	29
4.5.1	Procedure: sp_get_movies_with_details	29
4.5.2	Procedure: sp_get_customer_bookings	30
4.5.3	Procedure: sp_calculate_final_amount	30
4.5.4	Procedure: sp_generate_sales_report	30
4.6	Customer Dashboard Procedures	31
4.6.1	Procedure: sp_get_customer_dashboard	31
4.6.2	Procedure: sp_get_customer_points	31
4.6.3	Procedure: sp_get_membership_card	31
4.7	Gift Transaction Procedures	31
4.7.1	Procedure: sp_gift_coupon	31
4.7.2	Procedure: sp_send_gift	32
5	Functions	32
5.1	Validation Functions	32
5.1.1	Function: fn_validate_email	32
5.1.2	Function: fn_check_email_exists	33
5.1.3	Function: fn_validate_age	33
5.1.4	Function: fn_validate_password_hash	33
5.2	Calculation Functions	34
5.2.1	Function: fn_calculate_points	34
5.2.2	Function: fn_count_showtimes_in_range	35
5.3	Membership Management Functions	36
5.3.1	Function: fn_get_default_membership	36
5.4	Revenue Analysis Functions	36
5.4.1	Function: fn_calculate_avg_booking_value_by_month	36
5.4.2	Function: fn_calculate_total_revenue_by_customer	37
5.5	Function Integration with Database Objects	39
5.6	Function Call Examples	39
6	Application Implementation	41
6.1	Backend Implementation	42
6.1.1	Authentication Service Implementation	42
6.1.2	Booking Management Controller	43
6.2	Frontend Implementation	43
6.2.1	Booking Context State Management	43
6.3	Application Workflow and User Interface	45
6.3.1	User Registration and Authentication	45
6.3.2	Home Page and Movie Discovery	45
6.3.3	Movie Catalog and Filtering	46
6.3.4	Theater Page	47
6.3.5	Movie Detail and Showtime Selection	47
6.3.6	Seat Selection Interface	48
6.3.7	Food and Beverage Selection	49
6.3.8	Payment and Point Redemption	49
6.3.9	Booking Management and History	50
6.3.10	Membership Card	51
7	Conclusion	51
7.1	Summary	51
7.2	Future Works	52



A Additional Application Screenshots	52
A.1 FAQ Page	52
A.2 Membership Information Page	53
A.3 Payment Policy Page	53
A.4 Terms of Use Page	54
A.5 Refund Request Interface	54
A.6 Ticket View Interface	55
B Database Connection Configuration	56
Code Availability	56
Acknowledgement	56



1 Introduction

Building upon the conceptual database design established in Assignment 1, this report documents the practical implementation of the BKinema movie ticket booking system.

Following the Entity-Relationship Diagram (ERD) and relational schema designed in the previous assignment, this implementation addresses three critical aspects of database system development: **physical schema implementation, advanced database programming, and application connectivity**.

Application integration demonstrates how the database connects with a application stack. The implementation uses **NestJS** (a TypeScript-based Node.js framework) for the backend and **React.js** for the frontend, communicating with a **MySQL** database through stored procedure calls. This architecture ensures clear separation of concerns, improved security through parameterized queries, and better performance through server-side processing.

The remaining sections of this report present detailed documentation of each implementation component. Section 2 demonstrates the SQL statements for creating all database tables, including primary keys, foreign keys, and check constraints, along with sample data insertion. Section 3 documents the implementation of triggers for enforcing business rules and maintaining derived attributes. Section 4 presents stored procedures for insert, update, and delete operations with validation, as well as query procedures for data retrieval. Section 5 describes the implementation of functions with cursor operations and conditional logic. Section 6 illustrates the web application implementation, showing how frontend interfaces connect to database operations through stored procedure calls, accompanied by sequence diagrams and activity diagrams to visualize system workflows. Finally, Section 7 summarizes the achievements and lessons learned from this database implementation project.

2 Database Schema Implementation

The BKinema database consists of the following table categories:

- **User Management:** User, Customer, Staff, Membership (4 tables)
- **Theater Infrastructure:** Theater, Auditorium, Seat (3 tables)
- **Movie Information:** Movie, Director, Actor, Genre, Subtitle, Dubbing (6 tables)
- **Showtime and Bookings:** Showtime, Showtime_seat, Booking (3 tables)
- **Payments and Transactions:** Payment, Coupon, Refund (3 tables)
- **Food and Beverage:** FWB_Menu, FWB, Contains_item (3 tables)
- **Gift Relationships:** Give, Send_gift (2 tables)

2.1 Core Entity Tables

2.1.1 User Table

The User table stores information about all system users, serving as the base table for both customers and staff through a specialization relationship.

Key Constraints:

- Email uniqueness prevents duplicate accounts
- Password field stores bcrypt-hashed passwords (handled at application layer)
- `created_at` and `updated_at` provide audit trail
- Gender enum restricts values to valid options

```
1 CREATE TABLE "User" (
2     "id" bigint NOT NULL AUTO_INCREMENT,
3     "fname" varchar(255) NOT NULL,
4     "minit" varchar(50) DEFAULT NULL,
5     "lname" varchar(255) NOT NULL,
6     "birthday" date DEFAULT NULL,
7     "gender" enum('Male','Female','Other') NOT NULL DEFAULT 'Other',
8     "email" varchar(255) NOT NULL,
9     "password" varchar(255) NOT NULL,
10    "district" varchar(255) DEFAULT NULL,
11    "city" varchar(255) DEFAULT NULL,
12    "created_at" datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
13    "updated_at" datetime NOT NULL DEFAULT CURRENT_TIMESTAMP
14        ON UPDATE CURRENT_TIMESTAMP,
15    PRIMARY KEY ("id"),
16    UNIQUE KEY "uk_user_email" ("email")
17 ) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
```

Listing 1: User table structure from schema.sql

2.1.2 Membership Table

Defines the loyalty program tiers with point rates for earning rewards.

```
1 CREATE TABLE "membership" (
2     "tier_name" varchar(255) NOT NULL,
3     "min_spent" decimal(12,2) NOT NULL DEFAULT '0.00',
4     "box_office_point_rate" decimal(5,2) NOT NULL DEFAULT '0.00',
5     "concession_point_rate" decimal(5,2) NOT NULL DEFAULT '0.00',
6     PRIMARY KEY ("tier_name"),
7     CONSTRAINT "chk_membership_box_rate" CHECK (
8         ("box_office_point_rate" >= 0) AND
9         ("box_office_point_rate" <= 100)
10    ),
11    CONSTRAINT "chk_membership_con_rate" CHECK (
12        ("concession_point_rate" >= 0) AND
13        ("concession_point_rate" <= 100)
14    )
15 ) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
```

Listing 2: Membership table structure

Business Rules:

- Point rates are percentages (0-100) applied to spending amounts
- Separate rates for box office (tickets) and concessions (F&B)
- Membership tiers: Member, U22, VIP, VVIP

2.1.3 Customer Table

Extends the User table with customer-specific attributes including membership and points.



```
1 CREATE TABLE "customer" (
2     "user_id" bigint NOT NULL,
3     "accumulated_points" int NOT NULL DEFAULT '0',
4     "total_spent" decimal(12,2) NOT NULL DEFAULT '0.00',
5     "membership_name" varchar(50) DEFAULT NULL,
6     "membership_valid_until" date DEFAULT NULL,
7     PRIMARY KEY ("user_id"),
8     KEY "fk_customer_membership" ("membership_name"),
9     CONSTRAINT "fk_customer_membership" FOREIGN KEY ("membership_name")
10        REFERENCES "membership" ("tier_name")
11        ON DELETE SET NULL ON UPDATE CASCADE,
12     CONSTRAINT "fk_customer_user" FOREIGN KEY ("user_id")
13        REFERENCES "User" ("id"),
14     CONSTRAINT "customer_chk_1" CHECK (("accumulated_points" >= 0)),
15     CONSTRAINT "customer_chk_2" CHECK (("total_spent" >= 0))
16 ) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
```

Listing 3: Customer table structure

Key Features:

- Automatically created via trigger when User is inserted
- Points and spending track customer loyalty
- Membership valid until date implements annual renewal cycle
- Check constraints prevent negative values

2.1.4 Staff Table

Extends the User table for staff members with role and shift information.

```
1 CREATE TABLE "staff" (
2     "user_id" bigint NOT NULL,
3     "shift" varchar(255) NOT NULL,
4     "role" varchar(255) NOT NULL,
5     "theater_id" bigint DEFAULT NULL,
6     PRIMARY KEY ("user_id"),
7     KEY "fk_staff_theater" ("theater_id"),
8     CONSTRAINT "fk_staff_theater" FOREIGN KEY ("theater_id")
9        REFERENCES "theater" ("id"),
10    CONSTRAINT "fk_staff_user" FOREIGN KEY ("user_id")
11        REFERENCES "User" ("id")
12 ) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
```

Listing 4: Staff table structure

2.2 Theater Infrastructure Tables

2.2.1 Theater Table

Represents cinema branches across different locations.

```
1 CREATE TABLE "theater" (
2   "id" bigint NOT NULL AUTO_INCREMENT,
3   "name" varchar(255) NOT NULL,
4   "street" varchar(255) NOT NULL,
5   "district" varchar(255) NOT NULL,
6   "city" varchar(255) NOT NULL,
7   "image" varchar(255) DEFAULT NULL,
8   "description" text,
9   PRIMARY KEY ("id")
10 ) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
```

Listing 5: Theater table structure

2.2.2 Auditorium Table

Represents screening rooms within theaters with different formats.

```
1 CREATE TABLE "auditorium" (
2   "number" int NOT NULL,
3   "theater_id" bigint NOT NULL,
4   "type" enum('2D','ScreenX','IMAX','4DX') NOT NULL DEFAULT '2D',
5   "capacity" int NOT NULL DEFAULT '0',
6   "image" varchar(255) DEFAULT NULL,
7   "description" text,
8   PRIMARY KEY ("number","theater_id"),
9   KEY "fk_auditorium_theater" ("theater_id"),
10  CONSTRAINT "fk_auditorium_theater" FOREIGN KEY ("theater_id")
11    REFERENCES "theater" ("id")
12 ) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
```

Listing 6: Auditorium table structure

Design Notes:

- Composite primary key (`number, theater_id`) allows same auditorium numbers across different theaters
- Capacity is automatically maintained by triggers when seats are added/removed
- Type enum supports various cinema formats

2.2.3 Seat Table

Individual seats within auditoriums with position and pricing information.

Seat Identification:

- `row_char + column_number` creates human-readable seat names (e.g., "A1", "B5")
- Composite PK ensures seat uniqueness within auditorium
- Triggers update auditorium capacity when seats are modified

```
1 CREATE TABLE "seat" (
2     "id" bigint NOT NULL,
3     "au_number" int NOT NULL,
4     "au_theater_id" bigint NOT NULL,
5     "row_char" varchar(10) NOT NULL,
6     "column_number" int NOT NULL,
7     "type" varchar(255) NOT NULL DEFAULT 'Standard',
8     "price" decimal(10,2) NOT NULL,
9     PRIMARY KEY ("id", "au_number", "au_theater_id"),
10    KEY "fk_seat_auditorium" ("au_number", "au_theater_id"),
11    CONSTRAINT "fk_seat_auditorium" FOREIGN KEY
12        ("au_number", "au_theater_id")
13        REFERENCES "auditorium" ("number", "theater_id"),
14    CONSTRAINT "seat_chk_1" CHECK (("price" >= 0))
15 ) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
```

Listing 7: *Seat table structure*

2.3 Movie and Showtime Tables

2.3.1 Movie Table

Core movie information with release scheduling.

```
1 CREATE TABLE "movie" (
2     "id" bigint NOT NULL AUTO_INCREMENT,
3     "name" varchar(255) NOT NULL,
4     "duration" int NOT NULL,
5     "language" varchar(255) DEFAULT NULL,
6     "release_date" date DEFAULT NULL,
7     "end_date" date DEFAULT NULL,
8     "age_rating" varchar(3) NOT NULL,
9     "poster_file" varchar(255) DEFAULT NULL,
10    "url_slug" varchar(255) DEFAULT NULL,
11    "description" text,
12    "trailer_url" varchar(255) DEFAULT NULL,
13    PRIMARY KEY ("id"),
14    UNIQUE KEY "uk_movie_url_slug" ("url_slug"),
15    CONSTRAINT "movie_chk_1" CHECK (("duration" > 0))
16 ) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
```

Listing 8: *Movie table structure*

2.3.2 Movie Multivalued Attributes

The following tables implement multivalued attributes for movies:

```
1 CREATE TABLE "director" (
2     "movie_id" bigint NOT NULL,
3     "director" varchar(255) NOT NULL,
4     PRIMARY KEY ("movie_id","director"),
5     CONSTRAINT "fk_director_movie" FOREIGN KEY ("movie_id")
6         REFERENCES "movie" ("id")
7 );
8
9 CREATE TABLE "actor" (
10    "movie_id" bigint NOT NULL,
11    "actor" varchar(255) NOT NULL,
12    "name" varchar(255) DEFAULT NULL,
13    "age" int DEFAULT NULL,
14    PRIMARY KEY ("movie_id","actor"),
15    CONSTRAINT "fk_actor_movie" FOREIGN KEY ("movie_id")
16        REFERENCES "movie" ("id"),
17    CONSTRAINT "actor_chk_1" CHECK (((age IS NULL) OR (age >= 0)))
18 );
19
20 CREATE TABLE "genre" (
21    "movie_id" bigint NOT NULL,
22    "genre" varchar(255) NOT NULL,
23    PRIMARY KEY ("movie_id","genre"),
24    CONSTRAINT "fk_genre_movie" FOREIGN KEY ("movie_id")
25        REFERENCES "movie" ("id")
26 );
27
28 CREATE TABLE "subtitle" (
29    "movie_id" bigint NOT NULL,
30    "subtitle" varchar(255) NOT NULL,
31    PRIMARY KEY ("movie_id","subtitle"),
32    CONSTRAINT "fk_subtitle_movie" FOREIGN KEY ("movie_id")
33        REFERENCES "movie" ("id")
34 );
35
36 CREATE TABLE "dubbing" (
37    "movie_id" bigint NOT NULL,
38    "dubbing" varchar(255) NOT NULL,
39    PRIMARY KEY ("movie_id","dubbing"),
40    CONSTRAINT "fk_dubbing_movie" FOREIGN KEY ("movie_id")
41        REFERENCES "movie" ("id")
42 );
```

Listing 9: Director table (multivalued attribute)

2.3.3 Showtime Table

Links movies to specific auditoriums at scheduled times.

Important Business Logic:

- Triggers prevent overlapping showtimes in same auditorium
- 15-minute gap enforced between consecutive showtimes
- Booking cutoff time validated by trigger (15 minutes before start for online)



```
1 CREATE TABLE "showtime" (
2     "id" bigint NOT NULL AUTO_INCREMENT,
3     "date" date NOT NULL,
4     "start_time" time NOT NULL,
5     "end_time" time NOT NULL,
6     "movie_id" bigint NOT NULL,
7     "au_number" int NOT NULL,
8     "au_theater_id" bigint NOT NULL,
9     PRIMARY KEY ("id"),
10    KEY "fk_showtime_movie" ("movie_id"),
11    KEY "fk_showtime_auditorium" ("au_number", "au_theater_id"),
12    CONSTRAINT "fk_showtime_auditorium" FOREIGN KEY
13        ("au_number", "au_theater_id")
14            REFERENCES "auditorium" ("number", "theater_id"),
15    CONSTRAINT "fk_showtime_movie" FOREIGN KEY ("movie_id")
16        REFERENCES "movie" ("id")
17 ) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
```

Listing 10: Showtime table structure

2.4 Booking and Transaction Tables

2.4.1 Booking Table

Central transaction hub tracking customer bookings.

```
1 CREATE TABLE "booking" (
2     "id" bigint NOT NULL AUTO_INCREMENT,
3     "created_time_at" datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
4     "status" varchar(255) NOT NULL,
5     "booking_method" varchar(255) NOT NULL,
6     "is_gift" tinyint(1) NOT NULL DEFAULT '0',
7     "live_direction" varchar(255) DEFAULT NULL,
8     "points_earned" int NOT NULL DEFAULT '0',
9     "points_used" int NOT NULL DEFAULT '0',
10    "customer_id" bigint DEFAULT NULL,
11    "staff_id" bigint DEFAULT NULL,
12    PRIMARY KEY ("id"),
13    KEY "fk_booking_staff" ("staff_id"),
14    KEY "idx_booking_status_created" ("status", "created_time_at"),
15    KEY "idx_booking_customer" ("customer_id"),
16    CONSTRAINT "fk_booking_customer" FOREIGN KEY ("customer_id")
17        REFERENCES "customer" ("user_id"),
18    CONSTRAINT "fk_booking_staff" FOREIGN KEY ("staff_id")
19        REFERENCES "staff" ("user_id")
20 ) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
```

Listing 11: Booking table structure

Status Values: Pending, Paid, Cancelled, Refunded

Booking Method: Online, Counter

2.4.2 Showtime_seat Table

Represents individual ticket bookings linking seats to showtimes.

```
1 CREATE TABLE "showtime_seat" (
2     "ticketid" bigint NOT NULL AUTO_INCREMENT,
3     "st_id" bigint NOT NULL,
4     "seat_id" bigint NOT NULL,
5     "seat_au_number" int NOT NULL,
6     "seat_au_theater_id" bigint NOT NULL,
7     "status" varchar(255) NOT NULL DEFAULT 'Valid',
8     "price" decimal(10,2) NOT NULL,
9     "booking_id" bigint DEFAULT NULL,
10    PRIMARY KEY ("ticketid"),
11    KEY "fk_showtimeseat_seat"
12        ("seat_id", "seat_au_number", "seat_au_theater_id"),
13    KEY "idx_ss_stid_seatid_status" ("st_id", "seat_id", "status"),
14    KEY "idx_ss_booking_status" ("booking_id", "status"),
15    CONSTRAINT "fk_showtimeseat_booking" FOREIGN KEY ("booking_id")
16        REFERENCES "booking" ("id"),
17    CONSTRAINT "fk_showtimeseat_seat" FOREIGN KEY
18        ("seat_id", "seat_au_number", "seat_au_theater_id")
19            REFERENCES "seat" ("id", "au_number", "au_theater_id"),
20    CONSTRAINT "fk_showtimeseat_showtime" FOREIGN KEY ("st_id")
21        REFERENCES "showtime" ("id"),
22    CONSTRAINT "showtime_seat_chk_1" CHECK (("price" >= 0))
23 ) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
```

Listing 12: *Showtime_seat table structure*

Status Lifecycle: Available → Held → Booked → Refunded

2.4.3 Payment Table

Records payment transactions for bookings.

```
1 CREATE TABLE "payment" (
2     "id" bigint NOT NULL AUTO_INCREMENT,
3     "payment_method" varchar(255) NOT NULL,
4     "status" varchar(255) NOT NULL,
5     "created_time_at" datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
6     "transaction_id" varchar(255) NOT NULL,
7     "expired_time_at" datetime DEFAULT NULL,
8     "duration" int DEFAULT NULL,
9     "booking_id" bigint NOT NULL,
10    PRIMARY KEY ("id"),
11    UNIQUE KEY "uk_payment_transaction" ("transaction_id"),
12    UNIQUE KEY "uk_payment_booking" ("booking_id"),
13    KEY "idx_payment_booking" ("booking_id"),
14    CONSTRAINT "fk_payment_booking" FOREIGN KEY ("booking_id")
15        REFERENCES "booking" ("id")
16 ) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
```

Listing 13: *Payment table structure*

2.4.4 Coupon Table

Manages promotional coupons and gift cards.

```
1 CREATE TABLE "coupon" (
2     "id" bigint NOT NULL AUTO_INCREMENT,
3     "name" varchar(255) NOT NULL,
4     "gift" tinyint(1) NOT NULL DEFAULT '0',
5     "balance" decimal(10,2) NOT NULL DEFAULT '0.00',
6     "coupon_type" enum('Amount','Percent','GiftCard') NOT NULL,
7     "date_expired" date DEFAULT NULL,
8     "booking_id" bigint DEFAULT NULL,
9     "customer_id" bigint DEFAULT NULL,
10    "discount_amount" decimal(10,2) DEFAULT '0.00',
11    PRIMARY KEY ("id"),
12    KEY "fk_coupon_booking" ("booking_id"),
13    KEY "fk_coupon_customer" ("customer_id"),
14    CONSTRAINT "fk_coupon_booking" FOREIGN KEY ("booking_id")
15        REFERENCES "booking" ("id"),
16    CONSTRAINT "fk_coupon_customer" FOREIGN KEY ("customer_id")
17        REFERENCES "customer" ("user_id"),
18    CONSTRAINT "coupon_chk_1" CHECK ((balance >= 0))
19 ) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
```

Listing 14: Coupon table structure

2.4.5 Refund Table

Tracks refund requests and processing with optional coupon compensation.

```
1 CREATE TABLE "refund" (
2     "id" bigint NOT NULL AUTO_INCREMENT,
3     "booking_id" bigint NOT NULL,
4     "amount" decimal(10,2) NOT NULL,
5     "reason" varchar(255) DEFAULT NULL,
6     "status" varchar(255) NOT NULL DEFAULT 'Requested',
7     "created_time_at" datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
8     "processed_time_at" datetime DEFAULT NULL,
9     "coupon_id" bigint DEFAULT NULL,
10    PRIMARY KEY ("id","booking_id"),
11    KEY "fk_refund_coupon" ("coupon_id"),
12    KEY "idx_refund_booking_status" ("booking_id","status"),
13    CONSTRAINT "fk_refund_booking" FOREIGN KEY ("booking_id")
14        REFERENCES "booking" ("id"),
15    CONSTRAINT "fk_refund_coupon" FOREIGN KEY ("coupon_id")
16        REFERENCES "coupon" ("id"),
17    CONSTRAINT "refund_chk_1" CHECK ((amount >= 0))
18 ) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
```

Listing 15: Refund table structure

2.5 Food and Beverage Tables

2.5.1 FWB_Menu Table

Catalog of available food and beverage items.

```
1 CREATE TABLE "fwb_menu" (
2     "id" bigint NOT NULL AUTO_INCREMENT,
3     "name" varchar(255) NOT NULL,
4     "description" text,
5     "image" varchar(255) DEFAULT NULL,
6     "price" decimal(10,2) NOT NULL,
7     "category" varchar(255) NOT NULL,
8     "capacity" varchar(255) DEFAULT NULL,
9     PRIMARY KEY ("id"),
10    CONSTRAINT "fwb_menu_chk_1" CHECK (("price" >= 0))
11 ) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
```

Listing 16: FWB_Menu table structure

2.5.2 FWB and Contains_item Tables

These tables manage the N:M relationship between bookings and menu items.

```
1 CREATE TABLE "fwb" (
2     "id" bigint NOT NULL,
3     "booking_id" bigint NOT NULL,
4     "quantity" int NOT NULL DEFAULT '1',
5     "price" decimal(10,2) NOT NULL,
6     PRIMARY KEY ("id", "booking_id"),
7     KEY "idx_fwb_booking" ("booking_id"),
8     CONSTRAINT "fk_fwb_booking" FOREIGN KEY ("booking_id")
9         REFERENCES "booking" ("id"),
10        CONSTRAINT "fwb_chk_1" CHECK (("quantity" > 0)),
11        CONSTRAINT "fwb_chk_2" CHECK (("price" >= 0))
12 );
13
14 CREATE TABLE "contains_item" (
15     "fwb_menu_id" bigint NOT NULL,
16     "fwb_id" bigint NOT NULL,
17     "fwb_booking_id" bigint NOT NULL,
18     "quantity" int NOT NULL DEFAULT '1',
19     PRIMARY KEY ("fwb_menu_id", "fwb_id", "fwb_booking_id"),
20     KEY "fk_contains_item_fwb" ("fwb_id", "fwb_booking_id"),
21     KEY "idx_contains_fwb_booking_menu" ("fwb_booking_id", "fwb_menu_id"),
22     CONSTRAINT "fk_contains_item_fwb" FOREIGN KEY
23         ("fwb_id", "fwb_booking_id")
24         REFERENCES "fwb" ("id", "booking_id"),
25     CONSTRAINT "fk_contains_item_fwbmenu" FOREIGN KEY ("fwb_menu_id")
26         REFERENCES "fwb_menu" ("id"),
27     CONSTRAINT "contains_item_chk_1" CHECK (("quantity" > 0))
28 );
```

Listing 17: FWB and Contains_item structures



2.6 Gift Relationship Tables

These tables track coupon and booking gift transactions between customers.

```
1 CREATE TABLE "give" (
2     "coupon_id" bigint NOT NULL,
3     "sender_id" bigint NOT NULL,
4     "receiver_id" bigint NOT NULL,
5     PRIMARY KEY ("coupon_id"),
6     KEY "fk_give_sender" ("sender_id"),
7     KEY "fk_give_receiver" ("receiver_id"),
8     CONSTRAINT "fk_give_coupon" FOREIGN KEY ("coupon_id")
9         REFERENCES "coupon" ("id"),
10    CONSTRAINT "fk_give_receiver" FOREIGN KEY ("receiver_id")
11        REFERENCES "customer" ("user_id"),
12    CONSTRAINT "fk_give_sender" FOREIGN KEY ("sender_id")
13        REFERENCES "customer" ("user_id")
14 );
15
16 CREATE TABLE "send_gift" (
17     "booking_id" bigint NOT NULL,
18     "sender_id" bigint NOT NULL,
19     "receiver_id" bigint NOT NULL,
20     PRIMARY KEY ("booking_id"),
21     KEY "fk_sendgift_sender" ("sender_id"),
22     KEY "fk_sendgift_receiver" ("receiver_id"),
23     CONSTRAINT "fk_sendgift_booking" FOREIGN KEY ("booking_id")
24         REFERENCES "booking" ("id"),
25     CONSTRAINT "fk_sendgift_receiver" FOREIGN KEY ("receiver_id")
26         REFERENCES "customer" ("user_id"),
27     CONSTRAINT "fk_sendgift_sender" FOREIGN KEY ("sender_id")
28         REFERENCES "customer" ("user_id")
29 );
```

Listing 18: Gift relationship tables

2.7 Indexing Strategy

The database implements a indexing strategy for query optimization:

- **Primary Keys:** All tables have primary key indexes (B-Tree, Unique)
- **Foreign Keys:** Automatic indexes on all foreign key columns
- **Unique Constraints:** uk_user_email, uk_movie_url_slug, uk_payment_transaction
- **Composite Indexes:**
 - idx_booking_status_created (status, created_time_at) for filtering
 - idx_ss_stid_seatid_status for seat availability queries
 - idx_ss_booking_status for ticket lookup
- **Performance Indexes:**
 - idx_booking_customer for customer booking history
 - idx_payment_booking for payment lookups
 - idx_refund_booking_status for refund queries

3 Triggers

3.1 User Management Triggers

3.1.1 Trigger: Auto-Create Customer Record

Purpose: Automatically creates a customer record when a new user is inserted, implementing the User-Customer specialization relationship from the ERD.

Timing: AFTER INSERT on User table

Implementation:

```
1 DELIMITER $$  
2 CREATE TRIGGER "trg_after_user_insert"  
3 AFTER INSERT ON "User"  
4 FOR EACH ROW  
5 BEGIN  
6     DECLARE v_membership VARCHAR(50);  
7     DECLARE v_age INT;  
8     DECLARE v_valid_until DATE;  
9  
10    IF NOT EXISTS (SELECT 1 FROM staff WHERE user_id = NEW.id) THEN  
11        IF NEW.birthday IS NOT NULL THEN  
12            SET v_age = TIMESTAMPDIFF(YEAR, NEW.birthday, CURDATE());  
13        ELSE  
14            SET v_age = NULL;  
15        END IF;  
16  
17        IF v_age IS NOT NULL AND v_age < 22 THEN  
18            SET v_membership = 'U22';  
19        ELSE  
20            SET v_membership = 'Member';  
21        END IF;  
22  
23        IF MONTH(CURDATE()) < 6 OR  
24            (MONTH(CURDATE()) = 6 AND DAY(CURDATE()) = 1) THEN  
25            SET v_valid_until = DATE(CONCAT(YEAR(CURDATE()), '-06-01'));  
26        ELSE  
27            SET v_valid_until = DATE(CONCAT(YEAR(CURDATE()) + 1, '-06-01'));  
28        END IF;  
29  
30        INSERT INTO customer (  
31            user_id, accumulated_points, total_spent,  
32            membership_name, membership_valid_until  
33        )  
34        VALUES (  
35            NEW.id, 0, 0, v_membership, v_valid_until  
36        );  
37    END IF;  
38 END$$  
39 DELIMITER ;
```

Listing 19: Trigger: *trg_after_user_insert*

Business Logic:

- Only creates customer record if user is not a staff member



- Determines initial membership based on age (U22 for users under 22, otherwise Member)
- Sets membership valid until next June 1st (annual renewal cycle)
- Initializes `accumulated_points` and `total_spent` to 0

Integration: This trigger works seamlessly with the `sp_register_user` stored procedure, eliminating the need for application code to manually create customer records.

3.1.2 Trigger: Validate Customer Updates

Purpose: Validates customer data before updates to prevent negative values and maintain data integrity.

Timing: BEFORE UPDATE on `customer` table

Implementation:

```
1 DELIMITER $$  
2 CREATE TRIGGER "trg_before_customer_update"  
3 BEFORE UPDATE ON "customer"  
4 FOR EACH ROW  
5 BEGIN  
6     IF NEW.accumulated_points < 0 THEN  
7         SIGNAL SQLSTATE '45000'  
8         SET MESSAGE_TEXT = 'Accumulated points cannot be negative';  
9     END IF;  
10  
11    IF NEW.total_spent < 0 THEN  
12        SIGNAL SQLSTATE '45000'  
13        SET MESSAGE_TEXT = 'Total spent cannot be negative';  
14    END IF;  
15  
16 END$$  
17 DELIMITER ;
```

Listing 20: Trigger: `trg_before_customer_update`

3.2 Seat Capacity Management Triggers

These triggers automatically maintain the auditorium capacity field as seats are added, removed, or moved between auditoriums.

3.2.1 Trigger: Update Capacity on Seat Insert

Purpose: Increments auditorium capacity when a new seat is added.

Timing: AFTER INSERT on `seat` table

Implementation:



```
1 DELIMITER $$  
2 CREATE TRIGGER "trg_after_seat_insert"  
3 AFTER INSERT ON "seat"  
4 FOR EACH ROW  
5 BEGIN  
6     UPDATE auditorium  
7     SET capacity = (  
8         SELECT COUNT(*)  
9             FROM seat  
10            WHERE au_number = NEW.au_number  
11            AND au_theater_id = NEW.au_theater_id  
12    )  
13    WHERE number = NEW.au_number  
14    AND theater_id = NEW.au_theater_id;  
15 END$$  
16 DELIMITER ;
```

Listing 21: Trigger: *trg_after_seat_insert*

3.2.2 Trigger: Update Capacity on Seat Delete

Purpose: Decrements auditorium capacity when a seat is removed.

Timing: AFTER DELETE on `seat` table

Implementation:

```
1 DELIMITER $$  
2 CREATE TRIGGER "trg_after_seat_delete"  
3 AFTER DELETE ON "seat"  
4 FOR EACH ROW  
5 BEGIN  
6     UPDATE auditorium  
7     SET capacity = (  
8         SELECT COUNT(*)  
9             FROM seat  
10            WHERE au_number = OLD.au_number  
11            AND au_theater_id = OLD.au_theater_id  
12    )  
13    WHERE number = OLD.au_number  
14    AND theater_id = OLD.au_theater_id;  
15 END$$  
16 DELIMITER ;
```

Listing 22: Trigger: *trg_after_seat_delete*

3.2.3 Trigger: Update Capacity on Seat Move

Purpose: Updates capacity for both source and destination auditoriums when a seat is moved.

Timing: AFTER UPDATE on `seat` table

Implementation:



```
1 DELIMITER $$  
2 CREATE TRIGGER "trg_after_seat_update"  
3 AFTER UPDATE ON "seat"  
4 FOR EACH ROW  
5 BEGIN  
6     IF OLD.au_number != NEW.au_number OR  
7         OLD.au_theater_id != NEW.au_theater_id THEN  
8         UPDATE auditorium  
9             SET capacity = (  
10                 SELECT COUNT(*)  
11                     FROM seat  
12                     WHERE au_number = OLD.au_number  
13                         AND au_theater_id = OLD.au_theater_id  
14             )  
15             WHERE number = OLD.au_number  
16                 AND theater_id = OLD.au_theater_id;  
17     END IF;  
18  
19     UPDATE auditorium  
20         SET capacity = (  
21             SELECT COUNT(*)  
22                 FROM seat  
23                 WHERE au_number = NEW.au_number  
24                     AND au_theater_id = NEW.au_theater_id  
25             )  
26             WHERE number = NEW.au_number  
27                 AND theater_id = NEW.au_theater_id;  
28 END$$  
29 DELIMITER ;
```

Listing 23: Trigger: *trg_after_seat_update*

3.3 Showtime Scheduling Triggers

3.3.1 Trigger: Prevent Overlapping Showtimes on Insert

Purpose: Enforces the business rule that showtimes in the same auditorium must not overlap and must have at least 15 minutes gap for cleaning.

Timing: BEFORE INSERT on showtime table

Business Rules:

- Two showtimes cannot overlap in the same auditorium on the same date
- Minimum 15-minute gap required between consecutive showtimes
- Validation uses ADDTIME/SUBTIME for buffer calculation

Implementation:



```
1 DELIMITER $$  
2 CREATE TRIGGER "trg_before_showtime_insert"  
3 BEFORE INSERT ON "showtime"  
4 FOR EACH ROW  
5 BEGIN  
6     DECLARE conflict_count INT;  
7  
8     SELECT COUNT(*) INTO conflict_count  
9     FROM showtime  
10    WHERE au_number = NEW.au_number  
11        AND au_theater_id = NEW.au_theater_id  
12        AND date = NEW.date  
13        AND (  
14            (NEW.start_time < ADDTIME(end_time, '00:15:00')  
15            AND NEW.end_time > SUBTIME(start_time, '00:15:00'))  
16        );  
17  
18    IF conflict_count > 0 THEN  
19        SIGNAL SQLSTATE '45000'  
20        SET MESSAGE_TEXT =  
21            'Showtime conflicts with existing schedule. ' ||  
22            'Showtimes must not overlap and must have at least ' ||  
23            '15 minutes gap between them.';  
24    END IF;  
25 END$$  
26 DELIMITER ;
```

Listing 24: Trigger: *trg_before_showtime_insert*

3.3.2 Trigger: Prevent Overlapping Showtimes on Update

Purpose: Enforces the same scheduling rules when modifying existing showtimes.

Timing: BEFORE UPDATE on `showtime` table

Implementation:



```
1 DELIMITER $$  
2 CREATE TRIGGER "trg_before_showtime_update"  
3 BEFORE UPDATE ON "showtime"  
4 FOR EACH ROW  
5 BEGIN  
6     DECLARE conflict_count INT;  
7  
8     IF NEW.date != OLD.date  
9         OR NEW.start_time != OLD.start_time  
10        OR NEW.end_time != OLD.end_time  
11        OR NEW.au_number != OLD.au_number  
12        OR NEW.au_theater_id != OLD.au_theater_id THEN  
13  
14         SELECT COUNT(*) INTO conflict_count  
15         FROM showtime  
16         WHERE id != NEW.id  
17             AND au_number = NEW.au_number  
18             AND au_theater_id = NEW.au_theater_id  
19             AND date = NEW.date  
20             AND (  
21                 (NEW.start_time < ADDTIME(end_time, '00:15:00')  
22                     AND NEW.end_time > SUBTIME(start_time, '00:15:00'))  
23 );  
24  
25     IF conflict_count > 0 THEN  
26         SIGNAL SQLSTATE '45000'  
27         SET MESSAGE_TEXT =  
28             'Showtime conflicts with existing schedule.';  
29     END IF;  
30     END IF;  
31 END$$  
32 DELIMITER ;
```

Listing 25: Trigger: *trg_before_showtime_update*

3.4 Booking and Payment Triggers

3.4.1 Trigger: Validate Booking Time and Seat Availability

Purpose: Prevents booking tickets for showtimes that have already started or are too close to start time for online bookings.

Timing: BEFORE INSERT on `showtime_seat` table

Business Rules:

- Cannot book tickets for showtimes that have already started or passed
- Online bookings must be completed at least 15 minutes before showtime
- Staff counter bookings can be made closer to showtime (bypass 15-minute rule)
- Uses Vietnam timezone (GMT+7) for time comparisons

Implementation:



```
1 DELIMITER $$  
2 CREATE TRIGGER "trg_before_showtime_seat_insert"  
3 BEFORE INSERT ON "showtime_seat"  
4 FOR EACH ROW  
5 BEGIN  
6     DECLARE v_showtime_datetime DATETIME;  
7     DECLARE v_booking_method VARCHAR(255);  
8     DECLARE v_minutes_until_showtime INT;  
9     DECLARE v_current_datetime_vn DATETIME;  
10  
11     SET v_current_datetime_vn = CONVERT_TZ(NOW(), '+00:00', '+07:00');  
12  
13     SELECT TIMESTAMP(date, start_time) INTO v_showtime_datetime  
14     FROM showtime  
15     WHERE id = NEW.st_id;  
16  
17     IF v_showtime_datetime <= v_current_datetime_vn THEN  
18         SIGNAL SQLSTATE '45000'  
19         SET MESSAGE_TEXT =  
20             'Cannot book tickets for showtimes that have ' ||  
21             'already started or passed.';  
22     END IF;  
23  
24     IF NEW.status = 'Held' AND NEW.booking_id IS NOT NULL THEN  
25         SELECT booking_method INTO v_booking_method  
26         FROM booking  
27         WHERE id = NEW.booking_id;  
28  
29         IF v_booking_method = 'Online' THEN  
30             SET v_minutes_until_showtime =  
31                 TIMESTAMPDIFF(MINUTE, v_current_datetime_vn,  
32                             v_showtime_datetime);  
33  
34             IF v_minutes_until_showtime < 15 THEN  
35                 SIGNAL SQLSTATE '45000'  
36                 SET MESSAGE_TEXT =  
37                     'Online booking must be completed at least ' ||  
38                     '15 minutes before showtime start time.';  
39             END IF;  
40         END IF;  
41     END IF;  
42 END$$  
43 DELIMITER ;
```

Listing 26: Trigger: *trg_before_showtime_seat_insert*

3.4.2 Trigger: Set Booking Gift Flag

Purpose: Automatically sets the `is_gift` flag on a booking when a gift transaction is recorded.

Timing: AFTER INSERT on `send_gift` table

Implementation:



```
1 DELIMITER $$  
2 CREATE TRIGGER "trg_sendgift_set_isgift"  
3 AFTER INSERT ON "send_gift"  
4 FOR EACH ROW  
5 BEGIN  
6     UPDATE booking  
7         SET is_gift = 1  
8         WHERE id = NEW.booking_id;  
9 END$$  
10 DELIMITER ;
```

Listing 27: Trigger: *trg_sendgift_set_isgift*

4 Stored Procedures

4.1 Authentication and User Management Procedures

4.1.1 Procedure: *sp_register_user*

Implementation:

```
1 CREATE PROCEDURE sp_register_user(...)  
2 BEGIN  
3     DECLARE v_email_valid BOOLEAN;  
4     DECLARE v_age_valid BOOLEAN;  
5  
6     DECLARE EXIT HANDLER FOR SQLEXCEPTION  
7     BEGIN  
8         GET DIAGNOSTICS CONDITION 1 v_error_msg = MESSAGE_TEXT;  
9         ROLLBACK;  
10        SET p_success = FALSE;  
11        SET p_message = CONCAT('Database error: ', v_error_msg);  
12    END;  
13  
14    START TRANSACTION;  
15  
16    SET v_email_valid = fn_validate_email(p_email);  
17    IF NOT v_email_valid THEN  
18        SET p_success = FALSE;  
19        SET p_message = 'Invalid email format';  
20        ROLLBACK;  
21    ELSE  
22        INSERT INTO `User` (...) VALUES (...);  
23        SET p_user_id = LAST_INSERT_ID();  
24        COMMIT;  
25    END IF;  
26 END;
```

Listing 28: *sp_register_user* procedure

Purpose: Registers a new user with validation.

Parameters:



- IN: p_fname, p_minit, p_lname, p_email, p_password, p_birthday, p_gender, p_district, p_city
- OUT: p_user_id (generated ID), p_success (boolean), p_message (status message)

Validation Logic:

- Email format validation using fn_validate_email()
- Email uniqueness check using fn_check_email_exists()
- Age validation (must be ≥ 13 years) using fn_validate_age()
- Password must be bcrypt-hashed before calling (handled by application)

4.1.2 Procedure: sp_login_user

Purpose: Retrieves user credentials for authentication.

Parameters:

- IN: p_email
- OUT: p_user_id, p_password, p_fname, p_lname, p_success, p_message

Note: Password verification using bcrypt is performed in the application layer (NestJS) after retrieving the hashed password.

4.1.3 Procedure: sp_update_user_profile

Purpose: Updates user profile information.

Business Rules:

- Only non-null parameters are updated (partial updates supported)
- birthday cannot be changed after registration (security requirement)
- email cannot be changed (would require separate email change with verification)

4.2 Showtime Management Procedures

4.2.1 Procedure: sp_delete_showtime

Purpose: Deletes a showtime from the system with proper authorization and validation checks.

Parameters:

- IN: p_showtime_id (ID of showtime to delete)
- IN: p_staff_id (ID of staff requesting deletion)

Business Rules:

1. If any customer has booked tickets for this showtime
2. If staff role is not Admin or Manager (e.g., regular Staff)
3. If showtime ID does not exist

Implementation:

```
1 CREATE PROCEDURE sp_delete_showtime(
2     IN p_showtime_id BIGINT,
3     IN p_staff_id BIGINT
4 )
5 BEGIN
6     DECLARE v_showtime_exists INT;
7     DECLARE v_ticket_count INT;
8     DECLARE v_staff_role VARCHAR(255);
9
10    SELECT `role` INTO v_staff_role
11    FROM staff
12    WHERE user_id = p_staff_id;
13
14    IF v_staff_role IS NULL THEN
15        SIGNAL SQLSTATE '45000'
16        SET MESSAGE_TEXT = 'Staff not found or unauthorized';
17    END IF;
18
19    IF v_staff_role NOT IN ('Admin', 'Manager') THEN
20        SIGNAL SQLSTATE '45000'
21        SET MESSAGE_TEXT =
22            'Only Admin or Manager staff can delete showtime';
23    END IF;
24
25    SELECT COUNT(*) INTO v_showtime_exists
26    FROM showtime
27    WHERE id = p_showtime_id;
28
29    IF v_showtime_exists = 0 THEN
30        SIGNAL SQLSTATE '45000'
31        SET MESSAGE_TEXT = 'Showtime not found';
32    END IF;
33
34    SELECT COUNT(*) INTO v_ticket_count
35    FROM showtime_seat
36    WHERE st_id = p_showtime_id
37    AND booking_id IS NOT NULL;
38
39    IF v_ticket_count > 0 THEN
40        SIGNAL SQLSTATE '45000'
41        SET MESSAGE_TEXT =
42            'Cannot delete showtime: ' ||
43            'There are tickets already booked';
44    END IF;
45
46    DELETE FROM showtime_seat WHERE st_id = p_showtime_id;
47    DELETE FROM showtime WHERE id = p_showtime_id;
48
49 END;
```

Listing 29: *sp_delete_showtime* procedure

Deletion Strategy:

- **Hard Delete:** Showtime is permanently removed from database



- **Cascade Deletion:** All associated `showtime_seat` records are deleted first
- **No Soft Delete:** System does not use status-based soft deletion for showtimes
- **Rationale:** Showtimes without bookings have no historical value and can be safely removed to maintain database cleanliness

Alternative Approach for Booked Showtimes:

- For showtimes with existing bookings, use `sp_cancel_booking` to process refunds
- After all bookings are cancelled and refunded, the showtime can be deleted
- This ensures customer rights are protected and refund policies are enforced

4.3 Booking Management Procedures

4.3.1 Procedure: `sp_start_booking`

Purpose: Creates a new booking with seat reservations and optional F&B items in a single transaction.
Parameters:

- IN: `p_customer_id`, `p_showtime_id`, `p_seat_ids_json` (array), `p_fwb_items_json` (array or NULL)
- OUT: `p_booking_id`

Transaction Flow:

1. Validate showtime exists and get auditorium info
2. Create booking record with `status='Pending'`
3. Loop through seat IDs from JSON array:
 - Verify seat exists in showtime's auditorium
 - Check seat is not already Held or Booked
 - Insert `showtime_seat` with `status='Held'`
4. If F&B items provided:
 - Create parent FWB record
 - Loop through items, validate menu IDs, insert `contains_item` records
 - Calculate total F&B price
5. COMMIT if all seats successfully held, ROLLBACK on any error

Implementation:



```
1 CREATE PROCEDURE sp_start_booking(...)
2 BEGIN
3     DECLARE EXIT HANDLER FOR SQLEXCEPTION
4     BEGIN
5         ROLLBACK;
6         RESIGNAL;
7     END;
8
9     START TRANSACTION;
10
11    INSERT INTO booking (created_time_at, status, booking_method, ...)
12    VALUES (NOW(), 'Pending', 'Online', ...);
13    SET p_booking_id = LAST_INSERT_ID();
14
15    WHILE v_idx < v_len DO
16        SET v_seat_id = CAST(JSON_UNQUOTE(
17            JSON_EXTRACT(p_seat_ids_json, CONCAT('$[', v_idx, ']'))
18        ) AS UNSIGNED);
19
20        IF EXISTS (
21            SELECT 1 FROM showtime_seat
22            WHERE st_id = p_showtime_id
23            AND seat_id = v_seat_id
24            AND status IN ('Held', 'Booked')
25        ) THEN
26            SIGNAL SQLSTATE '45000'
27            SET MESSAGE_TEXT = 'Seat already taken';
28        END IF;
29
30        INSERT INTO showtime_seat (... )
31        VALUES (... , 'Held', ... );
32
33        SET v_idx = v_idx + 1;
34    END WHILE;
35
36    COMMIT;
37 END;
```

Listing 30: *sp_start_booking - seat locking logic*

4.3.2 Procedure: sp_confirm_payment

Purpose: Processes payment confirmation, updates booking status, converts held seats to booked, and awards loyalty points.

Parameters:

- IN: p_booking_id, p_payment_method, p_transaction_id, p_total_amount, p_duration
- OUT: p_payment_id

Complex Business Logic:

1. Validate booking is in 'Pending' status
2. Deduct used points from customer accumulated_points
3. Insert payment record with status='Success'



4. Update booking status to 'Paid'
5. Update showtime_seat status from 'Held' to 'Booked'
6. Calculate final amount paid (after discounts)
7. Calculate points earned based on membership tier rates
8. Update customer total_spent and accumulated_points
9. Store points_earned in booking record

Points Calculation Logic:

```
1 CALL sp_calculate_final_amount(p_booking_id,
2     v_base_seat_price, v_fwb_price, v_subtotal,
3     v_coupon_discount, v_box_office_discount,
4     v_concession_discount, v_points_discount,
5     v_membership_tier, v_coupon_type, v_final_amount);
6
7 SELECT m.box_office_point_rate, m.concession_point_rate
8 INTO v_ticket_rate, v_fnb_rate
9 FROM customer c
10 JOIN membership m ON m.tier_name = c.membership_name
11 WHERE c.user_id = v_customer_id;
12
13 SET v_ticket_amount = v_final_amount * (v_base_seat_price / v_subtotal);
14 SET v_fnb_amount = v_final_amount * (v_fwb_price / v_subtotal);
15
16 SET v_ticket_points = fn_calculate_points(v_ticket_amount, v_ticket_rate);
17 SET v_fnb_points = fn_calculate_points(v_fnb_amount, v_fnb_rate);
18 SET v_total_points = v_ticket_points + v_fnb_points;
19
20 UPDATE customer
21 SET total_spent = total_spent + v_final_amount,
22     accumulated_points = accumulated_points + v_total_points
23 WHERE user_id = v_customer_id;
```

Listing 31: *sp_confirm_payment - points calculation*

4.3.3 Procedure: sp_cancel_expired_bookings

Purpose: Automatically cancels bookings that have exceeded the payment timeout period (batch operation for cleanup job).

Parameters:

- IN: p_timeout_minutes (e.g., 15 minutes)
- OUT: p_cancelled_count, p_failed_count

Implementation Strategy:

- Uses CURSOR to iterate through expired pending bookings
- Each booking processed in separate transaction to prevent all-or-nothing rollback
- Creates cancelled payment record for audit trail
- Releases held seats back to Available status
- Continues processing even if individual cancellations fail
- Returns success and failure counts



4.4 Payment and Refund Procedures

4.4.1 Procedure: sp_apply_coupon

Purpose: Applies a coupon to a booking with validation.

Validation Rules:

- Coupon must belong to customer
- Coupon must not be already used (`booking_id IS NULL`)
- Coupon must not be expired
- Coupon must have positive `balance`

Key Implementation Detail:

```
1 UPDATE coupon
2 SET booking_id = p_booking_id,
3     discount_amount = balance,
4     balance = 0
5 WHERE id = p_coupon_id;
```

Listing 32: `sp_apply_coupon` - discount tracking

4.4.2 Procedure: sp_apply_points

Purpose: Applies loyalty points to a booking for payment discount.

Business Rules:

- Minimum 10 points required for redemption
- Points can cover up to 90% of transaction value
- 1 point = 1,000 VND
- Points are only deducted from customer when payment is confirmed (not when applied)

4.4.3 Procedure: sp_create_refund_coupon

Purpose: Creates a refund and issues a compensation coupon valid for 1 year.

Transaction Flow:

1. Verify booking ownership and status
2. Validate refund amount > 0
3. Create refund record with `status='Completed'`
4. Create compensation coupon (`type=GiftCard`, valid 1 year)
5. Link coupon to refund
6. Update booking `status` to '`Cancelled`'

4.5 Query and Reporting Procedures

4.5.1 Procedure: sp_get_movies_with_details

Purpose: Optimized query that returns movies with ALL related data in a single database call (eliminates N+1 query problem).

Features:

- Joins movie, director, actor, genre, subtitle, dubbing tables
- Uses GROUP_CONCAT to aggregate multivalued attributes
- Filters by status (now, upcoming, ended, all)
- Supports pagination with LIMIT and OFFSET
- Returns denormalized result set for efficient frontend rendering

Implementation:

```
1 CREATE PROCEDURE sp_get_movies_with_details(
2     IN p_status VARCHAR(20),
3     IN p_limit INT,
4     IN p_offset INT
5 )
6 BEGIN
7     SELECT
8         m.id, m.name, m.duration, m.language, m.release_date,
9         m.end_date, m.age_rating, m.poster_file, m.url_slug,
10        m.description, m.trailer_url,
11        GROUP_CONCAT(DISTINCT d.director ORDER BY d.director
12                      SEPARATOR '|||') AS directors,
13        GROUP_CONCAT(DISTINCT a.actor ORDER BY a.actor
14                      SEPARATOR '|||') AS actors,
15        GROUP_CONCAT(DISTINCT g.genre ORDER BY g.genre
16                      SEPARATOR '|||') AS genres,
17        GROUP_CONCAT(DISTINCT s.subtitle ORDER BY s.subtitle
18                      SEPARATOR '|||') AS subtitles,
19        GROUP_CONCAT(DISTINCT db.dubbing ORDER BY db.dubbing
20                      SEPARATOR '|||') AS dubbing_options
21     FROM movie m
22     LEFT JOIN director d ON m.id = d.movie_id
23     LEFT JOIN actor a ON m.id = a.movie_id
24     LEFT JOIN genre g ON m.id = g.movie_id
25     LEFT JOIN subtitle s ON m.id = s.movie_id
26     LEFT JOIN dubbing db ON m.id = db.movie_id
27     WHERE (
28         p_status = 'all'
29         OR (p_status = 'now' AND m.release_date <= CURDATE()
30             AND (m.end_date IS NULL OR m.end_date >= CURDATE()))
31         OR (p_status = 'upcoming' AND m.release_date > CURDATE())
32         OR (p_status = 'ended' AND m.end_date < CURDATE())
33     )
34     GROUP BY m.id, ...
35     ORDER BY m.release_date DESC
36     LIMIT p_limit OFFSET p_offset;
37 END;
```

Listing 33: sp_get_movies_with_details - optimized query



4.5.2 Procedure: sp_get_customer_bookings

Purpose: Retrieves customer's booking history with complete details including tickets, F&B, and calculated totals.

Return Values:

- Result Set 1: Array of bookings with nested details (JSON for F&B items)
- Result Set 2: Pagination metadata (`totalCount`, `limit`, `offset`, `hasMore`)

Complex Calculation:

- Uses temporary table to store calculated final amounts per booking
- Calls `sp_calculate_final_amount` for each booking
- Converts UTC timestamps to Vietnam timezone (GMT+7)
- Aggregates seat names into comma-separated list
- Returns F&B items as JSON array

4.5.3 Procedure: sp_calculate_final_amount

Purpose: Centralized calculation of booking total with all discounts applied.

Parameters:

- IN: `p_booking_id`
- OUT: `p_base_seat_price`, `p_fwb_price`, `p_subtotal`, `p_membership_tier`, `p_coupon_discount`, `p_box_office_discount`, `p_concession_discount`, `p_points_discount`, `p_membership_tier`, `p_final_amount`

Calculation Logic:

1. Calculate base ticket price (sum of `showtime_seat` prices)
2. Calculate F&B total (sum of `fwb price` \times `quantity`)
3. Get coupon discount (from `coupon.discount_amount` where `booking_id` matches)
4. Calculate points discount (`points_used` \times 1000)
5. Final amount = `subtotal` - `coupon_discount` - `points_discount`
6. Ensure final amount ≥ 0

Note: Current implementation does not apply membership percentage discounts (set to 0), only tracks tier for display purposes.

4.5.4 Procedure: sp_generate_sales_report

Purpose: Generates sales report for a theater within a date range.

Features:

- Uses CURSOR to iterate through showtimes
- Calculates tickets sold and revenue per showtime
- Includes F&B revenue linked to showtime bookings
- Creates temporary table for results
- Returns aggregated data ordered by date

Parameters Validation:

- Theater ID required and must exist
- Start date and end date required
- Start date must be \leq end date
- Returns meaningful error messages via SIGNAL

4.6 Customer Dashboard Procedures

4.6.1 Procedure: sp_get_customer_dashboard

Purpose: Retrieves summary statistics and recent activities for customer dashboard.

Return Values:

- Result Set 1: Stats (`totalPoints`, `totalGiftCards`, `totalVouchers`, `totalBookings`)
- Result Set 2: Recent activities (last 10 activities combining bookings and coupons)

4.6.2 Procedure: sp_get_customer_points

Purpose: Retrieves customer's point balance and transaction history with pagination.

Point History Categories:

- EARNED: Points earned from completed payments
- USED: Points applied for payment discounts
- REFUNDED: Points deducted when booking is refunded

Implementation Note: Uses UNION ALL to combine point transactions from different sources, then applies pagination.

4.6.3 Procedure: sp_get_membership_card

Purpose: Generates membership card information for display.

Unique Card Number Generation:

```
1 CONCAT(
2     LPAD(c.user_id, 4, '0'), '-',
3     LPAD(MOD(c.user_id * 7919, 10000), 4, '0'), '-',
4     LPAD(MOD(c.user_id * 4999 + 1000, 10000), 4, '0'), '-',
5     LPAD(MOD(c.user_id * 9973 + 5000, 10000), 4, '0')
6 ) AS card_number
```

Listing 34: Membership card number generation

4.7 Gift Transaction Procedures

4.7.1 Procedure: sp_gift_coupon

Purpose: Transfers coupon ownership from sender to receiver.

Business Rules:

- Cannot gift to yourself
- Receiver must exist and be a customer
- Sender must own the coupon
- Coupon cannot be already applied to booking



- Coupon cannot be expired
- Coupon can only be gifted once (gift flag check)

Transaction Safety:

- Uses row-level locking (FOR UPDATE) to prevent race conditions
- Checks ROW_COUNT() after UPDATE to detect concurrent modifications
- Records transaction in give table for audit trail

4.7.2 Procedure: sp_send_gift

Purpose: Transfers booking ownership as a gift.

Validation and Process:

- Validates booking status (must be Confirmed or Paid)
- Checks booking hasn't been previously gifted
- Updates booking customer_id to receiver
- Sets is_gift flag to 1 (via trigger)
- Records transaction in send_gift table

5 Functions

Functions in MySQL provide reusable computational logic that can be called within SQL statements, stored procedures, or triggers. The BKinema system implements 9 functions that handle validation, calculation, data transformation, and revenue analysis tasks. This section documents each function with its purpose, implementation, and integration with other database objects.

5.1 Validation Functions

These functions perform data validation and are used extensively by stored procedures and triggers to enforce business rules.

5.1.1 Function: fn_validate_email

Purpose: Validates email format using regex pattern matching.

Returns: BOOLEAN (1 if valid, 0 if invalid)

Implementation:

```
1 CREATE FUNCTION fn_validate_email(p_email VARCHAR(255))
2 RETURNS BOOLEAN
3 DETERMINISTIC
4 BEGIN
5     RETURN p_email REGEXP
6         '^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.\.[A-Za-z]{2,}+$';
7 END;
```

Listing 35: fn_validate_email function

Usage: Called by sp_register_user to validate email format before account creation.

Regex Pattern Explanation:

- [A-Za-z0-9._%+-]+: Local part (before @) allows alphanumeric and common special characters



- @: Literal @ symbol
- [A-Za-z0-9.-]+: Domain name allows alphanumeric, hyphens, dots
- .[A-Za-z]{2,}: Top-level domain (.com, .net, etc.) requires at least 2 letters

5.1.2 Function: fn_check_email_exists

Purpose: Checks if an email is already registered in the system.

Returns: BOOLEAN (1 if exists, 0 if not)

Implementation:

```
1 CREATE FUNCTION fn_check_email_exists(p_email VARCHAR(255))
2 RETURNS TINYINT(1)
3 READS SQL DATA
4 BEGIN
5     DECLARE email_count INT;
6     SELECT COUNT(*) INTO email_count
7     FROM `User`
8     WHERE email = p_email;
9     RETURN email_count > 0;
10 END;
```

Listing 36: fn_check_email_exists function

Usage: Called by sp_register_user to prevent duplicate email registrations.

Declaration Note: READS SQL DATA indicates the function queries the database but doesn't modify data.

5.1.3 Function: fn_validate_age

Purpose: Validates that user's age meets minimum requirement (13 years).

Returns: BOOLEAN (1 if valid, 0 if invalid)

Implementation:

```
1 CREATE FUNCTION fn_validate_age(p_birthday DATE)
2 RETURNS BOOLEAN
3 DETERMINISTIC
4 BEGIN
5     RETURN TIMESTAMPDIFF(YEAR, p_birthday, CURDATE()) >= 13;
6 END;
```

Listing 37: fn_validate_age function

Usage: Called by sp_register_user and user update procedures to enforce minimum age policy.

TIMESTAMPDIFF Note: Calculates complete years between birthday and current date, properly handling leap years and varying month lengths.

5.1.4 Function: fn_validate_password_hash

Purpose: Validates that password is properly bcrypt-hashed.

Returns: BOOLEAN (1 if valid hash, 0 if not)

Implementation:

```
1 CREATE FUNCTION fn_validate_password_hash(p_password VARCHAR(255))
2 RETURNS TINYINT(1)
3 DETERMINISTIC
4 BEGIN
5     RETURN p_password IS NOT NULL
6         AND LENGTH(p_password) = 60
7         AND p_password LIKE '$2%';
8 END;
```

Listing 38: *fn_validate_password_hash function*

Validation Criteria:

- Password must not be NULL
- Length must be exactly 60 characters (bcrypt hash format)
- Must start with \$2 (bcrypt algorithm identifier: \$2a\$, \$2b\$, \$2y\$)

Security Note: Password hashing is performed in the application layer (NestJS with bcrypt library) before calling stored procedures. This function only validates the format.

5.2 Calculation Functions

5.2.1 Function: fn_calculate_points

Purpose: Calculates loyalty points earned from spending amount based on membership tier rate.

Parameters:

- p_amount: Spending amount in VND
- p_rate: Point earning rate as percentage (0-100)

Returns: INT (points earned, rounded to nearest integer)

Implementation:

```
1 CREATE FUNCTION fn_calculate_points(
2     p_amount DECIMAL(10,2),
3     p_rate DECIMAL(5,2)
4 )
5 RETURNS INT
6 DETERMINISTIC
7 BEGIN
8     DECLARE v_raw_points DECIMAL(10,2);
9     DECLARE v_points INT;
10
11    SET v_raw_points = (p_amount * p_rate / 100) / 1000;
12    SET v_points = ROUND(v_raw_points, 0);
13
14    RETURN v_points;
15 END;
```

Listing 39: *fn_calculate_points function*

Calculation Example:

- Spending: 100,000 VND

- Member rate: 5% box office, 3% concession
- Ticket points: $(100,000 \times 5 / 100) / 1000 = 5$ points
- F&B points: $(50,000 \times 3 / 100) / 1000 = 1.5 \rightarrow 2$ points (rounded)

Usage: Called by `sp_confirm_payment` and `sp_update_customer_membership` to calculate points earned from completed transactions.

Design Rationale:

- Centralized calculation ensures consistency across all point-earning scenarios
- Rounding applied after rate calculation prevents accumulation of rounding errors
- Different rates for box office vs. concessions supported through separate function calls

5.2.2 Function: `fn_count_showtimes_in_range`

Purpose: Counts total number of showtimes for a theater within a date range.

Implementation:

```
1 CREATE FUNCTION "fn_count_showtimes_in_range"(
2     p_theater_id INT,
3     p_start_date DATE,
4     p_end_date DATE
5 ) RETURNS int
6     READS SQL DATA
7 BEGIN
8     DECLARE v_total INT DEFAULT 0;
9     DECLARE v_current_date DATE;
10
11    IF p_theater_id IS NULL OR p_theater_id <= 0 THEN RETURN -1; END IF;
12    IF p_start_date IS NULL OR p_end_date IS NULL THEN RETURN -2; END IF;
13    IF p_start_date > p_end_date THEN RETURN -3; END IF;
14
15    SET v_current_date = p_start_date;
16
17    date_loop: LOOP
18        IF v_current_date > p_end_date THEN
19            LEAVE date_loop;
20        END IF;
21
22        SET v_total = v_total + (
23            SELECT COUNT(*)
24            FROM showtime
25            WHERE au_theater_id = p_theater_id
26            AND date = v_current_date
27        );
28
29        SET v_current_date = v_current_date + INTERVAL 1 DAY;
30    END LOOP;
31
32    RETURN v_total;
33 END;
```

Listing 40: `fn_count_showtimes_in_range` function

Parameters:

- p_theater_id: Theater ID to query
- p_start_date: Range start date
- p_end_date: Range end date

Returns: INT (total showtime count, or negative error code)

Error Codes:

- -1: Invalid theater ID (NULL or ≤ 0)
- -2: NULL date parameters
- -3: Start date after end date

5.3 Membership Management Functions

5.3.1 Function: fn_get_default_membership

Purpose: Returns the default membership tier for new customers.

Returns: VARCHAR(50) (tier name)

Implementation:

```
1 CREATE FUNCTION fn_get_default_membership()
2 RETURNS VARCHAR(50) CHARSET utf8mb4 COLLATE utf8mb4_unicode_ci
3 READS SQL DATA
4 DETERMINISTIC
5 BEGIN
6     RETURN 'Member';
7 END;
```

Listing 41: fn_get_default_membership function

Usage: Called by trg_after_user_insert trigger when creating customer records for users who don't qualify for U22 (based on age).

Design Note: This function encapsulates the default tier logic, allowing future changes to default tier assignment without modifying trigger code.

5.4 Revenue Analysis Functions

These functions implement complex business analytics using cursors and loops to calculate revenue metrics that support management decision-making and performance analysis.

5.4.1 Function: fn_calculate_avg_booking_value_by_month

Purpose: Calculates the average booking value (tickets + F&B) for all paid bookings within a specific month and year. This metric helps management track booking trends and revenue performance over time.

Parameters:

- p_year: Year to analyze (INT)
- p_month: Month to analyze (1-12)

Returns: DECIMAL(10,2) (average booking value in VND, or negative error code)

Implementation:

```
1 CREATE FUNCTION fn_calculate_avg_booking_value_by_month(p_year INT, p_month INT)
2 RETURNS DECIMAL(10,2)
3 READS SQL DATA
4 BEGIN
5     DECLARE v_total_amount DECIMAL(12,2) DEFAULT 0.00;
6     DECLARE v_booking_id BIGINT;
7     DECLARE v_booking_value DECIMAL(12,2);
8     DECLARE v_booking_status VARCHAR(255);
9     DECLARE v_booking_count INT DEFAULT 0;
10    DECLARE v_done INT DEFAULT 0;
11
12    DECLARE booking_cursor CURSOR FOR
13        SELECT b.id, b.status
14        FROM booking b
15        WHERE YEAR(b.created_time_at) = p_year AND MONTH(b.created_time_at) =
16            ↪ p_month;
17
18    DECLARE CONTINUE HANDLER FOR NOT FOUND SET v_done = 1;
19
20    IF p_year IS NULL OR p_year < 2000 OR p_year > 2100 THEN RETURN -1; END IF;
21    IF p_month IS NULL OR p_month < 1 OR p_month > 12 THEN RETURN -2; END IF;
22
23    OPEN booking_cursor;
24    booking_loop: LOOP
25        FETCH booking_cursor INTO v_booking_id, v_booking_status;
26        IF v_done = 1 THEN LEAVE booking_loop; END IF;
27        IF v_booking_status = 'Paid' THEN
28            SET v_booking_value = (
29                SELECT COALESCE(SUM(ss.price), 0)
30                FROM showtime_seat ss
31                WHERE ss.booking_id = v_booking_id AND ss.status IN ('Held',
32                    ↪ 'Booked')
33            ) +
34            (SELECT COALESCE(SUM(f.price * f.quantity), 0)
35                FROM fwb f
36                WHERE f.booking_id = v_booking_id
37            );
38            SET v_total_amount = v_total_amount + v_booking_value;
39            SET v_booking_count = v_booking_count + 1;
40        END IF;
41    END LOOP;
42    CLOSE booking_cursor;
43
44    IF v_booking_count = 0 THEN RETURN 0.00; END IF;
45
46    RETURN ROUND(v_total_amount / v_booking_count, 2);
47END;
```

Listing 42: *fn_calculate_avg_booking_value_by_month function*

5.4.2 Function: fn_calculate_total_revenue_by_customer

Purpose: Calculates the total lifetime revenue generated by a specific customer across all their paid bookings, including both ticket sales and F&B purchases. This function supports customer value analysis and VIP tier qualification.

Implementation:

```
1 CREATE FUNCTION fn_calculate_total_revenue_by_customer(p_customer_id BIGINT)
2 RETURNS DECIMAL(12,2)
3 READS SQL DATA
4 BEGIN
5     DECLARE v_total_revenue DECIMAL(12,2) DEFAULT 0.00;
6     DECLARE v_item_price DECIMAL(10,2);
7     DECLARE v_done INT DEFAULT 0;
8
9     DECLARE ticket_cursor CURSOR FOR
10        SELECT ss.price
11        FROM showtime_seat ss
12        INNER JOIN booking b ON ss.booking_id = b.id
13        WHERE b.customer_id = p_customer_id AND b.status = 'Paid' AND ss.status IN
14            ('Held', 'Booked');
15
16     DECLARE fwb_cursor CURSOR FOR
17        SELECT f.price * f.quantity
18        FROM fwb f
19        INNER JOIN booking b ON f.booking_id = b.id
20        WHERE b.customer_id = p_customer_id AND b.status = 'Paid';
21
22     DECLARE CONTINUE HANDLER FOR NOT FOUND SET v_done = 1;
23
24     IF p_customer_id IS NULL OR p_customer_id <= 0 THEN RETURN -1; END IF;
25
26     IF NOT EXISTS (SELECT 1 FROM customer WHERE user_id = p_customer_id)
27     THEN RETURN -2;
28     END IF;
29
30     OPEN ticket_cursor;
31     ticket_loop: LOOP
32        FETCH ticket_cursor INTO v_item_price;
33        IF v_done = 1 THEN LEAVE ticket_loop; END IF;
34        SET v_total_revenue = v_total_revenue + v_item_price;
35    END LOOP;
36     CLOSE ticket_cursor;
37
38     SET v_done = 0;
39     OPEN fwb_cursor;
40     fwb_loop: LOOP
41        FETCH fwb_cursor INTO v_item_price;
42        IF v_done = 1 THEN LEAVE fwb_loop; END IF;
43        SET v_total_revenue = v_total_revenue + v_item_price;
44    END LOOP;
45     CLOSE fwb_cursor;
46
47     RETURN ROUND(v_total_revenue, 2);
END;
```

Listing 43: *fn_calculate_total_revenue_by_customer function***Parameters:**

- **p_customer_id:** Customer user ID to analyze (BIGINT)

Returns: DECIMAL(12,2) (total revenue in VND, or negative error code)

Business Use Cases:

- Customer lifetime value (CLV) calculation
- VIP and VVIP tier qualification based on total spending
- Identification of high-value customers for targeted marketing
- Customer segmentation for personalized offers and promotions
- Revenue attribution per customer for financial reporting

5.5 Function Integration with Database Objects

Usage in Stored Procedures:

- `sp_register_user`: Uses `fn_validate_email`, `fn_check_email_exists`, `fn_validate_age`
- `sp_confirm_payment`: Uses `fn_calculate_points` for loyalty point calculation
- `sp_update_customer_membership`: Uses `fn_calculate_points` for separate box office and concession calculations

Usage in Triggers:

- `trg_after_user_insert`: Uses `fn_get_default_membership` to assign initial tier

5.6 Function Call Examples

Example 1: Email Validation in Registration

```
1 SET v_email_valid = fn_validate_email('user@example.com');
2 IF NOT v_email_valid THEN
3     SET p_message = 'Invalid email format';
4     ROLLBACK;
5 END IF;
```

Listing 44: Email validation usage example

Example 2: Points Calculation with Different Rates

```
1 -- Member tier: 5% box office, 3% concession
2 SET v_ticket_points = fn_calculate_points(100000, 5.00);      -- 5 points
3 SET v_fnb_points = fn_calculate_points(50000, 3.00);          -- 1.5 -> 2 points
4 SET v_total_points = v_ticket_points + v_fnb_points;           -- 7 points total
5
6 -- VIP tier: 8% box office, 6% concession
7 SET v_ticket_points = fn_calculate_points(100000, 8.00);      -- 8 points
8 SET v_fnb_points = fn_calculate_points(50000, 6.00);          -- 3 points
9 SET v_total_points = v_ticket_points + v_fnb_points;           -- 11 points total
```

Listing 45: Points calculation usage example

Example 3: Showtime Count for Reporting



```
1 -- Count showtimes for theater 1 in November 2025
2 SELECT fn_count_showtimes_in_range(1, '2025-11-01', '2025-11-30')
3 AS november_showtimes;
4
5 -- Handle error codes
6 SET v_count = fn_count_showtimes_in_range(p_theater_id, p_start, p_end);
7 IF v_count < 0 THEN
8     CASE v_count
9         WHEN -1 THEN SET v_error = 'Invalid theater ID';
10        WHEN -2 THEN SET v_error = 'Missing date parameters';
11        WHEN -3 THEN SET v_error = 'Invalid date range';
12    END CASE;
13 END IF;
```

Listing 46: Showtime count usage example

Example 4: Average Booking Value Analysis

```
1 -- Calculate average booking value for November 2025
2 SELECT fn_calculate_avg_booking_value_by_month(2025, 11)
3 AS avg_booking_value;
4 -- Result: 250000.00 (VND)
5
6 -- Compare Q4 2025 monthly averages
7 SELECT
8     'October' AS month,
9     fn_calculate_avg_booking_value_by_month(2025, 10) AS avg_value
10 UNION ALL
11 SELECT 'November', fn_calculate_avg_booking_value_by_month(2025, 11)
12 UNION ALL
13 SELECT 'December', fn_calculate_avg_booking_value_by_month(2025, 12);
14
15 -- Handle error cases
16 SET v_avg = fn_calculate_avg_booking_value_by_month(2025, 13);
17 IF v_avg = -2 THEN
18     SET v_error = 'Invalid month parameter';
19 END IF;
```

Listing 47: Average booking value calculation usage example

Example 5: Customer Lifetime Value Calculation

```
1  -- Calculate total revenue from customer ID 1001
2  SELECT fn_calculate_total_revenue_by_customer(1001)
3  AS customer_lifetime_value;
4  -- Result: 5250000.00 (VND)
5
6  -- Identify top 10 customers by revenue
7  SELECT
8      c.user_id,
9      CONCAT(u.fname, ' ', u.lname) AS customer_name,
10     fn_calculate_total_revenue_by_customer(c.user_id) AS total_revenue
11    FROM customer c
12   JOIN User u ON c.user_id = u.id
13  ORDER BY total_revenue DESC
14  LIMIT 10;
15
16  -- Check customer qualification for VIP tier (>= 2M VND)
17  SET v_revenue = fn_calculate_total_revenue_by_customer(p_customer_id);
18  IF v_revenue >= 2000000 THEN
19      SET v_tier = 'VIP';
20  ELSEIF v_revenue >= 1000000 THEN
21      SET v_tier = 'Member';
22  END IF;
23
24  -- Handle error cases
25  IF v_revenue = -1 THEN
26      SET v_error = 'Invalid customer ID';
27  ELSEIF v_revenue = -2 THEN
28      SET v_error = 'Customer not found';
29  END IF;
```

Listing 48: Customer total revenue calculation usage example

These 9 functions work together with stored procedures (Section 4) and triggers (Section 3) to create a comprehensive database programming layer for the BKinema system.

6 Application Implementation

The backend is implemented using NestJS, a modular Node.js framework that supports capabilities such as dependency injection, middleware, and robust error handling. Its native integration with TypeORM facilitates database operations while maintaining compatibility with stored procedures and triggers. The frontend is developed with React.js, employing a component-based structure that enhances maintainability and rendering performance through the virtual DOM—particularly beneficial for dynamic interfaces such as seat selection and real-time availability updates. React Router manages client-side navigation, the Context API handles global state management, and Axios enables communication with backend services. Authentication is implemented using JSON Web Tokens (JWT) in conjunction with Passport middleware, and user passwords are secured through bcrypt hashing. API documentation is generated automatically via Swagger/OpenAPI specifications.

6.1 Backend Implementation

6.1.1 Authentication Service Implementation

```
1 class AuthService {
2     constructor(
3         @InjectDataSource() private dataSource: DataSource,
4         private jwtService: JwtService,
5     ) {}
6
7     async register(registerDto: RegisterDto): Promise<AuthResponseDto> {
8         const { fname, minit, lname, email, password, birthday,
9             gender, district, city } = registerDto;
10
11     const hashedPassword = await bcrypt.hash(password, 10);
12
13     await this.dataSource.query(
14         `CALL sp_register_user(?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
15             @user_id, @success, @message)`,
16         [fname, minit || null, lname, email, hashedPassword,
17             birthday || null, gender || 'Other',
18             district || null, city || null],
19     );
20
21     const [result] = await this.dataSource.query(
22         `SELECT @user_id as userId, @success as success,
23             @message as message`
24     );
25
26     if (!result.success) {
27         throw new ConflictException(result.message);
28     }
29
30     const payload = { email, sub: result.userId, role: 'customer' };
31     return {
32         access_token: this.jwtService.sign(payload),
33         user: { id: result.userId, email, fname, lname }
34     };
35 }
36 }
```

Listing 49: Authentication service - user registration

User authentication is managed through a dedicated service that interfaces with the database's stored procedures for registration and login operations. The authentication flow validates credentials, generates JWT tokens, and maintains session security. Listing 49 presents the core authentication logic for user registration.

The registration process leverages the `sp_register_user` stored procedure, which handles database insertion and triggers the automatic creation of a customer record and membership initialization. By delegating business logic to stored procedures, the application layer remains thin and focused on request orchestration and response formatting.

6.1.2 Booking Management Controller

The booking controller exposes RESTful endpoints for initiating bookings, updating food and beverage selections, and retrieving booking history. The implementation coordinates between multiple services to validate seat availability, hold reservations, and manage booking lifecycles. Listing 50 demonstrates the booking endpoint implementations.

```
1 class BookingController {
2     constructor(private readonly bookingService: BookingService) {}
3
4     @Post('start')
5     @ApiOperation({ summary: 'Create booking and hold seats' })
6     async startBooking(@Body() dto: StartBookingDto) {
7         return this.bookingService.startBooking(dto);
8     }
9
10    @Post('fwb')
11    @ApiOperation({ summary: 'Update food & beverage items' })
12    async updateFwb(@Body() dto: UpdateFwbDto) {
13        return this.bookingService.updateBookingFwb(dto);
14    }
15
16    @Get('my-bookings')
17    @UseGuards(JwtAuthGuard)
18    @ApiOperation({ summary: 'Get customer bookings with pagination' })
19    async getMyBookings(
20        @Request() req,
21        @Query('limit') limit?: string,
22        @Query('offset') offset?: string,
23    ) {
24        const parsedLimit = limit ? parseInt(limit) : 5;
25        const parsedOffset = offset ? parseInt(offset) : 0;
26        return this.bookingService.getMyBookings(
27            req.user.userId, parsedLimit, parsedOffset
28        );
29    }
30}
```

Listing 50: *Booking controller endpoints*

The `JwtAuthGuard` decorator ensures that only authenticated users can access protected endpoints such as booking history retrieval. The pagination parameters in the `getMyBookings` endpoint default to 5 items per page, aligning with the user interface design that displays bookings in manageable chunks.

6.2 Frontend Implementation

The frontend application is structured as a single-page application (SPA) utilizing React Router for navigation and component composition. The architecture emphasizes component reusability, state management through Context API, and responsive design principles to ensure optimal user experience across devices.

6.2.1 Booking Context State Management

The booking context implements a centralized state management solution for the multi-step booking process, maintaining data consistency as users progress through seat selection, combo selection, and payment. Listing 51 demonstrates the context implementation with state persistence.

```
1 const BookingContext = createContext();
2
3 export function BookingProvider({ children }) {
4   const [bookingData, setBookingData] = useState(() => {
5     const saved = sessionStorage.getItem('bookingData');
6     return saved ? JSON.parse(saved) : {
7       selectedSeats: [],
8       selectedCombos: [],
9       showtimeId: null,
10      theaterId: null,
11      totalPrice: 0,
12      comboTotal: 0,
13    };
14  });
15
16  useEffect(() => {
17    sessionStorage.setItem('bookingData',
18      JSON.stringify(bookingData));
19  }, [bookingData]);
20
21  const updateBookingData = (updates) => {
22    setBookingData(prev => ({ ...prev, ...updates }));
23  };
24
25  const clearBookingData = () => {
26    setBookingData({
27      selectedSeats: [], selectedCombos: [],
28      showtimeId: null, theaterId: null,
29      totalPrice: 0, comboTotal: 0,
30    });
31    sessionStorage.removeItem('bookingData');
32  };
33
34  return (
35    <BookingContext.Provider value={{{
36      bookingData, updateBookingData, clearBookingData
37    }}}>
38      {children}
39      <BookingContext.Provider>
40    );
41 }
42
43 export const useBooking = () => useContext(BookingContext);
```

Listing 51: Booking context for global state management

The context utilizes `sessionStorage` for state persistence, ensuring that booking data survives page refreshes but is automatically cleared when the browser session ends. This approach balances user convenience with security, as sensitive payment information is never persisted beyond the current session.



6.3 Application Workflow and User Interface

6.3.1 User Registration and Authentication

The user onboarding process begins with the registration interface, which collects essential user information including name, email, contact details, and demographic data. Figure 1 displays the registration form with client-side validation to ensure data quality before submission.

The screenshot shows the BKinema registration page. At the top, there's a navigation bar with links for News & Offers, My Tickets, EN, VN, & Account, a search bar, and a Book Tickets button. The main area has 'MOVIES', 'THEATERS', 'MEMBERSHIP', and 'SERVICES' tabs. A 'LOGIN' and 'REGISTER' button are at the top left of the form. The 'REGISTER' button is highlighted in blue. The form fields include:

- First Name***: Tien Long
- Middle Initial (Optional)**: P
- Last Name***: Le
- Email***: long.lephanhien@hmcut.edu.vn
- Password***: [REDACTED]
- Date of Birth (Optional)**: 23 December 2005
- Gender**: Male
- City***: Ho Chi Minh City
- District (Optional)**: District 10

Below the form is a CAPTCHA field with the code "PHHA9G". There are several checkboxes for terms and conditions:

- By clicking the "Register" button below, I agree to allow BKinema Vietnam to process my personal data in accordance with purposes that BKinema Vietnam has announced in the Privacy Policy.
- Personal information provided here is accurate and matches information in ID card and/or Birth Certificate. Also, the email provided here is accurate and under my sole control.
- Correctly enter the email is correct and date of birth matches your Citizen Identity Card's information. If not, this information will not be supported to update and you may not receive Member Benefits
- I agree to the BKinema's Terms Of Use

A large blue 'REGISTER' button is at the bottom right of the form.

Figure 1: User registration interface with form validation

Upon successful registration, the system automatically creates a customer profile with a default membership tier and initializes the customer's point balance. The registration process invokes the `sp_register_user` stored procedure, which triggers the `trg_after_user_insert` trigger to establish the customer relationship and membership record as described in Section 3.

6.3.2 Home Page and Movie Discovery

The home page serves as the primary entry point, featuring promotional banners, highlighted movies, and quick access to currently showing films. The interface employs a carousel for featured content and grid layouts for movie browsing, as illustrated in Figure 2. The home page dynamically loads movie data from the backend API, displaying poster images, titles, ratings, and formats. Users can quickly navigate to detailed movie information or initiate the booking process directly from this interface.

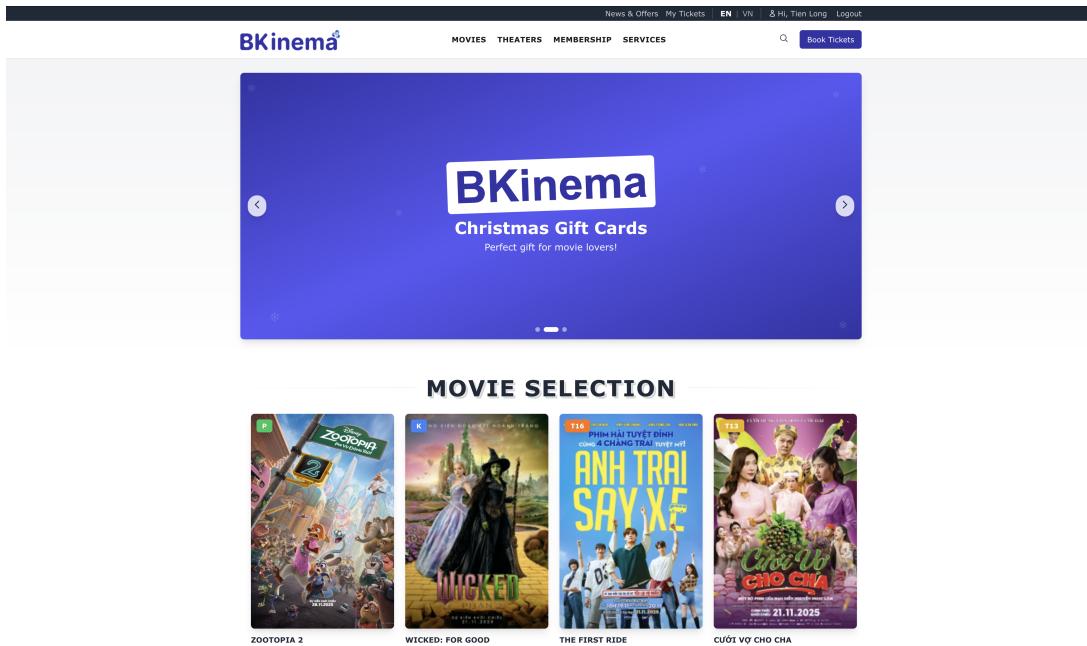


Figure 2: Home page with featured banners and movie highlights

6.3.3 Movie Catalog and Filtering

The movies page presents a comprehensive catalog of all currently showing films with filtering and sorting capabilities. Figure 3 demonstrates the movie listing interface with visual cards displaying essential information including poster, title, genre, duration, and rating.

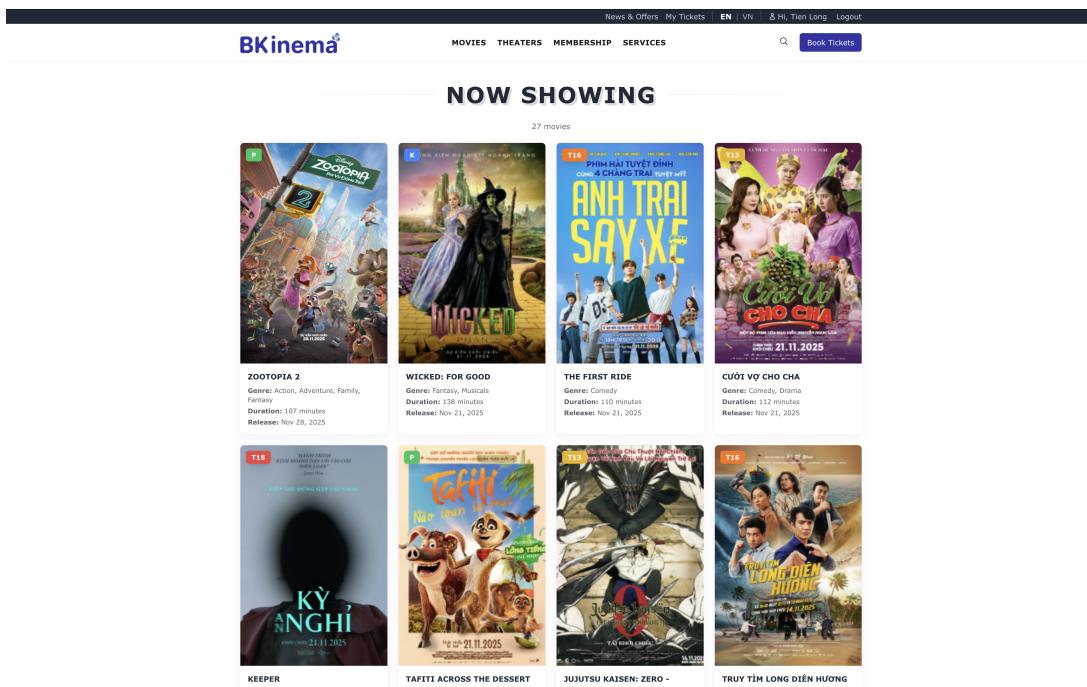


Figure 3: Movie catalog page showing all now-showing films

The interface enables users to filter movies by genre and rating, facilitating efficient movie discovery based on personal preferences. Each movie card serves as a clickable link to the detailed movie information page.



6.3.4 Theater Page

The screenshot shows a theater listing page for BKinema. At the top, there's a navigation bar with links for 'News & Offers', 'My Tickets', 'EN', 'VN', 'Logout', and a search bar with a 'Book Tickets' button. Below the navigation is the BKinema logo. A dropdown menu labeled 'THEATERS' is open, showing a list of cities: Ho Chi Minh City, Ha Noi, Da Nang, Can Tho, Dong Nai, Hai Phong, Quang Ninh, Ba Ria - Vung Tau, Binh Dinh, Binh Duong, Dak Lak, Tra Vinh, Yen Bai, Vinh Long, Kien Giang, Hau Giang, Ha Tinh, Phu Yen, Dong Thap, Bac Lieu, Hung Yen, Khanh Hoa, Kon Tum, Lang Son, Nghe An, Phu Tho, Quang Ngai, Soc Trang, Son La, Tay Ninh, and Thai Nguyen, Tien Giang. Below this is another section with a list of theaters: BKinema Aeon Binh Tan, BKinema Aeon Tan Phu, BKinema Crescent Mall, BKinema GigaMall Thu Duc, BKinema Hoang Van Thu, BKinema Hung Vuong Plaza, BKinema Liberty Citypoint, BKinema Ly Chinh Thang, BKinema Menas Mall, BKinema Pandora City, BKinema Pearl Plaza, BKinema Saigonres Nguyen Xi, BKinema Satra Cu Chi, BKinema Su Van Hanh, BKinema Thao Dien Pearl, BKinema Vincom Center Landmark 81, BKinema Vincom Dong Khoi, BKinema Vincom Go Vap, BKinema Vincom Mega Mall Grand Park, and BKinema Vivo City. Below this is a section titled 'THEATER' with a thumbnail image of a theater interior and the text 'BKinema Su Van Hanh'.

Figure 4: Theater listing with city selection and showtime display

Figure 4 presents the theater listing with city-based filtering and showtime availability indicators. Users first select their preferred city from a dropdown menu, which dynamically filters the theater list to display only theaters in that location. Upon selecting a theater, the interface expands to show the theater's address and current showtimes for all movies.

6.3.5 Movie Detail and Showtime Selection

The screenshot shows a movie detail page for 'ZOOTOPIA 2'. At the top, there's a navigation bar with links for 'News & Offers', 'My Tickets', 'EN', 'VN', 'Logout', and a search bar with a 'BOOKING' button. Below the navigation is the BKinema logo. To the left is a large movie poster for 'Zootopia 2'. To the right of the poster is the movie title 'ZOOTOPIA 2' and a detailed description box. The description includes: Director: Byron Howard, Jared Bush; Cast: Fortune Feimster, Jason Bateman, Quinta Brunson; Genre: Action, Adventure, Family, Fantasy; Release date: Nov 28, 2025; Duration: 107 minutes; Language: English – Subtitle: Vietnamese – Dubbing: Vietnamese; Rated: P - Movies suitable for all ages. Below the description is a 'Description' section with the text: 'Brave rabbit cop Judy Hopps and her friend, the fox Nick Wilde, team up again to crack a new case, the most perilous and intricate of their careers.' At the bottom is a 'Trailer' section featuring a video player showing a scene from the movie where two characters are driving a car. There are also 'Copy link' and 'BOOKING' buttons.

Figure 5: Movie detail page with comprehensive film information



The movie detail page aggregates comprehensive information about a selected film, including synopsis, director, cast, duration, language, and age rating. Figure 5 illustrates the detailed movie information layout with integrated booking initiation. The booking button on the movie detail page triggers a showtime selection modal, allowing users to specify their preferred viewing date, city, and theater. Figure 6 demonstrates the showtime selection interface that appears as an overlay, displaying available showtimes organized by date and theater.

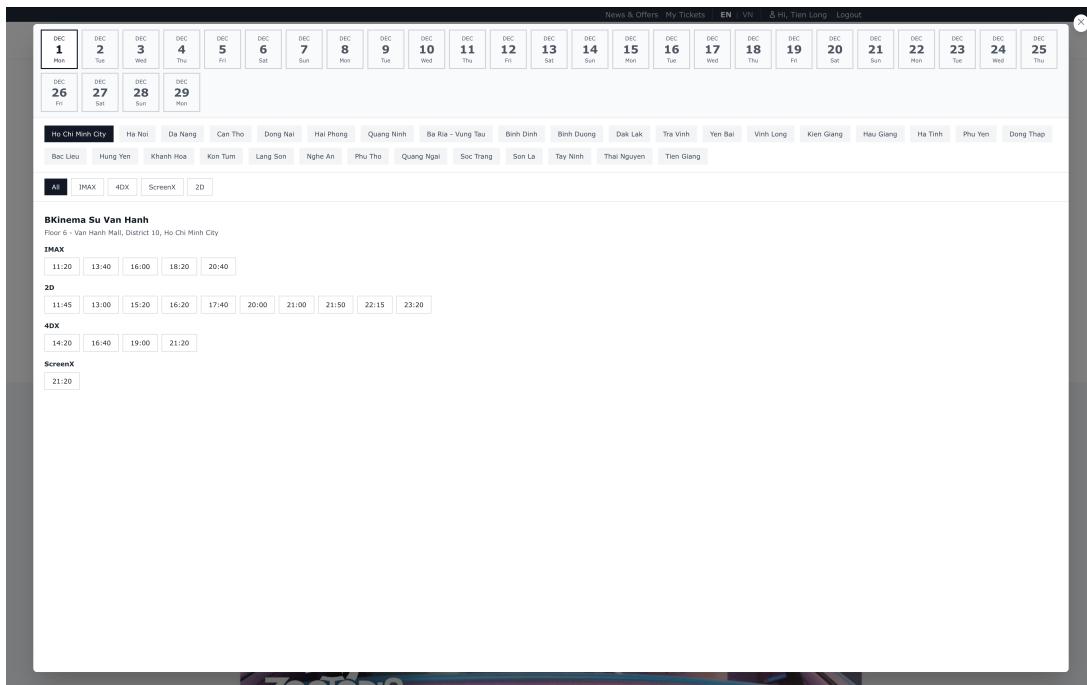


Figure 6: Showtime selection modal with date, city, and theater filtering

6.3.6 Seat Selection Interface

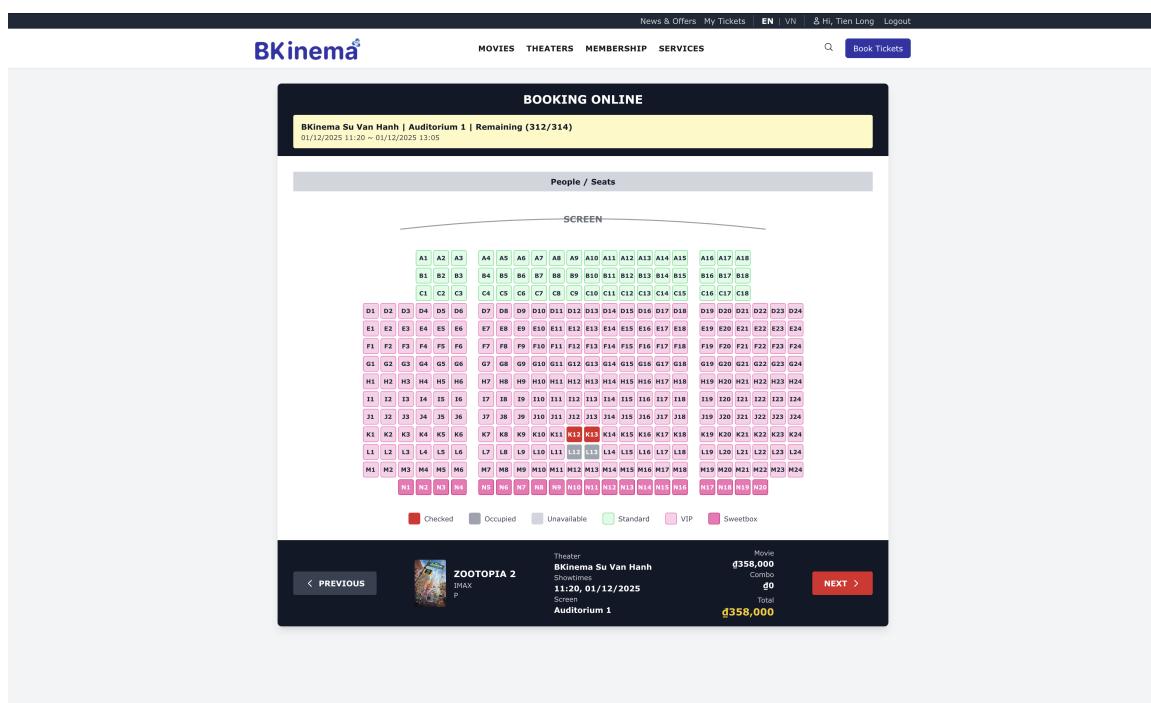


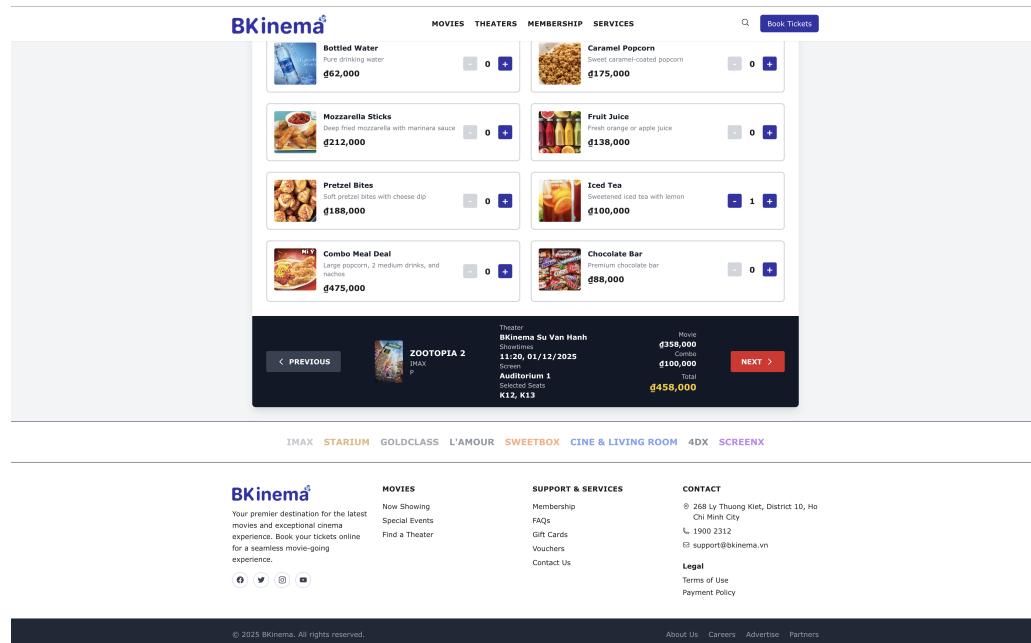
Figure 7: Interactive seat selection interface with real-time availability



The seat selection page presents an interactive auditorium layout where users can view seat availability and select their preferred seats. Figure 7 shows the seat map with color-coded indicators for available, occupied, and selected seats.

6.3.7 Food and Beverage Selection

Following seat selection, users proceed to the food and beverage (F&B) selection page where they can add combo items to their order. Figure 8 displays the F&B menu with item descriptions, prices, and quantity selectors.



The screenshot shows the BKinema food and beverage selection interface. At the top, there are tabs for MOVIES, THEATERS, MEMBERSHIP, and SERVICES, along with a search bar and a 'Book Tickets' button. Below this is a grid of food items with quantity selectors:

Item	Description	Price	Quantity
Bottled Water	Pure drinking water	đ62,000	0 +
Caramel Popcorn	Sweet caramel-coated popcorn	đ175,000	0 +
Mozzarella Sticks	Deep fried mozzarella with marinara sauce	đ212,000	0 +
Fruit Juice	Fresh orange or apple juice	đ138,000	0 +
Pretzel Bites	Soft pretzel bites with cheese dip	đ188,000	0 +
Iced Tea	Sweetened iced tea with lemon	đ100,000	1 +
Combo Meal Deal	Large popcorn, 2 medium drinks, and nachos	đ475,000	0 +
Chocolate Bar	Premium chocolate bar	đ88,000	0 +

At the bottom, it shows the movie information: Zootopia 2 (IMAX), showing time 11:30, 01/12/2025, Auditorium 1, Seats K12, K13. It also shows the total cost of the booking: Movie ticket (đ358,000), Concessions (đ100,000), Total (đ458,000). Navigation buttons for PREVIOUS and NEXT are also present.

Figure 8: Food and beverage selection interface

6.3.8 Payment and Point Redemption

The payment page consolidates the booking summary, calculates the final total including taxes, and provides payment options. Figure 9 illustrates the payment interface with membership point application functionality.

Authenticated customers can apply their accumulated membership points toward the booking cost, with the system displaying available points, conversion rate, and the resulting discount. The point application logic invokes the `sp_apply_points` stored procedure, which validates point availability, calculates the redemption amount, and updates the customer's point balance within a database transaction to ensure atomicity.

Upon payment confirmation, the system processes the transaction through the `sp_process_payment` stored procedure, which creates ticket records, updates seat availability, records the transaction in the payment ledger, and awards membership points based on the customer's tier multiplier.



The payment page shows a step-by-step process for applying discounts. Step 1: METHOD OF DISCOUNT includes options for BKinema Voucher, Discount Coupon, and BKinema Point. It displays a user has 21 points available for use. Step 2: BKinema GIFT CARD allows selecting a gift card. Step 3: FINAL PAYMENT lists various payment methods: ATM card (Vietnam Domestic), Credit/Debit Card (Visa, Mastercard), MoMo, ZaloPay, VNPay, and ShopeePay. On the right side, the Total Payment is shown as ₫458,000, with a BKinema Points discount of ₫15,000 applied, resulting in a final payment of ₫443,000. A Countdown Clock indicates 4 minutes and 29 seconds remaining.

Figure 9: Payment page with point redemption interface

6.3.9 Booking Management and History

Logged-in customers can access their booking history through the customer account dashboard. Figure 10 shows the booking history interface with pagination support for efficient navigation through past bookings.

The customer booking history interface shows a list of five past bookings. Each entry includes a thumbnail image of the movie, booking ID, movie title, theater name, seats, showtime, auditorium, food/beverage items, and total amount. Buttons for 'View' and 'Refund' are provided for each booking. The interface uses a dark theme with blue highlights for buttons and links.

Booking ID	Movie	Theater	Seats	Showtime	Auditorium	Food & Beverage	Total Amount
#450	ZOOTOPIA 2	BKinema Su Van Hanh	K12, K13	2025-12-01 11:20:00	Auditorium 1	Iced Tea x1	₩443,000
#449	ZOOTOPIA 2	BKinema Su Van Hanh	L12, L13	2025-12-01 11:20:00	Auditorium 1	Chocolate Bar x1	₩446,000
#448	ZOOTOPIA 2	BKinema Su Van Hanh	L12, L13	2025-12-01 11:20:00	Auditorium 1	Chocolate Bar x1	₩446,000
#447	ZOOTOPIA 2	BKinema Su Van Hanh	L12, L13	2025-12-01 11:20:00	Auditorium 1	Chocolate Bar x1	₩446,000

Figure 10: Customer booking history with pagination (5 bookings per page)

Each booking entry displays essential information including movie title, showtime, theater, seats, total cost, and booking status (Pending, Confirmed, Cancelled). The interface implements server-side pagination with a fixed limit of 5 bookings per page, querying the database through the `getMyBookings` service method that executes appropriate SELECT statements with LIMIT and OFFSET clauses.



6.3.10 Membership Card

The membership dashboard provides customers with visibility into their current membership tier, accumulated points, and points transaction history. Figure 11 displays the membership interface showing tier benefits and point balance.

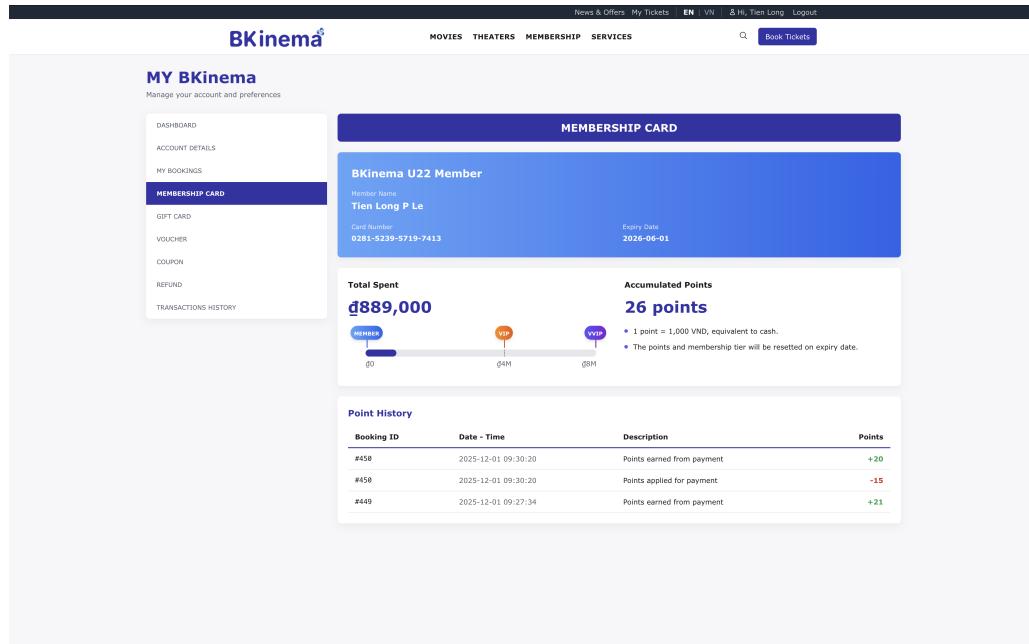


Figure 11: Membership dashboard showing tier status and points history

The points history section displays all point transactions including earnings from bookings, redemptions for discounts, and expirations due to the 365-day validity period. The interface invokes the `fn_get_available_points` function to calculate the current valid point balance, excluding expired points from the total.

The tier display shows progression toward the next membership level, calculated based on the total booking amount over the past 365 days. This information is retrieved through queries that aggregate booking totals and compare against the tier thresholds defined in the `Membership` table, as established in Section 2.

7 Conclusion

7.1 Summary

This report presented the complete implementation of the BKinema movie ticket booking system, demonstrating a fully functional solution that integrates an optimized database design with modern full-stack web development. The system successfully evolved from conceptual modeling to a deployable application capable of managing complex workflows including booking, membership, and payment processing.

The database consists of 14 relational tables supporting authentication with role-based access control, theater and auditorium configuration, rich movie metadata, dynamic seat allocation, multi-method payment processing, a tiered membership model, food and beverage management, and coupon/refund operations. Referential integrity is enforced through foreign keys, CHECK constraints, and ENUM definitions.

Advanced SQL programming is implemented through four triggers and five stored procedures to automate business rules such as showtime conflict validation, age-rating enforcement, member profile creation, booking and seat allocation, and payment handling with point accumulation. Four user-defined functions provide reusable logic for point calculation, tier upgrade cost evaluation, seat availability, and age verification. Comprehensive validation, transaction management, and detailed error reporting ensure data consistency.

The backend, built with NestJS, exposes a type-safe API layer that interacts with the database via parameterized stored procedure calls. The React-based frontend delivers responsive interfaces for browsing

movies, selecting seats, completing bookings, and managing payments. Critical operations rely on transactional guarantees, while performance optimization uses aggregated queries, efficient JOIN patterns, and server-side pagination.

7.2 Future Works

Future development can improve scalability, automation, and user experience. Booking timeout handling may be enhanced through MySQL event scheduling to automatically release expired seat locks. Search functionality could be upgraded using MySQL FULLTEXT or external engines such as Elasticsearch for relevance ranking and fuzzy matching.

Analytical features could be expanded through stored procedures and views to support reporting on revenue, utilization, membership behavior, and sales insights. Integration with email/SMS services would enable automated confirmations, reminders, and promotions. Caching with Redis would improve performance for frequently requested data, while cloud storage (e.g., AWS S3, GCS, Cloudinary) would enhance media management and delivery.

Architecture evolution toward microservices and WebSocket-based real-time seat updates would improve scalability and responsiveness. Native mobile applications could extend accessibility and integrate biometric authentication and push notifications. An administrative dashboard would provide comprehensive operational control.

Advanced extensions include machine-learning-based recommendation systems and dynamic pricing models, as well as third-party integrations such as TMDB/IMDb for metadata and VNPay, MoMo, or ZaloPay for localized payment support.

A Additional Application Screenshots

This appendix contains additional screenshots of the BKinema web application user interface that complement the main application flow described in Section 6.

A.1 FAQ Page

The FAQ (Frequently Asked Questions) page provides users with answers to common questions about the booking process, payment methods, refund policies, and membership benefits.

Frequently Asked Questions
Find answers to common questions about BKinema

Search for questions...

Movies

What is P, C13, C16, C18?
What is an Early Release?
Early Release allows viewers to watch blockbuster movies before the official release date. This special screening is typically available for highly anticipated films and may have limited showtimes and theaters.
What is a Sneak Show?

★ Membership Program

What is BKinema membership program?
How to apply for BKinema membership?
Is there an expiration date for BKinema membership card?
If I forget my member card, can I receive reward points?

♥ Reward Points

What are membership points (reward points)?
How do I use my points?
What is the expiry date of reward points?

Figure 12: FAQ page interface



A.2 Membership Information Page

The membership page displays detailed information about BKinema's loyalty program tiers (Member, U22, VIP, VVIP), including benefits, point accumulation rates, and tier requirements.

The screenshot shows the BKinema membership page. At the top, there are navigation links for News & Offers, My Tickets, EN | VN, & Hi, Tien Long, Logout, and a search bar with a Book Tickets button. The main title is "BKinema Membership" with the subtitle "Unlock exclusive benefits and rewards".

What is BKinema Membership?

The BKinema membership program is designed to bring our loyal customers an array of special offers, resulting in once-in-a-lifetime experiences that are both enriching and meaningful.

Membership Levels

U22 Member	Member	VIP	VVIP
Under 22 years old	Standard membership	Premium benefits	Elite membership
Box Office: 5%	Box Office: 5%	Box Office: 7%	Box Office: 10%
Concession: 3%	Concession: 3%	Concession: 4%	Concession: 5%
Birthday Gift: 1 Free Combo + 1 Free Ticket (on 23rd birthday)	Birthday Gift: 1 Free Combo	Birthday Gift: 1 Free Combo + 1 Free 2D/3D Ticket	Birthday Gift: 1 Free Combo + 2 Free 2D/3D Tickets

Loyalty Points

Every time you spend at BKinema, your spending will be converted to points. 1 point = 1,000 VND, equivalent to cash, which can be redeemed for drinks, combos, and movie tickets.

Figure 13: Membership information page

A.3 Payment Policy Page

This page outlines the payment policies, accepted payment methods, security measures, and terms and conditions for online transactions.

The screenshot shows the BKinema payment policy page. At the top, there are navigation links for News & Offers, My Tickets, EN | VN, & Hi, Tien Long, Logout, and a search bar with a Book Tickets button. The main title is "Payment Policy" with the subtitle "Secure and convenient payment options".

1. Payment Regulation

Customers can choose the following payment methods for online booking transactions on the BKinema website:

<input type="checkbox"/> Member Reward Points	<input type="checkbox"/> BKinema Gift Card
<input type="checkbox"/> Ticket Voucher	<input type="checkbox"/> ATM Card
<input type="checkbox"/> Credit/Debit Card	<input type="checkbox"/> International Cards

2. Online Payment Methods

Membership Points

1 point is equivalent to 1,000 VND. With these reward points, you can pay for all products and services available at BKinema similarly to using cash.

- Minimum 20 points required for any purchase
- Present membership card at box office or use online
- Check points and transaction history in your account

BKinema Gift Card

Gift cards can be used to buy movie tickets and concession items at all BKinema cinemas and for online booking.

• Minimum balance: 100,000 VND
• Available amounts: 100,000 / 200,000 / 300,000 / 500,000 / 1,000,000 VND
• One-year validity period
• Can be topped up to extend validity

ATM Cards (Domestic)

To make online payments using domestic ATM cards, the card must be registered for internet banking by the card issuer. The transaction must be...

Figure 14: Payment policy page



A.4 Terms of Use Page

The Terms of Use page contains the legal terms and conditions that govern the use of the BKinema platform, including user responsibilities, prohibited activities, and liability disclaimers.

Welcome to BKinema. Our company is BKinema Vietnam, located at 268 Lý Thongoose Street, District 10, Ho Chi Minh City, Vietnam.

Once you visit our website or use our services, it means that you have agreed with these terms. This website is entitled to change, edit, add and remove any part of Terms and Conditions at any time. The changes will come into effect as soon as they are posted on the website without any announcements in advance.

Please check regularly for our updates.

1. Scope of Application

This section applies specifically to the booking function available on this website. When using online booking function to buy tickets, you signify your agreement to all applicable terms, conditions, and notices, including but not limited to the Terms of Use. If this is not your intention and/or you disagree with any part of applicable terms and conditions, DO NOT USE this facility.

2. Account Conditions

You must register an account with real information and update if there are any changes. Each visitor is responsible for their password, account and activity on the website. Furthermore, you must notify us when your account is accessed without permission.

We do not accept any responsibility, whether direct or indirect, for any loss or damage caused by your non-compliance.

3. Regulations for Online Booking

- The function of booking online tickets only applies for BKinema members. For details about BKinema member registration, please visit our membership page.
- Usually, BKinema opens online booking before a movie's release date, though this depends on each movie and cinema. If the showtime you want is not displayed on the website, please check back later or contact our hotline 1900 2312 for more information.
- The close time of buying online tickets is 30 minutes before showtime or when a session is sold out. After this time you can come to the cinema to buy tickets directly.

Figure 15: Terms of use page

A.5 Refund Request Interface

The refund request interface allows customers to submit refund requests for their bookings. The system validates refund eligibility based on showtime timing and processes the request through stored procedures.

Figure 16: Refund request interface



A.6 Ticket View Interface

After successful booking, customers can view their ticket details including QR code, seat information, showtime details, and booking reference number.

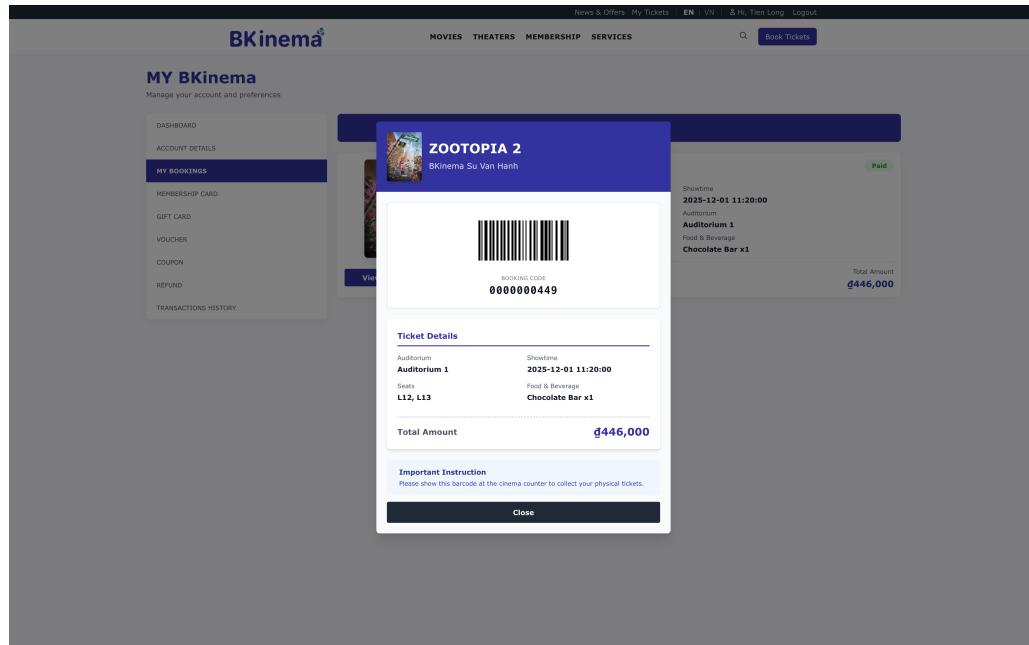


Figure 17: Ticket view interface

B Database Connection Configuration

```
1 TypeOrmModule.forRootAsync({
2   imports: [ConfigModule],
3   inject: [ConfigService],
4   useFactory: (config: ConfigService) => {
5     const db = config.get('database') as {
6       host?: string;
7       port?: number;
8       username?: string;
9       password?: string;
10      database?: string;
11      ca?: string;
12    } | undefined;
13
14    const ssl = db?.ca ? { ca: db.ca } : undefined;
15
16    return {
17      type: 'mysql' as const,
18      host: db?.host,
19      port: db?.port,
20      username: db?.username,
21      password: db?.password,
22      database: db?.database,
23      extra: ssl ? { ssl } : {},
24    };
25  },
26},
27
28 export class AppModule {}
```

Listing 52: Application module with database configuration

The backend connects to MySQL using TypeORM with connection pooling for optimal performance. The application module configures the database connection asynchronously using the ConfigService to load environment variables.

Code Availability

The complete implementation of the BKinema system, including all source code, stored procedures, triggers, and application code, is available on GitHub. Please access the repository via <https://github.com/longlephamtien/DS251-Assignment>.

Acknowledgement

We would like to express our sincere gratitude to Mr. Le Duc Hoang Nam for valuable guidance and support throughout this study.