HOLY ANGEL UNIVERSITY

6IMAN

FINAL GROUP PROJECT

Guzman, Keziah Claudine C.

Gatbonton, Keith Andre C.

Rivera, Jeriel Jair G.

Prof. Chris Almocera

S. Y. 2024 - 2025

April 2, 2025

TABLE OF CONTENTS

I. PROJECT DESCRIPTION/BUSINESS DESCRIPTION

WEBSITE NAME: Luxx Stay

Luxx Stay is a room reservation app that allows guests, travelers, and corporate executives to reserve rooms easily. The app provides customers with a variety of hotels, ranging from budget hotels to luxury resorts, to suit various tastes and budgets. With a smooth and user-friendly interface, Luxx Stay allows easy and fast room reservation.

Built with Kotlin, Luxx Stay has a robust PostgreSQL database that effectively handles hotel information, room bookings, user accounts, payments, and customer reviews. Security is the top priority on the platform with strong encryption and authentication. The hotels are authenticated prior to listing, ensuring the hotels are actual, and users are required to undergo email or phone verification for security purposes. All these ensure that fraudulent activities are avoided, and user information is safeguarded.

The app is both web and mobile-based with instant confirmation of booking, strong search filters, and AI-based recommendations for a customized experience. With an easy-to-use interface, Luxx Stay offers a safe, easy, and hassle-free method of searching and booking the desired accommodations without any hassle.

II. RELATIONAL DATABASE DESIGN

1) UUID Primary Keys: Ensures uniqueness across tables.
2) Foreign Key Relationships: users.id → bookings.user_id, hotels.id → bookings.hotel_id, bookings.id → payments.booking_id.
3) Timestamps: created_at and updated_at for tracking record changes.
4) Row Level Security (RLS): Policies to restrict access:
    a) Users can only view and update their own profiles.
    b) Anyone can view hotels.
    c) Users can only view and create their own bookings.
    d) Users can only view payments related to their bookings.
5) Constraints and Checks:
    a) Booking status: pending, confirmed, cancelled, completed.
    b) Payment status: pending, completed, failed.

Original Table Structure (Unnormalized Table)

1. Users

Attribute	Data Type	Relationship	Constraints
Id	UUID (Primary Key)	References auth.users(id)	PRIMARY KEY, ON DELETE CASCADE
email	TEXT	-	UNIQUE, NOT NULL
full_name	TEXT	-	-
phone_number	TEXT	-	-
address	TEXT	-	-
created_at	TIMESTAMP WITH TIME ZONE	-	DEFAULT timezone('utc', now()), NOT NULL
updated_at	TIMESTAMP WITH TIME ZONE	-	DEFAULT timezone('utc', now()), NOT NULL

2. Bookings

Attribute	Data Type	Relationship	Constraints
id	UUID (Primary Key)	-	PRIMARY KEY, DEFAULT uuid_generate_v4()
user_id	UUID	References users(id)	FOREIGN KEY, ON DELETE CASCADE, NOT NULL
hotel_id	UUID	References hotels(id)	FOREIGN KEY, ON DELETE CASCADE, NOT NULL
check_in_date	DATE	-	NOT NULL
check_out_date	DATE	-	NOT NULL
number_of_guests	INTEGER	-	NOT NULL
total_price	DECIMAL	-	NOT NULL
status	TEXT	-	DEFAULT 'pending', CHECK (status IN ('pending', 'confirmed', 'cancelled', 'completed'))
created_at	TIMESTAMP WITH TIME ZONE	-	DEFAULT timezone('utc', now()), NOT NULL
updated_at	TIMESTAMP WITH TIME ZONE	-	DEFAULT timezone('utc', now()), NOT NULL

3. Hotels

Attribute	Data Type	Relationships	Constraints
id	UUID (Primary Key)	-	PRIMARY KEY, DEFAULT uuid_generate_v4()
name	TEXT	-	NOT NULL
description	TEXT	-	-
price_per_night	DECIMAL	-	NOT NULL
image_url	TEXT	-	-
created_at	TIMESTAMP WITH TIME ZONE	-	DEFAULT timezone('utc', now()), NOT NULL
updated_at	TIMESTAMP WITH TIME ZONE	-	DEFAULT timezone('utc', now()), NOT NULL

4. Payments

Attribute	Data Type	Relationships	Constraints
id	UUID (Primary Key)	-	PRIMARY KEY, DEFAULT uuid_generate_v4()
booking_id	UUID	References bookings(id)	FOREIGN KEY, ON DELETE CASCADE, NOT NULL
amount	DECIMAL	-	NOT NULL
payment_method	TEXT	-	NOT NULL
payment_status	TEXT	-	DEFAULT 'pending', CHECK (payment_status IN ('pending', 'completed', 'failed'))
payment_date	TIMESTAMP WITH TIME ZONE	-	-
created_at	TIMESTAMP WITH TIME ZONE	-	DEFAULT timezone('utc', now()), NOT NULL
updated_at	TIMESTAMP WITH TIME ZONE	-	DEFAULT timezone('utc', now()), NOT NULL

Normalized Tables

Description of the Separated Normalized Tables

The following tables present a well-structured data model that has been normalized to remove redundancy and enhance data integrity. Initially stored in a single unnormalized table, the data has been systematically divided into four distinct tables, each representing specific entities and their attributes. This structured approach improves clarity, organization, and ease of data management and analysis.

Table 1: Users

ID	UserID	Full Name	Email	Phone	Address
12	35	Guzman, Keziah Claudine C.	keziahguzman@gmail.com	09661958802	Porac, Pampanga
13	36	Gatbonton, Keith Andre C.	keithandre@gmail.com	09772538001	Porac, Pampanga
14	38	Rivera, Jeriel Jair G.	jerieljair@gmail.com	09552634291	Porac, Pampanga

Table 2: Bookings

Hotel	Check in	Check out	No. Guest	Total Price
ABC Hotel	2025-04-12	2025-04-13	2	400.00 $
Widus Hotel	2025-06-23	2025-06-24	3	480.00 $
Park Inn	2025-11-09	2025-11-11	4	760.00 $

Table 3: Hotels

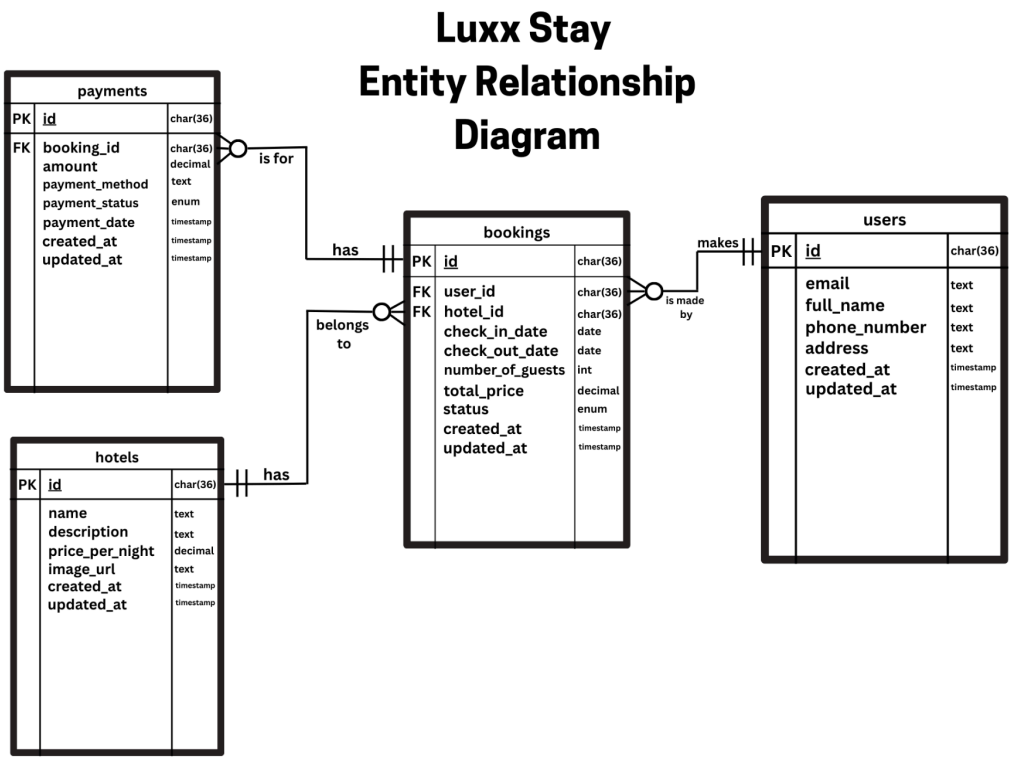
Name	Description	Price per Night	Created At
ABC Hotel	Luxury accommodation in the heart of the city	200$	2025-03-28 07:35:41
Clark Marriott Hotel	World-class luxury and service	250$	2025-03-28 07:39:22
Elenem Hotel	Your perfect getaway destination	180$	2025-03-28 07:45:36

Glory Hotel	Where comfort meets elegance	150$	2025-03-28 07:45:36
Hilton Clark	World-class luxury and service	250$	2025-03-28 07:52:48
Midori Clark	Japanese-inspired luxury living	220$	2025-03-28 07:35:41
One Euphoria	Experience pure bliss and luxury	280$	2025-03-28 07:39:22
Park Inn	Modern comfort in the heart of the city	190$	2025-03-28 07:45:36
Quest Plus	Premium hospitality and comfort	210$	2025-03-28 07:45:36
Widus Hotel	Luxury and elegance redefined	240$	2025-03-28 07:52:48

Table 4: Payments

Payment Method	Payment Status	Payment Date	Created At
Credit Card	Paid	2025-03-27	2025-03-28 07:52:48
Paypal	Paid	2025-04-26	2025-03-28 06:39:56
GCash	Paid	2025-02-13	2025-03-28 09:42:48

Entity Relationship Diagram (ERD)



Data Dictionary for Each Table

1. Users

Attribute	Data Type	Constraints	Description
id	UUID	PRIMARY KEY, NOT NULL	Unique Identifier for user account
email	VARCHAR(255)	NOT NULL, UNIQUE	Email address of the user
full_name	VARCHAR(255)	NOT NULL	Full name of the user
phone_number	VARCHAR(255)	NOT NULL	Phone number of the user
address	VARCHAR(255)	NOT NULL	Address of the user
created_at	TIMESTAMP WITH TIME ZONE	NOT NULL, DEFAULT NOW()	Account creation date
updated_at	TIMESTAMP WITH TIME ZONE	NOT NULL, DEFAULT NOW()	Account last updated date

2. Bookings

Attribute	Data Type	Constraints	Description
id	UUID	PRIMARY KEY, NOT NULL	Unique Identifier for booking
user_id	UUID	NOT NULL, FOREIGN KEY REFERENCES users(id)	References the user associated

			with the booking
hotel_id	UUID	NOT NULL, FOREIGN KEY REFERENCES hotels(id)	References the hotel associated with the booking
check_in_date	DATE	NOT NULL	Date of check-in
check_out_date	DATE	NOT NULL	Date of check-out
number_of_guests	INTEGER	NOT NULL	Number of guests for the booking
total_price	NUMERIC	NOT NULL	Total price of the booking
status	TEXT	NOT NULL, CHECK (status IN ('pending', 'confirmed', 'cancelled', 'completed'))	Status of the booking
created_at	TIMESTAMP WITH TIME ZONE	NOT NULL, DEFAULT NOW()	Booking creation date
updated_at	TIMESTAMP WITH TIME ZONE	NOT NULL, DEFAULT NOW()	Last updated booking date

3. Hotels

Attribute	Data Type	Constraints	Description
id	UUID	PRIMARY KEY, NOT NULL	Unique Identifier for hotel
name	TEXT	NOT NULL	Name of the hotel
description	TEXT	DEFAULT NULL	Description of the hotel
price_per_night	NUMERIC	NOT NULL	Price per night of the hotel
image_url	TEXT	DEFAULT NULL	Image URL of the hotel
created_at	TIMESTAMP WITH TIME ZONE	NOT NULL, DEFAULT NOW()	Hotel record creation date
updated_at	TIMESTAMP WITH TIME ZONE	NOT NULL, DEFAULT NOW()	Last updated date for hotel

4. Payments

Attribute	Data Type	Constraints	Description
id	UUID	PRIMARY KEY, NOT NULL	Unique Identifier for payment
booking_id	UUID	NOT NULL, FOREIGN KEY REFERENCES bookings(id)	References the associated booking
amount	NUMERIC	NOT NULL	Total amount for the booking
payment_method	TEXT	NOT NULL	Payment method used
payment_status	TEXT	NOT NULL, CHECK (payment_status IN ('pending', 'completed', 'failed'))	Status of the payment
payment_date	TIMESTAMP WITH TIME ZONE	DEFAULT NULL	Date when payment was completed
created_at	TIMESTAMP WITH TIME ZONE	NOT NULL, DEFAULT NOW()	Payment processing date
updated_at	TIMESTAMP WITH TIME ZONE	NOT NULL, DEFAULT NOW()	Last updated payment record

III. Database Screenshots:

Bookings

Table structure | Relation view

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	**id**	char(36)	utf8mb4_general_ci		No	uuid()			Change Drop More
2	**user_id**	char(36)	utf8mb4_general_ci		No	*None*			Change Drop More
3	**hotel_id**	char(36)	utf8mb4_general_ci		No	*None*			Change Drop More
4	**check_in_date**	date			No	*None*			Change Drop More
5	**check_out_date**	date			No	*None*			Change Drop More
6	**number_of_guests**	int(11)			No	*None*			Change Drop More
7	**total_price**	decimal(10,2)			No	*None*			Change Drop More
8	**status**	enum('pending', 'confirmed', 'cancelled', 'complet...	utf8mb4_general_ci		Yes	pending			Change Drop More
9	**created_at**	timestamp			No	current_timestamp()			Change Drop More
10	**updated_at**	timestamp			No	current_timestamp()		ON UPDATE CURRENT_TIMESTAMP()	Change Drop More

Hotels

Table structure | Relation view

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	**id**	char(36)	utf8mb4_general_ci		No	uuid()			Change Drop More
2	**name**	text	utf8mb4_general_ci		No	*None*			Change Drop More
3	**description**	text	utf8mb4_general_ci		Yes	*NULL*			Change Drop More
4	**price_per_night**	decimal(10,2)			No	*None*			Change Drop More
5	**image_url**	text	utf8mb4_general_ci		Yes	*NULL*			Change Drop More
6	**created_at**	timestamp			No	current_timestamp()			Change Drop More
7	**updated_at**	timestamp			No	current_timestamp()		ON UPDATE CURRENT_TIMESTAMP()	Change Drop More

Payments

Table structure | Relation view

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	**id**	char(36)	utf8mb4_general_ci		No	uuid()			Change Drop More
2	**booking_id**	char(36)	utf8mb4_general_ci		No	*None*			Change Drop More
3	**amount**	decimal(10,2)			No	*None*			Change Drop More
4	**payment_method**	text	utf8mb4_general_ci		No	*None*			Change Drop More
5	**payment_status**	enum('pending', 'completed', 'failed')	utf8mb4_general_ci		Yes	pending			Change Drop More
6	**payment_date**	timestamp			No	current_timestamp()		ON UPDATE CURRENT_TIMESTAMP()	Change Drop More
7	**created_at**	timestamp			No	current_timestamp()			Change Drop More
8	**updated_at**	timestamp			No	current_timestamp()		ON UPDATE CURRENT_TIMESTAMP()	Change Drop More

Users

Table structure | Relation view

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	**id**	char(36)	utf8mb4_general_ci		No	uuid()			Change Drop More
2	**email**	text	utf8mb4_general_ci		No	*None*			Change Drop More
3	**full_name**	text	utf8mb4_general_ci		Yes	*NULL*			Change Drop More
4	**phone_number**	text	utf8mb4_general_ci		Yes	*NULL*			Change Drop More
5	**address**	text	utf8mb4_general_ci		Yes	*NULL*			Change Drop More
6	**created_at**	timestamp			No	current_timestamp()			Change Drop More
7	**updated_at**	timestamp			No	current_timestamp()		ON UPDATE CURRENT_TIMESTAMP()	Change Drop More

Sample Queries:

Inserting Users:

```sql
INSERT INTO users (email, full_name, phone_number, address) VALUES
    ('keziahguzman@gmail.com', 'Guzman, Keziah Claudine C.', '09661958802', 'Porac, Pampanga'),
    ('keithandre@gmail.com', 'Gatbonton, Keith Andre C.', '09772538001', 'Porac, Pampanga'),
    ('jerieljair@gmail.com', 'Rivera, Jeriel Jair G.', '0955263429', 'Porac, Pampanga');
```

Inserting Bookings:

Run SQL query/queries on table luxx_booking_app.hotels:

```sql
INSERT INTO bookings (user_id, hotel_id, check_in_date, check_out_date, number_of_guests, total_price) VALUES
    ('84949663-0f0d-11f0-a67c-d0abd569ac11', 'ffa73faf-0f0d-11f0-a67c-d0abd569ac11', '2025-04-12', '2025-04-13', 2, 400.00),
    ('849499de-0f0d-11f0-a67c-d0abd569ac11', 'ffa74376-0f0d-11f0-a67c-d0abd569ac11', '2025-05-15', '2025-05-18', 2, 750.00),
    ('84949ab0-0f0d-11f0-a67c-d0abd569ac11', 'ffa74427-0f0d-11f0-a67c-d0abd569ac11', '2025-08-10', '2025-08-12', 1, 500.00);
```

id
name
description
price_per_night
image_url
created_at
updated_at

Inserting Payments:

Run SQL query/queries on table luxx_booking_app.bookings:

```sql
INSERT INTO payments (booking_id, amount, payment_method, payment_status, payment_date) VALUES
    ('bbd50373-0f10-11f0-a67c-d0abd569ac11', 400.00, 'Credit Card', 'completed', NOW()),
    ('bbd508b6-0f10-11f0-a67c-d0abd569ac11', 750.00, 'Bank Transfer', 'pending', NOW()),
    ('bbd50963-0f10-11f0-a67c-d0abd569ac11', 500.00, 'PayPal', 'completed', NOW());
```

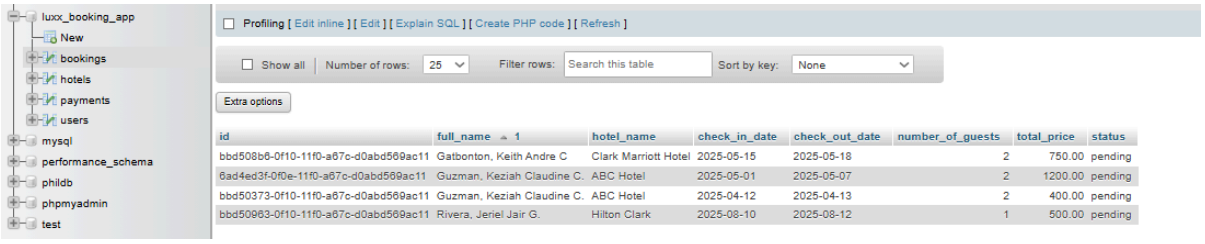
id
user_id
hotel_id
check_in_date
check_out_date
number_of_guests
total_price
status
created_at
updated_at

First Sample Query: Select all bookings for each user (command & execution)

Run SQL query/queries on table luxx_booking_app.bookings:

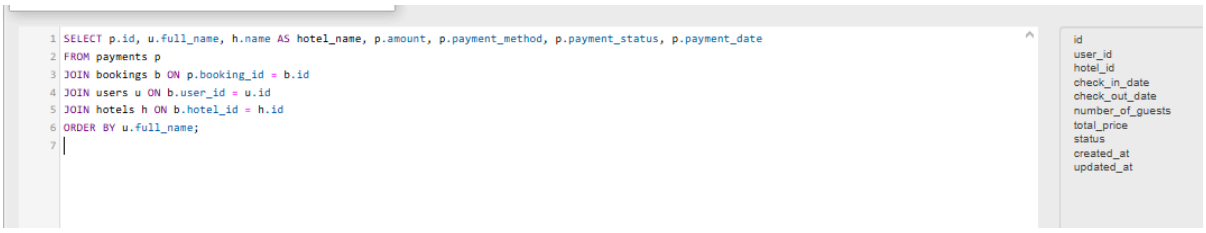
```sql
SELECT b.id, u.full_name, h.name AS hotel_name, b.check_in_date, b.check_out_date, b.number_of_guests, b.total_price, b.status
FROM bookings b
JOIN users u ON b.user_id = u.id
JOIN hotels h ON b.hotel_id = h.id
ORDER BY u.full_name;
```

id
user_id
hotel_id
check_in_date
check_out_date
number_of_guests
total_price
status
created_at
updated_at



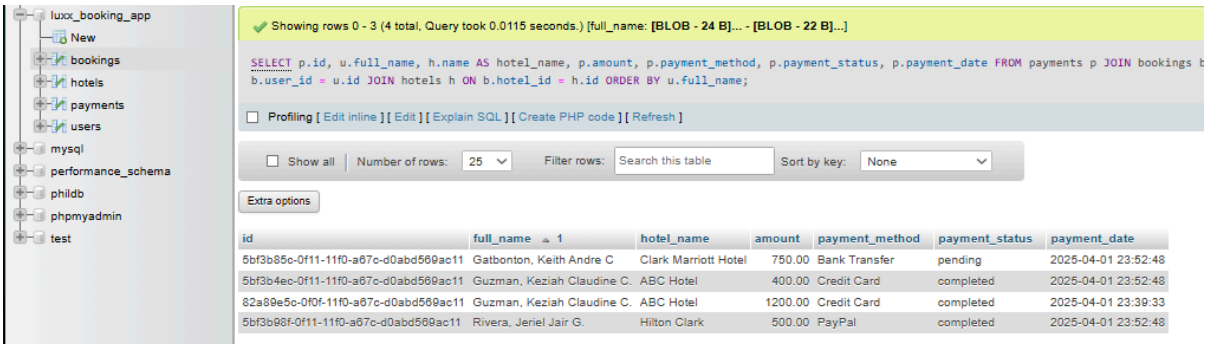
id	full_name 1	hotel_name	check_in_date	check_out_date	number_of_guests	total_price	status
bbd508b6-0f10-11f0-a67c-d0abd569ac11	Gatbonton, Keith Andre C	Clark Marriott Hotel	2025-05-15	2025-05-18	2	750.00	pending
6ad4ed3f-0f0e-11f0-a67c-d0abd569ac11	Guzman, Keziah Claudine C.	ABC Hotel	2025-05-01	2025-05-07	2	1200.00	pending
bbd50373-0f10-11f0-a67c-d0abd569ac11	Guzman, Keziah Claudine C.	ABC Hotel	2025-04-12	2025-04-13	2	400.00	pending
bbd50963-0f10-11f0-a67c-d0abd569ac11	Rivera, Jeriel Jair G.	Hilton Clark	2025-08-10	2025-08-12	1	500.00	pending

Second Sample Query: Select all payments for each user (command and execution)



```
SELECT p.id, u.full_name, h.name AS hotel_name, p.amount, p.payment_method, p.payment_status, p.payment_date
FROM payments p
JOIN bookings b ON p.booking_id = b.id
JOIN users u ON b.user_id = u.id
JOIN hotels h ON b.hotel_id = h.id
ORDER BY u.full_name;
```

id
user_id
hotel_id
check_in_date
check_out_date
number_of_guests
total_price
status
created_at
updated_at



id	full_name 1	hotel_name	amount	payment_method	payment_status	payment_date
5bf3b85c-0f11-11f0-a67c-d0abd569ac11	Gatbonton, Keith Andre C	Clark Marriott Hotel	750.00	Bank Transfer	pending	2025-04-01 23:52:48
5bf3b4ec-0f11-11f0-a67c-d0abd569ac11	Guzman, Keziah Claudine C.	ABC Hotel	400.00	Credit Card	completed	2025-04-01 23:52:48
82a89e5c-0f0f-11f0-a67c-d0abd569ac11	Guzman, Keziah Claudine C.	ABC Hotel	1200.00	Credit Card	completed	2025-04-01 23:39:33
5bf3b98f-0f11-11f0-a67c-d0abd569ac11	Rivera, Jeriel Jair G.	Hilton Clark	500.00	PayPal	completed	2025-04-01 23:52:48

IV. Business Rules & Code Snippets

BUSINESS RULES

1. Date Validation

- Check-in date must be before check-out date.
- Cannot create bookings for past dates.

Implementation:

Welcome | POLICIES.TXT

POLICIES.TXT

```
CREATE OR REPLACE FUNCTION check_dates_valid()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.check_in_date >= NEW.check_out_date THEN
        RAISE EXCEPTION 'Check-in date must be before check-out date';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER validate_dates
BEFORE INSERT OR UPDATE ON public.bookings
FOR EACH ROW
EXECUTE FUNCTION check_dates_valid();

CREATE OR REPLACE FUNCTION check_booking_dates()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.check_in_date < CURRENT_DATE THEN
        RAISE EXCEPTION 'Cannot create booking for past dates';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER validate_booking_dates
BEFORE INSERT ON public.bookings
FOR EACH ROW
EXECUTE FUNCTION check_booking_dates();
```

2. Guest Management

- Number of guests must be between 1 and 4.

Implementation:

Welcome | POLICIES.TXT

POLICIES.TXT

```
CREATE OR REPLACE FUNCTION check_guests_valid()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.number_of_guests < 1 OR NEW.number_of_guests > 4 THEN
        RAISE EXCEPTION 'Number of guests must be between 1 and 4';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER validate_guests
BEFORE INSERT OR UPDATE ON public.bookings
FOR EACH ROW
EXECUTE FUNCTION check_guests_valid();
```

3. Pricing Rules

- Total price is automatically calculated based on nights and price per night.
- Payment amount must match the total booking price.

Implementation:

Welcome | POLICIES.TXT

POLICIES.TXT

```
CREATE OR REPLACE FUNCTION calculate_total_price()
RETURNS TRIGGER AS $$
DECLARE
    nights INTEGER;
    price_per_night DECIMAL;
BEGIN
    -- Calculate number of nights
    nights := EXTRACT(DAY FROM (NEW.check_out_date - NEW.check_in_date));

    -- Get price per night from hotels table
    SELECT price_per_night INTO price_per_night
    FROM public.hotels
    WHERE id = NEW.hotel_id;

    -- Set total price
    NEW.total_price := nights * price_per_night;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER set_total_price
BEFORE INSERT OR UPDATE ON public.bookings
FOR EACH ROW
EXECUTE FUNCTION calculate_total_price();
```

4. Payment Rules

- Payment amount must match the total booking price.
- Payment status tracking is enforced.

Implementation:

Welcome | POLICIES.TXT

POLICIES.TXT

```
CREATE OR REPLACE FUNCTION validate_payment_amount()
RETURNS TRIGGER AS $$
DECLARE
    booking_total DECIMAL;
BEGIN
    SELECT total_price INTO booking_total
    FROM public.bookings
    WHERE id = NEW.booking_id;

    IF NEW.amount != booking_total THEN
        RAISE EXCEPTION 'Payment amount must match booking total price';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER check_payment_amount
BEFORE INSERT OR UPDATE ON public.payments
FOR EACH ROW
EXECUTE FUNCTION validate_payment_amount();
```

5. Booking Status Management

- Cannot complete a booking without a completed payment.
- Prevents overlapping bookings for the same hotel on the same dates.

Implementation:

Welcome | POLICIES.TXT

POLICIES.TXT

```
CREATE OR REPLACE FUNCTION check_booking_payment_status()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.status = 'completed' THEN
        IF NOT EXISTS (
            SELECT 1 FROM public.payments
            WHERE booking_id = NEW.id
            AND payment_status = 'completed'
        ) THEN
            RAISE EXCEPTION 'Cannot complete booking without completed payment';
        END IF;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER validate_booking_completion
BEFORE UPDATE ON public.bookings
FOR EACH ROW
EXECUTE FUNCTION check_booking_payment_status();

CREATE OR REPLACE FUNCTION check_booking_overlap()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (
        SELECT 1 FROM public.bookings
        WHERE hotel_id = NEW.hotel_id
        AND id != NEW.id
        AND status != 'cancelled'
        AND (
            (NEW.check_in_date BETWEEN check_in_date AND check_out_date)
            OR (NEW.check_out_date BETWEEN check_in_date AND check_out_date)
            OR (check_in_date BETWEEN NEW.check_in_date AND NEW.check_out_date)
            OR (check_out_date BETWEEN NEW.check_in_date AND NEW.check_out_date)
        )
    ) THEN
        RAISE EXCEPTION 'Hotel is already booked for these dates';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER check_booking_overlap
BEFORE INSERT OR UPDATE ON public.bookings
FOR EACH ROW
EXECUTE FUNCTION check_booking_overlap();
```

6. User Management

- Cannot delete users with active bookings.

Implementation:

Welcome | POLICIES.TXT

POLICIES.TXT

```
CREATE OR REPLACE FUNCTION check_user_deletion()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (
        SELECT 1 FROM public.bookings
        WHERE user_id = OLD.id
        AND status NOT IN ('cancelled', 'completed')
    ) THEN
        RAISE EXCEPTION 'Cannot delete user with active bookings';
    END IF;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER prevent_user_deletion
BEFORE DELETE ON public.users
FOR EACH ROW
EXECUTE FUNCTION check_user_deletion();
```

CODE SNIPPETS:

```
📁 luxx/
├── 📁 app/
│   └── 📁 src/
│       └── 📁 main/
│           ├── 📄 AndroidManifest.xml
│           └── 📁 java/com/example/luxx/
│               ├── 📄 MainActivity.kt
│               │
│               ├── 📁 data/
│               │   ├── 📄 SupabaseClient.kt
│               │   ├── 📄 SupabaseConfig.kt
│               │   ├── 📁 repository/
│               │   │   ├── 📄 UserRepository.kt
│               │   │   ├── 📄 HotelRepository.kt
│               │   │   └── 📄 BookingRepository.kt
│               │   └── 📁 model/
│               │
│               ├── 📁 navigation/
│               │   ├── 📄 BottonNavigationBar.kt
│               │   ├── 📄 Screen.kt
│               │   ├── 📄 AppNavigation.kt
│               │   ├── 📄 Routes.kt
│               │   └── 📄 NavGraph.kt
│               │
│               └── 📁 ui/
│                   ├── 📁 auth/
│                   │   ├── 📄 LoginScreen.kt
│                   │   ├── 📄 RegisterScreen.kt
│                   │   └── 📄 AuthViewModel.kt
│                   │
│                   ├── 📁 booking/
│                   │   └── 📄 BookingViewModel.kt
│                   │
│                   ├── 📁 components/
│                   │   └── 📄 AuthComponents.kt
│                   │
│                   ├── 📁 hotels/
│                   │   ├── 📄 ABCHotelScreen.kt
│                   │   ├── 📄 ClarkMarriottScreen.kt
│                   │   ├── 📄 ElenemResortScreen.kt
│                   │   ├── 📄 GloryHotelScreen.kt
│                   │   ├── 📄 HiltonClarkScreen.kt
│                   │   ├── 📄 MidoriClarkScreen.kt
│                   │   ├── 📄 OneEuphoriaScreen.kt
│                   │   ├── 📄 ParkInnScreen.kt
│                   │   ├── 📄 QuestPlusScreen.kt
│                   │   └── 📄 WidusHotelScreen.kt
│                   │
│                   ├── 📁 payment/
│                   │   └── 📄 PaymentViewModel.kt
│                   │
│                   ├── 📁 screens/
│                   │   ├── 📄 PaymentScreen.kt
│                   │   ├── 📄 BookingScreen.kt
│                   │   ├── 📄 HomeScreen.kt
│                   │   ├── 📄 LoginScreen.kt
│                   │   ├── 📄 HotelsScreen.kt
│                   │   ├── 📄 AdminDashboardScreen.kt
│                   │   └── 📄 ProfileScreen.kt
│                   │
│                   └── 📁 theme/
│                       ├── 📄 Colors.kt
│                       ├── 📄 Theme.kt
│                       ├── 📄 Color.kt
│                       └── 📄 Type.kt
```

V. SYSTEM DOCUMENTATION

1. Overview of the Software Functionalities

- This application is intended for use in managing hotel bookings, user accounts, and admin controls. The functionalities are user login, user profiles, access to dashboard, and booking capabilities.

1.1 User Registration

- Users can sign up with their password and email.
- Registration requires minimal data such as full name, phone number, and address.

1.2 User Login

- The users can log in using their registered credentials.
- Password recovery is possible if needed.

1.3 Dashboard

- The users are redirected back to their personal dashboard when logging in.

- Shows future and past reservations.

1.4 Profile Management

- They can edit their own details such as name, phone number, and address.

- Feature to reset the password for security purposes.

1.5 Admin Dashboard

- Administrators are able to see user management and administer the system.

- User account deletion feature if necessary.

2. Usage Instructions

2.1 Accessing the System

- Open the application via the web or mobile interface.

- Ensure you have an active internet connection.

2.2 Registering as a User

- Press the "Sign Up" on the front page.

- Fill out required registration details.

- Complete the application form to apply.

2.3 Logging In

- Enter your password and your email on the login page.

- Click on "Login" to view your dashboard.

2.4 Navigating the Dashboard

- The dashboard provides an overview of profile settings and bookings.

- They are able to see the booking history and book again.

2.5 Booking a Hotel

- Proceed to the "Hotels" section.

- Choose a hotel and choose your desired check-in and check-out dates.

- Specify the number of visitors and ensure the overall amount.

- Complete the booking process by making a payment.

- The confirmation message will be displayed in case of successful

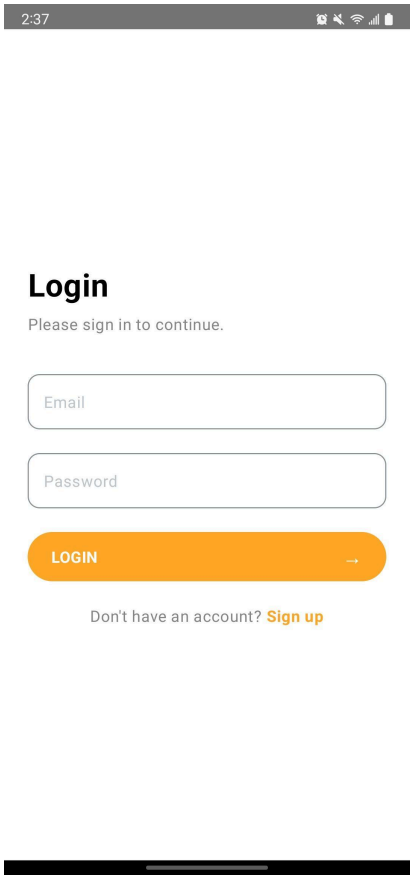
booking.

3. Data Protection and Security

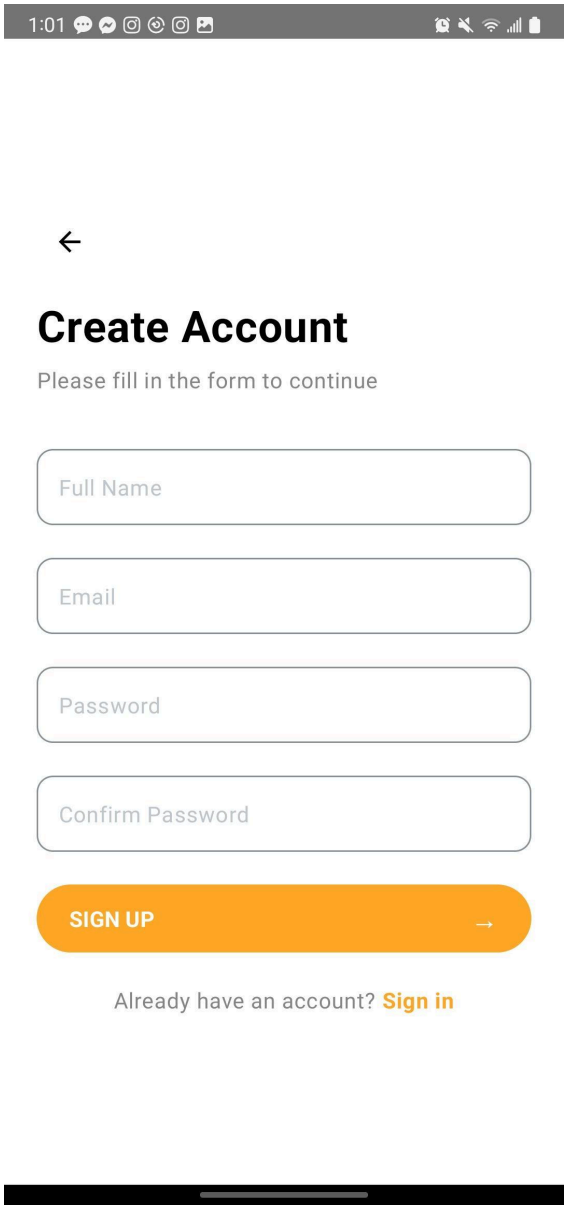
- The system employs secure authentication and encryption methods.
- The users can update their login information whenever needed.
- Admins control activity to avoid unauthorized access.

3. System Screenshots

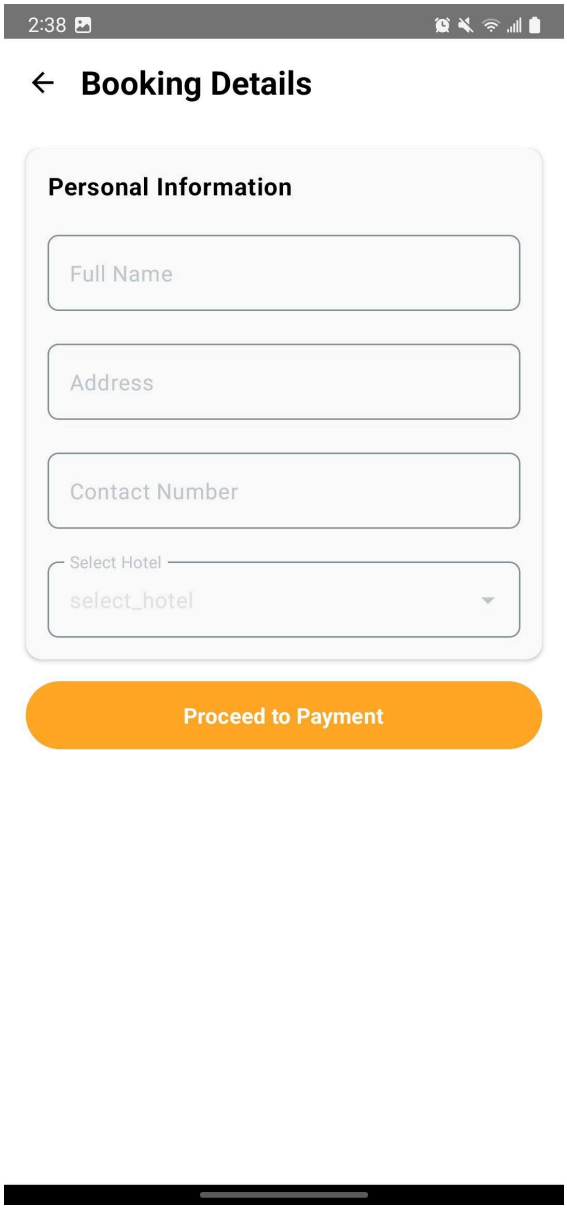
3.1 Login Page



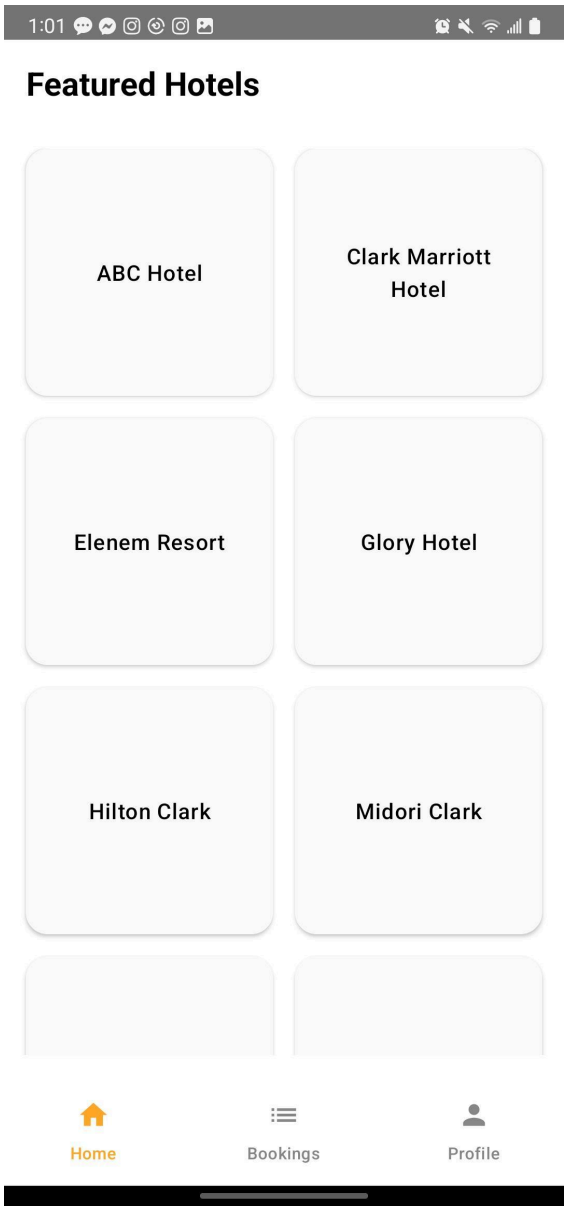
3.2 Create Account Page

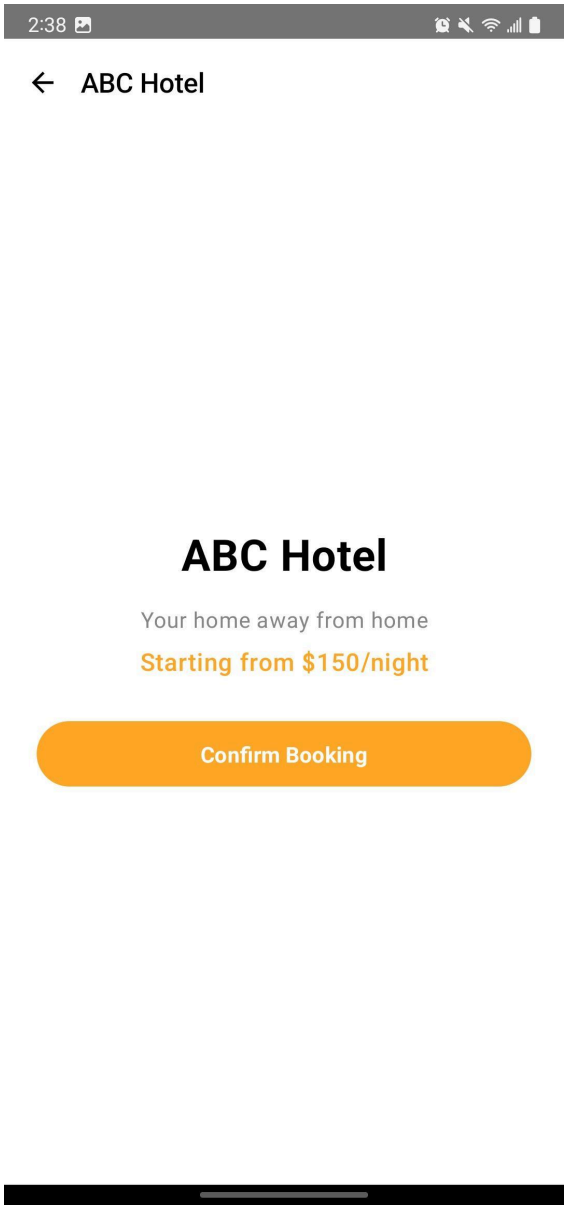


3.3 Profile Page/Booking Details

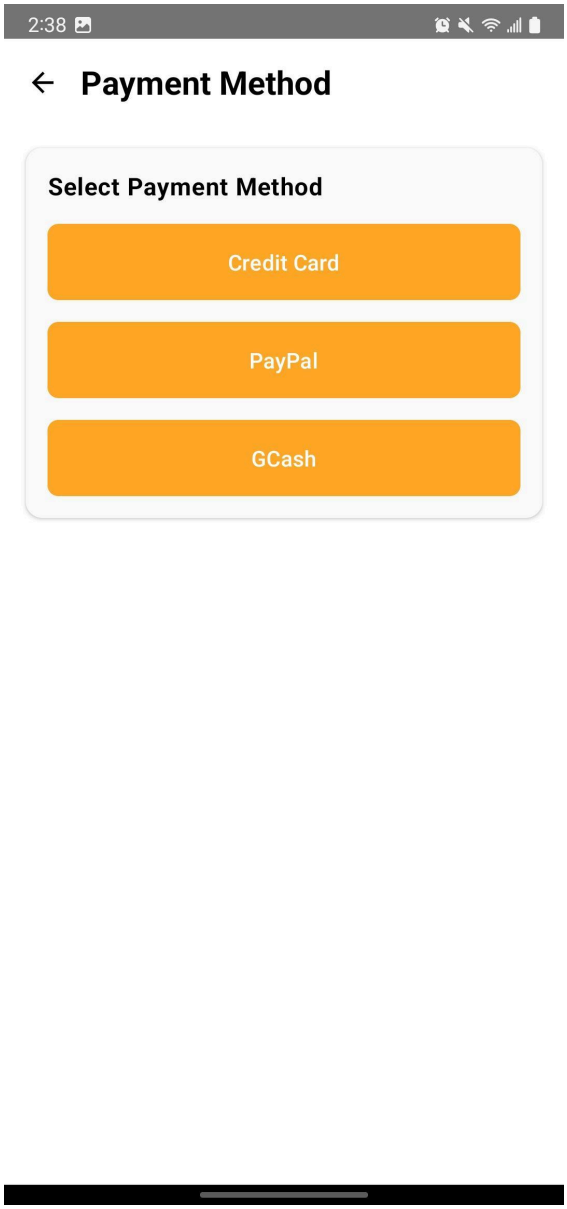

2:38
Booking Details
Personal Information
Full Name
Address
Contact Number
Select Hotel
select_hotel
Proceed to Payment

3.4 Hotel Options Page



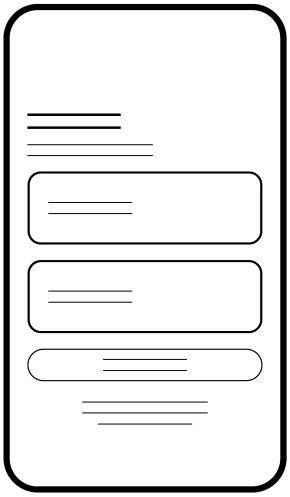

2:38
ABC Hotel
ABC Hotel
Your home away from home
Starting from $150/night
Confirm Booking

3.5 Payment Page

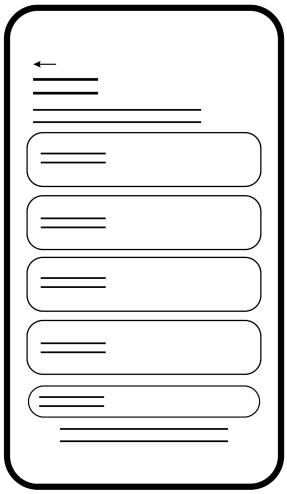


4. Wireframes

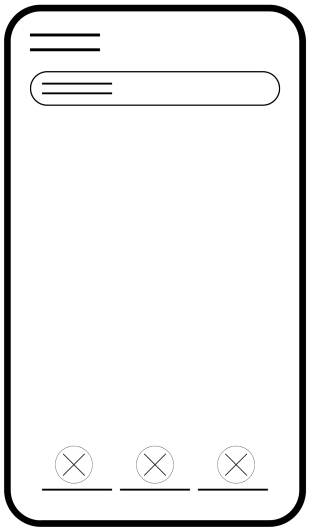
Wireframe For Login Page



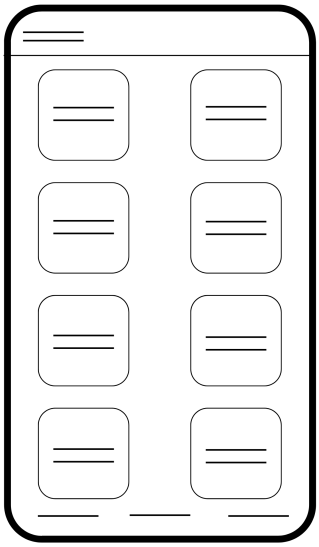
Wireframe for Create Account Page



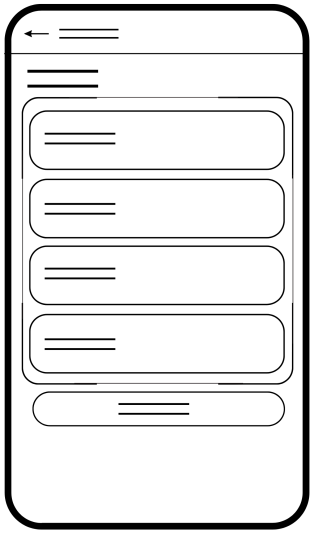
Wireframe For Profile Page



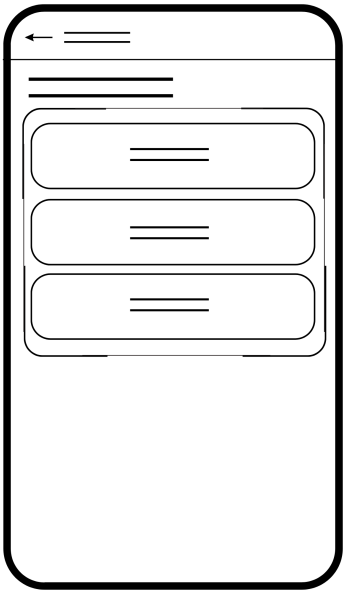
Wireframe For Featured Hotel Options



Wireframe for Booking Details and Personal Information



Wireframe For Payment Methods Page



VI. Members & Roles

Members:

1. Gatbonton, Keith Andrē C.
2. Guzman, Keziah Claudine C.
3. Rivera, Jeriel Jair G.

Roles:

1. Gatbonton, Keith Andrē C.

- Leader and Manager of Project
- Diagram and Wireframe Creator
- Proofreader and Documentation Creator

2. Guzman Keziah Claudine

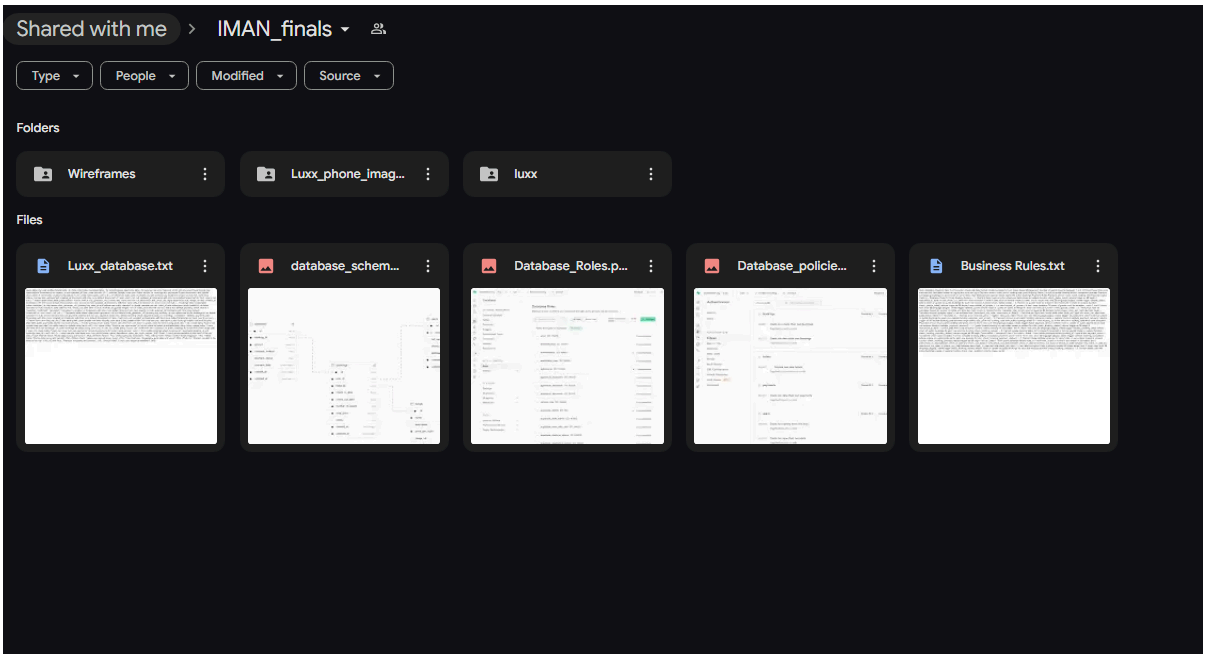
- Main Creator of Documentation
- Planner of UI Design

3. Rivera, Jeriel Jair G.

- Main Interface Creator
- Main Coder and Creator of Source Code Implementation
- Main Tester and Initiator of Troubleshooting

VII. CODEBASE:

Source Code:



https://drive.google.com/drive/folders/1nxAac5MsyAc-_4n0RGdYbLilyzGLxkKE?usp=sharing