# Student dorm cleaning service app

## 1 . Purpose and pain point

My project aims to create a booking platform designed to simplify and improve dormitory cleaning management. Many existing cleaning services rely on manual bookings, resulting in disorganized scheduling, misplaced records, double-bookings, and unexpected cancellations, while lacking subscription plan support. This project solves these pain points by centralizing customer, cleaner, schedule, and service data into a digital system that supports structured bookings, clear time slots, and recurring subscription plans. By replacing manual coordination with organized data flow and subscription options, the platform improves reliability, prevents double-booking conflicts, and scales efficiently for student dorm cleaning services.

## 2. Service Price

Regular Service price: ฿250–350 per 1 hour, varies by room size and cleaning time.
Monthly Subscription: ฿200–220 per 1 hour ,varies by room size and cleaning time.
Platform commission: 15% per transaction, deducted from each booking payment.

## 3. Key Features & Functions

### 3.1. Login & Authentication

The function `fn_login_with_username_and_password(username, password)` validates user credentials and returns success or failure, using hashed password checks and role verification to block unauthorized or fake access so only registered users can proceed with bookings.

### 3.2 Homepage / Available Services

The function `fn_list_active_services()` retrieves active cleaning services, while `fn_search_keyword(keyword)` enables fast service filtering; screens display service name, description, duration, hourly price, and number of cleaners, supported by indexed search for quick browsing.

## 3.3 Cleaner Profiles

The system uses `fn_get_top_rated(min_rating)` to return cleaner name, bio, and star rating, helping students choose trusted cleaners and increasing transparency and booking confidence.

## 3.4 Date & Time Matching

The function `fn_find_available_cleaners(username, date)` checks the availability table and detects time conflicts, preventing double bookings and ensuring reliable scheduling.

## 3.5 User Location / Saved Addresses

The function `fn_get_user_location(username)` loads saved dorm building and room number, enabling one-tap location selection and faster, error-free address linking for service delivery.

## 3.6 Booking Confirmation

The function `fn_creating_booking(user, cleaner, service, time, location)` creates a booking record and returns a unique `booking_id`, which links the user, cleaner, service, and location; this ID is reused on tracking and payment screens.

## 3.7 Payment System

The function  fn_save_payment(booking, amount, currency, method) stores transaction details with amount, payment method (e.g. cash, card, or mobile banking), timestamp, and status, returning a payment_id;  indexing on booking_id  allows fast payment validation during heavy queries.

### 3.8 Subscription

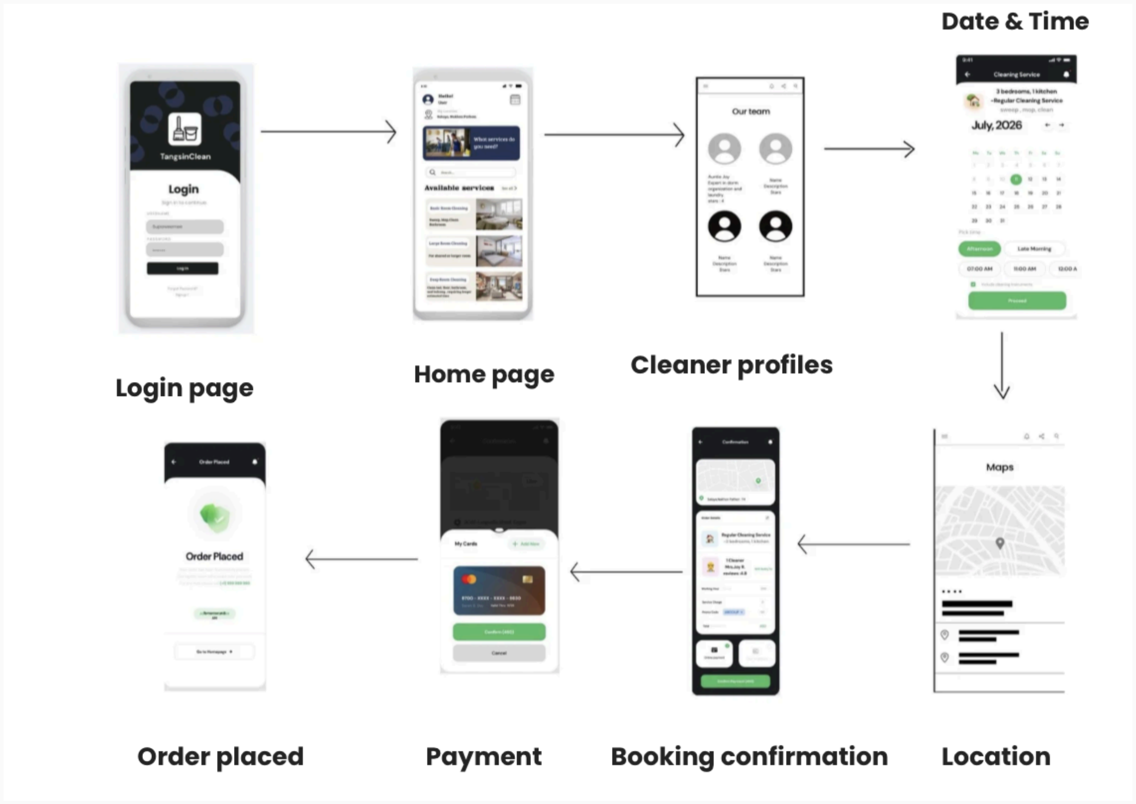
The function fn_user_subscription(user, plan, start_date)  registers recurring membership plans, stores the plan reference, calculates the next billing date, and returns a subscription_id ; the monthly plan pricing is structured to be cheaper per hour, improving affordability while generating predictable recurring revenue.

## 4. User security

- fn_login_with_username_and_password validates username, hashed password, and role to secure login.
- fn_list_active_services returns only active services, protecting hidden or inactive data.
- fn_find_matching_cleaners checks cleaner availability with conflict filtering to avoid double-booking exploits.
- fn_get_user_location retrieves saved dorm addresses and links them safely by location_id, limiting user access to their own stored locations.

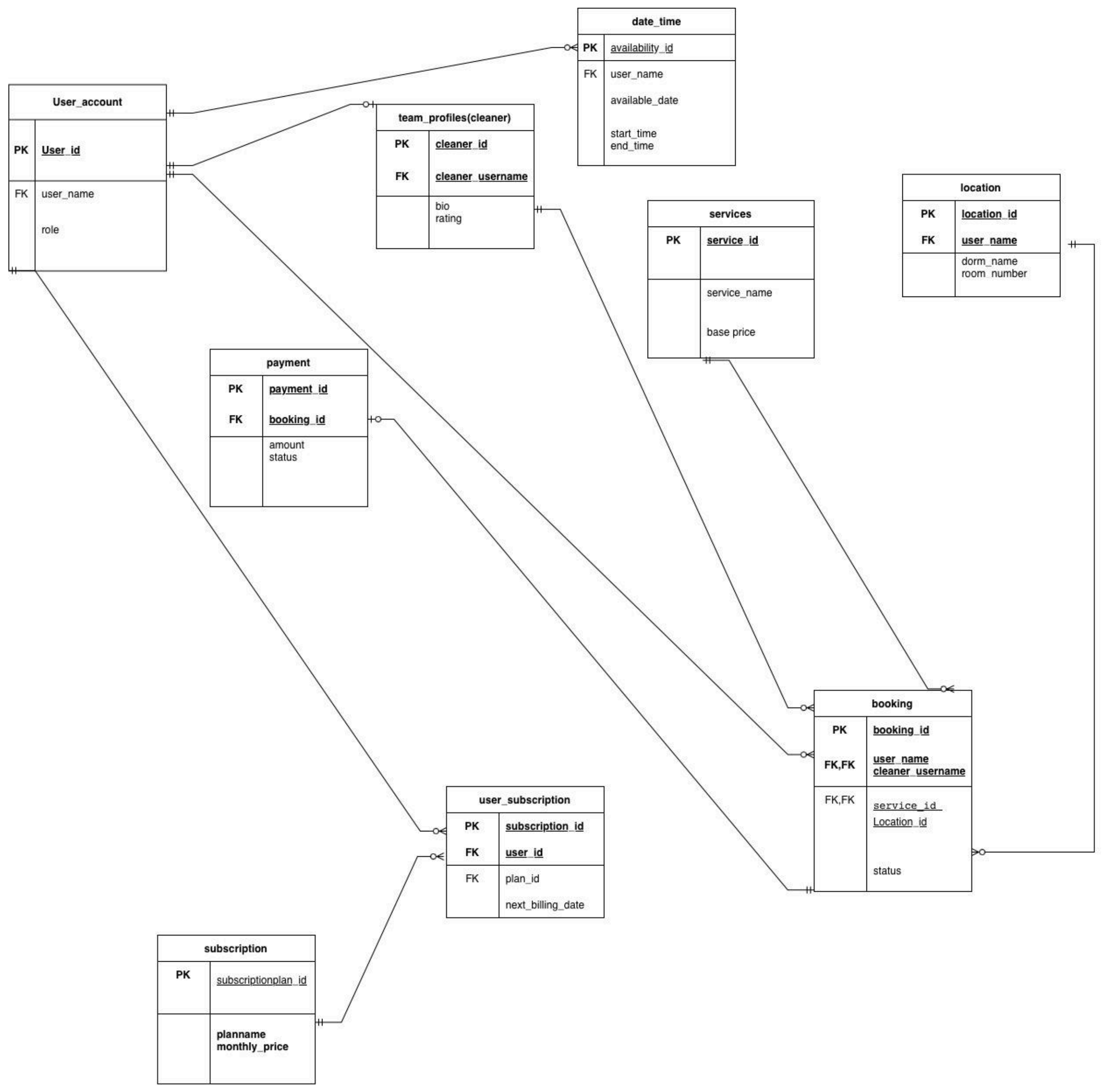
## Site Map

The sitemap shows the user flow of the cleaning service platform. It starts from the Login page, followed by the Home page where available services are displayed. Users can view Cleaner Profiles, select Date & Time, and proceed to Booking. Next steps include Location, Booking Confirmation, Payment, and Order Placed, completing the service request process.



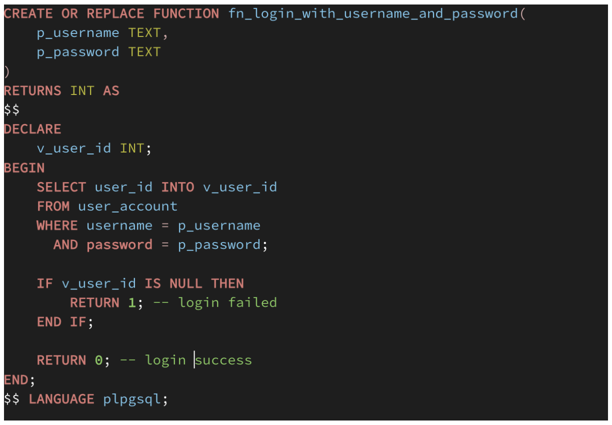
ER diagram

All data entities are connected using primary keys and foreign keys. A customer record in the database is uniquely identified in the User_account table by its User_id and role (e.g. student or cleaner). Cleaning providers are stored in team_profiles with a cleaner_id and referenced by username. When a booking is created, the booking table stores a booking_id and links the user, cleaner, service, and address by including user_name, cleaner_username, service_id, and location_id as foreign keys. Payment records then connect directly to the booking using booking_id to store the amount and status, while subscriptions link customers to isolated subscription plans through subscription_id and track future billing dates. The booking_id acts as the central connection point that ties the main app data together.

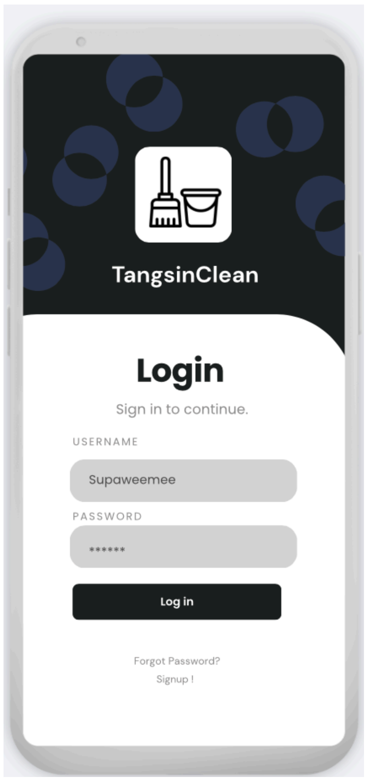


## UI from the app

1) Login page



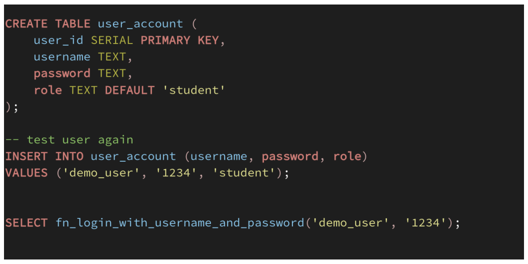
```
CREATE OR REPLACE FUNCTION fn_login_with_username_and_password(
    p_username TEXT,
    p_password TEXT
)
RETURNS INT AS
$$
DECLARE
    v_user_id INT;
BEGIN
    SELECT user_id INTO v_user_id
    FROM user_account
    WHERE username = p_username
      AND password = p_password;

    IF v_user_id IS NULL THEN
        RETURN 1; -- login failed
    END IF;

    RETURN 0; -- login success
END;
$$ LANGUAGE plpgsql;
```



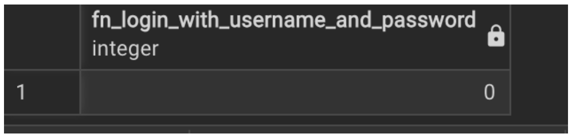
Call: fn_login_with_username_and_password(username, password)

Result :

Return 0 if login success

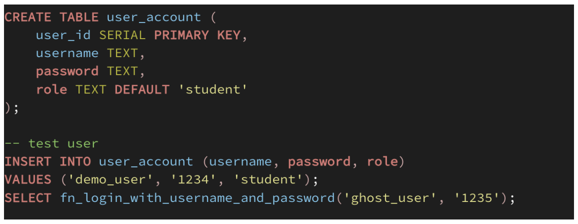


```
CREATE TABLE user_account (
    user_id SERIAL PRIMARY KEY,
    username TEXT,
    password TEXT,
    role TEXT DEFAULT 'student'
);

-- test user again
INSERT INTO user_account (username, password, role)
VALUES ('demo_user', '1234', 'student');


SELECT fn_login_with_username_and_password('demo_user', '1234');
```

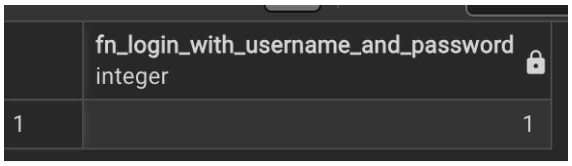


```
  | fn_login_with_username_and_password
  | integer
1 |                                   0
```

Return 1 if login failed



```
CREATE TABLE user_account (
    user_id SERIAL PRIMARY KEY,
    username TEXT,
    password TEXT,
    role TEXT DEFAULT 'student'
);

-- test user
INSERT INTO user_account (username, password, role)
VALUES ('demo_user', '1234', 'student');
SELECT fn_login_with_username_and_password('ghost_user', '1235');
```

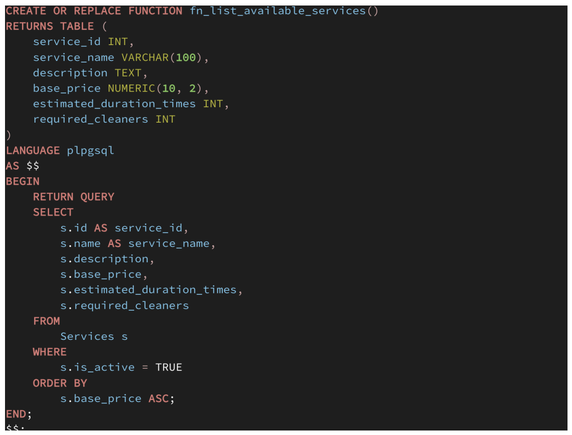


```
  | fn_login_with_username_and_password
  | integer
1 |                                   1
```

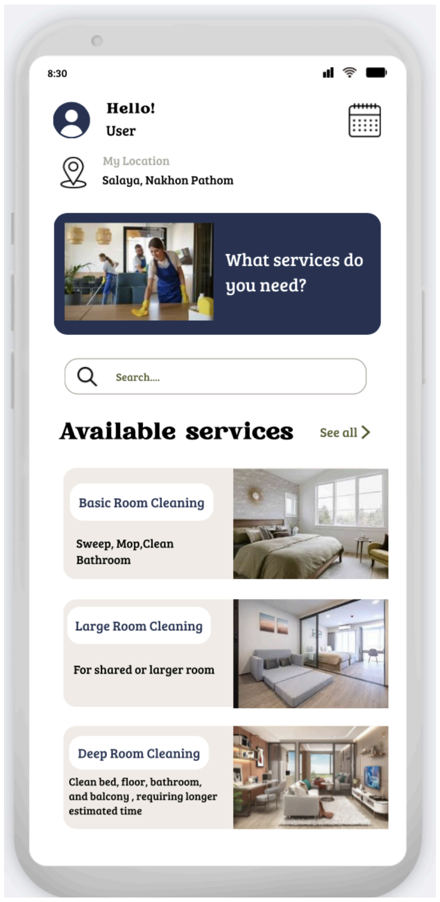
Supawee Meersripong 6481317

2) Available services

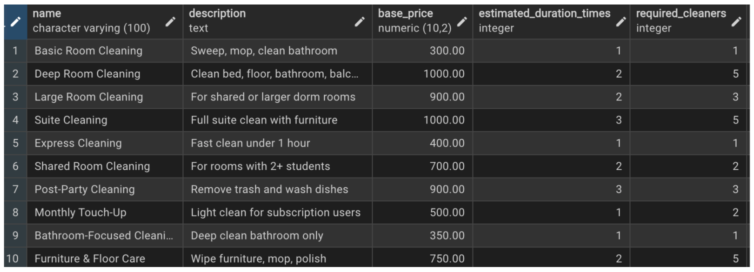
The function **fn_list_available_services** retrieves a list of all active cleaning services and returns their main details ( service id , name, description, estimated duration, price, and number of cleaners needed).



```
CREATE OR REPLACE FUNCTION fn_list_available_services()
RETURNS TABLE (
    service_id INT,
    service_name VARCHAR(100),
    description TEXT,
    base_price NUMERIC(10, 2),
    estimated_duration_times INT,
    required_cleaners INT
)
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN QUERY
    SELECT
        s.id AS service_id,
        s.name AS service_name,
        s.description,
        s.base_price,
        s.estimated_duration_times,
        s.required_cleaners
    FROM
        Services s
    WHERE
        s.is_active = TRUE
    ORDER BY
        s.base_price ASC;
END;
$$;
```

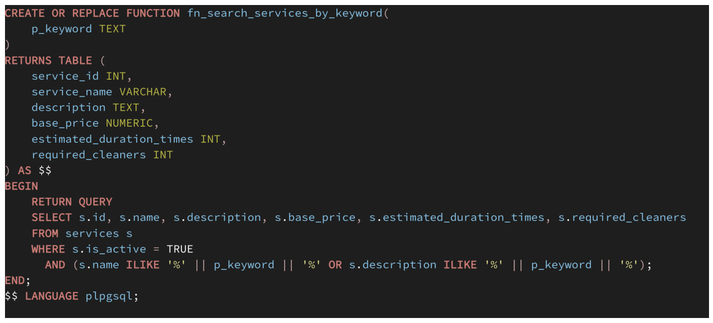

8:30
Hello!
User
My Location
Salaya, Nakhon Pathom
What services do you need?
Search....
Available services
See all
Basic Room Cleaning
Sweep, Mop,Clean Bathroom
Large Room Cleaning
For shared or larger room
Deep Room Cleaning
Clean bed, floor, bathroom, and balcony , requiring longer estimated time

**Result :** Lists of all available services with details such as prices, estimated times in hour and the number of cleaners.

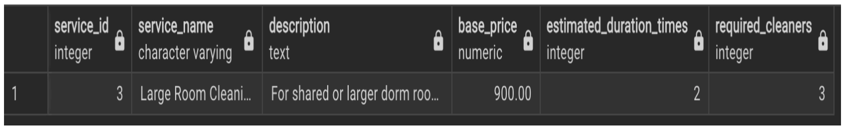


| | name<br>character varying (100) | description<br>text | base_price<br>numeric (10,2) | estimated_duration_times<br>integer | required_cleaners<br>integer |
|---|---|---|---|---|---|
| 1 | Basic Room Cleaning | Sweep, mop, clean bathroom | 300.00 | 1 | 1 |
| 2 | Deep Room Cleaning | Clean bed, floor, bathroom, balc... | 1000.00 | 2 | 5 |
| 3 | Large Room Cleaning | For shared or larger dorm rooms | 900.00 | 2 | 3 |
| 4 | Suite Cleaning | Full suite clean with furniture | 1000.00 | 3 | 5 |
| 5 | Express Cleaning | Fast clean under 1 hour | 400.00 | 1 | 1 |
| 6 | Shared Room Cleaning | For rooms with 2+ students | 700.00 | 2 | 2 |
| 7 | Post-Party Cleaning | Remove trash and wash dishes | 900.00 | 3 | 3 |
| 8 | Monthly Touch-Up | Light clean for subscription users | 500.00 | 1 | 2 |
| 9 | Bathroom-Focused Cleani... | Deep clean bathroom only | 350.00 | 1 | 1 |
| 10 | Furniture & Floor Care | Wipe furniture, mop, polish | 750.00 | 2 | 5 |

For **fn_search_services_by_keyword**( ‘keyword’),users can quickly find services by searching
Example of input keyword: “ Large Room”



```
CREATE OR REPLACE FUNCTION fn_search_services_by_keyword(
    p_keyword TEXT
)
RETURNS TABLE (
    service_id INT,
    service_name VARCHAR,
    description TEXT,
    base_price NUMERIC,
    estimated_duration_times INT,
    required_cleaners INT
) AS $$
BEGIN
    RETURN QUERY
    SELECT s.id, s.name, s.description, s.base_price, s.estimated_duration_times, s.required_cleaners
    FROM services s
    WHERE s.is_active = TRUE
      AND (s.name ILIKE '%' || p_keyword || '%' OR s.description ILIKE '%' || p_keyword || '%');
END;
$$ LANGUAGE plpgsql;
```

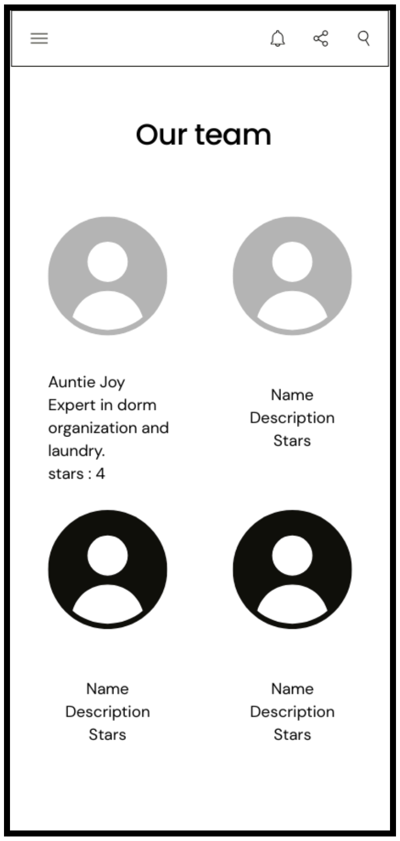


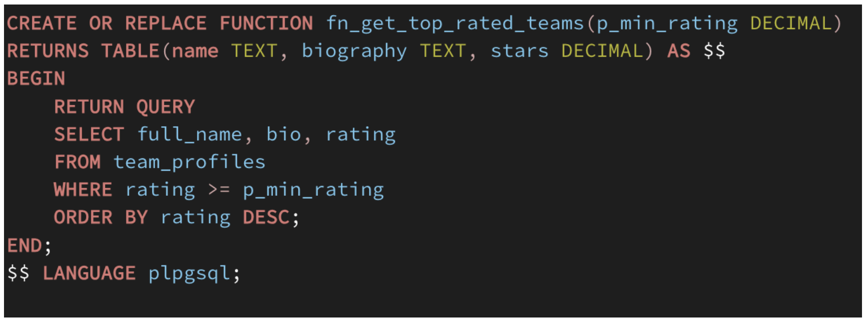
| | service_id<br>integer | service_name<br>character varying | description<br>text | base_price<br>numeric | estimated_duration_times<br>integer | required_cleaners<br>integer |
|---|---|---|---|---|---|---|
| 1 | 3 | Large Room Cleani... | For shared or larger dorm roo... | 900.00 | 2 | 3 |

Supawee Meersripong 6481317

3.Cleaner profiles

**fn_get_top_rated_teams (min_rating):** This function will filter cleaners based on their star rating to help students select trusted cleaners.

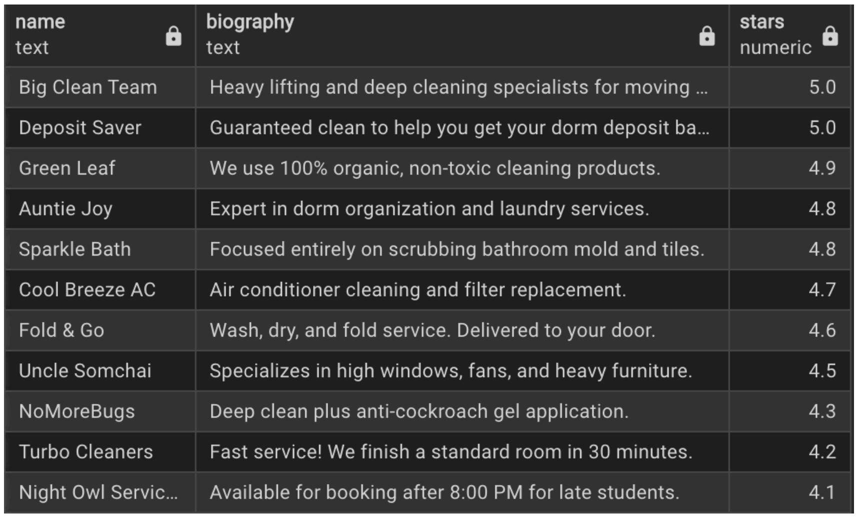




```
CREATE OR REPLACE FUNCTION fn_get_top_rated_teams(p_min_rating DECIMAL)
RETURNS TABLE(name TEXT, biography TEXT, stars DECIMAL) AS $$
BEGIN
    RETURN QUERY
    SELECT full_name, bio, rating
    FROM team_profiles
    WHERE rating >= p_min_rating
    ORDER BY rating DESC;
END;
$$ LANGUAGE plpgsql;
```

Example of input : fn_get_top_rated_teams (4.1)

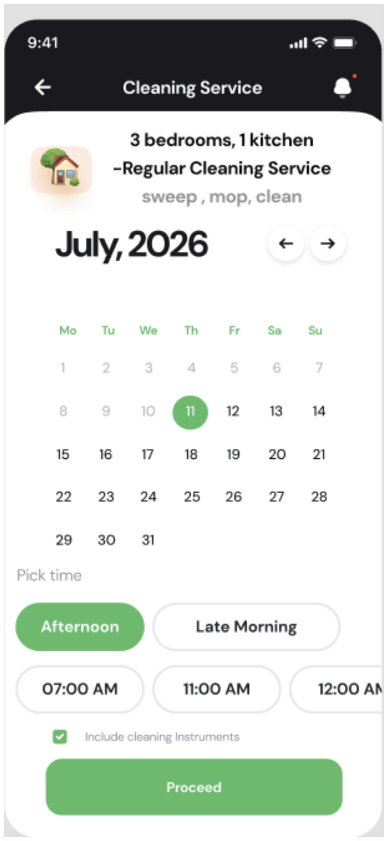
Result: It will return list of all cleaners from 4.1 stars or higher

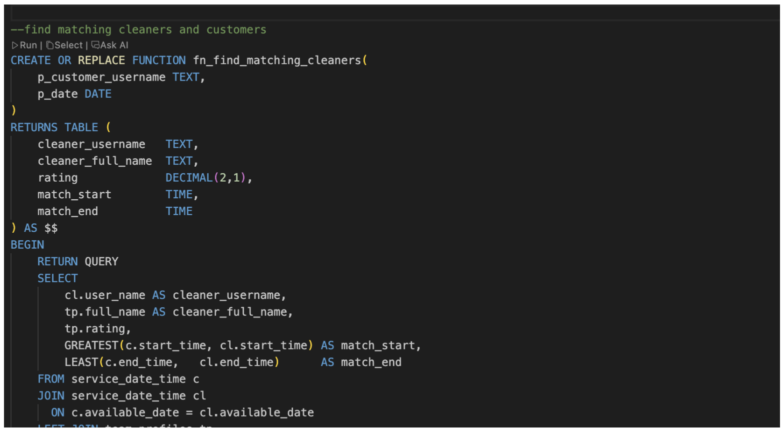


| name<br>text | biography<br>text | stars<br>numeric |
|---|---|---|
| Big Clean Team | Heavy lifting and deep cleaning specialists for moving ... | 5.0 |
| Deposit Saver | Guaranteed clean to help you get your dorm deposit ba... | 5.0 |
| Green Leaf | We use 100% organic, non-toxic cleaning products. | 4.9 |
| Auntie Joy | Expert in dorm organization and laundry services. | 4.8 |
| Sparkle Bath | Focused entirely on scrubbing bathroom mold and tiles. | 4.8 |
| Cool Breeze AC | Air conditioner cleaning and filter replacement. | 4.7 |
| Fold & Go | Wash, dry, and fold service. Delivered to your door. | 4.6 |
| Uncle Somchai | Specializes in high windows, fans, and heavy furniture. | 4.5 |
| NoMoreBugs | Deep clean plus anti-cockroach gel application. | 4.3 |
| Turbo Cleaners | Fast service! We finish a standard room in 30 minutes. | 4.2 |
| Night Owl Servic... | Available for booking after 8:00 PM for late students. | 4.1 |

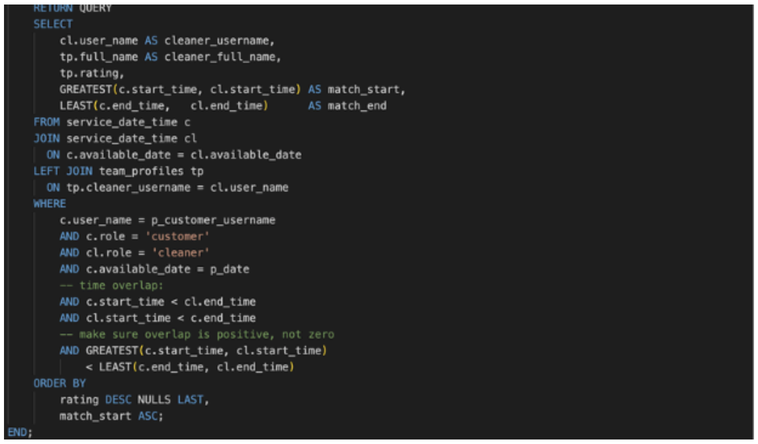
Key points: The function returns cleaner identity, biography, and rating to help students select trusted cleaners, increasing booking confidence and service transparency.

4) Date & Time





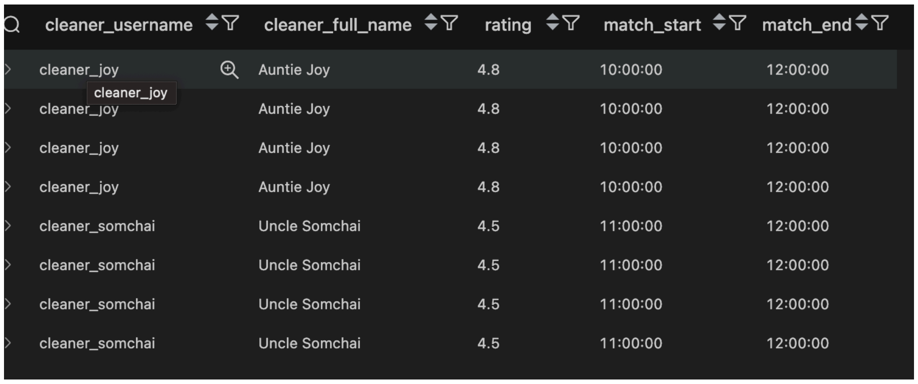
```
--find matching cleaners and customers
▷Run | Select | Ask AI
CREATE OR REPLACE FUNCTION fn_find_matching_cleaners(
    p_customer_username TEXT,
    p_date DATE
)
RETURNS TABLE (
    cleaner_username   TEXT,
    cleaner_full_name  TEXT,
    rating             DECIMAL(2,1),
    match_start        TIME,
    match_end          TIME
) AS $$
BEGIN
    RETURN QUERY
    SELECT
        cl.user_name AS cleaner_username,
        tp.full_name AS cleaner_full_name,
        tp.rating,
        GREATEST(c.start_time, cl.start_time) AS match_start,
        LEAST(c.end_time,   cl.end_time)      AS match_end
    FROM service_date_time c
    JOIN service_date_time cl
      ON c.available_date = cl.available_date
```



```
    RETURN QUERY
    SELECT
        cl.user_name AS cleaner_username,
        tp.full_name AS cleaner_full_name,
        tp.rating,
        GREATEST(c.start_time, cl.start_time) AS match_start,
        LEAST(c.end_time,   cl.end_time)      AS match_end
    FROM service_date_time c
    JOIN service_date_time cl
      ON c.available_date = cl.available_date
    LEFT JOIN team_profiles tp
      ON tp.cleaner_username = cl.user_name
    WHERE
        c.user_name = p_customer_username
        AND c.role = 'customer'
        AND cl.role = 'cleaner'
        AND c.available_date = p_date
        -- time overlap:
        AND c.start_time < cl.end_time
        AND cl.start_time < c.end_time
        -- make sure overlap is positive, not zero
        AND GREATEST(c.start_time, cl.start_time)
            < LEAST(c.end_time, cl.end_time)
    ORDER BY
        rating DESC NULLS LAST,
        match_start ASC;
END;
```

**fn_find_matching_cleaners** :checks cleaner availability and returns all cleaners who are free at the same date and time the student selects.

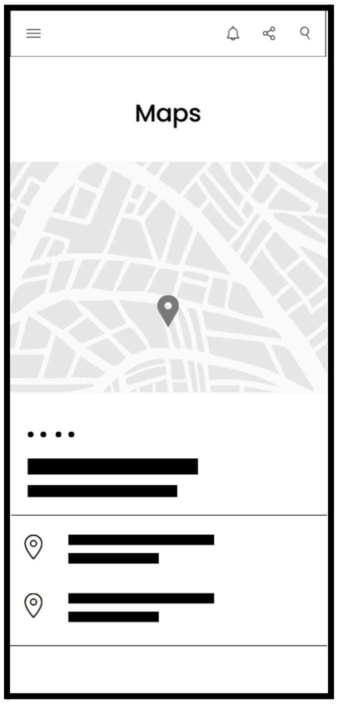
Example of input : fn_find_matching_cleaners('student_tang', '2025-12-05')

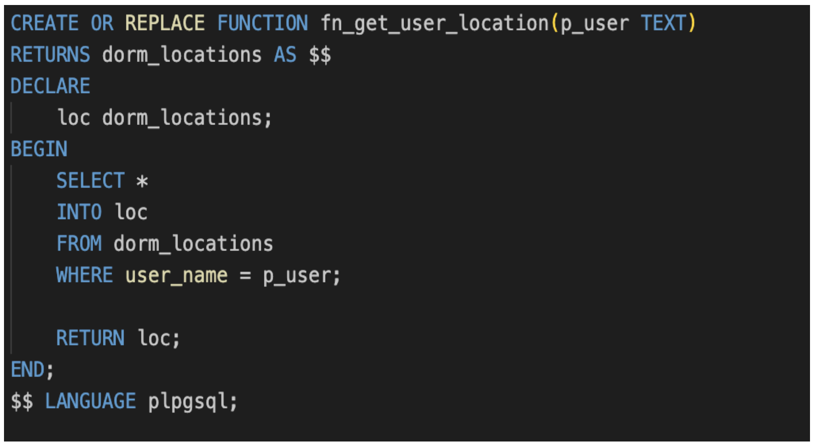


| cleaner_username | cleaner_full_name | rating | match_start | match_end |
|---|---|---|---|---|
| cleaner_joy | Auntie Joy | 4.8 | 10:00:00 | 12:00:00 |
| cleaner_joy | Auntie Joy | 4.8 | 10:00:00 | 12:00:00 |
| cleaner_joy | Auntie Joy | 4.8 | 10:00:00 | 12:00:00 |
| cleaner_joy | Auntie Joy | 4.8 | 10:00:00 | 12:00:00 |
| cleaner_somchai | Uncle Somchai | 4.5 | 11:00:00 | 12:00:00 |
| cleaner_somchai | Uncle Somchai | 4.5 | 11:00:00 | 12:00:00 |
| cleaner_somchai | Uncle Somchai | 4.5 | 11:00:00 | 12:00:00 |
| cleaner_somchai | Uncle Somchai | 4.5 | 11:00:00 | 12:00:00 |

5) User Location

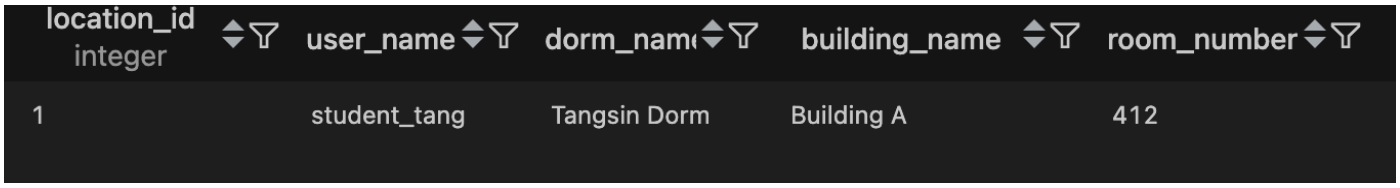
**Fn_get_user_location (username) :** This function saves user addresses and allow users to quickly select their dorm location during booking.





```
CREATE OR REPLACE FUNCTION fn_get_user_location(p_user TEXT)
RETURNS dorm_locations AS $$
DECLARE
    loc dorm_locations;
BEGIN
    SELECT *
    INTO loc
    FROM dorm_locations
    WHERE user_name = p_user;

    RETURN loc;
END;
$$ LANGUAGE plpgsql;
```

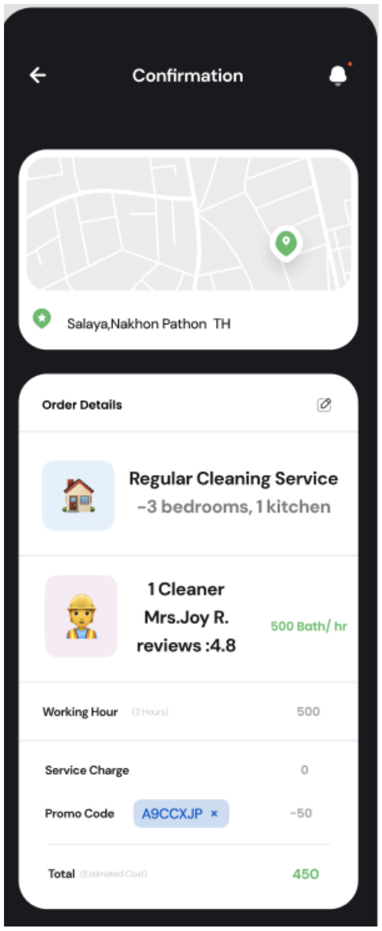
Example of input : “student_tang (username)” this will return user location information
The selected address reference is stored as location_id and linked to booking for later use.



| location_id integer | user_name | dorm_nam | building_name | room_number |
|---|---|---|---|---|
| 1 | student_tang | Tangsin Dorm | Building A | 412 |

6) Confirm bookings

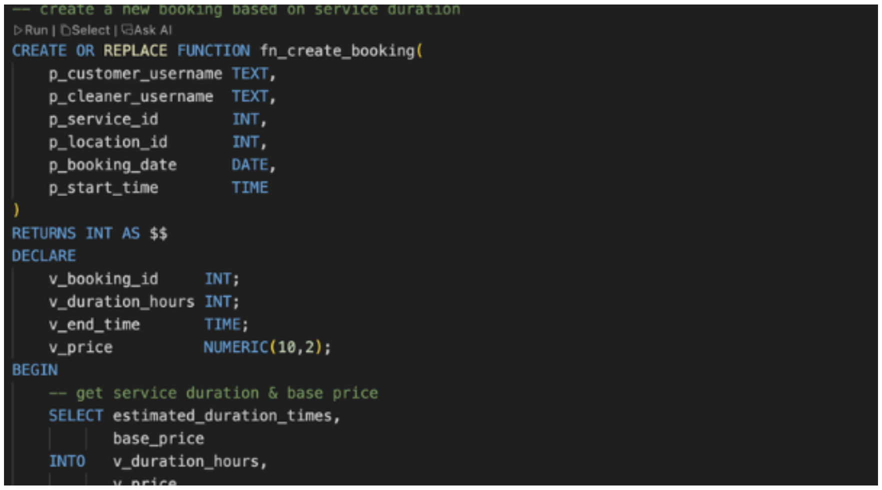
**fn_creating_booking** : This function will create a booking id for a booking list.



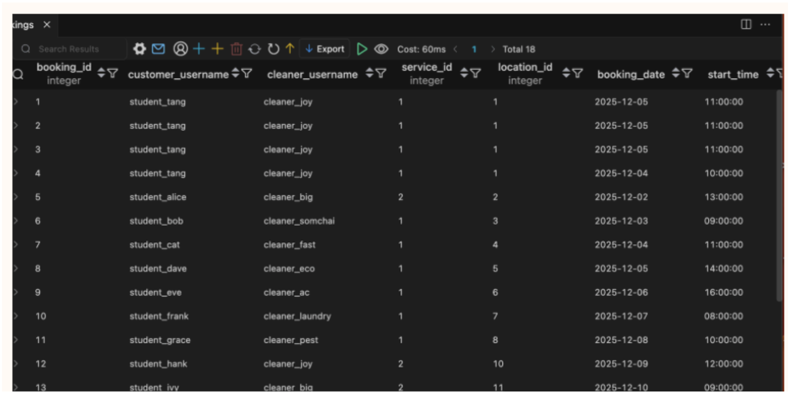
Regular Cleaning Service
-3 bedrooms, 1 kitchen

1 Cleaner
Mrs.Joy R.
reviews :4.8
500 Bath/ hr

| | | |
|---|---|---|
| Working Hour (2 Hours) | | 500 |
| Service Charge | | 0 |
| Promo Code | A9CCXJP × | -50 |
| Total (Estimated Cost) | | 450 |



```
-- create a new booking based on service duration
CREATE OR REPLACE FUNCTION fn_create_booking(
    p_customer_username TEXT,
    p_cleaner_username  TEXT,
    p_service_id        INT,
    p_location_id       INT,
    p_booking_date      DATE,
    p_start_time        TIME
)
RETURNS INT AS $$
DECLARE
    v_booking_id     INT;
    v_duration_hours INT;
    v_end_time       TIME;
    v_price          NUMERIC(10,2);
BEGIN
    -- get service duration & base price
    SELECT estimated_duration_times,
           base_price
    INTO   v_duration_hours,
           v_price
```



| booking_id integer | customer_username | cleaner_username | service_id integer | location_id integer | booking_date | start_time |
|---|---|---|---|---|---|---|
| 1 | student_tang | cleaner_joy | 1 | 1 | 2025-12-05 | 11:00:00 |
| 2 | student_tang | cleaner_joy | 1 | 1 | 2025-12-05 | 11:00:00 |
| 3 | student_tang | cleaner_joy | 1 | 1 | 2025-12-05 | 11:00:00 |
| 4 | student_tang | cleaner_joy | 1 | 1 | 2025-12-04 | 10:00:00 |
| 5 | student_alice | cleaner_big | 2 | 2 | 2025-12-02 | 13:00:00 |
| 6 | student_bob | cleaner_somchai | 1 | 3 | 2025-12-03 | 09:00:00 |
| 7 | student_cat | cleaner_fast | 1 | 4 | 2025-12-04 | 11:00:00 |
| 8 | student_dave | cleaner_eco | 1 | 5 | 2025-12-05 | 14:00:00 |
| 9 | student_eve | cleaner_ac | 1 | 6 | 2025-12-06 | 16:00:00 |
| 10 | student_frank | cleaner_laundry | 1 | 7 | 2025-12-07 | 08:00:00 |
| 11 | student_grace | cleaner_pest | 1 | 8 | 2025-12-08 | 10:00:00 |
| 12 | student_hank | cleaner_joy | 2 | 10 | 2025-12-09 | 12:00:00 |
| 13 | student_ivy | cleaner_big | 2 | 11 | 2025-12-10 | 09:00:00 |

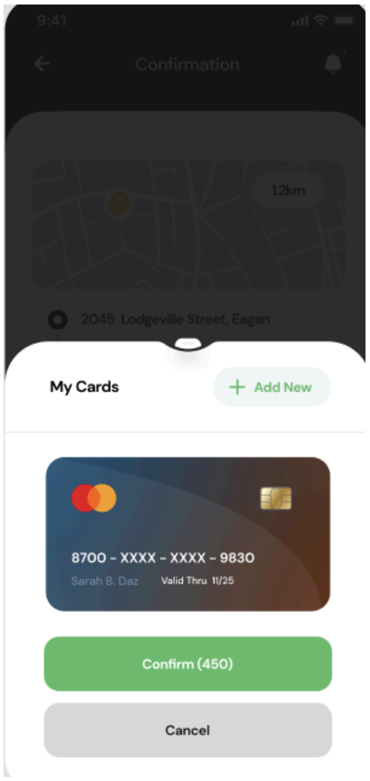
Example Input: p_customer_username: "student_tang" ,p_cleaner_username: "cleaner_joy"

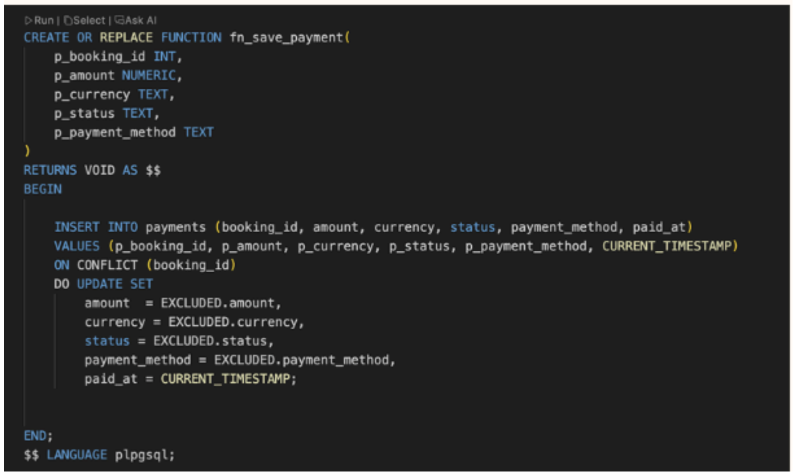
,p_service_id: 1 (Standard Clean),p_location_id: 1 (Tangsin Dorm),p_booking_date: "2025-12-05", p_start_time: "10:00"

Result:The function returns the new Booking ID (e.g., 15) with booking details so the app can show the "Success" screen.

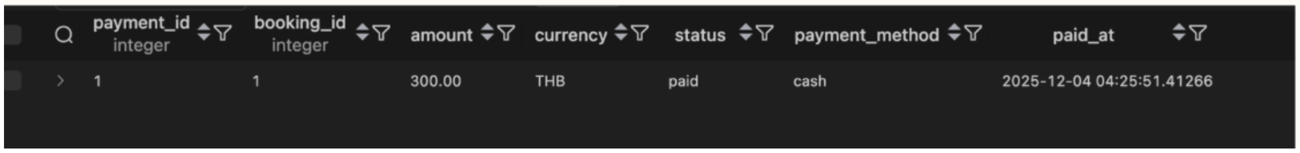
7) Payment

**fn_save_payment** : This saves payment amount, currency, method—such as cash, card, or digital payment—and the current timestamp. It returns a payment_id, which the app uses to confirm that payment was successful.





```
CREATE OR REPLACE FUNCTION fn_save_payment(
    p_booking_id INT,
    p_amount NUMERIC,
    p_currency TEXT,
    p_status TEXT,
    p_payment_method TEXT
)
RETURNS VOID AS $$
BEGIN

    INSERT INTO payments (booking_id, amount, currency, status, payment_method, paid_at)
    VALUES (p_booking_id, p_amount, p_currency, p_status, p_payment_method, CURRENT_TIMESTAMP)
    ON CONFLICT (booking_id)
    DO UPDATE SET
        amount  = EXCLUDED.amount,
        currency = EXCLUDED.currency,
        status = EXCLUDED.status,
        payment_method = EXCLUDED.payment_method,
        paid_at = CURRENT_TIMESTAMP;


END;
$$ LANGUAGE plpgsql;
```



| payment_id integer | booking_id integer | amount | currency | status | payment_method | paid_at |
|---|---|---|---|---|---|---|
| 1 | 1 | 300.00 | THB | paid | cash | 2025-12-04 04:25:51.41266 |

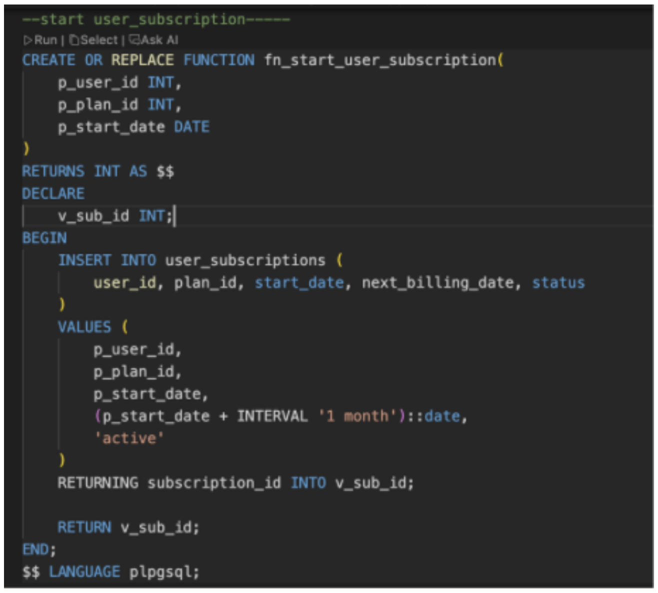
Supawee Meersripong 6481317

## 8) Subscription

**fn_user_subscription**: It automates the billing cycle and records the user ID, plan ID, and start date. The system then calculates the next billing date, so the user is charged correctly each month.

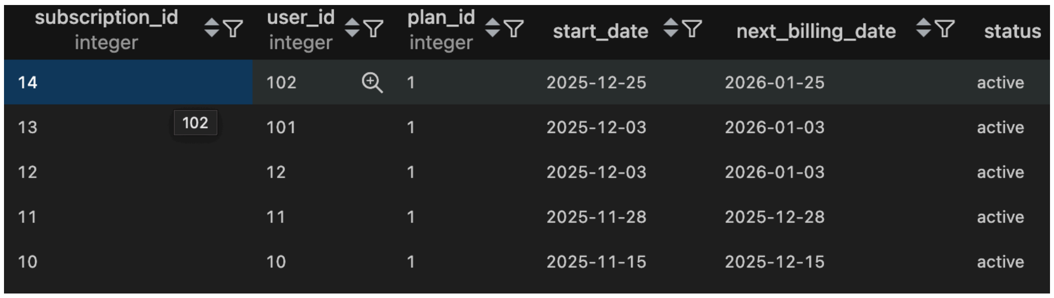
**Input :**

- p_user_id (INT): The ID of the cleaner (e.g., 101).
- p_plan_id (INT): The plan they chose (e.g., 1 for Standard).
- p_start_date (DATE): Today's date (e.g., '2025-12-05')



```sql
--start user_subscription-----
CREATE OR REPLACE FUNCTION fn_start_user_subscription(
    p_user_id INT,
    p_plan_id INT,
    p_start_date DATE
)
RETURNS INT AS $$
DECLARE
    v_sub_id INT;
BEGIN
    INSERT INTO user_subscriptions (
        user_id, plan_id, start_date, next_billing_date, status
    )
    VALUES (
        p_user_id,
        p_plan_id,
        p_start_date,
        (p_start_date + INTERVAL '1 month')::date,
        'active'
    )
    RETURNING subscription_id INTO v_sub_id;

    RETURN v_sub_id;
END;
$$ LANGUAGE plpgsql;
```

Returns the subscription ID (INT), which the app will use to confirm for active plan



| subscription_id integer | user_id integer | plan_id integer | start_date | next_billing_date | status |
|---|---|---|---|---|---|
| 14 | 102 | 1 | 2025-12-25 | 2026-01-25 | active |
| 13 | 101 | 1 | 2025-12-03 | 2026-01-03 | active |
| 12 | 12 | 1 | 2025-12-03 | 2026-01-03 | active |
| 11 | 11 | 1 | 2025-11-28 | 2025-12-28 | active |
| 10 | 10 | 1 | 2025-11-15 | 2025-12-15 | active |

Supawee Meersripong 6481317

External Database URL
postgresql://database_dnjb_user:7S34SOVKbJamvSHg3PwKt3qvDgQlHDet@dpg-d450j3i4d50c73es1iu0-a.singapore-postgres.render.com/database_dnjb (expired on 4/12/2025)

postgresql://db2_rmg6_user:zM61nYvMUBhWK94ppdXnw1s2pYlzSyOI@dpg-d4or0u7pm1nc73cjgp50-a.singapore-postgres.render.com/db2_rmg6 (new one)

Git repository
https://github.com/supaweemeesripong/database_project.git

Supawee Meersripong 6481317