# Randevu App Backend Development Document

- The **Randevu App** project is being developed as a **SaaS platform** to sell subscriptions to hair salons.
- All features in the **Flutter frontend** (Randevu App) must work properly and be fully compatible with the backend.
- For now, there will only be a **standard package** with **no limitations**.
- Subscribed salons will be given multitenant-style domains in the format **“randevu.app/examplesalonname”** (I want this to work using **subfolder logic**, not subdomains).
- Since there may be a mobile application in the future, the system should be built with a **centralized API structure**.
- The server should include a **Super Admin CLI**, capable of handling all super admin operations and viewing reports.
- The **backend should be built using Python and Flask**.
- The **database should be PostgreSQL**.
- Once the backend and database design are complete, we can move on to frontend adjustments.

I'll create a detailed API contract and database schema design based on the Flutter frontend provided. This will serve as a blueprint for the backend implementation using Flask and PostgreSQL.

# API Contract and Database Schema for Randevu App

## Overview

The Randevu App is a SaaS application for hair salons, allowing them to manage appointments, services, hairdressers, and customers. The system follows a multi-tenant architecture where each salon has its own subdomain (`randevu.app/salonnname`).

# Database Schema

## Core Tables

### 1. Salons

```
sql
CREATE TABLE salons (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    name VARCHAR(100) NOT NULL,
    address TEXT,
    phone VARCHAR(20),
    email VARCHAR(100),
    website VARCHAR(200),
    logo_url TEXT,
    created_at TIMESTAMP NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMP,
    domain_prefix VARCHAR(50) UNIQUE NOT NULL,
    owner_id UUID NOT NULL REFERENCES users(id),
    is_active BOOLEAN NOT NULL DEFAULT TRUE
);

CREATE TABLE salon_settings (
    salon_id UUID PRIMARY KEY REFERENCES salons(id) ON DELETE CASCAD
E,
    allow_online_booking BOOLEAN NOT NULL DEFAULT TRUE,
    default_appointment_duration INTEGER NOT NULL DEFAULT 30,
    minimum_notice_time INTEGER NOT NULL DEFAULT 60,
    cancelation_time_limit INTEGER,
    send_sms_reminders BOOLEAN NOT NULL DEFAULT TRUE,
```

```
    reminder_time_before_appointment INTEGER NOT NULL DEFAULT 24,
    require_customer_email BOOLEAN NOT NULL DEFAULT TRUE,
    created_at TIMESTAMP NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMP
);

CREATE TABLE salon_working_schedule (
    salon_id UUID NOT NULL REFERENCES salons(id) ON DELETE CASCADE,
    day_of_week VARCHAR(10) NOT NULL,-- 'monday', 'tuesday', etc.
    is_active BOOLEAN NOT NULL DEFAULT TRUE,
    open_time TIME NOT NULL,
    close_time TIME NOT NULL,
    PRIMARY KEY (salon_id, day_of_week)
);

CREATE TABLE salon_sms_settings (
    salon_id UUID PRIMARY KEY REFERENCES salons(id) ON DELETE CASCAD
E,
    is_active BOOLEAN NOT NULL DEFAULT FALSE,
    api_key TEXT,
    sender_id VARCHAR(100),
    appointment_confirmation_template TEXT,
    appointment_reminder_template TEXT,
    appointment_cancel_template TEXT,
    created_at TIMESTAMP NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMP
);
```

## 2. Users

```
sql
CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
```

```
    username VARCHAR(50) UNIQUE NOT NULL,
    password_hash TEXT NOT NULL,
    email VARCHAR(100),
    user_role VARCHAR(20) NOT NULL,-- 'superAdmin', 'salonOwner', 'hairdres
ser'
    salon_id UUID REFERENCES salons(id) ON DELETE CASCADE,
    is_active BOOLEAN NOT NULL DEFAULT TRUE,
    created_at TIMESTAMP NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMP,
    last_login TIMESTAMP
);
```

## 3. Hairdressers

```
sql
CREATE TABLE hairdressers (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    user_id UUID REFERENCES users(id) ON DELETE SET NULL,
    salon_id UUID NOT NULL REFERENCES salons(id) ON DELETE CASCADE,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100),
    phone VARCHAR(20),
    profile_image TEXT,
    is_active BOOLEAN NOT NULL DEFAULT TRUE,
    created_at TIMESTAMP NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMP
);

CREATE TABLE hairdresser_working_schedule (
    hairdresser_id UUID NOT NULL REFERENCES hairdressers(id) ON DELETE
CASCADE,
    day_of_week VARCHAR(10) NOT NULL,-- 'monday', 'tuesday', etc.
    is_active BOOLEAN NOT NULL DEFAULT TRUE,
```

```
    open_time TIME NOT NULL,
    close_time TIME NOT NULL,
    PRIMARY KEY (hairdresser_id, day_of_week)
);

CREATE TABLE hairdresser_holiday_dates (
    id SERIAL PRIMARY KEY,
    hairdresser_id UUID NOT NULL REFERENCES hairdressers(id) ON DELETE
CASCADE,
    holiday_date DATE NOT NULL,
    created_at TIMESTAMP NOT NULL DEFAULT NOW()
);
```

## 4. Services

```
sql
CREATE TABLE services (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    salon_id UUID NOT NULL REFERENCES salons(id) ON DELETE CASCADE,
    name VARCHAR(100) NOT NULL,
    price DECIMAL(10, 2) NOT NULL,
    duration_minutes INTEGER NOT NULL,
    description TEXT,
    is_active BOOLEAN NOT NULL DEFAULT TRUE,
    created_at TIMESTAMP NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMP
);
```

## 5. Hairdresser Services

```
sql
```

```
CREATE TABLE hairdresser_services (
    hairdresser_id UUID NOT NULL REFERENCES hairdressers(id) ON DELETE
CASCADE,
    service_id UUID NOT NULL REFERENCES services(id) ON DELETE CASCAD
E,
    PRIMARY KEY (hairdresser_id, service_id)
);
```

## 6. Customers

```
sql
CREATE TABLE customers (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    salon_id UUID NOT NULL REFERENCES salons(id) ON DELETE CASCADE,
    name VARCHAR(100) NOT NULL,
    phone VARCHAR(20) NOT NULL,
    email VARCHAR(100),
    created_at TIMESTAMP NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMP,
    UNIQUE (salon_id, phone)
);
```

## 7. Appointments

```
sql
CREATE TABLE appointments (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    salon_id UUID NOT NULL REFERENCES salons(id) ON DELETE CASCADE,
    customer_id UUID NOT NULL REFERENCES customers(id) ON DELETE CAS
CADE,
    hairdresser_id UUID NOT NULL REFERENCES hairdressers(id) ON DELETE
```

```
CASCADE,
    date_time TIMESTAMP NOT NULL,
    status VARCHAR(20) NOT NULL DEFAULT 'pending',-- 'pending', 'confirme
d', 'completed', 'cancelled', 'rejected'
    customer_note TEXT,
    appointment_code VARCHAR(20) UNIQUE NOT NULL,
    created_at TIMESTAMP NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMP
);

CREATE TABLE appointment_services (
    appointment_id UUID NOT NULL REFERENCES appointments(id) ON DELET
E CASCADE,
    service_id UUID NOT NULL REFERENCES services(id) ON DELETE CASCAD
E,
    price DECIMAL(10, 2) NOT NULL,-- Price at the time of appointment
    PRIMARY KEY (appointment_id, service_id)
);
```

## 8. Authentication

```
sql
CREATE TABLE refresh_tokens (
    id SERIAL PRIMARY KEY,
    user_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    token TEXT NOT NULL,
    expires_at TIMESTAMP NOT NULL,
    created_at TIMESTAMP NOT NULL DEFAULT NOW()
);

CREATE TABLE password_reset_tokens (
    id SERIAL PRIMARY KEY,
    user_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,
```

```
    token TEXT NOT NULL,
    expires_at TIMESTAMP NOT NULL,
    created_at TIMESTAMP NOT NULL DEFAULT NOW()
);
```

# API Endpoints

## Authentication API

## 1. Login

- **URL**: `/api/auth/login`
- **Method**: `POST`
- **Request Body**:

```
json
{
  "username": "string",
  "password": "string"
}
```

- **Response**:

```
json
{
  "success": true,
  "data": {
    "user_id": "uuid-string",
    "user_name": "string",
    "user_role": "string",
    "token": "jwt-token-string",
```

```
    "salon_id": "uuid-string"
  }
}
```

## 2. Refresh Token

- **URL**: `/api/auth/refresh`
- **Method**: `POST`
- **Request Body**:

```
json
{
  "refresh_token": "string"
}
```

- **Response**:

```
json
{
  "success": true,
  "data": {
    "token": "jwt-token-string",
    "refresh_token": "refresh-token-string"
  }
}
```

## 3. Logout

- **URL**: `/api/auth/logout`
- **Method**: `POST`

- **Headers**: Authorization: Bearer {token}
- **Response**:

```
json
{
  "success": true
}
```

## Salon API

## 1. Get Salon

- **URL**: `/api/salons/{salon_id}`
- **Method**: `GET`
- **Headers**: Authorization: Bearer {token}
- **Response**:

```
json
{
  "success": true,
  "data": {
    "id": "uuid-string",
    "name": "string",
    "address": "string",
    "phone": "string",
    "email": "string",
    "website": "string",
    "logo_url": "string",
    "owner_id": "uuid-string",
    "settings": {
```

```
    "allow_online_booking": true,
    "default_appointment_duration": 30,
    "minimum_notice_time": 60,
    "cancelation_time_limit": 24,
    "send_sms_reminders": true,
    "reminder_time_before_appointment": 24,
    "require_customer_email": true
  },
  "sms_settings": {
    "is_active": false,
    "api_key": "string",
    "sender_id": "string",
    "appointment_confirmation_template": "string",
    "appointment_reminder_template": "string",
    "appointment_cancel_template": "string"
  },
  "working_schedule": {
    "monday": {"is_active": true, "open_time": "09:00", "close_time": "18:0
0"},
    "tuesday": {"is_active": true, "open_time": "09:00", "close_time": "18:0
0"},
    "wednesday": {"is_active": true, "open_time": "09:00", "close_time": "1
8:00"},
    "thursday": {"is_active": true, "open_time": "09:00", "close_time": "18:0
0"},
    "friday": {"is_active": true, "open_time": "09:00", "close_time": "18:0
0"},
    "saturday": {"is_active": true, "open_time": "09:00", "close_time": "16:0
0"},
    "sunday": {"is_active": false, "open_time": "09:00", "close_time": "18:0
0"}
  }
 }
}
```

## 2. Update Salon

- **URL**: `/api/salons/{salon_id}`
- **Method**: `PUT`
- **Headers**: Authorization: Bearer {token}
- **Request Body**:

```
json
{
  "name": "string",
  "address": "string",
  "phone": "string",
  "email": "string",
  "website": "string"
}
```

- **Response**:

```
json
{
  "success": true,
  "data": {
    "id": "uuid-string",
    "name": "string",
    "address": "string",
    "phone": "string",
    "email": "string",
    "website": "string"
  }
}
```

## 3. Update Salon Settings

- **URL**: `/api/salons/{salon_id}/settings`
- **Method**: `PUT`
- **Headers**: Authorization: Bearer {token}
- **Request Body**:

```
json
{
  "allow_online_booking": true,
  "default_appointment_duration": 30,
  "minimum_notice_time": 60,
  "cancelation_time_limit": 24,
  "send_sms_reminders": true,
  "reminder_time_before_appointment": 24,
  "require_customer_email": true
}
```

- **Response**:

```
json
{
  "success": true
}
```

## 4. Update Salon Working Schedule

- **URL**: `/api/salons/{salon_id}/schedule`
- **Method**: `PUT`
- **Headers**: Authorization: Bearer {token}

- **Request Body**:

```
json
{
  "working_schedule": {
    "monday": {"is_active": true, "open_time": "09:00", "close_time": "18:0
0"},
    "tuesday": {"is_active": true, "open_time": "09:00", "close_time": "18:0
0"},
    "wednesday": {"is_active": true, "open_time": "09:00", "close_time": "1
8:00"},
    "thursday": {"is_active": true, "open_time": "09:00", "close_time": "18:0
0"},
    "friday": {"is_active": true, "open_time": "09:00", "close_time": "18:00"},
    "saturday": {"is_active": true, "open_time": "09:00", "close_time": "16:0
0"},
    "sunday": {"is_active": false, "open_time": "09:00", "close_time": "18:0
0"}
  }
}
```

- **Response**:

```
json
{
  "success": true
}
```

## 5. Update SMS Settings

- **URL**: `/api/salons/{salon_id}/sms-settings`

- **Method**: `PUT`
- **Headers**: Authorization: Bearer {token}
- **Request Body**:

```
json
{
  "is_active": true,
  "api_key": "string",
  "sender_id": "string",
  "appointment_confirmation_template": "string",
  "appointment_reminder_template": "string",
  "appointment_cancel_template": "string"
}
```

- **Response**:

```
json
{
  "success": true
}
```

## Hairdresser API

## 1. Get Hairdressers

- **URL**: `/api/hairdressers`
- **Method**: `GET`
- **Headers**: Authorization: Bearer {token}
- **Query Parameters**:

- `salon_id` (optional): Filter by salon
- `is_active` (optional): Filter by active status

- **Response**:

```
json
{
  "success": true,
  "data": [
    {
      "id": "uuid-string",
      "name": "string",
      "email": "string",
      "phone": "string",
      "profile_image": "string",
      "username": "string",
      "is_active": true,
      "salon_id": "uuid-string",
      "working_schedule": {
        "monday": {"is_active": true, "open_time": "09:00", "close_time": "18:
00"},
        "tuesday": {"is_active": true, "open_time": "09:00", "close_time": "18:
00"},
        "wednesday": {"is_active": true, "open_time": "09:00", "close_time":
"18:00"},
        "thursday": {"is_active": true, "open_time": "09:00", "close_time": "18:
00"},
        "friday": {"is_active": true, "open_time": "09:00", "close_time": "18:0
0"},
        "saturday": {"is_active": true, "open_time": "09:00", "close_time": "16:
00"},
        "sunday": {"is_active": false, "open_time": "09:00", "close_time": "18:
00"}
      },
```

```
    "service_ids": ["uuid-string", "uuid-string"],
    "holiday_dates": ["2025-05-01", "2025-05-15"]
   }
  ]
}
```

## 2. Get Single Hairdresser

- **URL**: `/api/hairdressers/{hairdresser_id}`
- **Method**: `GET`
- **Headers**: Authorization: Bearer {token}
- **Response**:

```
json
{
  "success": true,
  "data": {
    "id": "uuid-string",
    "name": "string",
    "email": "string",
    "phone": "string",
    "profile_image": "string",
    "username": "string",
    "is_active": true,
    "salon_id": "uuid-string",
    "working_schedule": {
      "monday": {"is_active": true, "open_time": "09:00", "close_time": "18:0
0"},
      "tuesday": {"is_active": true, "open_time": "09:00", "close_time": "18:0
0"},
      "wednesday": {"is_active": true, "open_time": "09:00", "close_time": "1
8:00"},
```

```
    "thursday": {"is_active": true, "open_time": "09:00", "close_time": "18:0
0"},
    "friday": {"is_active": true, "open_time": "09:00", "close_time": "18:0
0"},
    "saturday": {"is_active": true, "open_time": "09:00", "close_time": "16:0
0"},
    "sunday": {"is_active": false, "open_time": "09:00", "close_time": "18:0
0"}
   },
   "service_ids": ["uuid-string", "uuid-string"],
   "holiday_dates": ["2025-05-01", "2025-05-15"]
 }
}
```

## 3. Create Hairdresser

- **URL**: `/api/hairdressers`
- **Method**: `POST`
- **Headers**: Authorization: Bearer {token}
- **Request Body**:

```
json
{
  "name": "string",
  "email": "string",
  "phone": "string",
  "salon_id": "uuid-string",
  "username": "string",
  "password": "string",
  "is_active": true,
  "working_schedule": {
    "monday": {"is_active": true, "open_time": "09:00", "close_time": "18:0
```

```
0"},
    "tuesday": {"is_active": true, "open_time": "09:00", "close_time": "18:0
0"},
    "wednesday": {"is_active": true, "open_time": "09:00", "close_time": "1
8:00"},
    "thursday": {"is_active": true, "open_time": "09:00", "close_time": "18:0
0"},
    "friday": {"is_active": true, "open_time": "09:00", "close_time": "18:00"},
    "saturday": {"is_active": true, "open_time": "09:00", "close_time": "16:0
0"},
    "sunday": {"is_active": false, "open_time": "09:00", "close_time": "18:0
0"}
  }
}
```

- **Response**:

```
json
{
  "success": true,
  "data": {
    "id": "uuid-string",
    "name": "string",
    "email": "string",
    "phone": "string",
    "is_active": true,
    "salon_id": "uuid-string"
  }
}
```

## 4. Update Hairdresser

- **URL**: `/api/hairdressers/{hairdresser_id}`

- **Method**: `PUT`
- **Headers**: Authorization: Bearer {token}
- **Request Body**:

```
json
{
  "name": "string",
  "email": "string",
  "phone": "string",
  "is_active": true,
  "password": "string"// Optional
}
```

- **Response**:

```
json
{
  "success": true
}
```

## 5. Update Hairdresser Working Hours

- **URL**: `/api/hairdressers/{hairdresser_id}/working-hours`
- **Method**: `PUT`
- **Headers**: Authorization: Bearer {token}
- **Request Body**:

```
json
```

```
{
  "working_schedule": {
    "monday": {"is_active": true, "open_time": "09:00", "close_time": "18:0
0"},
    "tuesday": {"is_active": true, "open_time": "09:00", "close_time": "18:0
0"},
    "wednesday": {"is_active": true, "open_time": "09:00", "close_time": "1
8:00"},
    "thursday": {"is_active": true, "open_time": "09:00", "close_time": "18:0
0"},
    "friday": {"is_active": true, "open_time": "09:00", "close_time": "18:00"},
    "saturday": {"is_active": true, "open_time": "09:00", "close_time": "16:0
0"},
    "sunday": {"is_active": false, "open_time": "09:00", "close_time": "18:0
0"}
  }
}
```

- **Response**:

```
json
{
  "success": true
}
```

## 6. Update Hairdresser Holiday Dates

- **URL**: `/api/hairdressers/{hairdresser_id}/holiday-dates`
- **Method**: `PUT`
- **Headers**: Authorization: Bearer {token}
- **Request Body**:

```
json
{
  "holiday_dates": ["2025-05-01", "2025-05-15"]
}
```

- **Response**:

```
json
{
  "success": true
}
```

## 7. Update Hairdresser Services

- **URL**: `/api/hairdressers/{hairdresser_id}/services`
- **Method**: `PUT`
- **Headers**: Authorization: Bearer {token}
- **Request Body**:

```
json
{
  "service_ids": ["uuid-string", "uuid-string"]
}
```

- **Response**:

```
json
{
  "success": true
}
```

## 8. Delete Hairdresser

- **URL**: `/api/hairdressers/{hairdresser_id}`
- **Method**: `DELETE`
- **Headers**: Authorization: Bearer {token}
- **Response**:

```
json
{
  "success": true
}
```

## Service API

## 1. Get Services

- **URL**: `/api/services`
- **Method**: `GET`
- **Headers**: Authorization: Bearer {token}
- **Query Parameters**:
  - `salon_id` (optional): Filter by salon
  - `is_active` (optional): Filter by active status
- **Response**:

```
json
{
  "success": true,
  "data": [
    {
      "id": "uuid-string",
      "name": "string",
      "price": 100.00,
      "duration_minutes": 30,
      "description": "string",
      "is_active": true,
      "salon_id": "uuid-string"
    }
  ]
}
```

## 2. Get Single Service

- **URL**: `/api/services/{service_id}`
- **Method**: `GET`
- **Headers**: Authorization: Bearer {token}
- **Response**:

```
json
{
  "success": true,
  "data": {
    "id": "uuid-string",
    "name": "string",
    "price": 100.00,
```

```
    "duration_minutes": 30,
    "description": "string",
    "is_active": true,
    "salon_id": "uuid-string"
  }
}
```

## 3. Create Service

- **URL**: `/api/services`
- **Method**: `POST`
- **Headers**: Authorization: Bearer {token}
- **Request Body**:

```
json
{
  "name": "string",
  "price": 100.00,
  "duration_minutes": 30,
  "description": "string",
  "is_active": true,
  "salon_id": "uuid-string"
}
```

- **Response**:

```
json
{
  "success": true,
  "data": {
```

```
    "id": "uuid-string",
    "name": "string",
    "price": 100.00,
    "duration_minutes": 30,
    "description": "string",
    "is_active": true,
    "salon_id": "uuid-string"
  }
}
```

## 4. Update Service

- **URL**: `/api/services/{service_id}`
- **Method**: `PUT`
- **Headers**: Authorization: Bearer {token}
- **Request Body**:

```
json
{
  "name": "string",
  "price": 100.00,
  "duration_minutes": 30,
  "description": "string",
  "is_active": true
}
```

- **Response**:

```
json
{
```

```
  "success": true
}
```

## 5. Delete Service

- **URL**: `/api/services/{service_id}`
- **Method**: `DELETE`
- **Headers**: Authorization: Bearer {token}
- **Response**:

```
json
{
  "success": true
}
```

# Appointment API

## 1. Get Appointments

- **URL**: `/api/appointments`
- **Method**: `GET`
- **Headers**: Authorization: Bearer {token}
- **Query Parameters**:
  - `salon_id` (optional): Filter by salon
  - `hairdresser_id` (optional): Filter by hairdresser
  - `start_date` (optional): Filter by start date
  - `end_date` (optional): Filter by end date
  - `status` (optional): Filter by status
- **Response**:

```
json
{
  "success": true,
  "data": [
    {
      "id": "uuid-string",
      "customer_id": "uuid-string",
      "customer_name": "string",
      "customer_phone": "string",
      "customer_email": "string",
      "customer_note": "string",
      "hairdresser_id": "uuid-string",
      "hairdresser_name": "string",
      "date_time": "2025-04-15T14:30:00Z",
      "service_ids": ["uuid-string", "uuid-string"],
      "service_names": ["string", "string"],
      "total_price": 250.00,
      "status": "pending",
      "appointment_code": "ABC123",
      "created_at": "2025-04-01T10:00:00Z",
      "updated_at": "2025-04-01T10:00:00Z",
      "salon_id": "uuid-string"
    }
  ]
}
```

## 2. Get Single Appointment

- **URL**: `/api/appointments/{appointment_id}`
- **Method**: `GET`
- **Headers**: Authorization: Bearer {token}
- **Response**:

```
json
{
  "success": true,
  "data": {
    "id": "uuid-string",
    "customer_id": "uuid-string",
    "customer_name": "string",
    "customer_phone": "string",
    "customer_email": "string",
    "customer_note": "string",
    "hairdresser_id": "uuid-string",
    "hairdresser_name": "string",
    "date_time": "2025-04-15T14:30:00Z",
    "service_ids": ["uuid-string", "uuid-string"],
    "service_names": ["string", "string"],
    "total_price": 250.00,
    "status": "pending",
    "appointment_code": "ABC123",
    "created_at": "2025-04-01T10:00:00Z",
    "updated_at": "2025-04-01T10:00:00Z",
    "salon_id": "uuid-string"
  }
}
```

### 3. Get Appointment by Code

- **URL**: `/api/appointments/code/{code}`
- **Method**: `GET`
- **Response**:

```
json
{
  "success": true,
  "data": {
    "id": "uuid-string",
    "customer_id": "uuid-string",
    "customer_name": "string",
    "customer_phone": "string",
    "customer_email": "string",
    "customer_note": "string",
    "hairdresser_id": "uuid-string",
    "hairdresser_name": "string",
    "date_time": "2025-04-15T14:30:00Z",
    "service_ids": ["uuid-string", "uuid-string"],
    "service_names": ["string", "string"],
    "total_price": 250.00,
    "status": "pending",
    "appointment_code": "ABC123",
    "created_at": "2025-04-01T10:00:00Z",
    "updated_at": "2025-04-01T10:00:00Z",
    "salon_id": "uuid-string"
  }
}
```

## 4. Create Appointment

- **URL**: `/api/appointments`
- **Method**: `POST`
- **Request Body**:

```
json
{
  "customer_name": "string",
```

```
  "customer_phone": "string",
  "customer_email": "string",
  "customer_note": "string",
  "hairdresser_id": "uuid-string",
  "service_ids": ["uuid-string", "uuid-string"],
  "date_time": "2025-04-15T14:30:00Z",
  "salon_id": "uuid-string"
}
```

- **Response**:

```
json
{
  "success": true,
  "data": {
    "id": "uuid-string",
    "customer_name": "string",
    "customer_phone": "string",
    "customer_email": "string",
    "hairdresser_name": "string",
    "date_time": "2025-04-15T14:30:00Z",
    "service_names": ["string", "string"],
    "total_price": 250.00,
    "status": "pending",
    "appointment_code": "ABC123"
  }
}
```

## 5. Update Appointment

- **URL**: `/api/appointments/{appointment_id}`
- **Method**: `PUT`
- **Headers**: Authorization: Bearer {token}

- **Request Body**:

```
json
{
  "customer_name": "string",
  "customer_phone": "string",
  "customer_email": "string",
  "customer_note": "string",
  "hairdresser_id": "uuid-string",
  "service_ids": ["uuid-string", "uuid-string"],
  "date_time": "2025-04-15T14:30:00Z",
  "status": "string"
}
```

- **Response**:

```
json
{
  "success": true
}
```

## 6. Update Appointment Status

- **URL**: `/api/appointments/{appointment_id}/status`
- **Method**: `PUT`
- **Headers**: Authorization: Bearer {token}
- **Request Body**:

```
json
{
  "status": "string"// "pending", "confirmed", "completed", "cancelled", "rej
ected"
}
```

- **Response**:

```
json
{
  "success": true
}
```

## 7. Delete Appointment

- **URL**: `/api/appointments/{appointment_id}`
- **Method**: `DELETE`
- **Headers**: Authorization: Bearer {token}
- **Response**:

```
json
{
  "success": true
}
```

# Reports API (Salon Owner)

## 1. Revenue Report

- **URL**: `/api/reports/revenue`
- **Method**: `GET`
- **Headers**: Authorization: Bearer {token}
- **Query Parameters**:
  - `salon_id`: ID of the salon
  - `start_date`: Start date (YYYY-MM-DD)
  - `end_date`: End date (YYYY-MM-DD)
- **Response**:

```
json
{
  "success": true,
  "data": {
    "totalRevenue": 5000.00,
    "revenueByHairdresser": [
      {
        "hairdresser_id": "uuid-string",
        "hairdresser_name": "string",
        "revenue": 2500.00,
        "percentage": 50.0
      }
    ],
    "revenueByService": [
      {
        "service_id": "uuid-string",
        "service_name": "string",
        "revenue": 1500.00,
        "percentage": 30.0
      }
    ],
    "revenueByPeriod": [
```

```
    {
      "period": "2025-04",// YYYY-MM
      "revenue": 2500.00
    }
   ]
  }
}
```

## 2. Appointment Report

- **URL**: `/api/reports/appointments`
- **Method**: `GET`
- **Headers**: Authorization: Bearer {token}
- **Query Parameters**:
  - `salon_id`: ID of the salon
  - `start_date`: Start date (YYYY-MM-DD)
  - `end_date`: End date (YYYY-MM-DD)
- **Response**:

```
json
{
  "success": true,
  "data": {
    "totalAppointments": 100,
    "statusCounts": {
      "pending": 20,
      "confirmed": 30,
      "completed": 40,
      "cancelled": 5,
      "rejected": 5
    },
```

```
    "appointmentsByHairdresser": [
      {
        "hairdresser_id": "uuid-string",
        "hairdresser_name": "string",
        "count": 50,
        "percentage": 50.0
      }
    ],
    "appointmentsByHour": [
      {
        "hour": 9,
        "count": 10
      }
    ],
    "appointmentsByDay": [
      {
        "day": 1,// 1-7 (Monday-Sunday)
        "count": 15
      }
    ]
  }
}
```

## 3. Customer Report

- **URL**: `/api/reports/customers`
- **Method**: `GET`
- **Headers**: Authorization: Bearer {token}
- **Query Parameters**:
  - `salon_id`: ID of the salon
  - `start_date`: Start date (YYYY-MM-DD)
  - `end_date`: End date (YYYY-MM-DD)
- **Response**:

```
json
{
  "success": true,
  "data": {
    "totalCustomers": 50,
    "newCustomers": 10,
    "topSpendingCustomers": [
      {
        "customer_id": "uuid-string",
        "name": "string",
        "phone": "string",
        "email": "string",
        "totalSpent": 1000.00,
        "appointmentCount": 5,
        "lastAppointment": "2025-04-10T14:30:00Z"
      }
    ],
    "frequentCustomers": [
      {
        "customer_id": "uuid-string",
        "name": "string",
        "phone": "string",
        "email": "string",
        "appointmentCount": 10,
        "totalSpent": 800.00,
        "lastAppointment": "2025-04-10T14:30:00Z"
      }
    ]
  }
}
```

## 4. Service Report

- **URL**: `/api/reports/services`
- **Method**: `GET`
- **Headers**: Authorization: Bearer {token}
- **Query Parameters**:
  - `salon_id`: ID of the salon
  - `start_date`: Start date (YYYY-MM-DD)
  - `end_date`: End date (YYYY-MM-DD)
- **Response**:

```
json
{
  "success": true,
  "data": {
    "popularServices": [
      {
        "service_id": "uuid-string",
        "service_name": "string",
        "count": 30,
        "percentage": 40.0,
        "revenue": 1500.00
      }
    ],
    "averageServicesPerAppointment": 2.3
  }
}
```

## Super Admin API

## 1. Get All Salons

- **URL**: `/api/admin/salons`

- **Method**: `GET`
- **Headers**: Authorization: Bearer {token}
- **Query Parameters**:
  - `is_active` (optional): Filter by active status
  - `page` (optional): Page number
  - `per_page` (optional): Items per page
- **Response**:

```
json
{
  "success": true,
  "data": {
    "salons": [
      {
        "id": "uuid-string",
        "name": "string",
        "domain_prefix": "string",
        "owner_id": "uuid-string",
        "owner_name": "string",
        "is_active": true,
        "created_at": "2025-01-01T00:00:00Z"
      }
    ],
    "pagination": {
      "total": 50,
      "per_page": 10,
      "current_page": 1,
      "last_page": 5
    }
```

```
  }
}
```

## 2. Create Salon

- **URL**: `/api/admin/salons`
- **Method**: `POST`
- **Headers**: Authorization: Bearer {token}
- **Request Body**:

```
json
{
  "name": "string",
  "domain_prefix": "string",
  "owner": {
    "username": "string",
    "password": "string",
    "email": "string",
    "name": "string"
  }
}
```

- **Response**:

```
json
{
  "success": true,
  "data": {
    "id": "uuid-string",
    "name": "string",
```

```
    "domain_prefix": "string",
    "owner_id": "uuid-string"
  }
}
```

## 3. Update Salon Status

- **URL**: `/api/admin/salons/{salon_id}/status`
- **Method**: `PUT`
- **Headers**: Authorization: Bearer {token}
- **Request Body**:

```
json
{
  "is_active": true
}
```

- **Response**:

```
json
{
  "success": true
}
```

## 4. Get System Stats

- **URL**: `/api/admin/stats`
- **Method**: `GET`
- **Headers**: Authorization: Bearer {token}

- **Response**:

```
json
{
  "success": true,
  "data": {
    "totalSalons": 50,
    "activeSalons": 45,
    "totalUsers": 150,
    "totalAppointments": 5000,
    "appointmentsToday": 100,
    "revenueStats": {
      "totalRevenue": 50000.00,
      "revenueThisMonth": 5000.00,
      "revenueLastMonth": 4500.00
    },
    "topSalons": [
      {
        "id": "uuid-string",
        "name": "string",
        "appointmentCount": 500,
        "revenue": 10000.00
      }
    ]
  }
}
```

# Additional Implementation Details

## Multi-tenant Architecture

The application follows a multi-tenant architecture with a shared database approach:

1. **Tenant Identification**:
   - Each salon has a unique `domain_prefix` that serves as the tenant identifier
   - This is used in URLs: `randevu.app/{domain_prefix}`
2. **Middleware Implementation**:
   - A middleware will extract the tenant identifier from the request URL
   - It will validate the tenant exists and is active
   - It will set the salon/tenant context for the request

```
python
# Example middleware pseudocode
def tenant_middleware(request, next_handler):
    domain_prefix = extract_domain_prefix(request.url)

# Skip for admin routes
    if request.path.startswith('/api/admin'):
        return next_handler(request)

# Skip for public routes like auth
    if request.path in ['/api/auth/login', '/api/auth/refresh']:
        return next_handler(request)

# Get salon by domain_prefix
    salon = get_salon_by_domain_prefix(domain_prefix)
    if not salon:
        return error_response(404, "Salon not found")

    if not salon.is_active:
        return error_response(403, "Salon is inactive")

# Set salon context in request
    request.salon_context = salon
```

```
    return next_handler(request)
```

1. **Query Scoping**:
    - All queries to tenant-specific data will be automatically scoped to the current salon_id
    - This prevents data leakage between tenants

```
python
# Example query scope pseudocode
class HairdresserService:
    def get_hairdressers(self, salon_id, **filters):
# Always filter by salon_id
        filters['salon_id'] = salon_id
        return db.query(Hairdresser).filter_by(**filters).all()
```

## Authentication & Authorization

1. **JWT Authentication**:
    - JWT tokens will contain user ID, role, and salon ID
    - Tokens will expire after 1 hour
    - Refresh tokens will be stored in the database and valid for 30 days
2. **Role-based Access Control**:
    - Three main roles: `superAdmin`, `salonOwner`, `hairdresser`
    - Permission checks will be implemented at the controller level

```
python
# Example role check pseudocode
```

```
def require_role(roles):
    def decorator(function):
        def wrapper(request, *args, **kwargs):
            user_role = get_user_role_from_token(request)
            if user_role not in roles:
                return error_response(403, "Insufficient permissions")
            return function(request, *args, **kwargs)
        return wrapper
    return decorator

# Usage
@require_role(['salonOwner', 'superAdmin'])
def update_salon_settings(request, salon_id):
# Implementation
```

## Appointment Scheduling Logic

1. **Availability Checking**:
    - Check if the hairdresser is working on the selected day
    - Check for overlapping appointments
    - Check for holiday dates
    - Respect minimum notice time settings

```
python
# Example availability check pseudocode
def check_availability(salon_id, hairdresser_id, date_time, duration_minutes):
# Check if the date is in the future
    if date_time <= datetime.now():
        return False, "Cannot book appointments in the past"

# Check minimum notice time
    salon = get_salon(salon_id)
```

```
    if datetime.now() + timedelta(minutes=salon.settings.minimum_notice_time)
> date_time:
        return False, "Appointment time does not meet minimum notice requirem
ent"

# Check hairdresser working schedule
    day_of_week = date_time.strftime('%A').lower()
    working_hours = get_hairdresser_working_hours(hairdresser_id, day_of_we
ek)

    if not working_hours or not working_hours.is_active:
        return False, "Hairdresser is not working on this day"

# Parse working hours
    start_time = parse_time(working_hours.open_time)
    end_time = parse_time(working_hours.close_time)

# Check if appointment time is within working hours
    appointment_time = date_time.time()
    if appointment_time < start_time or appointment_time > end_time:
        return False, "Appointment time is outside working hours"

# Check if on holiday
    if is_holiday(hairdresser_id, date_time.date()):
        return False, "Hairdresser is on holiday on this date"

# Check for overlapping appointments
    appointment_end_time = date_time + timedelta(minutes=duration_minutes)
    overlapping = get_overlapping_appointments(
        hairdresser_id,
        date_time,
        appointment_end_time
    )

    if overlapping:
        return False, "Time slot is already booked"
```

```
    return True, "Time slot is available"
```

1. **Appointment Code Generation**:
    - Generate unique and customer-friendly codes for appointments
    - Format: 2 letters + 4 digits (e.g., AB1234)

```
python
def generate_appointment_code():
    letters = ''.join(random.choices(string.ascii_uppercase, k=2))
    numbers = ''.join(random.choices(string.digits, k=4))
    code = letters + numbers

# Ensure uniqueness
    while is_code_exists(code):
        letters = ''.join(random.choices(string.ascii_uppercase, k=2))
        numbers = ''.join(random.choices(string.digits, k=4))
        code = letters + numbers

    return code
```

## SMS Notifications

The backend will integrate with SMS providers to send notifications:

1. **SMS Gateway Integration**:
    - Support for multiple SMS providers
    - Template-based message composition
2. **Notification Types**:
    - Appointment confirmation
    - Appointment reminder

- Appointment cancellation

```
python
# Example SMS sending pseudocode
def send_appointment_sms(appointment_id, notification_type):
    appointment = get_appointment(appointment_id)
    salon = get_salon(appointment.salon_id)

    if not salon.sms_settings.is_active:
        return False, "SMS notifications are disabled"

# Get appropriate template
    if notification_type == 'confirmation':
        template = salon.sms_settings.appointment_confirmation_template
    elif notification_type == 'reminder':
        template = salon.sms_settings.appointment_reminder_template
    elif notification_type == 'cancellation':
        template = salon.sms_settings.appointment_cancel_template
    else:
        return False, "Invalid notification type"

# Format template with dynamic data
    message = format_template(template, {
        'ad': appointment.customer_name,
        'tarih': format_date(appointment.date_time),
        'saat': format_time(appointment.date_time),
        'kuafor': appointment.hairdresser_name,
        'salon': salon.name
    })

# Send SMS via provider
    result = sms_provider.send(
        api_key=salon.sms_settings.api_key,
```

```
        sender_id=salon.sms_settings.sender_id,
        to=appointment.customer_phone,
        message=message
    )

    return result.success, result.message
```

## Error Handling and Validation

1. **Centralized Error Handling**:
    - Consistent error response format
    - Detailed error messages in development, generic messages in production

```
python
def error_response(status_code, message, details=None):
    response = {
        'success': False,
        'message': message
    }

    if details and app.config['DEBUG']:
        response['details'] = details

    return jsonify(response), status_code
```

1. **Input Validation**:
    - Comprehensive validation schemas for all inputs
    - Validation errors return appropriate messages

```
python
```

```
# Example validation pseudocode
create_appointment_schema = {
    'customer_name': {'type': 'string', 'required': True, 'minlength': 2, 'maxlength': 100},
    'customer_phone': {'type': 'string', 'required': True, 'regex': '^[0-9]{10,11}$'},
    'customer_email': {'type': 'string', 'required': True, 'regex': r'^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$'},
    'hairdresser_id': {'type': 'uuid', 'required': True},
    'service_ids': {'type': 'list', 'required': True, 'schema': {'type': 'uuid'}},
    'date_time': {'type': 'datetime', 'required': True},
    'salon_id': {'type': 'uuid', 'required': True}
}

def validate_input(data, schema):
    validator = Validator(schema)
    if not validator.validate(data):
        return False, validator.errors
    return True, None
```

# Database Implementation Considerations

## Indexing Strategy

To optimize performance, the following indexes should be created:

```
sql
-- Salon lookup by domain prefix
CREATE INDEX idx_salons_domain_prefix ON salons(domain_prefix);

-- User lookup by username (for login)
CREATE INDEX idx_users_username ON users(username);
```

```
-- Filter appointments by salon, hairdresser, date range
CREATE INDEX idx_appointments_salon_id ON appointments(salon_id);
CREATE INDEX idx_appointments_hairdresser_id ON appointments(hairdresse
r_id);
CREATE INDEX idx_appointments_date_time ON appointments(date_time);
CREATE INDEX idx_appointments_status ON appointments(status);

-- Quick appointment lookup by code
CREATE INDEX idx_appointments_appointment_code ON appointments(appoi
ntment_code);

-- Combined index for reporting queries
CREATE INDEX idx_appointments_salon_date ON appointments(salon_id, date
_time);
CREATE INDEX idx_appointments_hairdresser_date ON appointments(hairdres
ser_id, date_time);
```

## Transaction Management

Ensure data integrity with proper transaction handling:

```
python
# Example transaction pseudocode
def create_appointment(data):
    with db.transaction():
# Find or create customer
        customer = find_customer_by_phone(data['customer_phone'], data['salo
n_id'])
        if not customer:
            customer = create_customer({
                'salon_id': data['salon_id'],
                'name': data['customer_name'],
                'phone': data['customer_phone'],
                'email': data['customer_email']
```

```
            })

    # Calculate total price
        services = get_services_by_ids(data['service_ids'])
        total_price = sum(service.price for service in services)

    # Create appointment
        appointment_code = generate_appointment_code()
        appointment = create_appointment_record({
            'salon_id': data['salon_id'],
            'customer_id': customer.id,
            'hairdresser_id': data['hairdresser_id'],
            'date_time': data['date_time'],
            'status': 'pending',
            'customer_note': data.get('customer_note', ''),
            'appointment_code': appointment_code,
            'total_price': total_price
        })

    # Create appointment-service relations with current prices
        for service in services:
            create_appointment_service({
                'appointment_id': appointment.id,
                'service_id': service.id,
                'price': service.price
            })

    # Send notification
        if should_send_notification(data['salon_id']):
            send_appointment_sms(appointment.id, 'confirmation')

        return appointment
```

## Security Considerations

1. **Password Storage**:
    - Use bcrypt for password hashing
    - Implement password complexity requirements
2. **HTTPS**:
    - All API endpoints must be served over HTTPS
    - HTTP Strict Transport Security (HSTS) headers
3. **Rate Limiting**:
    - Implement rate limiting for authentication endpoints
    - Apply per-tenant rate limits to prevent abuse
4. **Data Validation**:
    - Sanitize all inputs
    - Escape HTML in user-provided content
    - Validate UUIDs for database lookups

# Deployment Architecture

1. **Containerization**:
    - Docker for application components
    - Docker Compose for local development
    - Kubernetes for production
2. **Database**:
    - PostgreSQL with replication
    - Regular backups
    - Use connection pooling
3. **Caching**:
    - Redis for caching
    - Cache frequently accessed data (salon settings, working hours)

4. **Monitoring**:
    - Application metrics using Prometheus
    - Log aggregation with ELK stack
    - Error tracking

## CLI for Super Admin

Create a command-line interface for super admin operations:

```
python
@click.group()
def admin_cli():
    """CLI for Randevu.app Super Admin operations."""
    pass

@admin_cli.command()
@click.option('--username', required=True, help='Admin username')
@click.option('--password', required=True, help='Admin password')
@click.option('--email', required=True, help='Admin email')
def create_super_admin(username, password, email):
    """Create a new super admin user."""
# Implementation

@admin_cli.command()
@click.option('--salon-name', required=True, help='Salon name')
@click.option('--domain-prefix', required=True, help='Domain prefix')
@click.option('--owner-username', required=True, help='Owner username')
@click.option('--owner-password', required=True, help='Owner password')
@click.option('--owner-email', required=True, help='Owner email')
def create_salon(salon_name, domain_prefix, owner_username, owner_passw
ord, owner_email):
    """Create a new salon with owner."""
# Implementation
```

```
@admin_cli.command()
@click.option('--salon-id', required=True, help='Salon ID')
@click.option('--active/--inactive', default=True, help='Set salon active statu
s')
def set_salon_status(salon_id, active):
    """Set salon active status."""
# Implementation

@admin_cli.command()
@click.option('--days', default=30, help='Number of days for report')
def generate_system_report(days):
    """Generate system usage report."""
# Implementation
```

## Conclusion

This API contract and database schema provide a comprehensive foundation for developing the Randevu App backend. The design follows best practices for multi-tenant SaaS applications, with careful consideration for data isolation, security, and scalability.

The implementation should prioritize:

1. Data integrity through proper transaction management
2. Security through rigorous access controls
3. Performance optimization through indexing and caching
4. Maintainability through clean, modular code organization

This design aligns with the requirements of the Flutter frontend while providing a robust backend architecture that can support the growth of the application.

1. **Application-Specific Business Logic**: Although the templates I provided outline the general business logic and flow, they may not cover all edge cases and special conditions. You may need to expand this logic based on your specific business requirements.

2. **Integration Details**: More detailed code may be required for connecting to external services, such as integration with SMS providers.
3. **Test Scenarios**: Comprehensive unit tests and integration tests are necessary for all API and database functions.
4. **Scaling Strategies**: More detailed plans are needed for how the system will scale under high-traffic conditions.

The provided schema includes all the essential structure required for a backend using Flask and PostgreSQL. By following this schema, you can develop a functional backend with all the critical features. However, in real-world applications, unexpected situations and additional requirements may always arise.

In summary, the information provided offers a solid starting point for a reliable backend, but depending on the project's features and scale, further improvements and customizations may be necessary.

---