**The Hong Kong Polytechnic University**

**School of Professional Education and Executive Development**

**SEHS4701 Advanced Information Systems Development**

**2024-25, Semester Two**

**Group Project Assignment**  
**Functional Specification**

**Class 204 Group List:**

**HO Tsan (23067880S)**

**LO Pak Ho (23059830S)**

**MOK Yeung Man** **(23070533S)**

**NOA Sze Nga** **(23066937S)**

**WONG Kang Kap Sheldon (23070348S)**

**YUEN Wing Chung (23052400S)**

**Tables of Contents**

[The background and requirements of your client 4](#_Toc196682653)

[Client 4](#_Toc196682654)

[Overview 4](#_Toc196682655)

[Objective 4](#_Toc196682656)

[Core Functional Requirements: 4](#_Toc196682657)

[Additional Assumptions: 5](#_Toc196682658)

[Business Rules 6](#_Toc196682659)

[UI Design 8](#_Toc196682660)

[Database Design 9](#_Toc196682661)

[Database Overview 9](#_Toc196682662)

[Tables and Attributes 9](#_Toc196682663)

[Relationships 13](#_Toc196682664)

[Key Design Decisions 14](#_Toc196682665)

[Database Diagram 15](#_Toc196682666)

[System Diagram and Workflow 16](#_Toc196682667)

[System Architecture 16](#_Toc196682668)

[Components 17](#_Toc196682669)

[Backend Architecture 18](#_Toc196682670)

[Core Components 19](#_Toc196682671)

[Security Architecture 20](#_Toc196682672)

[Security Components 20](#_Toc196682673)

[JWT Implementation 20](#_Toc196682674)

[Data Access Layer 21](#_Toc196682675)

[Entity Relationship Structure 21](#_Toc196682676)

[Data Transfer Objects (DTOs) 22](#_Toc196682677)

[Sample DTOs 22](#_Toc196682678)

[The DTO pattern helps: 22](#_Toc196682679)

[Integration Points 22](#_Toc196682680)

[Error Handling and Validation 22](#_Toc196682681)

[Email Notification System 22](#_Toc196682682)

[Frontend Implementation 23](#_Toc196682683)

[Dental API Documentation 24](#_Toc196682684)

[Base URL 24](#_Toc196682685)

[Endpoints 24](#_Toc196682686)

[Authentication 34](#_Toc196682687)

[Notes 34](#_Toc196682688)

[Test Cases 35](#_Toc196682689)

# The background and requirements of your client

As Apex Innovations Limited we operate as a premier software development company that focuses on developing web applications for health-related services and services industries. We must design and implement HKDC’s dental booking system using J2EE technologies and MySQL database integration to achieve both operational efficiency and patient engagement objectives.

Client:

Hong Kong Dental Care Company Limited (HKDC)

Overview:

The Hong Kong Dental Center or HKDC operates as one of Hong Kong's biggest dental clinic franchises that provides treatment at five locations through its ten dental professionals. The existing dental booking system at HKDC requires a modern efficient solution for better patient experience and enhanced operational efficiency.

Objective:

Construct a comprehensive J2EE web application which connects with MySQL database to achieve smooth patient enrollment alongside appointment scheduling and clinic data management with security and scalability features.

## Core Functional Requirements:

1. **Patient Registration**: Enable patients to create accounts to access booking features or edit accounts.
2. **Forgot Password**: Allow patients to use forgot password function to reset their passwords via reset email function.
3. **Clinic Information**: Provide details such as clinic addresses, operating hours, and dentist profiles or pricing, availability.
4. **Dentist Treatment Schedule**: Display dentist availability across clinics (e.g., Dr. William Lam at TST clinic Mon/Wed/Fri 9:00 AM–12:00 PM, Kwai Chung Tue/Thu 3:00 PM–6:00 PM).
5. **Dental Booking**: Allow registered patients to book appointments with dentists within a 3-month window.
6. **Online Booking Enquiry**: Permit patients to view their appointment details online.
7. **Confirmation Email Acknowledgement**: Send email confirmations upon successful booking.
8. **ChatBot**: Customer can check the schedule or information about dentist via ChatBot.
9. **Blog**: Dental health articles and information.
10. **Contact Us**: Patients can ask the questions via submit the contact us form
11. **Security**: JWT-based authentication, secure password handling

## Additional Assumptions:

* Patients can cancel appointments.
* Implement user authentication and session management for enhanced security.
* Support multiple appointment types (e.g., Teeth Cleaning, Tooth Filling, Dental Checkup) to cater to diverse patient needs.

# Business Rules

1. **Patient Registration**
   1. Patients must provide a valid email address, full name, phone number, and password to register.
   2. Email addresses must be unique in the system.
   3. Email addresses must be vaild (xxx@xxx).
   4. Passwords must meet security requirements (minimum 8 characters).
   5. Passwords would be saved with hash function.
   6. Only registered patients can book appointments.
   7. Patients can edit profile after logined.
   8. Patients can use forgot password function and use the url form forgot password email to reset the password.
2. **Clinic Information**
   1. The system must display up-to-date information for all 5 clinics, including addresses, contact details, and operating hours.
   2. Dentist profiles must include name, profile image, and clinic assignments.
   3. Clinic information is viewable by all users (registered or unregistered).
3. **Dentist Treatment Schedule**
   1. Dentist schedules are predefined for each clinic (e.g., Dr. William Lam’s schedule as specified).
   2. Schedules are managed in 1 hour time slots for booking purposes.
   3. The system prevents double-booking of a dentist’s time slot.
   4. Schedules are visible for the next 3 months, updated dynamically.
4. **Dental Booking**
   1. Only registered, verified patients can book appointments.
   2. Patients can select a dentist, clinic, date, and time slot.
   3. Appointment types (e.g., Teeth Cleaning, Tooth Filling) must be specified during booking.
   4. A dentist can handle only one patient per time slot within the next 3 months.
   5. Patients cannot book overlapping appointments.
   6. Bookings must be made at least 24 hours in advance.
   7. The system checks the dentist's availability before confirming a booking.
   8. Patients can cancel the appointment and receive cancellation confirmation email.
5. **Online Booking Enquiry**
   1. Registered patients can view their upcoming and past appointments after login.
   2. Appointment details include dentist name, clinic, date, time, status, and appointment type.
   3. Access to booking details is restricted to the patient’s own records.
6. **Confirmation Email Acknowledgement**
   1. A confirmation email is sent to the patient’s registered email upon successful booking.
   2. The email includes appointment details (dentist, clinic, date, time, appointment type).
7. **Contact Us**
   1. Allow Patient to ask the question via contact us form
   2. Phone 8 digit validation
   3. Email address valid validation
8. **Blog**
   1. Allows the clinic to publish and manage dental health information articles to educate patients
9. **Security and Access Control**
   1. Patient data (e.g., password) is hashed in the MySQL database.
   2. Session timeouts are enforced after 15 minutes of inactivity.
10. **System Constraints**
    1. The system supports up to 10 dentists and 5 clinics initially, with scalability for future expansion.

# UI Design

# Database Design

The HKDC dental booking system database sustains patient enrollment together with clinic details retention along with dentist hours scheduling and appointment arrangements and supplementary functionality. The system operates within a dental database named **dental** which implements relational schema optimization principles for performance enhancement and data integrity and scalability purposes. The design takes its origin from the SQL schema framework while fulfilling the operational demands regarding patient enrollment as well as the management of clinics and dentists and scheduling system and appointment reservation and email notification functions.

## Database Overview

* Database Name: **dental**
* Character Set: **utf8mb4** with **utf8mb4\_general\_ci** collation to support multilingual data.
* Storage Engine: InnoDB, ensuring transactional integrity and foreign key support.
* Purpose: Store and manage data for clinics, dentists, patients, appointments, services, and schedules.

## Tables and Attributes

* **clinics**

Purpose: Stores information about HKDC’s five clinic locations.

|  |  |  |
| --- | --- | --- |
| Attributes | Properties | Description |
| id | INT, PK, AUTO\_INCREMENT | Unique identifier |
| name | VARCHAR(100), NOT NULL | Clinic name |
| address | TEXT, NOT NULL | Full address |
| district | VARCHAR(100), NOT NULL | District |
| phone | VARCHAR(100), NOT NULL | Contact number |
| open\_hours | TEXT, NOT NULL | Operating hours |

* **dentists**

Purpose: Stores profiles of the 10 dentists.

|  |  |  |
| --- | --- | --- |
| Attributes | Properties | Description |
| id | INT, PK, AUTO\_INCREMENT | Unique identifier |
| first\_name | VARCHAR(100), NOT NULL | Dentist’s first name |
| last\_name | VARCHAR(100), NOT NULL | Dentist’s last name |
| gender | VARCHAR(100), NOT NULL | Gender of the dentist |
| phone | VARCHAR(100), NOT NULL | Contact number |
| email\_addres | VARCHAR(255), NOT NULL | Email address |
| image\_url | TEXT, NOT NULL | Dentist’s profile image |

* **patients**

Purpose: Stores registered patient information for booking and authentication.

|  |  |  |
| --- | --- | --- |
| Attributes | Properties | Description |
| id | INT, PK, AUTO\_INCREMENT | Unique identifier |
| first\_name | VARCHAR(100), NOT NULL | Patient’s first name |
| last\_name | VARCHAR(100), NOT NULL | Patient’s last name |
| email\_address | VARCHAR(255), NOT NULL | Unique email for login |
| gender | VARCHAR(100), NOT NULL | Patient’s gender |
| dob | DATE, NOT NULL | Date of birth |
| phone | VARCHAR(100), NOT NULL | Contact number |
| password | TEXT, NOT NULL | Hashed password |
| created\_at | TIMESTAMP, NULL | Account creation timestamp |
| last\_login\_at | TIMESTAMP, NULL | Last login timestamp |

Indexes:

Index on appointment\_date to optimize queries for appointment listings and availability checks.

* **timeslots**

Purpose: Defines available time slots for dentist schedules and bookings.

|  |  |  |
| --- | --- | --- |
| Attributes | Properties | Description |
| id | INT, PK, AUTO\_INCREMENT | Unique identifier |
| start\_time | TIME, NOT NULL | Start time of the slot (e.g., 09:00:00) |
| end\_time | TIME, NOT NULL | End time of the slot (e.g., 10:00:00) |

* **clinic\_dentists**

Purpose: Maps dentists to clinics, specifying their schedules (day and time).

|  |  |  |
| --- | --- | --- |
| Attributes | Properties | Description |
| id | INT, PK, AUTO\_INCREMENT | Unique identifier |
| clinic\_id | INT, NOT NULL, FK | References clinics.id |
| dentist\_id | INT, NOT NULL, FK | References dentists.id |
| day\_of\_week | ENUM('Mon','Tue','Wed','Thu','Fri'), NOT NULL | Week Day |
| time\_slot\_id | INT, NOT NULL, FK | References timeslots.id |

Constraints:

* Unique constraint on (clinic\_id, dentist\_id, day\_of\_week, time\_slot\_id) to prevent duplicate schedules.
* Foreign keys ensure referential integrity.
* **items**

Purpose: Stores available dental services (appointment types).

|  |  |  |
| --- | --- | --- |
| Attributes | Properties | Description |
| id | INT, PK, AUTO\_INCREMENT | Unique identifier |
| name | VARCHAR(100), NOT NULL | Service name |
| image\_url | TEXT, NOT NULL | Service image |

* **dentist\_items**

Purpose: Associates dentists with services they offer and their fees.

|  |  |  |
| --- | --- | --- |
| Attributes | Properties | Description |
| id | INT, PK, AUTO\_INCREMENT | Unique identifier |
| dentist\_id | INT, NOT NULL, FK | References dentists.id |
| item\_id | INT, NOT NULL, FK | References items.id |
| fee | DECIMAL(12,2), NOT NULL | Fee for the service |

* **appointments**

Purpose: Stores patient appointments with dentists at specific clinics.

|  |  |  |
| --- | --- | --- |
| Attributes | Properties | Description |
| id | INT, PK, AUTO\_INCREMENT | Unique identifier |
| patient\_id | INT, NOT NULL, FK | References patients.id |
| clinic\_dentist\_id | INT, NOT NULL, FK | References clinic\_dentists.id |
| appointment\_date | DATE, NOT NULL | Date of the appointment |
| total\_amount | DECIMAL(12,2), NOT NULL | Total cost of the appointment |
| status | VARCHAR(100), NOT NULL | Appointment status |
| created\_at | TIMESTAMP, NULL | Appointment creation timestamp |

Indexes:

Index on appointment\_date to optimize queries for appointment listings and availability checks.

* **appointment\_items**

Purpose: Links appointments to specific services (items) selected.

|  |  |  |
| --- | --- | --- |
| Attributes | Properties | Description |
| id | INT, PK, AUTO\_INCREMENT | Unique identifier |
| appointment\_id | INT, NOT NULL, FK | References appointments.id |
| dentist\_item\_id | INT, NOT NULL, FK | References dentist\_items.id |

* **contact\_us**

Purpose: Stores the message from visitor via submit form.

|  |  |  |
| --- | --- | --- |
| Attributes | Properties | Description |
| id | INT, PK, AUTO\_INCREMENT | Unique identifier |
| first\_name | VARCHAR(100), NOT NULL | Visitor’s first name |
| last\_name | VARCHAR(100), NOT NULL | Visitor’s last name |
| phone | VARCHAR(100), NOT NULL | Visitor’s contact number |
| email\_address | VARCHAR(255), NOT NULL | Visitor’s email address |
| topic | VARCHAR(100), NOT NULL | Topic of message |
| password | TEXT, NOT NULL | Message content |
| created\_at | TIMESTAMP, NULL | Creation timestamp |

* **blogs**

Purpose: Stores the blog information.

|  |  |  |
| --- | --- | --- |
| Attributes | Properties | Description |
| id | INT, PK, AUTO\_INCREMENT | Unique identifier |
| title | VARCHAR(255), NOT NULL | Blog’s title |
| slug | VARCHAR(255), NOT NULL | Blog’s slug |
| snippet | TEXT, NOT NULL | Blog’s snippet |
| content | TEXT, NOT NULL | Blog’s content |
| post\_date | DATE, NOT NULL | Blog’s publish date |
| image | VARCHAR(255), NOT NULL | Blog’s content |

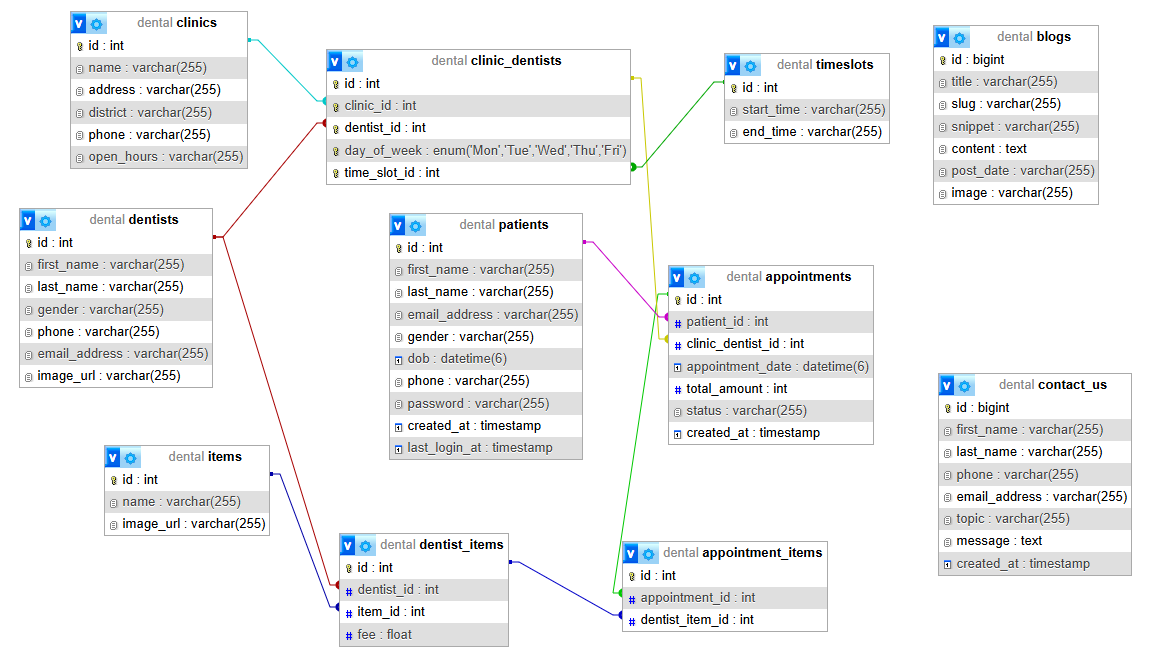
## Relationships

* **clinics ↔ clinic\_dentists**: One-to-many (one clinic has many dentist schedules).
* **dentists ↔ clinic\_dentists**: One-to-many (one dentist has many schedules).
* **timeslots ↔ clinic\_dentists**: One-to-many (one time slot is used in many schedules).
* **dentists ↔ dentist\_items**: One-to-many (one dentist offers many services).
* **items ↔ dentist\_items**: One-to-many (one service is offered by many dentists).
* **patients ↔ appointments**: One-to-many (one patient has many appointments).
* **clinic\_dentists ↔ appointments**: One-to-many (one schedule slot supports many appointments).
* **appointments ↔ appointment\_items**: One-to-many (one appointment includes many services).
* **dentist\_items ↔ appointment\_items**: One-to-many (one dentist-service pair is linked to many appointments).

## Key Design Decisions

* **Normalized Schema**: The database is normalized to at least **3NF** to reduce redundancy and ensure data integrity (e.g., separating items and dentist\_items for flexible pricing).
* **Foreign Keys**: Cascading deletes and updates (ON DELETE CASCADE, ON UPDATE CASCADE) ensure consistency when records are modified or removed.
* **Unique Constraints**: The clinic\_dentists table has a unique constraint to prevent scheduling conflicts for the same dentist, clinic, day, and time.
* **Decimal Precision**: total\_amount and fee use DECIMAL(12,2) to handle precise financial calculations (up to HKD 999,999,999.99).
* **Flexible Scheduling**: The clinic\_dentists table supports dynamic schedules by linking dentists to specific clinics, days, and time slots, aligning with requirements (e.g., Dr. William Lam’s schedule).
* **Scalability**: AUTO\_INCREMENT primary keys and InnoDB’s performance features support growth beyond 10 dentists and 5 clinics.
* **Security**: Patient passwords are stored as hashed values (TEXT)

## Database Diagram

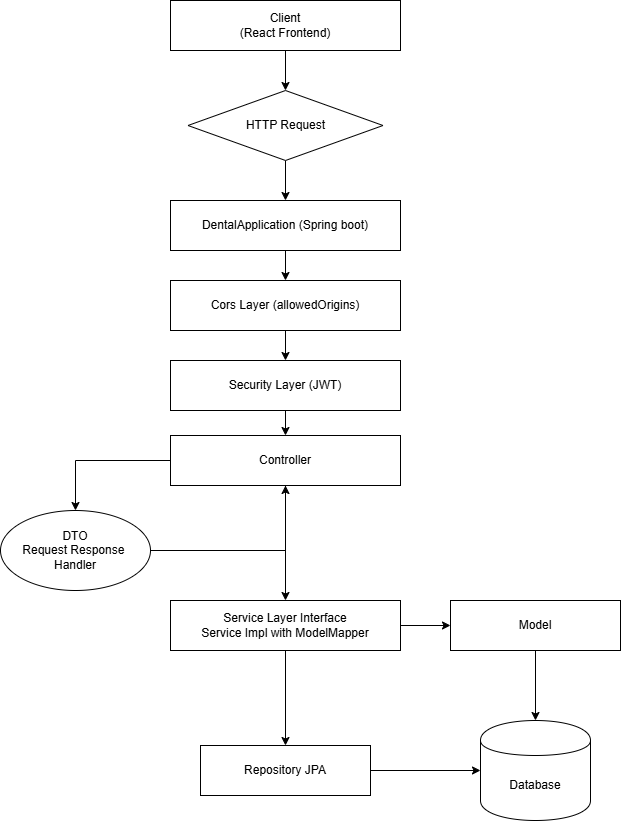


# System Diagram and Workflow

## System Architecture

The application is designed to be deployed in a containerized environment with Docker Compose, with separate containers for:

* Frontend (React/Next.js)
* Backend (Spring Boot)
* Database (MySQL)



## Components

**Client (React Frontend)**

* A React-based frontend sends HTTP requests to the backend.

**Spring Boot Backend**

* Handles incoming requests, processes business logic, and communicates with the database.

**Security Layer**

* Uses JWT for authentication for /appointment/\*\* endpoints
* Public endpoints like /dentist/\*\* do not require authentication.

**Controller**

* Receives HTTP requests from the client endpoints, invokes the appropriate Service methods, and returns responses with proper HTTP status codes.

**Service**

* the interface for business logic

**ServiceImpl**

* Implements the Service interface, handles the logic from the Repository, and maps entities to DTOs using ModelMapper.

**ModelMapper**

* Maps between Model entity and DTOs

**Repository**

* Provides CRUD operations for the Model entity using Spring Data JPA.

**Model**

* Represents the entity, defining the structure of data

**MySQL Database**

* Stores dentist data in the table

**Test Code**

* Test Function

## Backend Architecture

The dental clinic system follows a layered architecture pattern designed around Spring Boot principles to ensure scalability, maintainability, and separation of concerns.

一張含有 文字, 螢幕擷取畫面, 圖表, 收據 的圖片

AI 產生的內容可能不正確。

### Core Components

The backend is built using the Spring Boot framework and follows a classic layered architecture with the following key components:

一張含有 文字, 螢幕擷取畫面, 字型, 圖表 的圖片

AI 產生的內容可能不正確。

## Security Architecture

The application uses Spring Security with JWT (JSON Web Token) for authentication and authorization.

### Security Components

一張含有 文字, 圖表, 螢幕擷取畫面 的圖片

AI 產生的內容可能不正確。

### JWT Implementation

The JWT implementation provides secure authentication with the following features:

* Token generation upon successful login
* Token validation for protected resources
* Username extraction from tokens for identification

The **JwtUtil** class handles the core JWT operations:

* Token generation with configurable expiration (24 hours default)
* Token validation and parsing
* Username extraction from tokens

### Data Access Layer

The data access layer uses Spring Data JPA to interact with the MySQL database.

### Entity Relationship Structure

The backend uses JPA entities to model domain objects, with repositories providing data access operations.

一張含有 文字, 螢幕擷取畫面, 圖表, 字型 的圖片

AI 產生的內容可能不正確。

## Data Transfer Objects (DTOs)

The application uses DTOs to transfer data between client and server, decoupling the internal entity representation from the API contract.

### Sample DTOs

* **LoginResponse**: Contains patient information and authentication status
* **PatientByIdResponse**: Returns patient details by ID

### The DTO pattern helps:

* Control what data is exposed to clients
* Version APIs without changing entity models
* Optimize network traffic by transferring only needed data

### Integration Points

The backend integrates with several external systems and services:

1. **Email Service**: Uses SMTP to send notifications
2. **Database**: MySQL for persistent storage
3. **Frontend**: Exposes RESTful APIs consumed by the React frontend

### Error Handling and Validation

The application implements consistent error handling with:

* Global exception handling
* Validation for request payloads
* Standardized error responses

### Email Notification System

The Email Notification System is responsible for:

* Sending appointment confirmation emails when patients book appointments
* Sending password reset emails when patients request to reset their passwords
* Managing email templates with dynamic content
* Handling email delivery through SMTP

# Frontend Implementation

The frontend of the dental clinic system is built using React with Next.js framework and TypeScript. It follows a component-based architecture with page-based routing provided by Next.js.

一張含有 文字, 螢幕擷取畫面, 圖表, 字型 的圖片

AI 產生的內容可能不正確。

# Dental API Documentation

This document describes the API endpoints for the Dental application, based on the provided Postman collection. The API is hosted at [http://localhost:6616](http://localhost:6616/) and supports various operations related to items, dentists, clinics, patients, appointments, blogs, and contact forms. Most endpoints use the GET method for retrieval and POST for creation or updates, with some requiring Bearer token authentication.

## Base URL

[http://localhost:6616](http://localhost:6616/)

## Endpoints

**1. Items**

***List All Items***

* **URL**: /item/list
* **Method**: GET
* **Description**: Retrieves a list of all items.
* **Authentication**: None
* **Response**: Array of item objects (structure not specified).

**2. Dentists**

***List All Dentists***

* **URL**: /dentist/list
* **Method**: GET
* **Description**: Retrieves a list of all dentists.
* **Authentication**: None
* **Response**: Array of dentist objects.

***Get Dentist by ID***

* **URL**: /dentist/{id}
* **Method**: GET
* **Description**: Retrieves details of a specific dentist by ID.
* **Authentication**: None
* **Path Parameters**:
* id (integer): Dentist ID (e.g., 1)
* **Response**: Dentist object.

**3. Dentist Items**

***List All Dentist Items***

* **URL**: /dentistItem/list
* **Method**: GET
* **Description**: Retrieves a list of all dentist items.
* **Authentication**: None
* **Response**: Array of dentist item objects.

***Get Dentist Item by ID***

* **URL**: /dentistItem/{id}
* **Method**: GET
* **Description**: Retrieves details of a specific dentist item by ID.
* **Authentication**: None
* **Path Parameters**:
* id (integer): Dentist item ID (e.g., 1)
* **Response**: Dentist item object.

***Get Dentist Items by Dentist ID***

* **URL**: /dentistItem/dentist/{id}
* **Method**: GET
* **Description**: Retrieves all items associated with a specific dentist.
* **Authentication**: None
* **Path Parameters**:
* id (integer): Dentist ID (e.g., 1)
* **Response**: Array of dentist item objects.

***Get Dentist Items by Item ID***

* **URL**: /dentistItem/item/{id}
* **Method**: GET
* **Description**: Retrieves all dentist items associated with a specific item.
* **Authentication**: None
* **Path Parameters**:
* id (integer): Item ID (e.g., 1)
* **Response**: Array of dentist item objects.

**4. Clinics**

***List All Clinics***

* **URL**: /clinic/list
* **Method**: GET
* **Description**: Retrieves a list of all clinics.
* **Authentication**: None
* **Response**: Array of clinic objects.

***Get Clinic by ID***

* **URL**: /clinic/{id}
* **Method**: GET
* **Description**: Retrieves details of a specific clinic by ID.
* **Authentication**: None
* **Path Parameters**:
* id (integer): Clinic ID (e.g., 1)
* **Response**: Clinic object.

**5. Clinic Dentists**

***List All Clinic Dentists***

* **URL**: /clinicDentist/list
* **Method**: GET
* **Description**: Retrieves a list of all clinic-dentist associations.
* **Authentication**: None
* **Response**: Array of clinic-dentist objects.

***Get Clinic Dentist by ID***

* **URL**: /clinicDentist/{id}
* **Method**: GET
* **Description**: Retrieves details of a specific clinic-dentist association by ID.
* **Authentication**: None
* **Path Parameters**:
* id (integer): Clinic-dentist ID (e.g., 1)
* **Response**: Clinic-dentist object.

***Get Clinic Dentists by Dentist ID***

* **URL**: /clinicDentist/dentist/{id}
* **Method**: GET
* **Description**: Retrieves all clinic associations for a specific dentist.
* **Authentication**: None
* **Path Parameters**:
* id (integer): Dentist ID (e.g., 1)
* **Response**: Array of clinic-dentist objects.

***Get Clinic Dentists by Clinic ID***

* **URL**: /clinicDentist/clinic/{id}
* **Method**: GET
* **Description**: Retrieves all dentist associations for a specific clinic.
* **Authentication**: None
* **Path Parameters**:
* id (integer): Clinic ID (e.g., 1)
* **Response**: Array of clinic-dentist objects.

***Get Timeslot for Clinic Dentist***

* **URL**: /clinicDentist/timeslot/{id}
* **Method**: GET
* **Description**: Retrieves timeslot information for a specific clinic-dentist association.
* **Authentication**: None
* **Path Parameters**:
* id (integer): Clinic-dentist ID (e.g., 1)
* **Response**: Timeslot object.

**6. Patients**

***Register Patient***

* **URL**: /patient/register
* **Method**: POST
* **Description**: Registers a new patient.
* **Authentication**: None
* **Request Body**:{   
   "firstName": "string",   
   "lastName": "string",   
   "emailAddress": "string",   
   "password": "string",   
   "phone": "string",   
   "gender": "string",   
   "dob": "string (ISO 8601)"   
  }
* **Response**: Patient object or success message.

***Login Patient***

* **URL**: /patient/login
* **Method**: POST
* **Description**: Authenticates a patient and returns a token.
* **Authentication**: None
* **Request Body**:{   
   "emailAddress": "string",   
   "password": "string"   
  }
* **Response**: Authentication token (JWT).

***Edit Patient***

* **URL**: /patient/edit
* **Method**: POST
* **Description**: Updates patient information.
* **Authentication**: None
* **Request Body**:{   
   "id": "string",   
   "firstName": "string",   
   "lastName": "string",   
   "emailAddress": "string",   
   "password": "string",   
   "phone": "string",   
   "gender": "string",   
   "dob": "string (ISO 8601)"   
  }
* **Response**: Updated patient object or success message.

***Get Patient by ID***

* **URL**: /patient/{id}
* **Method**: GET
* **Description**: Retrieves details of a specific patient by ID.
* **Authentication**: None
* **Path Parameters**:
* id (integer): Patient ID (e.g., 1)
* **Response**: Patient object.

***Forgot Password***

* **URL**: /patient/forgot-password
* **Method**: POST
* **Description**: Initiates password reset process by sending a reset code.
* **Authentication**: None
* **Request Body**:{   
   "email": "string"   
  }
* **Response**: Success message.

***Reset Password***

* **URL**: /patient/reset-password
* **Method**: POST
* **Description**: Resets patient password using a reset code.
* **Authentication**: None
* **Request Body**:{   
   "email": "string",   
   "code": "string",   
   "newPassword": "string"   
  }
* **Response**: Success message.

**7. Appointments**

***Create Appointment***

* **URL**: /appointment/create
* **Method**: POST
* **Description**: Creates a new appointment.
* **Authentication**: Bearer Token
* **Request Body**:{   
   "patientId": integer,   
   "clinicDentistId": integer,   
   "appointmentDate": "string (ISO 8601)",   
   "totalAmount": number,   
   "status": "string",   
   "appointmentItems": [   
     {   
       "id": integer,   
       "dentistItemId": integer   
     }   
   ]   
  }
* **Response**: Appointment object or success message.

***Cancel Appointment***

* **URL**: /appointment/cancel
* **Method**: POST
* **Description**: Cancels an existing appointment.
* **Authentication**: Bearer Token
* **Request Body**:{   
   "appointmentId": integer   
  }
* **Response**: Success message.

***Get Appointment by ID***

* **URL**: /appointment/{id}
* **Method**: GET
* **Description**: Retrieves details of a specific appointment by ID.
* **Authentication**: Bearer Token
* **Path Parameters**:
* id (integer): Appointment ID (e.g., 1)
* **Response**: Appointment object.

***List All Appointments***

* **URL**: /appointment/list
* **Method**: GET
* **Description**: Retrieves a list of all appointments.
* **Authentication**: Bearer Token
* **Response**: Array of appointment objects.

***Get Appointments by Patient ID***

* **URL**: /appointment/patient/{id}
* **Method**: GET
* **Description**: Retrieves all appointments for a specific patient.
* **Authentication**: Bearer Token
* **Path Parameters**:
* id (integer): Patient ID (e.g., 1)
* **Response**: Array of appointment objects.

***Get Appointments by Clinic, Dentist, and Date***

* **URL**: /appointment/clinicId/{clinicId}/dentistId/{dentistId}/appointmentDate/{date}
* **Method**: GET
* **Description**: Retrieves appointments for a specific clinic, dentist, and date.
* **Authentication**: Bearer Token
* **Path Parameters**:
* clinicId (integer): Clinic ID (e.g., 1)
* dentistId (integer): Dentist ID (e.g., 1)
* date (string): Date in YYYY-MM-DD format (e.g., 2025-05-24)
* **Response**: Array of appointment objects.

**8. Blogs**

***List All Blogs***

* **URL**: /blog/list
* **Method**: GET
* **Description**: Retrieves a list of all blog posts.
* **Authentication**: None
* **Response**: Array of blog objects.

***Get Blog by ID***

* **URL**: /blog/{id}
* **Method**: GET
* **Description**: Retrieves details of a specific blog post by ID.
* **Authentication**: None
* **Path Parameters**:
* id (integer): Blog ID (e.g., 1)
* **Response**: Blog object.

**9. Contact Us**

***Submit Contact Form***

* **URL**: /contact\_us
* **Method**: POST
* **Description**: Submits a contact form with user inquiries.
* **Authentication**: None
* **Request Body**:{   
   "firstName": "string",   
   "lastName": "string",   
   "phone": "string",   
   "emailAddress": "string",   
   "topic": "string",   
   "message": "string"   
  }
* **Response**: Success message.

## Authentication

* **Bearer Token**: Required for appointment-related endpoints (/appointment/\*). Include the token in the Authorization header:Authorization: Bearer <token>
* **No Authentication**: Most endpoints do not require authentication, except for appointment-related operations.

## Notes

* All endpoints assume JSON as the content type for request and response bodies.
* Timestamps in request bodies (e.g., dob, appointmentDate) should follow the ISO 8601 format (e.g., 2025-05-24T10:00:00).
* The API runs on localhost:6616, indicating it is intended for local development or testing.
* Response structures (e.g., object schemas) are not specified in the Postman collection and should be verified with the API implementation.

# Test Cases

**User Acceptance Test**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Created Date | Author | Change Description |
| 0.1 | 29 Apr 2025 | Group 204 | Initial Version (internal testing) |

|  |  |  |  |
| --- | --- | --- | --- |
| Section | Test Case | Pass | Fail |
| A | Home Page |  |  |
|  | Our Treatments List | ■ | □ |
|  | Our Team List | ■ | □ |
|  | Our Locations | ■ | □ |
|  | Other Static Page | ■ | □ |
| B | Registration and Login |  |  |
|  | Register a New Account | ■ | □ |
|  | Login / Edit Account / Logout | ■ | □ |
|  | Forgot/Reset Password |  |  |
| C | Clinic Information |  |  |
|  | View Clinic List | ■ | □ |
| D | Dental Booking |  |  |
|  | Book an Appointment | ■ | □ |
|  | Cancel Booking |  |  |
| E | ChatBot | ■ | □ |
| F | Contact Us Form | ■ | □ |
| G | Email Function | ■ | □ |

**Web > Home > Registration/Login**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Function | Function Description | Test Scenario | Expected Result | Pass | Fail (Reason) |
| Register a New Account | Register a new patient account | **Pre-Condition in Web:**  1. Stable internet connection  2. Email is not previously registered in database  **Steps in Web:**  1. Fill in fields:    - First Name\* (e.g., John, < 100 chars)    - Last Name\* (e.g., Doe, < 100 chars)    - Email\* (e.g., john.doe@example.com)    - Phone\* (8 digits, e.g., 12345678)    - Gender\* (Male/Female)    - DOB\* (e.g., 1990-01-01)  - Password\* (≥8 chars, letters/numbers)    - Check T&C checkbox\*  2. Press “Submit” with/without mandatory fields  3. Press “Submit” with invalid data (e.g., email: abc, phone: 123)  4. Click T&C hyperlink to view terms | **Result in Web:**  1. With all mandatory fields correctly filled, proceed to confirmation page (Screen 3.1)    2. If mandatory fields are missing/invalid, “Submit” button is disabled, and alert messages appear    3. Invalid email/phone triggers specific error messages    4. T&C hyperlink opens terms page (Screen 4.0), can return to Screen 3.0  5. Press “Back” to return to Home (Screen 2.0), previous inputs cleared on re-entering registration | ■ | □ |

**Web > Home > Clinic Information**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Function | Function Description | Test Scenario | Expected Result | Pass | Fail (Reason) |
| View Clinic List | View list of all clinics | **Pre-Condition in Web:**  1. Stable internet connection  2. On Home Page (Screen 2.0)    **Steps in Web:**  1. Press “Clinics” menu item  2. View clinic list | **Result in Web:**  1. Proceed to Clinic List Page (Screen 5.0)  2. Display 5 clinics with details: name, address, district, phone, operating hours | ■ | □ |

**Web > Home > Book Appointment**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Function | Function Description | Test Scenario | Expected Result | Pass | Fail (Reason) |
| Book an Appointment | Book a dental appointment | **Pre-Condition in Web:**  1. Stable internet connection  2. User is logged in with registered account  3. Dentist ID 1 available at clinic ID 1 on 2025-05-25 10:00 AM  **Steps in Web:**  1. Press “Book Appointment” menu item  2. Select clinic, dentist, date, time slot, and service (e.g., Teeth Cleaning)  3. Press “Confirm Booking”  4. Attempt to book same time slot again  5. Receive confirm email | **Result in Web:**  1. Proceed to Clinic List Page (Screen 5.0)  2. Display 5 clinics with details: name, address, district, phone, operating hours  3. Details are correctly formatted and match database records | ■ | □ |