Design

Document

Doctor’s Appointment Application

**Index**

|  |  |  |
| --- | --- | --- |
| Sr No | Content | Page No |
| 1 | Introduction | 3 |
| 2 | Architectural Design | 3 |
| 3 | High Level Design | 6 |
|  | 3.1 E-R Diagram | 6 |
|  | 3.2 Page Navigation Diagram | 7 |
|  | 3.3 Data Flow Diagram | 9 |
| 4 | Low Level Design | 13 |
|  | 4.1 Database Design | 13 |
|  | 4.2 Stored Procedure | 16 |
|  | 4.3 Details Of Page Navigation | 17 |

* **Introduction:**
* This document explains the system requirements and scope for developing Doctor appointment application
* Doctor appointment application could divide the three main parts, Patient part, Doctor part, Admin part.
* This document describes the system requirement of the Account part.

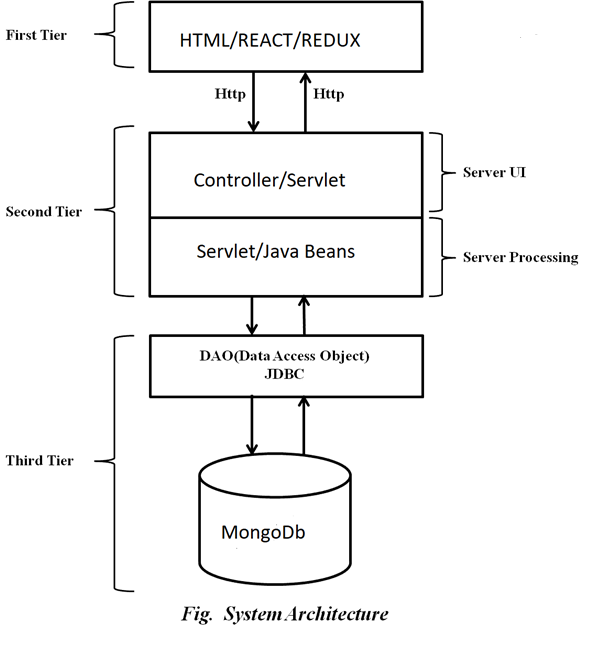
.

* **Architecture Design:**

Following diagram shows the details of the Doctor’s Appointment Application architecture.

This System consist of three tiers as listed below,

* First tier
* Second tier
* Third tier



**First Tier:**

1) This tier is used for user interface and it is called as client tier. In this tier we are using React because it is Component-Based Architecture: React is component-based architecture promotes modularity and reusability. In a Doctor’s appointment application, we create separate components for users, and other UI elements. This makes the codebase organized, easier to maintain, and allows for efficient collaboration among developers.

2)Virtual DOM:

React uses a virtual DOM to efficiently update the actual DOM. In a Doctor’s appointment application, where users might be added, updated, or removed frequently, this feature ensures that only the necessary changes are applied to the DOM, reducing the overall performance overhead.

**Second Tier:**

**HTTP Requests:**

The primary method of communication between the frontend and the backend is through HTTP requests. These requests are initiated from the frontend and sent to specific URLs (endpoints) on the backend server. We are using React JavaScript's built-in fetch function or third-party libraries like Axios to make HTTP requests to the backend.

**Service and Controller:**

Created a service layer that interacts with the repository and a controller to handle incoming requests. The service layer should contain business logic, and the controller should handle HTTP endpoints.

**API Endpoints:**

The backend provides various API endpoints that the frontend can interact with. These endpoints are defined based on the functionalities our application requires (e.g., fetching data, submitting forms, authentication).

For example, we have endpoints like /api/users to get a list of users or /api/login to handle user authentication.

**Authentication and Authorization (using JWT):**

User authentication and authorization are important aspects of connecting a frontend and backend. The backend typically provides mechanisms for user registration, login, and token-based authentication.

After authentication, the backend generates tokens (e.g., JWT) that the frontend includes in subsequent requests to access protected resources.

**Data Formats:**

Data exchanged between the frontend and backend is often in JSON format. The frontend sends data as JSON in the request body, and the backend responds with JSON data in the response body.

**CORS (Cross-Origin Resource Sharing):**

The frontend and backend are hosted on different domains or ports might encounter CORS issues. Therefore, CORS is a security feature implemented to prevent unauthorized cross-origin requests.

The backend has configured to allow requests from specific origins (domains) using appropriate response headers.

* **Server Process**

Spring Boot uses auto-configuration to automatically configure various components based on the class path and the included libraries. It scans for dependencies and sets up appropriate configurations, reducing the need for manual configuration.

**Third Tier:**

Third tier consist of a Data Access Object (DAO) and the back end i.e. the database of Online PG Management system.

**Data Access Object (DAO):**

Data access object layer has proven good in separate business logic layer and persistent layer. The DAO design pattern completely hides the data access implementation from its clients.

Using MongoDB with Spring Boot involves integrating MongoDB, a NoSQL database, in Spring Boot application. Spring Data MongoDB is a part of the Spring Data project that provides a higher-level abstraction for working with MongoDB in a Spring application. Here's a basic overview of how to use MongoDB with Spring Boot:

**Add Dependencies:**

Start by adding the necessary dependencies in Spring Boot project's pom.xml (Maven) file and need spring-boot-starter-data-mongodb to include Spring Data MongoDB and the MongoDB Java driver.

**Configure MongoDB Connection:**

In application properties configuration file, we specified the MongoDB connection details such as URL, database name, and authentication credentials.

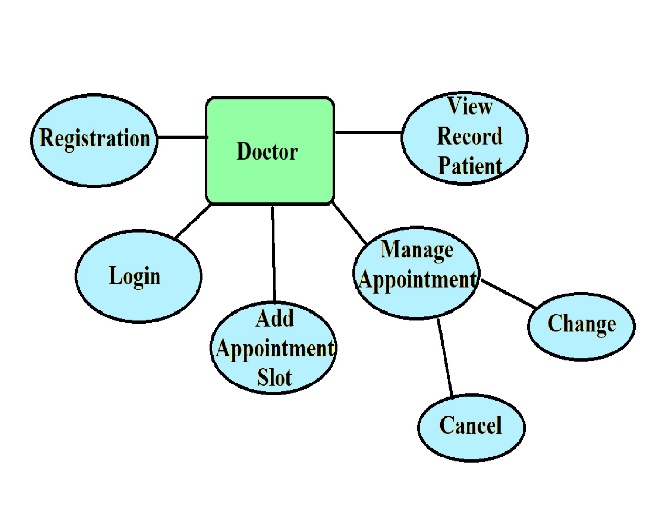
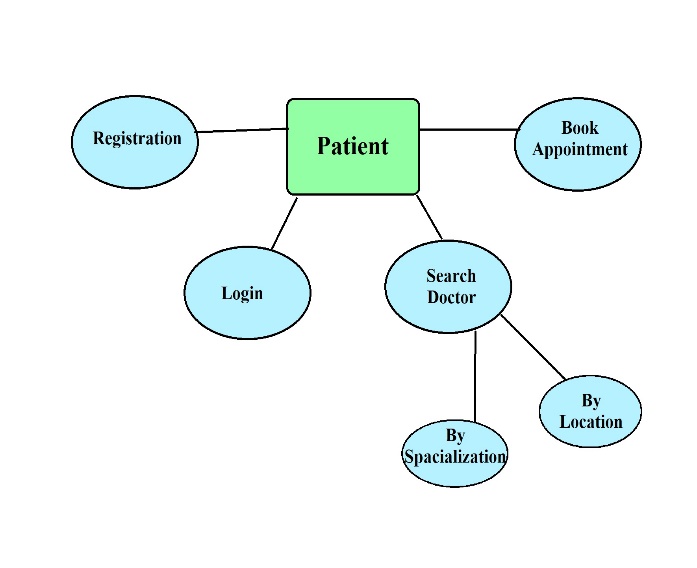
**Create Entity Classes:**

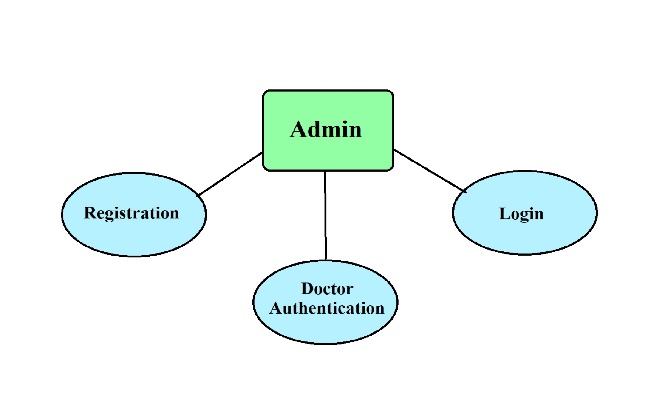
Defined Java classes that will represent our MongoDB documents. These classes should be annotated with @Document and other annotations to define their structure.

**Create Repositories:**

Spring Data MongoDB provides repository interfaces that extend MongoRepository or CrudRepository. These interfaces offer methods for common database operations.

* **High Level Design:**
* **E-R Diagram:**





Above E-R Diagram shows that database of Task Management system for Doctor appointment consist of following entities:

* **Registered Doctor**

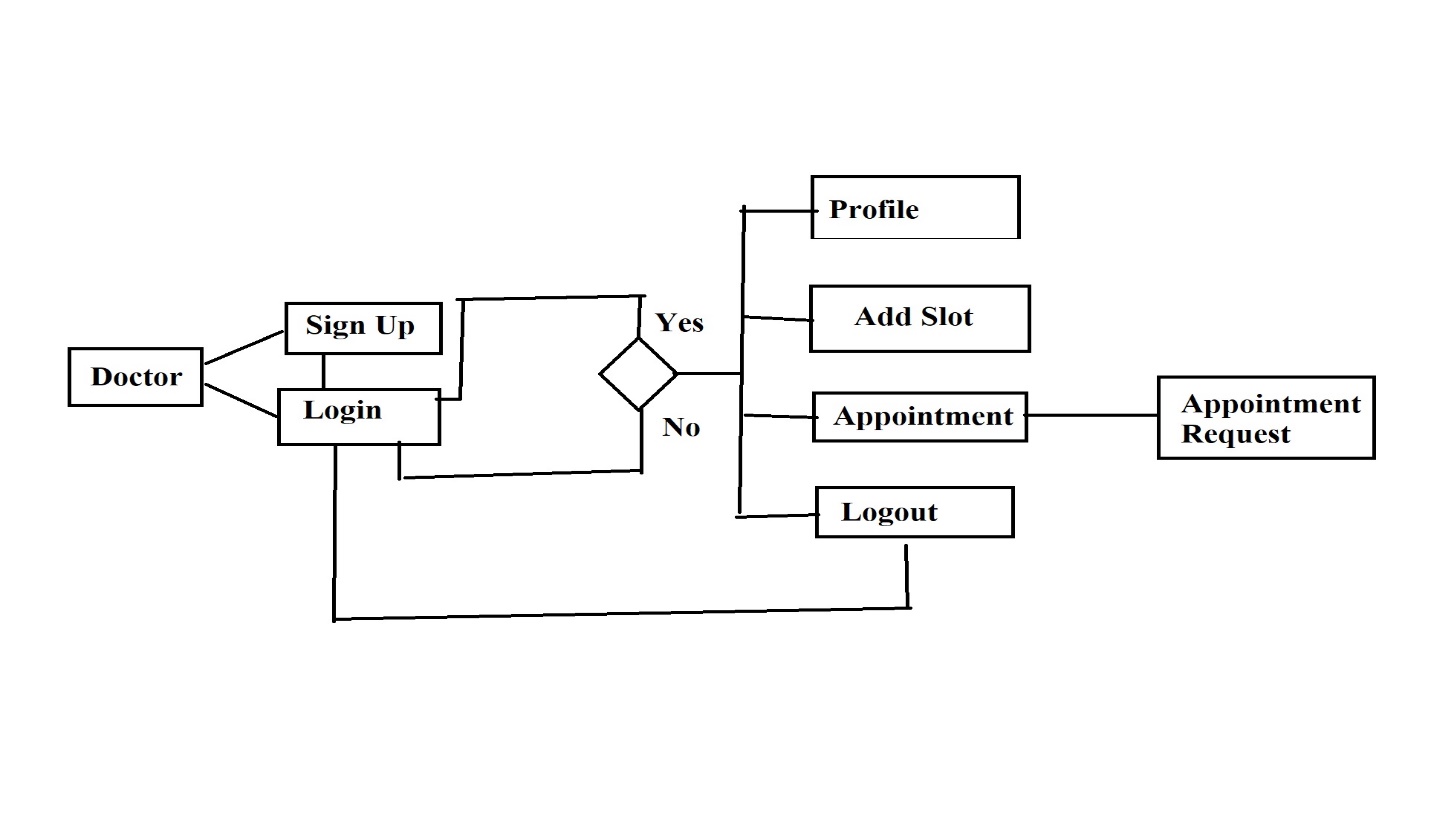
This Entity contains the Name , Email, Contact,Address, Spacialization, Registration number, Password.

* **Registered Patient**

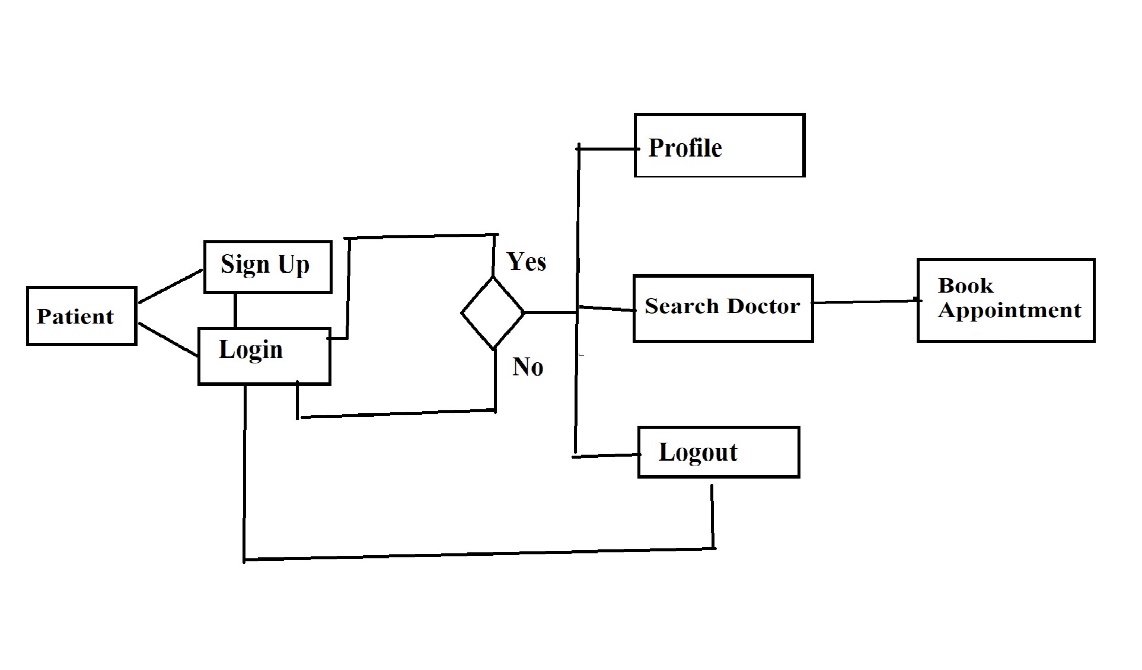
This Entity contains the ,Name ,Email, Contact, Address, Password.

* **Page Navigation Diagram:**
* **Doctor**

Following diagram explains the page navigation for the Doctor module:

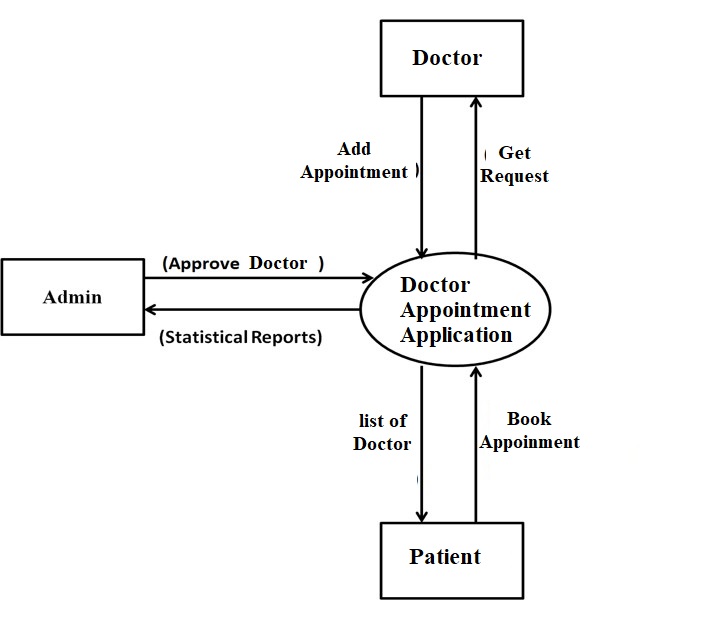


* **Patient :**

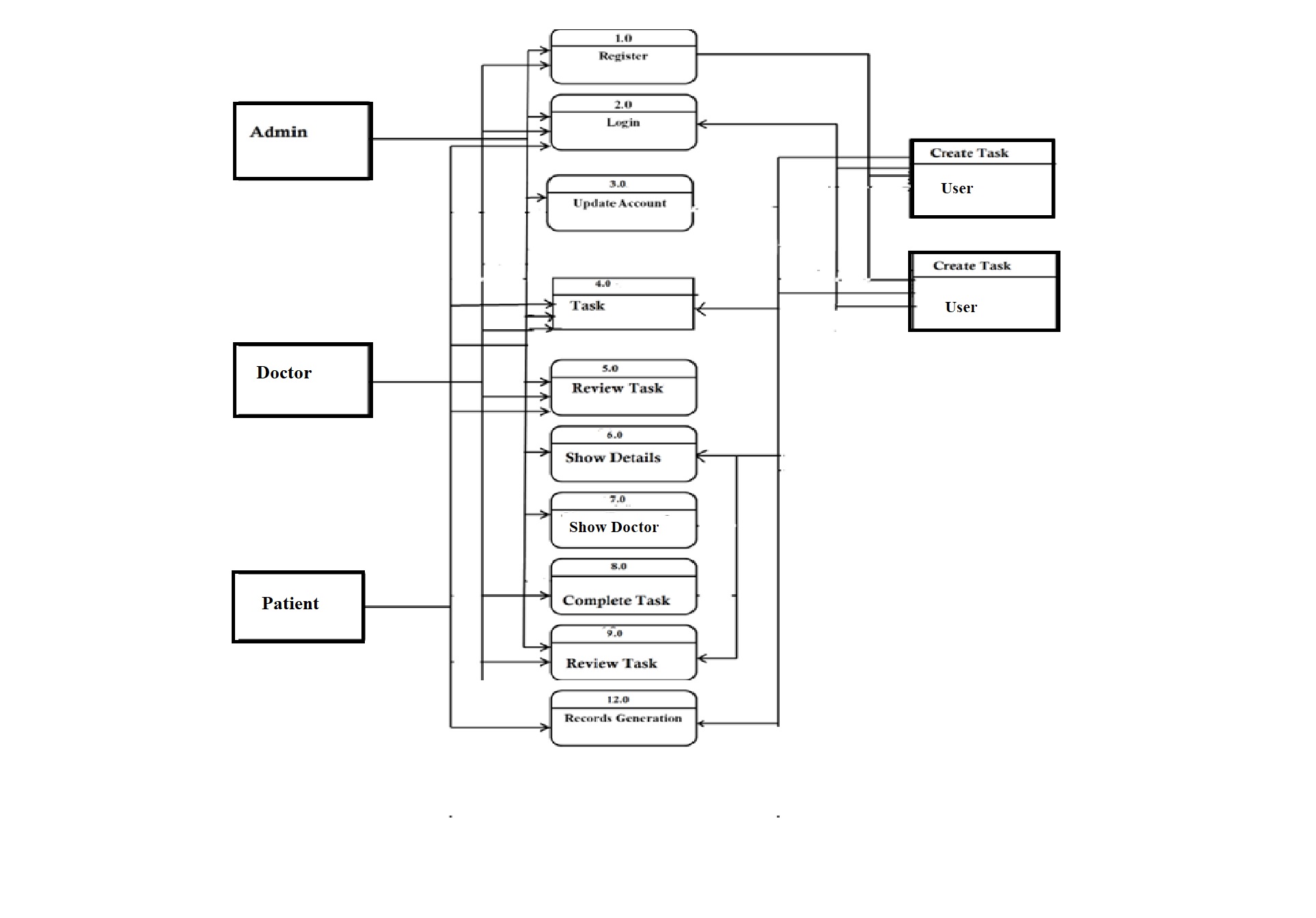


**Data Flow Diagram:**

* **0-Level DFD:**



* In 0-Level DFD, there are three Entities:
* Admin
* Doctor
* Patient
* **1-Level DFD:**

 *fig.1Level DFD*

In 1-Level DFD,

**Admin Entity**: having following processes:

* Login(Process 1.0)
* Create Login(Process 2.0)
* Update Account( process 3.0)
* Appointment Details(process 4.0)
* Show Doctor ( Process 5.0)
* Show Patient ( Process 6.0)
* Pofiles (Process 7.0)

**Doctor Entity** :having following processes:

* Appointment Details (Process 1.0)
* Add Slot(Process 2.0)
* View Appointments(Process 3.0)
* Confirm Appointment(Process 4.0)
* Cancel Appointment (Process 5.0)

**Patient Entity**: having following processes:

* Login(Process 1.0)
* Search Doctor (Process 5.0)
* View Slot(Process 6.0)
* Book Appontment (Process 7.0)
* **Low Level Design:**
* **Database Design:**

**1**] **Table Login**

|  |  |  |  |
| --- | --- | --- | --- |
| Sr.no | UserName | Password | Role |
| 1 | Email id | Any Password | Admin |
| 2 | Email id | Any password | Doctor |
| 3 | Email id | Any Password | Patient |

**2] Table Doctor**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Type** | **Null** | **Default** | **Description** |
| Doctor\_Id | Key , value | No | Null | Doctor ID |
| Full Name | Key , value | No | Null | Name of Doctor |
| Full Address | Key, Value | No | Null | Address of Doctor |
| Contact Number | Key, value | No | Null | Phone number of Doctor |
| Email | Key , value | No | Null | Email of Doctor |
| Password | Key , value | No | Null | Password of Doctor |
| Doctor’s License Number | Key , Value | No | Null | License number of Doctor |
| Clinic Address | Key , Value | No | Null | Clinic Address of Doctor |
| Doctor’s Specialization | Key , Value | No | Null | Specialization of Doctor |

**4] Table Patient**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Type** | **Null** | **Default** | **Description** |
| Patient\_id | Key, Value | No | Null | User Id |
| Patient Name | Key , Value | No | Null | Name of Patient |
| Patient Address | Key , Value | No | Null | Address of Patient |
| Patient Contact Number | Key, Value | No | Null | Contact number of patient |
| Patient Email | Key, Value | No | Null | Email of Patient |
| Patient Password | Key, Value | No | Null | Password of Patient |
| Emergency Contact | Key, Value | No | Null | Emergency Contact of Patient |
| Patient Health problems | Key, Value | No | Null | Patient health problems |

**5] Table Category**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Type** | **Null** | **Default** | **Description** |
| category\_id | Key ,value | NO | Null | Doctor category\_id |
| category\_description | Key ,value | YES | Null | Doctor category\_description |
| category\_image | Binary data | YES | Null | Doctor category\_image |
| category\_image\_path | Key ,value | YES | Null | Doctor category\_image\_path |
| category\_specialization | Key ,value | YES | Null | Doctor category\_specialization |

**6] Table Slot**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Type** | **Null** | **Default** | **Dsecription** |
| slot\_date | DateNow | NO | Null | Date of slot |
| slot\_doctor\_id | Key,value | NO | Null | Doctor Id |
| slot\_end\_time | DateNow | YES | Null | Slot time |
| slot\_id | Key,value | NO | Null | Slot Id |
| slot\_patient\_id | Key,value | YES | Null | Patient Id |
| slot\_start\_time | DateNow | NO | Null | Slot Start Time |
| slot\_end\_time | Key,value | NO | Null | Slot end time |

**6] Table Clinic**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Type** | **Null** | **Default** | **Dsecription** |
| clinic\_id | Key,value | NO | Null | Doctor Clinic Id |
| clinic\_doctor\_id | Key,value | YES | Null | Doctor Id |
| clinic\_image | Binary data | YES | Null | Clinic Image |