Software Requirement Specification Document

For

Hospital Management System

Prepared by

Aniruddh Dwivedi

(23BCE8304)

School of Computer Science and Engineering

10th January 2025

Table of Contents

1. Introduction

This Software Requirement Specification Document for the development of a hospital entails the different aspects involved in the creation and optimization of the Hospital Management System.

1.1. Purpose

The purpose of the hospital management system is to help a hospital or any other large-scale patient care system to be enabled to store patient, employee, and transport records in an organized manner over the course of its operation.

1.2. Intended Customers/Users

The intended users of the designed system are hospital staff and human resources managers who would find themselves eased in their jobs through the use of the provided software product.

1.3. Product Scope

The Hospital Management System aims to streamline the operational and administrative processes within a hospital environment by providing a centralized, user-friendly platform for managing patient records, employee details, appointments, inventory, billing, and other critical hospital functions.

1.4. Outline of the document

The document provides an overview on the workings of the Hospital Management System, including its purpose, intended users, and scope. It outlines the overall description of the product, detailing its perspective, key functions, user classes, and design constraints. Visual representations such as UML, ER, and DFD diagrams are included to illustrate the system architecture and workflows, along with assumptions and dependencies.

2. Overall Description

The Hospital Management System is a standalone software designed to digitize and centralize hospital operations, integrating seamlessly with existing hospital infrastructure, providing a cohesive platform for managing patient records, staff data, appointments, billing, and more.

2.1. Product Perspective

The Hospital Management System is designed to digitize and centralize hospital operations. It integrates with existing hospital infrastructure to provide a cohesive platform for managing patient records, staff data, appointments, billing, inventory, and more. The system is designed to be scalable and readily accessible by different levels of technically inclined individuals.

2.2. Product Functions

The hospital management system provides the following key functions:

- **Patient Management:** Registration and appointment scheduling.
- **Staff Management:** Storing employee details and performance metrics.
- **Billing and Payments:** Automating billing processes and payment tracking.
- **Inventory Management:** Monitoring stock levels of medicines and medical supplies.

2.3. User Classes

Three basic classes of users are decided to be availed to the system's potential user-base, these are:

- Administrators: Monitor the use of the system by sub-classes of users and make executive decisions regarding hospital administrative processes
- Standard Employees: Manage the regular record maintenance of the system's records.
- Technical Staff: Maintain the software and ensure data integrity.
- Contractors: Part-time workers hired to maintain the system

The different user classes may work alongside each other in the system's environment and access information that under normal circumstances, only some parties may be privy to.

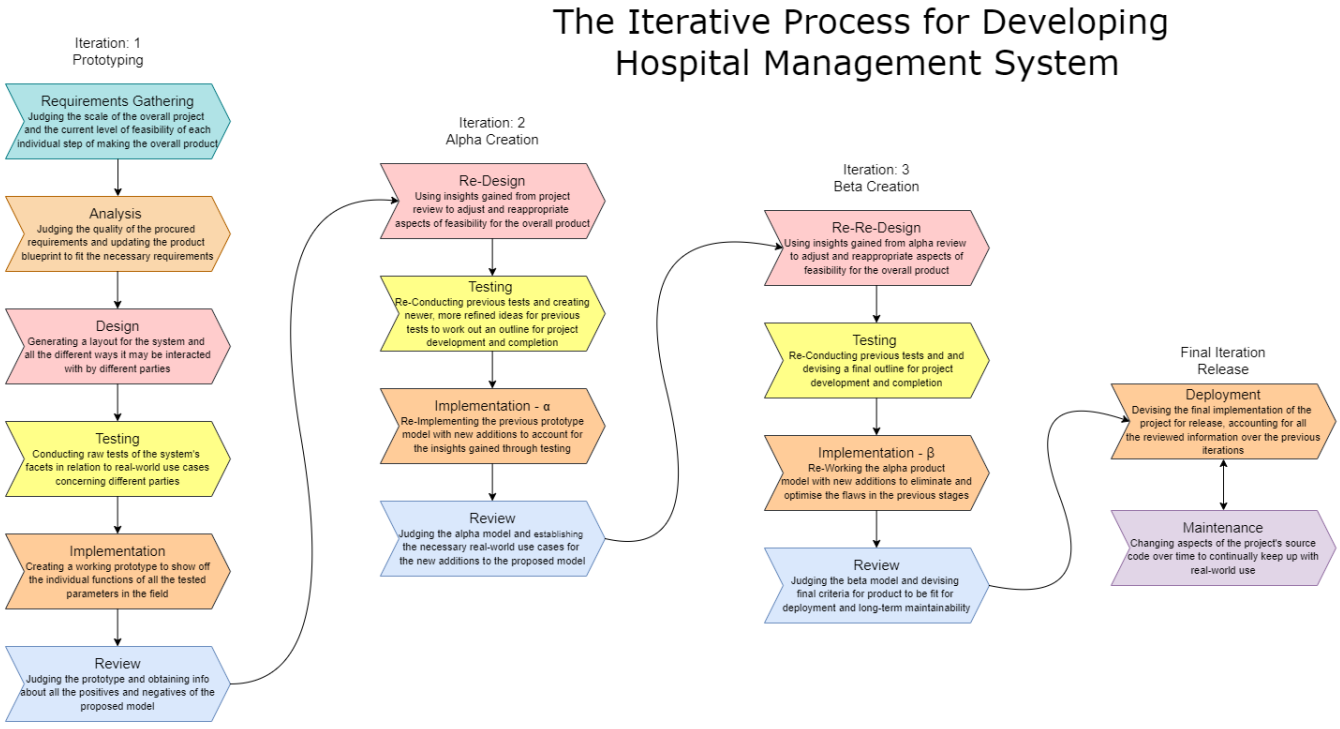
2.4. Design and Implementation Constraints

Data Security: Encryption and secure access are needed control mechanisms to protect sensitive data.

Hardware Limitations: The system must operate efficiently on existing hospital hardware.

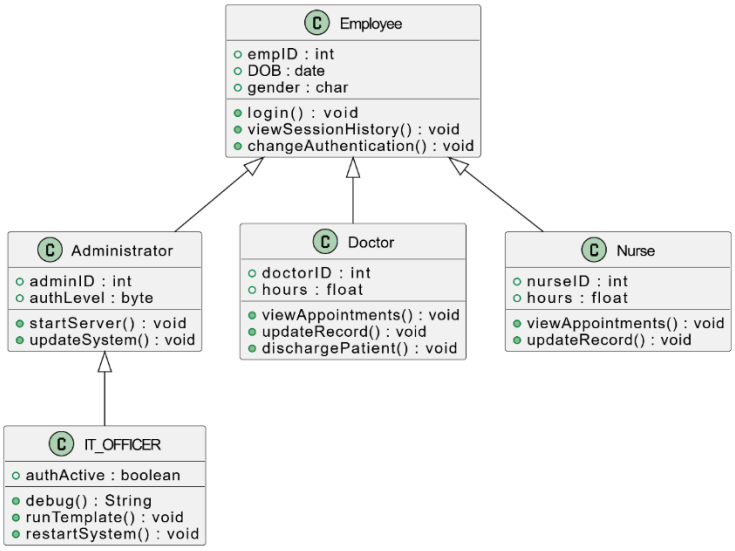
Technical Expertise: Hospital staff may have limited technical knowledge, which may lead to them being unable to manage the workings of the system.

3. Diagrams

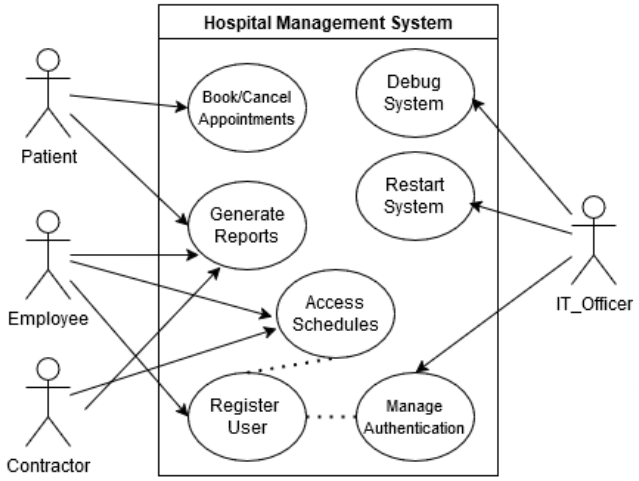


3.0.1: Diagrammatic Representation of the use of Iterative process for system development, design, and release

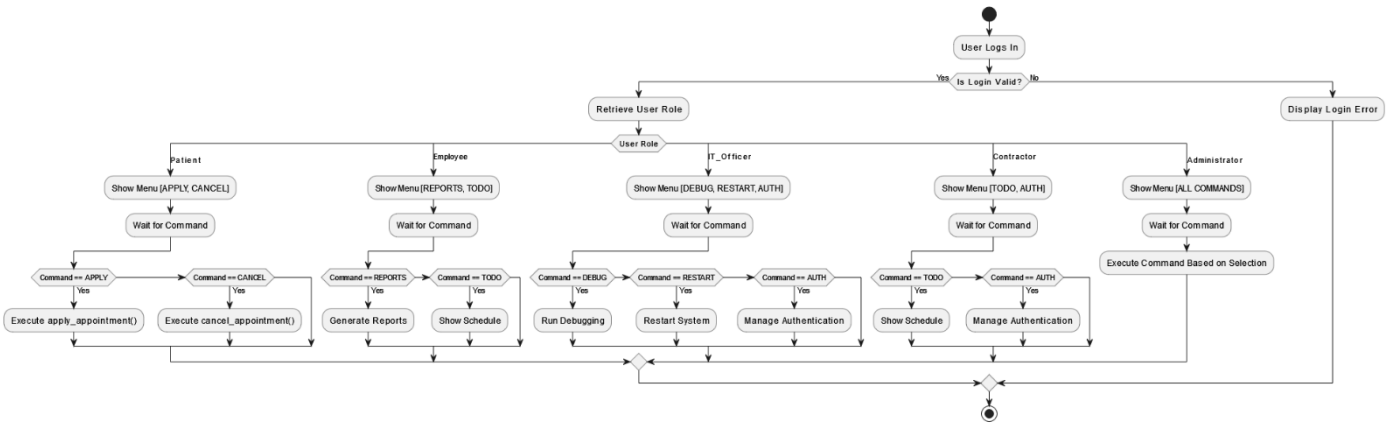
3.1. UML Diagrams



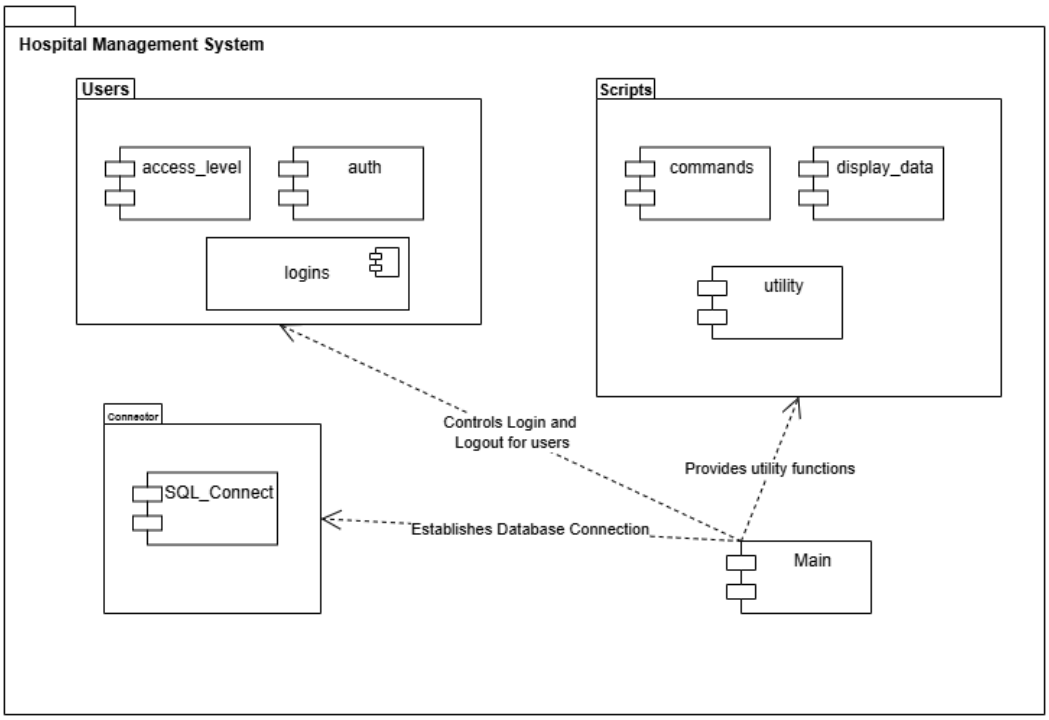
3.1.1. Class-Diagram for representation of the system's user profiles



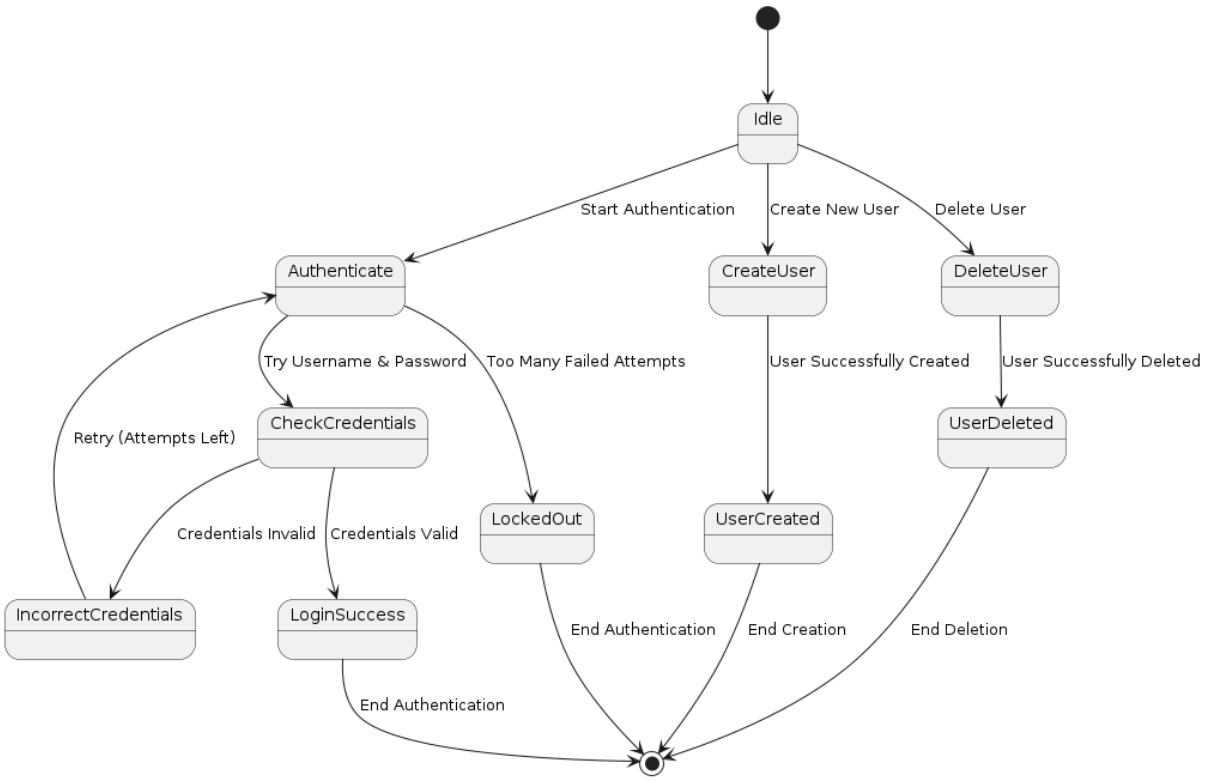
3.1.2. Use-Case diagram describing the different roles played by different user types



3.1.3. Activity Diagram showing system's flow of execution

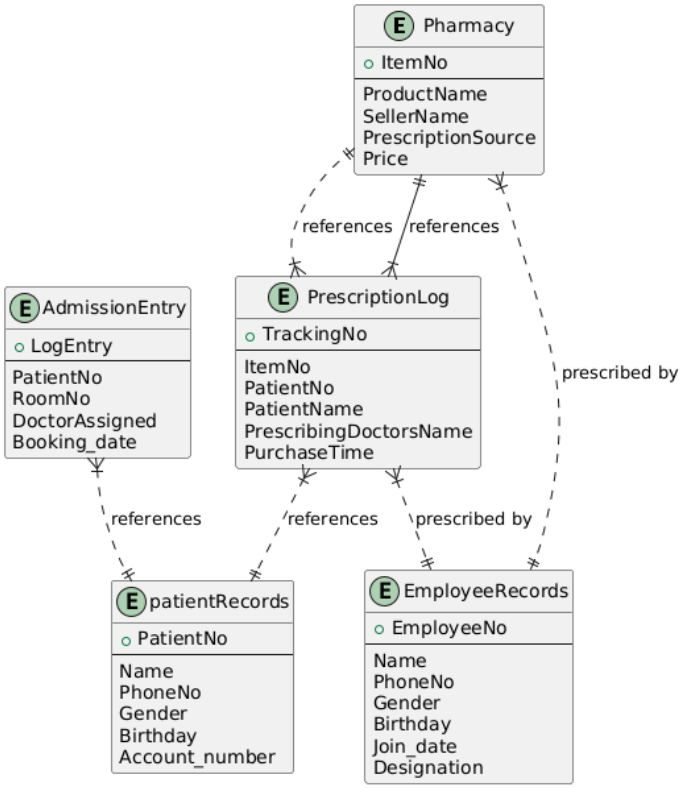


3.1.4. Package diagram describing the system's design

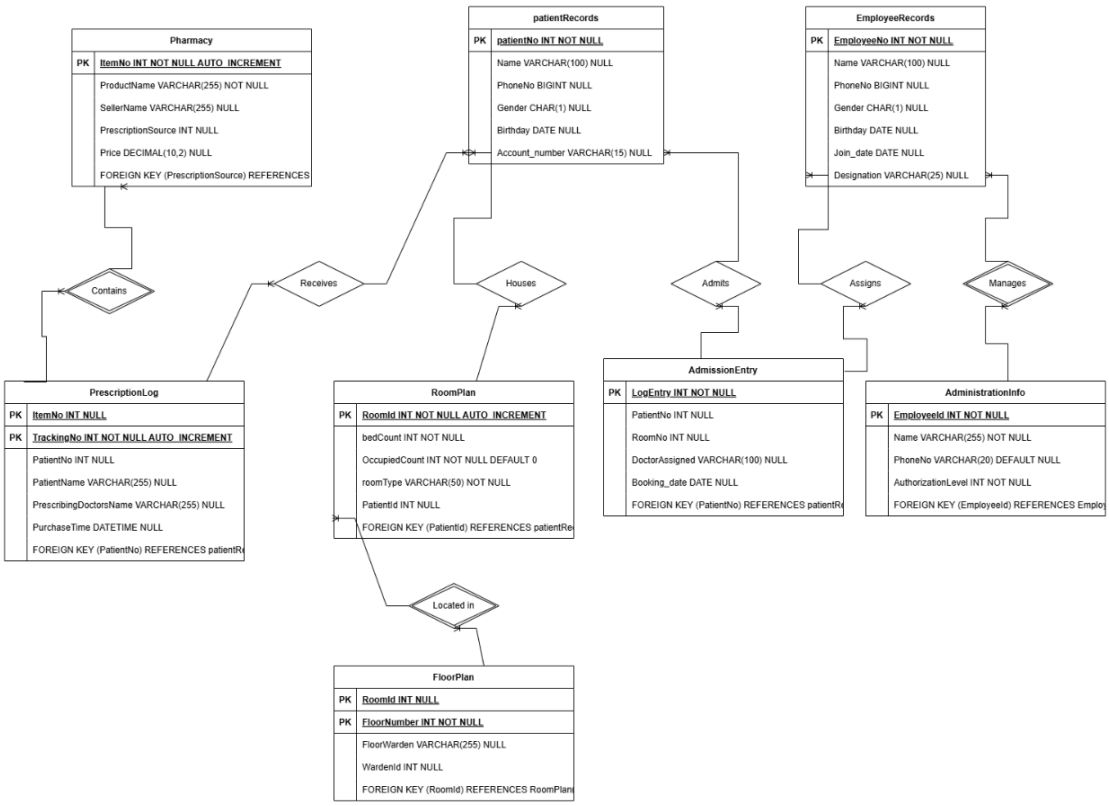


3.1.5. State-Machine Diagram describing the authentication and login process

3.2. ER Diagrams

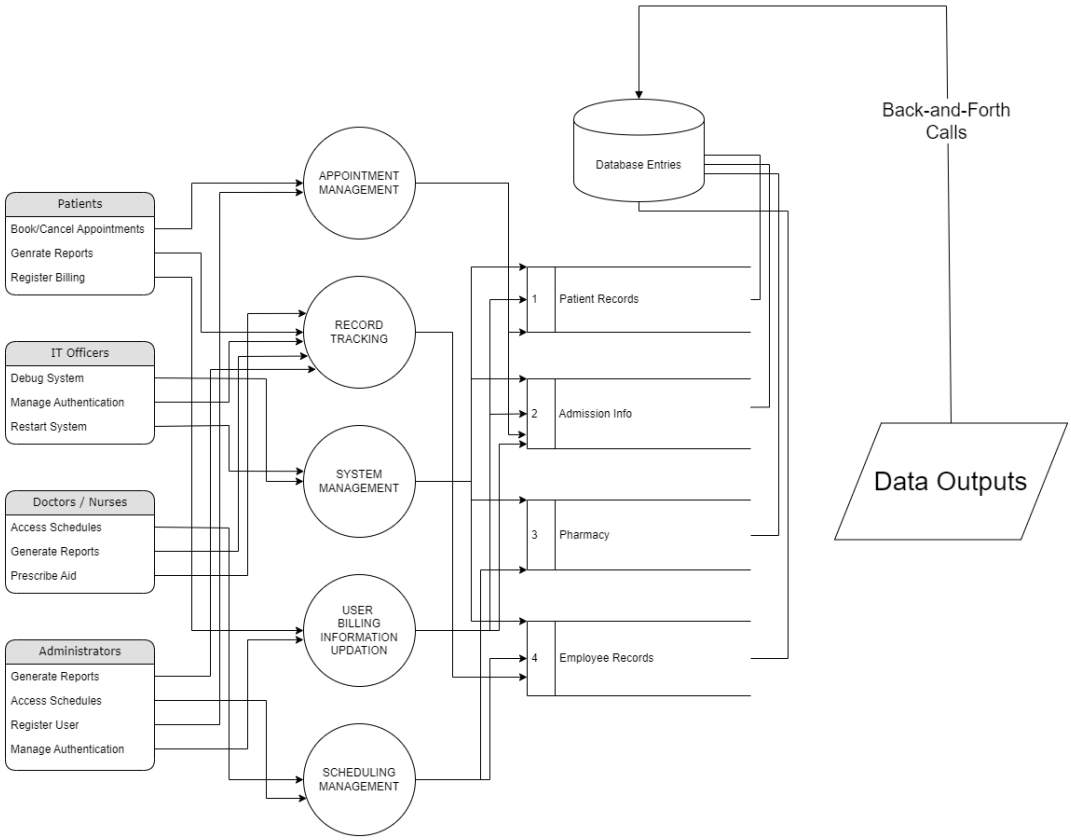


3.2.1. Crow's Foot ER diagram for representing Pharmacy Records

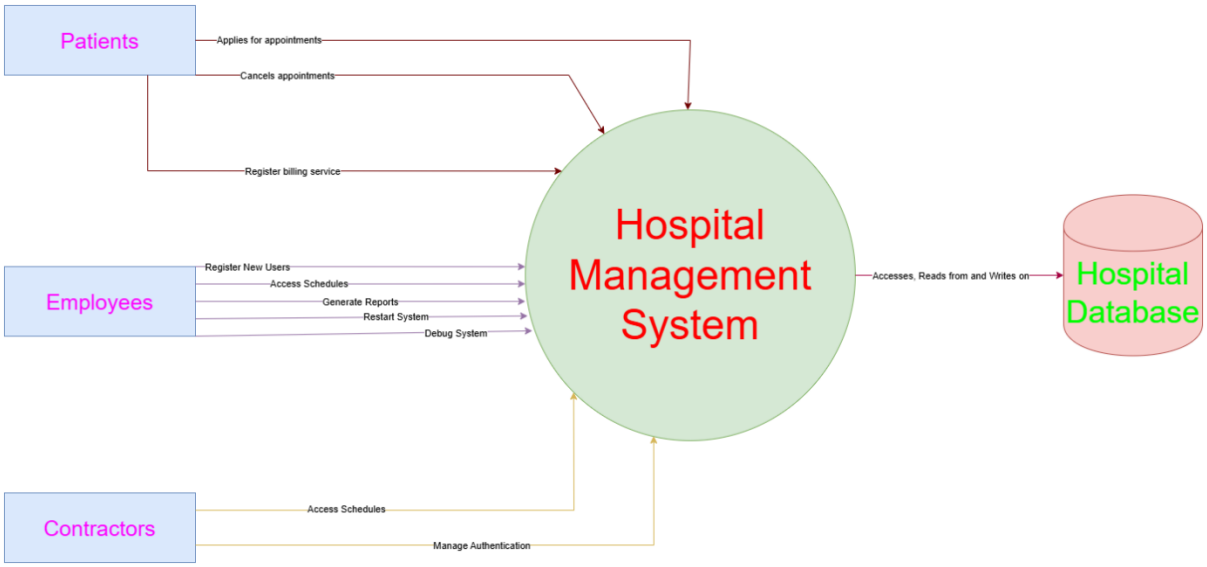


3.2.3. Chen's ER diagram notation for patient record management

3.3. DFD Diagrams



3.3.1. Level – 0 DFD diagram for representing the system's basic interactions



3.3.2. Level - 1 DFD diagram for representing in-depth workings of the system

3.4. Assumptions and Dependencies

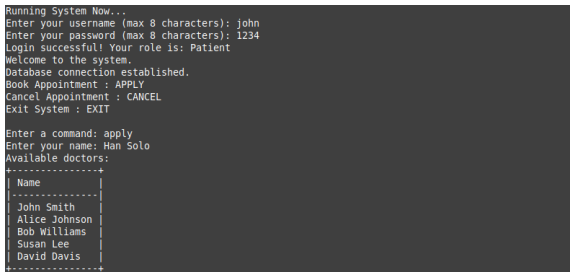
The current model of the system expects a decent level of technical know-how being available to all members of staff and may as such prove quite challenging to have to learn from scratch. The System relies on well-maintained infrastructure that must be regularly updated and be readily accessible for all members of staff at their respective authorization levels.

4. External Interface Requirements / Screenshots

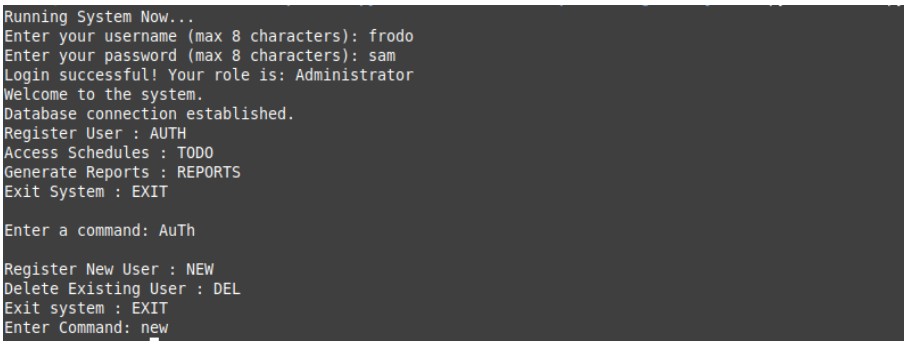
The Hospital Management System is a standalone software designed to digitize and centralize hospital operations, as part of its implementation interface, certain design prospects for different software components and their interactivity are detailed thus.

4.1. User Interfaces

The Present iteration of the system uses a command-line interface for displaying menus and performing system tasks. This allows the user to easily gain access to different system functionalities while keeping up high-level data abstraction as different users have access to only specific sets of commands and only ever access part of the data.



Commands for administrators in the system and their implements for accessibility



Commands for patients in the system and their implements for accessibility

4.2. Hardware Interfaces

The present iteration of the system relies on locally stored data accessed via simple and easy to organise computer architecture, a user need only connect to the physical server instance to access all records and manipulate them, these changes may in future iterations reflect across other concurrently running user instances.

The basic system specifications for running the system are as follows:

- 64-bit processor with CISC configuration
- 2 GB RAM
- 16 MB dedicated VRAM
- 100 MB available disc space
- 1920x1080 or equivalent 16:9 aspect ratio capable displays

Most modern computer systems would qualify these requirements, making the system very easy to access.

4.3. Software Interfaces

Commensurate to the previously specified hardware requirements, the peripheral media may run the system if it qualifies the following software requirements:

- 64-bit Linux or Windows Operating Systems with compatible drivers
- MySQL Server version: 8.0.41-0ubuntu0.24.04.1 (Ubuntu) or equivalent
- Python 3.12.3 with (GCC 13.3.0) on linux or equivalent
- Python modules:

    -tabulate -pymysql -enum

Most modern machines should qualify the stated software requirements and those which do not may be brought up to standard quite accessibly.

4.4. Communication Interfaces

The Present iteration of the system need only to have basic communication peripherals as availed in most present-day computers provided to it, including:

- Microphone-Based single-channel P.A. systems
- Digital display and alert systems
- Building wide well-organised intercoms

With the software front being configured to work seamlessly alongside the necessary hardware, communication interfaces in the system may help to reliably provide users access to different system peripherals.

5. Other Non-Functional Requirements

The Hospital Management System requires several non-functional aspects being met; these requirements may be elucidated as follows...

5.1. Performance Requirements

The system is to be able handle simultaneous access by multiple users, including doctors, nurses, administrators, and patients. For this, the response time for querying patient records should not exceed a few seconds under normal load conditions while supporting a large set of concurrent users without significant degradation in performance. The appointment booking process should be performant under high loads.

5.2. Safety Requirements

The system ensures zero data-loss in the case of power failures and automatically backs up patient records on a daily basis, with modifications being logged and auditable to maintain integrity, following industry-standard data encryption to prevent accidental data corruption.

5.3. Security Requirements

User authentication must be enforced via multi-faceted file security software. Role-based access control (RBAC) is implemented to ensure employees only access necessary information with all sensitive data, including patient records and payment details, being encrypted.

5.4. Software Quality Attributes

The system is highly accessible and made with a focus on scalability, accommodating any increase in patients and employees, all the while staying maintainable with minimal downtime. It is easy to use for the target audience and can be ran on most machines, being very well adjusted for portability.

5.5. Business Policies/Rules

Patient records are stored for all future endowments of every patient in compliance with health regulations, allowing only authorized doctors to approve discharge requests for patients, with them Patients being notified a day in advance before any scheduled appointment, with maintenance being scheduled outside peak hours.

6. Testing

It is essential to check the quality of software before launching to ensure that all requirements are fulfilled. The several tests used to ensure this are listed below.

6.1. Black-Box Testing

The system as tested from the end-user perspective without accessing internal code is done through functional testing to verify that patient registration, appointment booking, and medical record retrieval work as expected. Usability testing ensures that staff and patients can navigate the system easily. Scalability testing is used to judge whether the system may get scaled up or down based on user loads.

6.1.1. Unit Testing

Each module, such as authentication, patient registration, appointment booking, and billing, is tested individually, automated test cases verify the correct functioning of input validation, database queries, and session handling. Mock databases are used to simulate real-world data transactions.

Equivalence Class Partitions			
Variable Type	Invalid Regions <	Valid Regions	Invalid Regions >
Authentication Info	<= 3 Characters	4 - 8 Characters	>= 9 Characters
Age	<= -1	0 - 122	> 122
Employee/Patient Number / ID	<= 0	1 – Max Value	> Max Value

6.1.2. Usability Testing

The system's interface is evaluated for ease of navigation by doctors, nurses, patients, and administrators. Feedback from hospital staff and patients is collected to ensure intuitive workflows. Tests measure navigation efficiency, ensuring users can complete key tasks (e.g., booking an appointment) within minimal numbers of program steps.

Different Use Case parameters may exist due to different types of users, this leads to the need for using a varied set of computations being performed each time an action is performed.

Use Case Parameters				
Condition	Steps	Input	Expected Result	Output
User Login	1. Open Landing Screen 2. Ask for user authentication 3. Validate user	Username and Password	User gets logged in if authentic else the system is locked	User Login is successful
Employee Report Generation	1. Open Reports Screen 2. Ask for Employee ID 3. Present Reports	Employee ID and Report Generation Parameters	Database Records containing Reports	Report Generation is successful
User Registration	1. Access Hospital login 2. Update new user name 3. Add new user password	New user's details	The logins csv file is updated with new user	Authentication Updation is successful
Admission Entry	1. Open Admission Screen 2. Ask for patient details 3. Update database	Necessary Patient Details (like Patient No, Name, DOB etc.)	Database Records for the patient admission are updated	Admission Entry is successful

6.1.3. Scalability Testing

The system is to be tested with 5, 100, and 1000+ concurrent users to measure response time. Load testing ensures database queries, API calls, and appointment scheduling remain responsive under heavy usage, and the system's auto-scaling capabilities are evaluated to ensure smooth performance during peak hospital hours.

6.2. White-Box Testing

White box testing ensures proper internal structure and workings of a software application. The tests are conducted over the source code and are used to design test cases that can verify the correctness of the software at the code level, through code reviews and integration tests.

6.2.1. Integration Testing (Top-Down)

The system is tested module by module, starting with high-level modules (UI, API) and integrating lower-level components (database, authentication). Database integrity is validated by ensuring proper data storage, retrieval, and update mechanisms.

Different system components are taken under consideration like the previously discussed functions for debugging, accessing user data, et al and each is tested under a glass box environment before final integration.

6.2.2. Unit Testing

Individual functions, methods, and database queries are tested for correctness and efficiency, edge cases and boundary testing are performed on critical components such as login authentication, role-based access control, and appointment scheduling and code coverage tools are used to ensure all the code is tested successfully.

The built-in python library unittest is used along with different scripts as shown here:

Code:

```
def debug_system():
    print("Debugging system...")

    file_path = "../main.py"

    if os.name == "nt":  # Windows
        os.startfile(file_path)
    elif os.name == "posix":  # Linux/macOS
        os.system("xdg-open " + file_path)

    exit()

```

Tests:

```
import unittest
from unittest.mock import patch
import os
class TestDebugSystem(unittest.TestCase):
    @patch("os.system")  # Mock os.system
    @patch("builtins.exit")  # Mock exit to prevent stopping tests
    def test_debug_system(self, mock_exit, mock_startfile, mock_system):
        debug_system()

        # Check if correct function was called based on OS
        if os.name == "nt":
            mock_startfile.assert_called_once_with("../scripts.py")
        else:
            mock_system.assert_called_once_with("xdg-open ../scripts.py")

        # Ensure exit() is called
        mock_exit.assert_called_once()

if __name__ == "__main__":
    unittest.main()

```

6.2.3. System Testing

The fully integrated system is tested under real-world conditions to ensure that all modules function together as expected. Security audits being conducted to verify data encryption, access control enforcement, and compliance with healthcare data privacy regulations allow for successful deployment. Performance benchmarks ensure that the system meets response time and uptime requirements.

The procedure is much the same as the previously discussed black-box methodologies.

7. Conclusion

Through thorough testing and comprehensive evaluation, the Hospital Management System ensures efficiency, security, and scalability in managing hospital operations. The non-functional requirements, such as performance, security, and usability, have been carefully considered to provide a seamless experience for all users, including doctors, patients, administrators, and IT.

With a robust design, testing framework, and business policies in place, the HMS is well-equipped to handle the complex demands of modern healthcare environments, ultimately enhancing patient care, streamlining administrative tasks, and improving hospital workflow efficiency.

8. References

The SRS document uses various pieces of information sourced from the following websites alongside the python documentation forums.

UML and ER Diagrams: https://www.uml-diagrams.org/

SRS Fundamentals: https://www.utdallas.edu/~chung/RE/Presentations07S/Team_1_Doc/Documents/SRS4.0.doc

Official Python Documentation: https://www.python.org/doc/

9. Acknowledgement

I would like to express my gratitude to my teacher Dr. Suresh Dara who gave me the opportunity to do this wonderful project. The project helped me learn about many new things about software creation and management. I would also like to thank my group members who helped me create this project in due time.