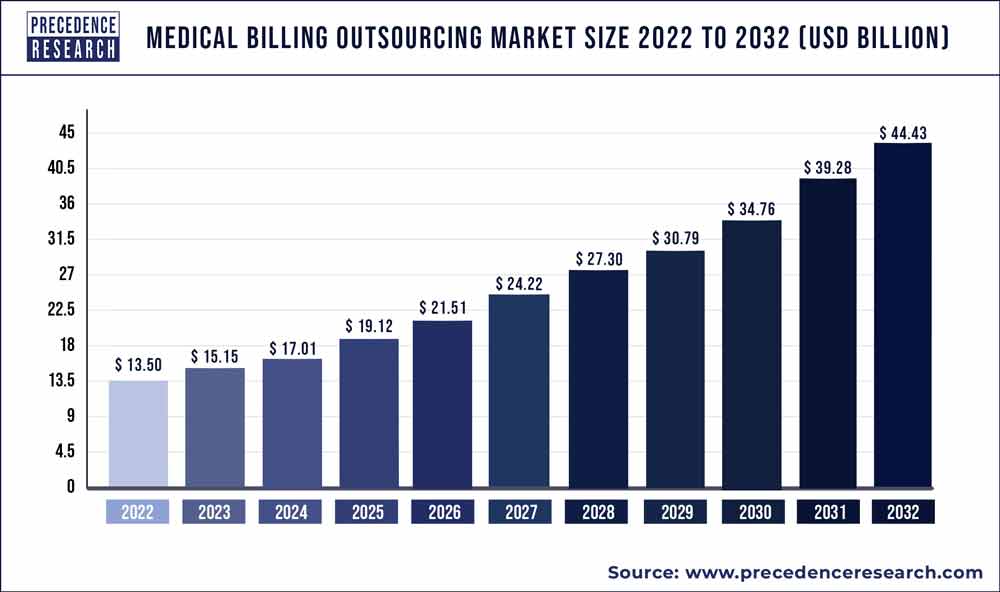
**I.INTRODUCTION**

Digitalization in the medical domain signifies a profound transformation brought about by the integration of cutting-edge technologies into healthcare practices. It involves leveraging information technology, data analytics, artificial intelligence, and various digital tools to enhance the efficiency, accessibility, and overall quality of healthcare services. The Electronic Health Records (EHRs) includes Medical Inventory and billing to shift old paper based health-records to digital health records. Which aims to create is an inventory management system which manages the medicines history. That pharmacist can easily maintain history of tablets and their expiry. E-Health is described by the European Commission As “the use of new communication and information technology to address the needs of people, patients, healthcare practitioners, healthcare providers, and decision-makers"[1]. The integration of inventory and billing ensures patient information and medication is protected. Ongoing training for healthcare professionals ensures effective use of digital tools, contributing to a future where technology empowers healthcare and improves patient outcomes. The graph below shows the growth of the Medical billing market over the past years and future prediction according to precedence research.



**Source:** www.precedenceresearch.com

**Graph 1.1:** Medical billing outsource market size [2]

**1.1 Objective:**

The primary objective of Medical Inventory and Billing is to digitalize and store information and make previous billing and inventory records accessible. Digital billing and inventory improves quality, safety, and efficiency while reducing costs. Through digital billing and inventory system past records are properly maintained. Additionally, Electronic Health Record (EHR) integration with medical inventory billing system addresses data standardization, security, and privacy, contributing to cost reduction and supporting proper management of records for better population health management.

**1.2 Problem Statement:**

Healthcare facilities face numerous challenges in managing their medical inventory and billing processes. Traditional methods of inventory tracking and billing management rely heavily on manual processes, leading to inefficiencies, errors, and increased operational costs. Healthcare suppliers require a smoothed out and integrated solution that automates inventory management. Furthermore, the system should guarantee consistence with regulatory standards, improving financial transparency.

**1.3 Motivation:**

The motivation behind this project stems from the critical need to address existing challenges in the healthcare system, specifically the inadequate recording and accessibility of patients' medication histories. The current traditional paper-based health records and billing system has proven inefficient and fragmented patient information. This inefficiency poses a significant threat to patient medication and overall healthcare quality. Medical inventory billing system offer a digital solution that goes beyond paper documentation. The motivation is not only to address existing challenges but to embrace a future where technology plays a pivotal role in empowering healthcare professionals and improving patient outcome.

The motivation behind our project is to bring the benefits of digitalization to the field of healthcare.

1. **Efficient Operations**: Through our system, we aim to streamline administrative tasks associated with inventory tracking and billing, ultimately enhancing overall efficiency in healthcare facilities.
2. **Financial Intelligence**: By harnessing digital technologies, our system provides healthcare administrators with valuable insights into financial data, facilitating informed decision-making and optimizing financial management practices.
3. **Patient Care Optimization**: Our system's efficient management of medical inventory ensures that healthcare providers have access to essential resources when needed, thereby improving patient care and satisfaction.
4. **Accessibility and Transparency**: Transitioning from traditional paper-based systems to digital solutions enhances accessibility and transparency in healthcare management. Our system enables quick and easy access to inventory and billing information, promoting greater accountability and transparency across the organization.

**1.4 Scope of the Project:**

The Medical Inventory Billing System project aims to create a digital system for managing patient medicine information. This involves choosing a user-friendly inventory and billing system, ensuring secure data, and training healthcare staff. The overall goal is to establish an efficient and secure healthcare data management system, emphasizing improved patient care and compliance with privacy regulations.

**II. LITERATURE REVIEW**

In Paper [3], the utilization of inventory management systems has developed over time, with different industries adopting automated systems to streamline their operations. Initially motivated by Hollerith’s punch card system in the 1930s, modern inventory management systems have become essential tools for tracking and managing inventory efficiently.

In the year 1970 [3] before the approach of automated inventory management systems, manual systems were inclined to mistakes and inefficiencies. The passing of stock information among employees caused errors and wasted time. Furthermore, employees often forgot batch numbers, leading to goods being sold to the wrong customers. Additionally, rearranging items in the warehouse made it difficult for employees to locate goods efficiently. A more streamlined communication and tracking system was deemed necessary to address these issues and improve operational efficiency.

In 2000 [3] the benefits of using Inventory Management Systems became more evident. These systems prevented companies from running out of stocks, eliminated human errors, and reduced time and money spent on manual management. However, manual stock management remained challenging and costly, leading to inefficiencies in inventory management.

In 2010 [3] the importance and purpose of Inventory Management Systems were emphasized. These systems aimed to avoid conflicts between demand and supply, with the introduction of automated warehouse systems improving efficiency in managing goods movement and storage.

In 2015 [3] the report provides a concise overview of the benefits of automated inventory management systems, emphasizing their ability to eliminate human error, reduce time and cost, and provide accurate sales figures through automated estimation.

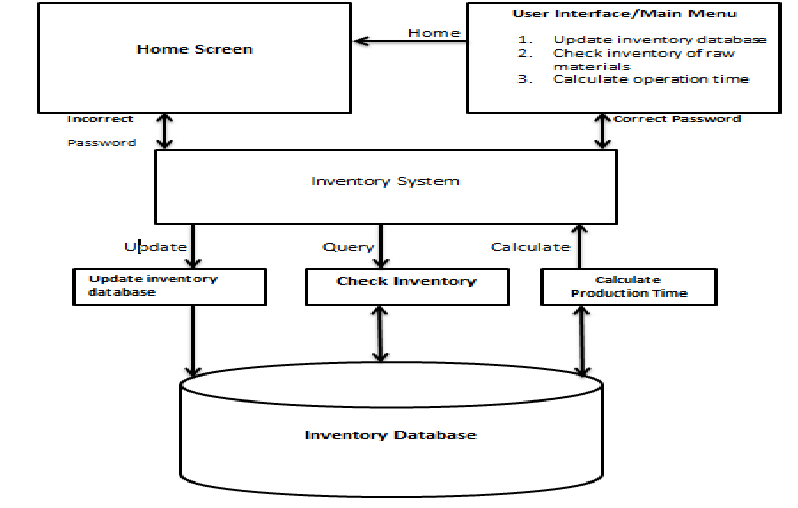
|  |
| --- |

In 2020 RFID technology offers real-time inventory tracking, reducing stock outs and overstocking. Ma et al. (2020) demonstrated its effectiveness in improving inventory accuracy and minimizing stock outs in hospitals. Standardized processes and training are vital for successful RFID implementation.

In 2019 Artificial neural networks (ANN) offer a promising approach to enhancing inventory management in healthcare facilities. By predicting demand for medical supplies, ANN enables better inventory optimization and waste reduction. In a study by Soman et al., ANN proved effective in forecasting medical supply demand within hospital settings, resulting in up to a 25% reduction in inventory levels and significant cost savings for healthcare facilities.

**Table 2.1:** Study of Previous approach and key findings according to the years

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.No.** | **Year** | **Approach** | **Key Finding** | **Gap** |
|  |  |  |  |  |
| 1 | |  |  | | --- | --- | |  | 1930 | | Punch card-based checkout system inspired by Hollerith's idea | First modern check-out system using punch cards for billing and inventory management. | High costs and slow lead time limited popularity. |
| 2 | 1970 | Transition from paper-based inventory systems to automated systems in universities | Automated inventory management improved efficiency and accuracy in tracking stocks. | Challenges with paper-based systems included errors, wasted time, and difficulty in managing stock efficiently. |
| 4 | 2010 | Importance and Purpose of Inventory Management Systems | Avoidance of conflicts between demand and supply. | Introduction of automated warehouse systems improved efficiency in managing goods movement and storage. |
| 5 | 2015 | Development of medical stock inventory management systems in healthcare facilities | Optimization of inventory control, reduction of medical waste, and improvement of resource utilization. | Challenges in accurate tracking of inventory levels, such as data inaccuracy and inadequate training |
| 6 | 2018 | RFID technology for real-time tracking of medical supplies | Effective in improving inventory accuracy and reducing the risk of stock outs. | Need for standardized processes and training to ensure successful implementation. |
| 7 | 2019 | Artificial Neural Networks (ANN) for predicting demand for medical supplies | Effective in predicting demand and reducing inventory levels by up to 25% reduction in inventory levels | Potential for further optimization and integration with other systems. |

**III. SYSTEM ARCHITECTURE**

**Fig 3.1:** Architecture for the Medical inventory and billing system

The first step for the Medical inventory and Billing is the User Interface which serves as the primary point of interaction for healthcare professionals, providing a user-friendly platform accessible through Python GUI. It offers intuitive navigation and clear layout to facilitate easy access to system functionalities.

In the second step is the Application Layer, where the backend logic and database management are handled. This layer encompasses various modules responsible for different aspects of the system's functionality, including inventory management, billing processes, authentication, and data storage.

The Inventory Module within the Application Layer is dedicated to managing medical inventory. It oversees tasks such as tracking medicine stock levels, recording details of medications, managing expiry dates, and facilitating inventory updates. This module ensures efficient management of medicines to meet patient needs while minimizing wastage.

In Step three of Medical Inventory and Billing management, the Billing Module handles invoicing and financial transactions related to medical services and products. It generates bills, processes payments, maintains transaction records, and supports billing reconciliation. This module ensures accurate and timely billing processes to streamline financial management.

The system's data is stored in a relational database, organized into tables for various entities such as Medicines, Patients, and Customers. The database facilitates efficient data management, retrieval, and manipulation, ensuring that essential information is readily accessible for system functionalities.

**IV. FEASIBILITY STUDY**

**Software Requirements:**

**4.1.4 Python**

Python serves as the fundamental language for developing the logic and functionality of the medical billing and inventory system, providing versatility and compatibility with healthcare algorithms. Registered access technology enhances data security by ensuring that only authorized individuals have access to patient records, bolstering user authentication within the Medical Billing and Inventory system.

**4.1.5 Tkinter**

Tkinter is employed to design interfaces tailored for healthcare professionals, offering an intuitive and user-friendly experience that simplifies navigation and interaction with the Medical Billing and Inventory system.

**4.1.6 Firebase**

(Real-time Cloud-Based Storage) Leveraging Firebase, the Medical inventory and database system benefits from real-time data synchronization and secure cloud-based storage, facilitating seamless accessibility to patient records from various devices.

**4.1.7 SQL Database**

An SQL database for a medical inventory and billing system is feasible, providing efficient data storage, retrieval, and management capabilities essential for tracking medical supplies, managing patient billing, and generating accurate reports. Its structured query language allows for seamless integration with the system's backend, ensuring robust data handling and scalability to meet the evolving needs of healthcare facilities.

**Hardware Requirements:**

Computer servers for data processing and storage

**Processor:** A multi-core processor with a clock speed of at least 2.5 Ghz.

**RAM:** At least 16 GB of RAM is recommended for running the system in production. However, the actual RAM requirements will depend on the amount of data being processed and the number of concurrent users accessing the system.

**Operating System:** The System can any operating system that supports Python and Graphical User Interface. However, Windows-based operating systems are recommended for running the system in production.

**Server:** A dedicated server is recommended for running the system in production. The server should have sufficient resources to handle the load generated by the system.

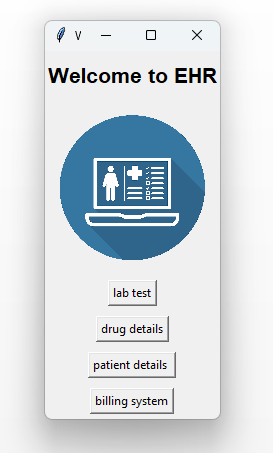
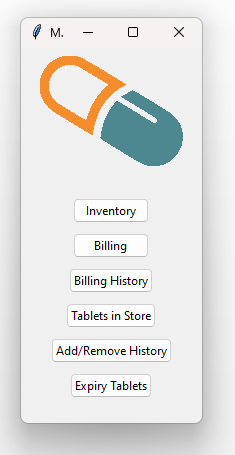
**Storage:** The system requires a minimum of 5 GB of storage for storing data. However, the actual storage requirements will depend on the amount of data being processed and the duration for which the data needs to be stored.

**Backup and Recovery:** The system should be backed up regularly to ensure data integrity and availability. A backup solution should be implemented to ensure that the system can be quickly restored in case of data loss or system failure.

**V. Modules**

**Module -1: User interference for inventory and billing**

The first stage of medical inventory and billing system is accessing user interference .The integration of Electronic Health Record and Medical Inventory and Billing helps to access and digitalize all the medical requirement at one place .The medical tablet inventory and billing system helps to keep all data organized.

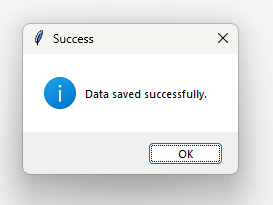
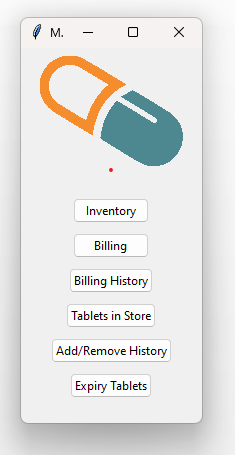
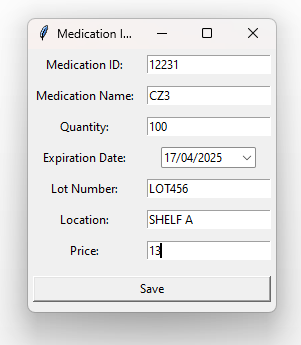


**Fig 5.1.1:** Accessing to the billing system

After accessing to the billing system, as shown in **Fig 5.1.1**, the billing interface shows multiple options for inventor registering, inventory history, checking expiry of the tablets, removal of expiry tablets and in the billing we have registration of the user and checking details of the registered user. These data later on helps for the data for billing of medicines for patient.

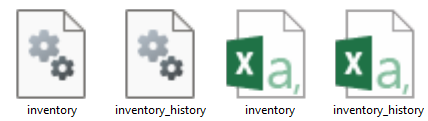
**Module -2: Registering Medications for Inventory**

The inventory registration helps pharmacist to register details of the medicines with other required details like medication ID, Medication Name, Quantity, Expiry Date, LOT Number, Location, price. The LOT number helps for the pharmacist to know about the batch of tablets received after manufacture. Location of tablets ensures proper placement of the tables, helps to locate medicines easily by location of shelf where the tablets are stored.



**Fig 5.2.1:** Entering medicine data

The registered data is stored in databases file , after registering of tablets the data is stored in two different database one inventory.db which shows live data of medication and inventory\_history.db which stores all the records of the medicines entered and removed , the inventory\_history stores all the inventory history of the medicines. The same information is also saved in CSV file for easy access and visibility of medication.

****

**Fig 5.2.2:** Database file and CSV file created after Registration

**Code for Registering Patient Details**

def register\_patient(self):

register\_window = tk.Toplevel(self.root)

register\_window.title("Register New Patient")

# Registration form elements

patient\_id\_label = ttk.Label(register\_window, text="Patient ID:")

patient\_id\_label.grid(row=0, column=0, padx=5, pady=5)

patient\_id\_entry = ttk.Entry(register\_window)

patient\_id\_entry.grid(row=0, column=1, padx=5, pady=5)

first\_name\_label = ttk.Label(register\_window, text="First Name:")

first\_name\_label.grid(row=1, column=0, padx=5, pady=5)

first\_name\_entry = ttk.Entry(register\_window)

first\_name\_entry.grid(row=1, column=1, padx=5, pady=5)

last\_name\_label = ttk.Label(register\_window, text="Last Name:")

last\_name\_label.grid(row=2, column=0, padx=5, pady=5)

last\_name\_entry = ttk.Entry(register\_window)

last\_name\_entry.grid(row=2, column=1, padx=5, pady=5)

dob\_label = ttk.Label(register\_window, text="Date of Birth:")

dob\_label.grid(row=3, column=0, padx=5, pady=5)

dob\_cal = DateEntry(register\_window, width=12, background='darkblue', foreground='white', borderwidth=2, date\_pattern='dd/mm/yyyy')

dob\_cal.grid(row=3, column=1, padx=5, pady=5)

gender\_label = ttk.Label(register\_window, text="Gender:")

gender\_label.grid(row=4, column=0, padx=5, pady=5)

gender\_var = tk.StringVar(register\_window)

gender\_var.set("Male") # Default value

gender\_option = ttk.OptionMenu(register\_window, gender\_var, "Male", "Female", "Others")

gender\_option.grid(row=4, column=1, padx=5, pady=5)

email\_label = ttk.Label(register\_window, text="Email:")

email\_label.grid(row=5, column=0, padx=5, pady=5)

email\_entry = ttk.Entry(register\_window)

email\_entry.grid(row=5, column=1, padx=5, pady=5)

phone\_label = ttk.Label(register\_window, text="Phone:")

phone\_label.grid(row=6, column=0, padx=5, pady=5)

phone\_entry = ttk.Entry(register\_window)

phone\_entry.grid(row=6, column=1, padx=5, pady=5)

def save\_registration():

# Get data from entry fields

patient\_id = patient\_id\_entry.get()

first\_name = first\_name\_entry.get()

last\_name = last\_name\_entry.get()

date\_of\_birth = dob\_cal.get\_date().strftime('%d/%m/%Y')

gender = gender\_var.get()

email = email\_entry.get()

phone = phone\_entry.get()

# Validate input

if not all([patient\_id, first\_name, last\_name, date\_of\_birth, gender, email, phone]):

messagebox.showerror("Error", "Please fill in all fields.")

return

# Save to SQLite database

self.cursor\_patient\_details.execute('''CREATE TABLE IF NOT EXISTS Patients

(PatientID INTEGER PRIMARY KEY, FirstName TEXT, LastName TEXT, DateOfBirth TEXT, Gender TEXT, Email TEXT, Phone TEXT)''')

self.cursor\_patient\_details.execute("INSERT INTO Patients VALUES (?, ?, ?, ?, ?, ?, ?)",

(int(patient\_id), first\_name, last\_name, date\_of\_birth, gender, email, phone))

self.conn\_patient\_details.commit()

messagebox.showinfo("Success", "Patient registered successfully.")

register\_window.destroy() # Close the registration window after successful registration

# Save button for registration

save\_button = ttk.Button(register\_window, text="Save", command=save\_registration)

save\_button.grid(row=7, column=0, columnspan=2, padx=5, pady=10, sticky="WE")

class BillingSystem:

def \_\_init\_\_(self, root, cursor\_patient\_details):

self.root = root

self.cursor\_patient\_details = cursor\_patient\_details # Store the cursor for patient details

self.root.title("Medical Billing System")

# Connect to the SQLite database for inventory

self.conn\_inventory = sqlite3.connect('inventory.db', check\_same\_thread=False)

self.cursor\_inventory = self.conn\_inventory.cursor()

# Connect to the SQLite database for billing history

self.conn\_billing\_history = sqlite3.connect('billing\_history.db', check\_same\_thread=False)

self.cursor\_billing\_history = self.conn\_billing\_history.cursor()

# Label and Entry for Medication ID

self.label\_medication\_id = ttk.Label(root, text="Medication ID:")

self.label\_medication\_id.grid(row=0, column=0, padx=5, pady=5)

self.entry\_medication\_id = ttk.Entry(root)

self.entry\_medication\_id.grid(row=0, column=1, padx=5, pady=5)

# Label and Entry for Quantity

self.label\_quantity = ttk.Label(root, text="Quantity:")

self.label\_quantity.grid(row=3, column=0, padx=5, pady=5)

self.entry\_quantity = ttk.Entry(root)

self.entry\_quantity.grid(row=3, column=1, padx=5, pady=5)

# Button to add medication to bill

self.button\_add\_medication = ttk.Button(root, text="Add Medication", command=self.add\_medication\_to\_bill)

self.button\_add\_medication.grid(row=4, column=0, padx=5, pady=5)

# Button to remove medication from bill

self.button\_remove\_medication = ttk.Button(root, text="Remove Medication", command=self.remove\_medication\_from\_bill)

self.button\_remove\_medication.grid(row=4, column=1, padx=5, pady=5)

# Button to print receipt

self.button\_print\_receipt = ttk.Button(root, text="Print Receipt", command=self.print\_receipt)

self.button\_print\_receipt.grid(row=4, column=2, padx=5, pady=5)

# Treeview to display bill

self.columns = ("id", "medication name", "price", "quantity", "PRICE", "GST", "Total") # Added Total column

self.treeview = ttk.Treeview(root, columns=self.columns, show="headings")

for col in self.columns:

self.treeview.heading(col, text=col)

self.treeview.grid(row=5, column=0, columnspan=3, padx=5, pady=5)

# Total price label

self.label\_total\_price = ttk.Label(root, text="Total Price:")

self.label\_total\_price.grid(row=6, column=0, padx=5, pady=5)

self.total\_price\_var = tk.DoubleVar() # Change to DoubleVar

self.label\_total\_price\_value = ttk.Label(root, textvariable=self.total\_price\_var)

self.label\_total\_price\_value.grid(row=6, column=1, padx=5, pady=5)

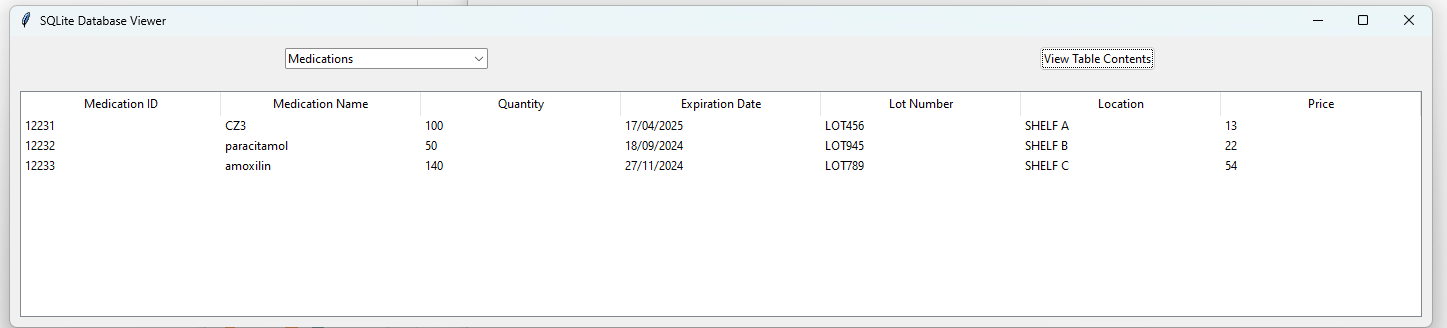
# Initialize bill details

self.bill = []

self.total\_price = 0

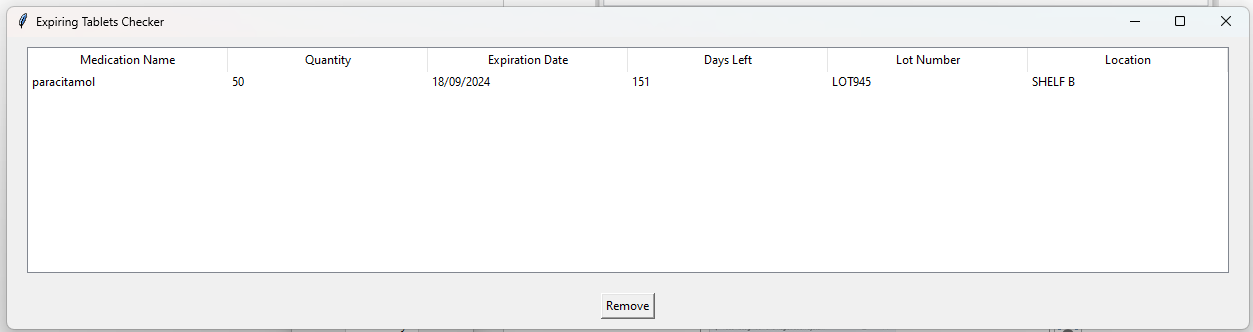
self.patient\_id = None

The medication details are collected as shown in **Fig 5.2.1** the details in order to be seen by the pharmacist must access to “tablets in Store” which shows live overview of tables present in the store.

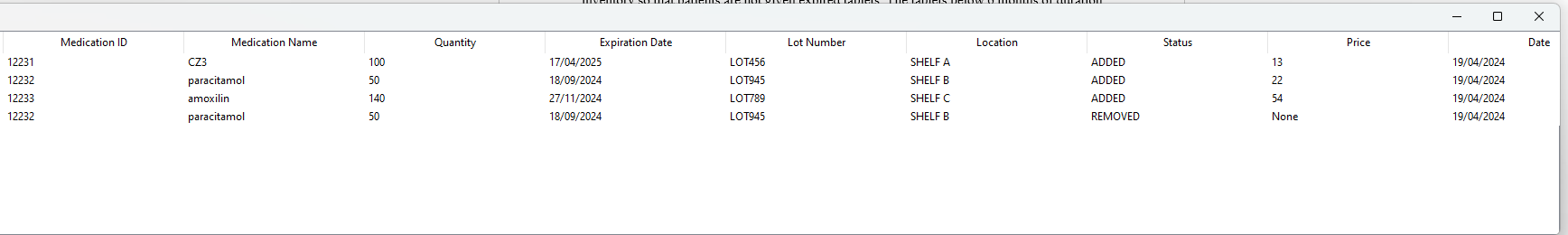


**Fig 5.2.3:** Registered medicines in database.

**Module -3: Removal of Expired tablets and viewing its history**

The main problem in medical inventory management is to remove expired tablets from the inventory so that patients are not given expired tablets. The tablets below 6 months of duration needs to be removed from the inventory and only tablets above 6 months must be kept in inventory.

**Fig 5.3.1:** Shows the near to expire tables in the inventory

As shown in **Fig 5.3.1** the expired tables are visible in list, the list shows paracetamol which has 151 days left from the current day. When remove button is pressed the tablet is removed from the list. The table is also removed from the inventory.db and the removed tablet is added to the inventory\_history.db

**Fig 5.3.2:** History of the removed medicine

**Code For Checking Medicine History**def view\_history():

# Connect to the SQLite database

conn = sqlite3.connect("inventory\_history.db")

cursor = conn.cursor()

# Create the InventoryHistory table if it doesn't exist

cursor.execute('''CREATE TABLE IF NOT EXISTS InventoryHistory

(MedicationID INTEGER, MedicationName TEXT, Quantity INTEGER,

ExpirationDate TEXT, LotNumber TEXT, Location TEXT, ADDED TEXT, CurrentDate TEXT)''') # Added CurrentDate column

conn.commit()

# Fetch data from the database

cursor.execute("SELECT \* FROM InventoryHistory")

rows = cursor.fetchall()

# Create and populate a new window to display the data

history\_window = tk.Toplevel(root)

history\_window.title("Inventory History Details")

treeview = ttk.Treeview(history\_window)

treeview.pack(expand=True, fill="both")

# Add columns to the Treeview

columns = ("Medication ID", "Medication Name", "Quantity", "Expiration Date", "Lot Number", "Location", "Status", "Price","Date") # Added "Current Date" column

treeview["columns"] = columns

for col in columns:

treeview.heading(col, text=col)

# Populate the Treeview with data

for row in rows:

treeview.insert("", "end", values=row)

# Close the database connection

conn.close()

# Create GUI

root = tk.Tk()

root.title("Inventory History Viewer")

# Button to view history

view\_button = tk.Button(root, text="View Inventory History", command=view\_history)

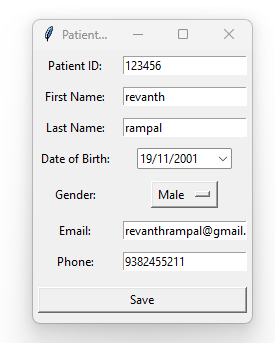
view\_button.pack(padx=10, pady=10)

root.mainloop()

**Module -4: Billing and Registering of Patient**

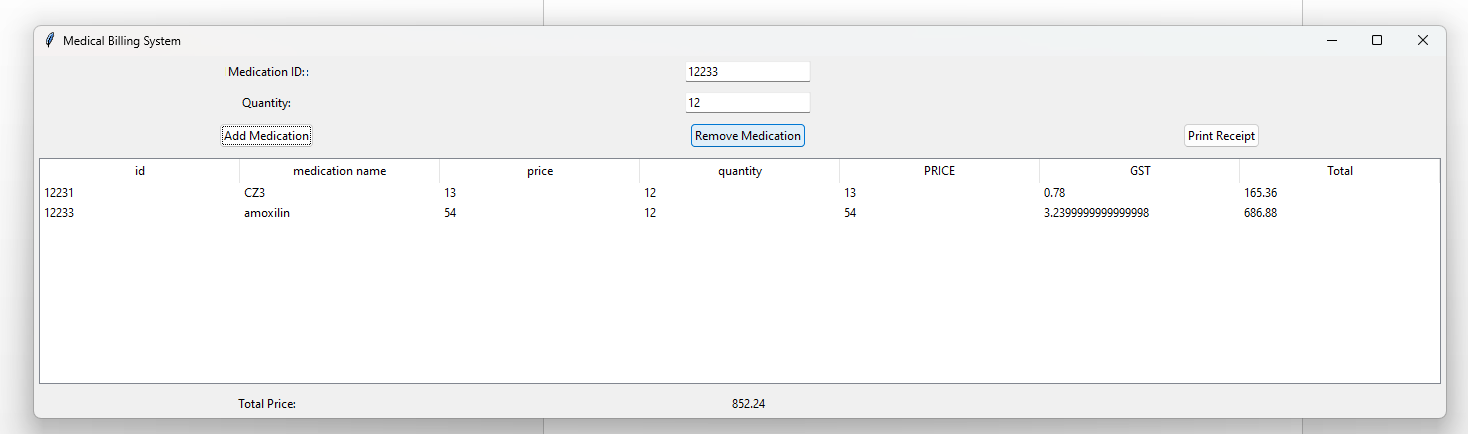
In this phase of the project, we focus on registering of user in case of there is no user registered.

The registering of users asks basic information of the patient which is used further for medical billing. The registerd user is stored in separate database patient details.db the registration of new use is shown below in **Fig 5.4.1** where the information like patient id, name ,date of birth,gender,email and phone number.



**Fig 5.4.1:** Registration of patient details

The registered patient details are saved in separate database file patient details. These database file is created and every new patient details are appended for the registration of new patient. The below figure **Fig 5.4.2** shows the billing interface of medical tablet id and its quantity by using past inventory data the price of the quantity is calculated.



**Fig 5.4.2:** Billing of Medical Tablets

**Code for Uploading Data**

class BillingSystem:

def \_\_init\_\_(self, root):

self.root = root

self.root.title("Medical Billing System")

# Connect to the SQLite database for inventory

self.conn\_inventory = sqlite3.connect('inventory.db')

self.cursor\_inventory = self.conn\_inventory.cursor()

# Connect to the SQLite database for billing history

self.conn\_billing\_history = sqlite3.connect('billing\_history.db')

self.cursor\_billing\_history = self.conn\_billing\_history.cursor()

# Label and Entry for Medication ID

self.label\_medication\_id = ttk.Label(root, text="Medication ID:")

self.label\_medication\_id.grid(row=0, column=0, padx=5, pady=5)

self.entry\_medication\_id = ttk.Entry(root)

self.entry\_medication\_id.grid(row=0, column=1, padx=5, pady=5)

# Label and Entry for Quantity

self.label\_quantity = ttk.Label(root, text="Quantity:")

self.label\_quantity.grid(row=3, column=0, padx=5, pady=5)

self.entry\_quantity = ttk.Entry(root)

self.entry\_quantity.grid(row=3, column=1, padx=5, pady=5)

# Button to add medication to bill

self.button\_add\_medication = ttk.Button(root, text="Add Medication", command=self.add\_medication\_to\_bill)

self.button\_add\_medication.grid(row=4, column=0, padx=5, pady=5)

# Button to remove medication from bill

self.button\_remove\_medication = ttk.Button(root, text="Remove Medication", command=self.remove\_medication\_from\_bill)

self.button\_remove\_medication.grid(row=4, column=1, padx=5, pady=5)

# Button to print receipt

self.button\_print\_receipt = ttk.Button(root, text="Print Receipt", command=self.print\_receipt)

self.button\_print\_receipt.grid(row=4, column=2, padx=5, pady=5)

# Treeview to display bill

self.columns = ("Medication Name", "Price", "Quantity", "Single Price", "GST", "Total Price")

self.treeview = ttk.Treeview(root, columns=self.columns, show="headings")

for col in self.columns:

self.treeview.heading(col, text=col)

self.treeview.grid(row=5, column=0, columnspan=3, padx=5, pady=5)

# Total price label

self.label\_total\_price = ttk.Label(root, text="Total Price:")

self.label\_total\_price.grid(row=6, column=0, padx=5, pady=5)

self.total\_price\_var = tk.DoubleVar() # Change to DoubleVar

self.label\_total\_price\_value = ttk.Label(root, textvariable=self.total\_price\_var)

self.label\_total\_price\_value.grid(row=6, column=1, padx=5, pady=5)

# Initialize bill details

self.bill = []

self.total\_price = 0

def add\_medication\_to\_bill(self):

medication\_id = self.entry\_medication\_id.get()

quantity = self.entry\_quantity.get()

# Fetch medication details from the inventory database

self.cursor\_inventory.execute("SELECT MedicationName, Price, Quantity FROM Medications WHERE MedicationID=?", (medication\_id,))

medication\_details = self.cursor\_inventory.fetchone()

if medication\_details:

medication\_name, price, available\_quantity = medication\_details

single\_price = price

gst = price \* 0.06 # 6% GST

# Check if entered quantity is valid

try:

quantity = int(quantity)

if quantity <= 0:

raise ValueError("Quantity must be a positive integer")

except ValueError:

messagebox.showerror("Error", "Quantity must be a positive integer")

return

# Check if entered quantity exceeds available quantity

total\_quantity\_in\_bill = sum(item[3] for item in self.bill if item[0] == medication\_id) # Sum of quantity of same medication in bill

if total\_quantity\_in\_bill + quantity > available\_quantity:

messagebox.showerror("Error", "Total quantity exceeds available stock")

return

total\_price = (float(quantity) \* price) + (float(quantity) \* gst) # Total price including GST

# Update the bill

self.bill.append((medication\_id, medication\_name, price, quantity, single\_price, gst, total\_price))

# Update the total price

self.total\_price += total\_price

# Update the treeview

self.treeview.insert("", "end", values=(medication\_name, price, quantity, single\_price, gst, total\_price))

# Update total price label with formatted value

self.total\_price\_var.set("{:.2f}".format(self.total\_price)) # Format to display only two decimal places

else:

messagebox.showerror("Error", f"Medication with ID {medication\_id} not found.")

def remove\_medication\_from\_bill(self):

selected\_item = self.treeview.selection()

if selected\_item:

item\_values = self.treeview.item(selected\_item, 'values')

total\_price = item\_values[5] # Total price is at index 5

# Ensure total\_price is a float

if isinstance(total\_price, str):

total\_price = float(total\_price)

self.total\_price -= total\_price

self.total\_price\_var.set("{:.2f}".format(self.total\_price)) # Format to display only two decimal places

self.treeview.delete(selected\_item)

for index, item in enumerate(self.bill):

if item[1] == item\_values[0]:

self.bill.pop(index)

break

else:

messagebox.showerror("Error", "Please select an item to remove from the bill.")

def print\_receipt(self):

try:

with sqlite3.connect('inventory.db') as conn\_inventory:

cursor\_inventory = conn\_inventory.cursor()

with sqlite3.connect('billing\_history.db') as conn\_billing\_history:

cursor\_billing\_history = conn\_billing\_history.cursor()

for item in self.bill:

medication\_id, medication\_name, price, quantity, single\_price, gst, total\_price = item

billing\_date = datetime.now().strftime('%Y-%m-%d %H:%M:%S') # Get current date and time

cursor\_billing\_history.execute("INSERT INTO BillingHistory (MedicationName, Quantity, Price, TotalPrice, BillingDate) VALUES (?, ?, ?, ?, ?)",

(medication\_name, quantity, price, total\_price, billing\_date))

# Update the inventory database

cursor\_inventory.execute("UPDATE Medications SET Quantity = Quantity - ? WHERE MedicationID = ?", (quantity, medication\_id))

except sqlite3.Error as e:

messagebox.showerror("Database Error", str(e))

else:

messagebox.showinfo("Receipt Printed", "Receipt printed successfully")

# Clear the bill and reset total price

self.bill = []

self.total\_price = 0

self.total\_price\_var.set(0.0)

self.treeview.delete(\*self.treeview.get\_children())

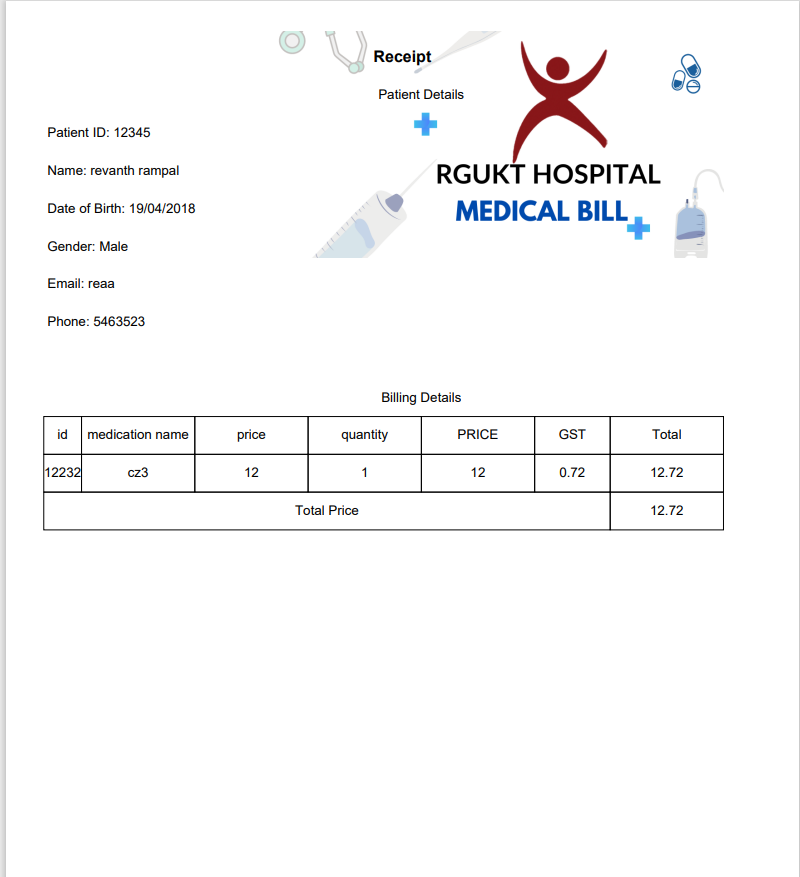
if \_\_name\_\_ == "\_\_main\_\_":

root = tk.Tk()

billing\_system = BillingSystem(root)

root.mainloop()

Once the medicines are entered in the print receipt button generates pdf on the name of patient and the medicines are generated in table format. The below shows the pdf generated by using patient details.

****

**Fig 5.4.3:** Printing of the generated bill.

**Code for pdf Generation:**

class BillingSystem:

def \_\_init\_\_(self, root):

self.root = root

self.root.title("Medical Billing System")

# Connect to the SQLite database for inventory

self.conn\_inventory = sqlite3.connect('inventory.db')

self.cursor\_inventory = self.conn\_inventory.cursor()

# Connect to the SQLite database for billing history

self.conn\_billing\_history = sqlite3.connect('billing\_history.db')

self.cursor\_billing\_history = self.conn\_billing\_history.cursor()

# Label and Entry for Medication ID

self.label\_medication\_id = ttk.Label(root, text="Medication ID:")

self.label\_medication\_id.grid(row=0, column=0, padx=5, pady=5)

self.entry\_medication\_id = ttk.Entry(root)

self.entry\_medication\_id.grid(row=0, column=1, padx=5, pady=5)

# Label and Entry for Quantity

self.label\_quantity = ttk.Label(root, text="Quantity:")

self.label\_quantity.grid(row=3, column=0, padx=5, pady=5)

self.entry\_quantity = ttk.Entry(root)

self.entry\_quantity.grid(row=3, column=1, padx=5, pady=5)

# Button to add medication to bill

self.button\_add\_medication = ttk.Button(root, text="Add Medication", command=self.add\_medication\_to\_bill)

self.button\_add\_medication.grid(row=4, column=0, padx=5, pady=5)

# Button to remove medication from bill

self.button\_remove\_medication = ttk.Button(root, text="Remove Medication", command=self.remove\_medication\_from\_bill)

self.button\_remove\_medication.grid(row=4, column=1, padx=5, pady=5)

# Button to print receipt

self.button\_print\_receipt = ttk.Button(root, text="Print Receipt", command=self.print\_receipt)

self.button\_print\_receipt.grid(row=4, column=2, padx=5, pady=5)

# Treeview to display bill

self.columns = ("Medication Name", "Price", "Quantity", "Single Price", "GST", "Total Price")

self.treeview = ttk.Treeview(root, columns=self.columns, show="headings")

for col in self.columns:

self.treeview.heading(col, text=col)

self.treeview.grid(row=5, column=0, columnspan=3, padx=5, pady=5)

# Total price label

self.label\_total\_price = ttk.Label(root, text="Total Price:")

self.label\_total\_price.grid(row=6, column=0, padx=5, pady=5)

self.total\_price\_var = tk.DoubleVar() # Change to DoubleVar

self.label\_total\_price\_value = ttk.Label(root, textvariable=self.total\_price\_var)

self.label\_total\_price\_value.grid(row=6, column=1, padx=5, pady=5)

# Initialize bill details

self.bill = []

self.total\_price = 0

def add\_medication\_to\_bill(self):

medication\_id = self.entry\_medication\_id.get()

quantity = self.entry\_quantity.get()

# Fetch medication details from the inventory database

self.cursor\_inventory.execute("SELECT MedicationName, Price, Quantity FROM Medications WHERE MedicationID=?", (medication\_id,))

medication\_details = self.cursor\_inventory.fetchone()

if medication\_details:

medication\_name, price, available\_quantity = medication\_details

single\_price = price

gst = price \* 0.06 # 6% GST

# Check if entered quantity is valid

try:

quantity = int(quantity)

if quantity <= 0:

raise ValueError("Quantity must be a positive integer")

except ValueError:

messagebox.showerror("Error", "Quantity must be a positive integer")

return

# Check if entered quantity exceeds available quantity

total\_quantity\_in\_bill = sum(item[3] for item in self.bill if item[0] == medication\_id) # Sum of quantity of same medication in bill

if total\_quantity\_in\_bill + quantity > available\_quantity:

messagebox.showerror("Error", "Total quantity exceeds available stock")

return

total\_price = (float(quantity) \* price) + (float(quantity) \* gst) # Total price including GST

# Update the bill

self.bill.append((medication\_id, medication\_name, price, quantity, single\_price, gst, total\_price))

# Update the total price

self.total\_price += total\_price

# Update the treeview

self.treeview.insert("", "end", values=(medication\_name, price, quantity, single\_price, gst, total\_price))

# Update total price label with formatted value

self.total\_price\_var.set("{:.2f}".format(self.total\_price)) # Format to display only two decimal places

else:

messagebox.showerror("Error", f"Medication with ID {medication\_id} not found.")

def remove\_medication\_from\_bill(self):

selected\_item = self.treeview.selection()

if selected\_item:

item\_values = self.treeview.item(selected\_item, 'values')

total\_price = item\_values[5] # Total price is at index 5

# Ensure total\_price is a float

if isinstance(total\_price, str):

total\_price = float(total\_price)

self.total\_price -= total\_price

self.total\_price\_var.set("{:.2f}".format(self.total\_price)) # Format to display only two decimal places

self.treeview.delete(selected\_item)

for index, item in enumerate(self.bill):

if item[1] == item\_values[0]:

self.bill.pop(index)

break

else:

messagebox.showerror("Error", "Please select an item to remove from the bill.")

def print\_receipt(self):

try:

with sqlite3.connect('inventory.db') as conn\_inventory:

cursor\_inventory = conn\_inventory.cursor()

with sqlite3.connect('billing\_history.db') as conn\_billing\_history:

cursor\_billing\_history = conn\_billing\_history.cursor()

for item in self.bill:

medication\_id, medication\_name, price, quantity, single\_price, gst, total\_price = item

billing\_date = datetime.now().strftime('%Y-%m-%d %H:%M:%S') # Get current date and time

cursor\_billing\_history.execute("INSERT INTO BillingHistory (MedicationName, Quantity, Price, TotalPrice, BillingDate) VALUES (?, ?, ?, ?, ?)",

(medication\_name, quantity, price, total\_price, billing\_date))

# Update the inventory database

cursor\_inventory.execute("UPDATE Medications SET Quantity = Quantity - ? WHERE MedicationID = ?", (quantity, medication\_id))

except sqlite3.Error as e:

messagebox.showerror("Database Error", str(e))

else:

messagebox.showinfo("Receipt Printed", "Receipt printed successfully")

# Clear the bill and reset total price

self.bill = []

self.total\_price = 0

self.total\_price\_var.set(0.0)

self.treeview.delete(\*self.treeview.get\_children())

if \_\_name\_\_ == "\_\_main\_\_":

root = tk.Tk()

app = MedicalApp(root)

root.mainloop()

**VI. Conclusion and Future Enhancement**

The development of the Medical Inventory and Billing System marks a significant advancement in healthcare management, streamlining inventory tracking and billing processes to enhance efficiency and accuracy. By transitioning from traditional paper-based systems to digital solutions, healthcare facilities can improve operational efficiency, ensure data accuracy, and provide better patient care. Future enhancements such as integration with Electronic Health Record (EHR) systems, advanced reporting capabilities, and mobile applications promise to further elevate the system's functionality and usability. Strengthening security measures, optimizing supply chain integration, and implementing predictive analytics will enable healthcare organizations to maximize cost-effectiveness and optimize resource utilization. Ultimately, the continued evolution of the Medical Inventory and Billing System holds promise for revolutionizing healthcare management, promoting transparency, and improving patient outcomes.

**REFERENCES**

[1] R. S. Dick ,E. B. Steen, The Computer-based Patient Record: An Essential Technology for Health Care, Institute of Medicine, National Academy Press, 2000.

[2] https://www.precedenceresearch.com/medical-billing-outsourcing-market

[3] Kennedy, J., & Tapp, A. (2019). Implementing an Inventory Management System in Healthcare. Journal of Healthcare Management, 64(6), 407-416.

[4] Dhamodaran, S., Lakshmi, M. (2019). "Ensampling data prediction using sparse data in mobile intelligent system". International Journal of Interactive Mobile Technologies, 13(10), pp. 106-119.

[5] Ayatollahi, H., Gholizadeh, M., & Vakili, R. (2019). Developing a Radio Frequency Identification (RFID)-based Medical Inventory Management System. Journal of Medical Systems, 43(6), 135.

[6] Islam, M. S., Hasan, M. M., & Alam, M. R. (2018). A Medical Inventory Management System for Resource Optimization in Healthcare. International Journal of Computer Applications, 180(4), 1-7.

[7] Geerts, M., & Lawless, A. (2018). A Hospital Inventory Management System to Improve Patient Care. Journal of Hospital Administration, 7(2), 7-16.

[8] Anderson, J. G., & Aydin, C. E. (2017). The role of technology in healthcare inventory management. Journal of Healthcare Engineering, 2017, 1-9.

[9] Dhamodaran, S., Baron Sam, B., Refonaa, J. "Study of land management system in urban area development using geographic information system", International Journal of Applied Engineering Research, 10(76), pp. 353-358, (2015).

[10] Collins, M. (2015, September 4). Top 5 Benefits of Automated Inventory Management System. Retrieved from Understanding eCommerce: https://www.understandingecommerce.com/top-5-benefits-of-automated-inventory-management-system.

[11] Tardieu, P. (2015). Introduction to a Dental and Medical Inventory, Order and Traceability. Oral Hygiene & Health.

[12] Vitez, O. (2017, September 26). Importance of Computerized Inventory Systems. Retrieved from bizfluent: <https://bizfluent.com/facts->6848349-importance-computerized-inventory-systems.html.

[13] Yinyeh, M. O., & Alhassan, S. (2013). Inventory Management System Software for Public Universities in Ghana. International Journal of Advanced Research in Computer Engineering & Technology (IJARCET).

**Web links:**

Here is a list of reference links that we have used for our project:

1. <https://yourstory.com/2018/01/healthtech-startups-india/>
2. [https://www.precedenceresearch.com/medical-billing-outsourcing-market](https://www.precedenceresearch.com/medical-billing-outsourcing-market%20)