ORDER SCHEDULING SYSTEM

A MINI-PROJECT REPORT

Submitted by

HARINI D S 231901009

KEERTHIKA S 231901024

in partial fulfillment of the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING (CYBER SECURITY)

RAJALAKSHMI ENGINEERING COLLEGE, CHENNAI

An Autonomous Institute

CHENNAI

MAY 2025

BONAFIDE CERTIFICATE

Certified that this project **"HOTEL ORDER SCHEDULING"** is the bonafide work of **"HARINI D S , KEERTHIKA S"** who carried out the project work under my supervision.

SIGNATURE
Mrs. JANANEE V
ASSISTANT PROFESSOR

Dept. of Computer Science and Engg,
Rajalakshmi Engineering College
Chennai

This mini project report is submitted for the viva voce examination to be held on _____

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ABSTRACT :

The Hotel Order Scheduling System is a web-based application designed to efficiently manage food orders in a restaurant or hotel kitchen. It allows users to add customer orders and dynamically schedule them using classical CPU scheduling algorithms like Priority, FirstCome-First-Serve (FCFS), Shortest Job First (SJF), and Round Robin. The goal is to simulate how these scheduling strategies can improve kitchen workflow by minimizing wait times and maximizing fairness and efficiency. The system is built using FastAPI for the backend and a lightweight HTML/JavaScript frontend for interaction. Orders are stored in a CSV file and visualized through an interactive dashboard using Chart.js. With real-time updates and scheduling visualization, the system serves as both a functional application and an educational tool to demonstrate how algorithmic scheduling can be applied to real-world service environments.

ACKNOWLEDGEMENT

We express our sincere thanks to our honorable chairman **MR. S. MEGANATHAN** and the chairperson **DR. M.THANGAM MEGANATHAN** for their timely support and encouragement.

We are greatly indebted to our respected and honorable principal **Dr. S.N. MURUGESAN**

for his able support and guidance.

No words of gratitude will suffice for the unquestioning support extended to us by our Head Of the Department **Mr. BENIDICT JAYAPRAKASH NICHOLAS M.E Ph.D.,** for being ever supporting force during our project work.

We also extend our sincere and hearty thanks to our internal guide **Mrs**.**V JANANEE,** for her valuable guidance and motivation during the completion of this project.

Our sincere thanks to our family members, friends and other staff members of computer science engineering.

1. **HARINI D S**
2. **KEERTHIKA S**

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

In hotel and restaurant environments, managing kitchen operations and food order timing is critical. Delays in order processing can impact service quality and customer satisfaction. The Hotel Order Scheduling System applies traditional CPU scheduling algorithms such as FCFS, SJF, Priority, and Round Robin to optimize the handling of customer orders in real-time. The system enables dynamic task scheduling, prioritization based on preparation time or urgency, and visualization for better kitchen workflow management.

1.2 SCOPE OF THE WORK

This system targets small to medium-sized restaurants and hotel kitchens. It allows kitchen staff or managers to input incoming orders and apply a selected scheduling algorithm to optimize the cooking sequence. The solution includes a simple web interface, real-time updates, and order visualization to ensure smoother operations and faster customer service.

1.3 PROBLEM STATEMENT

Manual order management often results in mismanagement, longer wait times, and inefficient use of kitchen resources. Orders might be delayed, lost, or prepared in suboptimal sequence. There is a need for a digital system that schedules orders fairly and efficiently based on predefined strategies.

1.4 AIM AND OBJECTIVES OF THE PROJECT

- To develop an order scheduling system for food preparation in hotels.
- To implement FCFS, SJF, Priority, and Round Robin algorithms to handle scheduling logic.
- To provide a web interface for inputting and viewing orders.
- To visualize scheduled orders using graphical representations (e.g., Gantt charts).
- To improve order tracking, reduce preparation delays, and increase customer satisfaction.

CHAPTER 2

SYSTEM SPECIFICATIONS:

2.1 HARDWARE SPECIFICATIONS

Component	: Specification
Processor	: Intel i5
Memory Size	: 8 GB (Minimum)
HDD/SSD	: 256 GB (Minimum)

2.2 SOFTWARE SPECIFICATIONS

Component	: Technology Used
Operating System	: Windows 10
Frontend	: HTML, JavaScript
Backend	: fastAPI(python framework)
Database	: CSV File
Libraries/Tools	: Chart.js, Uvicorn
Used	: Python, JavaScript, SQL Languages

CHAPTER 3

MODULE DESCRIPTION

The system is divided into the following functional modules:

- **Order Input Module:** Accepts new orders via a form (customer name, dish, preparation time, priority, etc.).
- **Scheduling Engine:** Processes orders based on selected algorithm:
    1. FCFS: First come, first served.
    2. SJF: Shortest preparation time first.
    3. Priority: Based on manually assigned priority values.
    4. Round Robin: Fair time slicing using a defined quantum.
- **Data Persistence Module:** Stores and retrieves orders from CSV.
- **Visualization Module:** Displays scheduled results using Chart.js in a Ganttstyle chart.
- **API Services:** Exposes REST endpoints for frontend interaction.

CHAPTER 4

CODING

SOURCE CODE

4.1. app.py

```
from fastapi import FastAPI, Request
from fastapi.responses import FileResponse, JSONResponse
from fastapi.staticfiles import StaticFiles
from fastapi.middleware.cors import CORSMiddleware
from pydantic import BaseModel
from scheduler import read_orders, schedule_orders
from datetime import datetime
import csv
import os

app = FastAPI()

BASE_DIR = os.path.dirname(__file__)
DATA_PATH = os.path.join(BASE_DIR, "data.csv")

# Ensure data.csv exists with headers if not present
if not os.path.exists(DATA_PATH):
    with open(DATA_PATH, "w", newline="") as f:
```

```
        writer = csv.writer(f)
        writer.writerow(["customer_name", "dish_name", "prep_time", "category",
"priority", "timestamp"])

# Enable CORS (for frontend integration if needed)
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],  # Change to specific domain(s) in production
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Mount static files (e.g., CSS, JS)
app.mount("/static", StaticFiles(directory=BASE_DIR), name="static")

# Serve index.html from root
@app.get("/")
async def root():
    return FileResponse(os.path.join(BASE_DIR, "index.html"))

# Serve other static files
@app.get("/{file_name}")
async def serve_file(file_name: str):
    file_path = os.path.join(BASE_DIR, file_name)
```

```
    if os.path.exists(file_path):
        return FileResponse(file_path)
    return JSONResponse(content={"error": "File not found"}, status_code=404)

# Pydantic model for input validation
class Order(BaseModel):
    customer_name: str
    dish_name: str
    prep_time: int
    category: str
    priority: int

# API endpoint to get scheduled orders
@app.get("/api/orders")
async def get_orders(algorithm: str = "Priority", quantum: int = 5):
    orders = read_orders(DATA_PATH)
    scheduled = schedule_orders(orders, algorithm, quantum)
    return {"algorithm": algorithm, "orders": scheduled}

# API endpoint to add a new order
@app.post("/api/orders")
async def add_order(order: Order):
    with open(DATA_PATH, "a", newline="") as f:
        writer = csv.writer(f)
        writer.writerow([
```

```
            order.customer_name,
            order.dish_name,
            order.prep_time,
            order.category,
            order.priority,
            datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        ])
    return {"message": "Order added"}
```

4.2. scheduler.py

```
from datetime import datetime, timedelta
import csv
from zoneinfo import ZoneInfo  # For timezone support

IST = ZoneInfo("Asia/Kolkata")  # Indian Standard Time

def read_orders(file_path):
    orders = []
    with open(file_path, newline="") as f:
        reader = csv.reader(f)
        for row in reader:
            if len(row) < 6 or row[0] == "customer_name":
                continue
            orders.append({
                "customer_name": row[0],
```

```
                "dish_name": row[1],
                "prep_time": int(row[2]),
                "category": row[3],
                "priority": int(row[4]),
                "timestamp": datetime.strptime(row[5], "%Y-%m-%d
%H:%M:%S").replace(tzinfo=IST)
            })
    return orders

def schedule_orders(orders, algorithm="Priority", quantum=5):
    now = datetime.now(IST)
    scheduled_orders = []

    if algorithm == "Priority":
        orders.sort(key=lambda x: x["priority"])
    elif algorithm == "FCFS":
        orders.sort(key=lambda x: x["timestamp"])
    elif algorithm == "SJF":
        orders.sort(key=lambda x: x["prep_time"])
    elif algorithm == "Round Robin":
        return round_robin_schedule(orders, quantum)

    for order in orders:
        start_time = order["timestamp"]
```

```
        end_time = start_time +
timedelta(minutes=order["prep_time"])

        if now < start_time:
            status = "Pending"
        elif start_time <= now < end_time:
            status = "In Progress"
        else:
            status = "Completed"

        scheduled_orders.append({
            **order,
            "start_time": start_time.strftime("%H:%M:%S"),
            "end_time": end_time.strftime("%H:%M:%S"),
            "status": status
        })

    return scheduled_orders

def round_robin_schedule(orders, quantum):
    now = datetime.now(IST)
    orders = sorted(orders, key=lambda x: x["timestamp"])
    remaining_time = {i: order["prep_time"] for i, order in
enumerate(orders)}
    current_time = orders[0]["timestamp"] if orders else now
```

```
finished = set()
schedule = []

while len(finished) < len(orders):
    for i, order in enumerate(orders):
        if i in finished:
            continue

        time_slice = min(quantum, remaining_time[i])
        start_time = current_time
        end_time = start_time + timedelta(minutes=time_slice)
        remaining_time[i] -= time_slice

        if remaining_time[i] <= 0:
            finished.add(i)

        if now < start_time:
            status = "Pending"
        elif start_time <= now < end_time:
            status = "In Progress"
        else:
            status = "Completed"

        schedule.append({
            **order,
```

```
            "start_time": start_time.strftime("%H:%M:%S"),
            "end_time": end_time.strftime("%H:%M:%S"),
            "status": status,
            "slice": time_slice
        })

        current_time = end_time

return schedule
```

CHAPTER 5

SCREENSHOTS

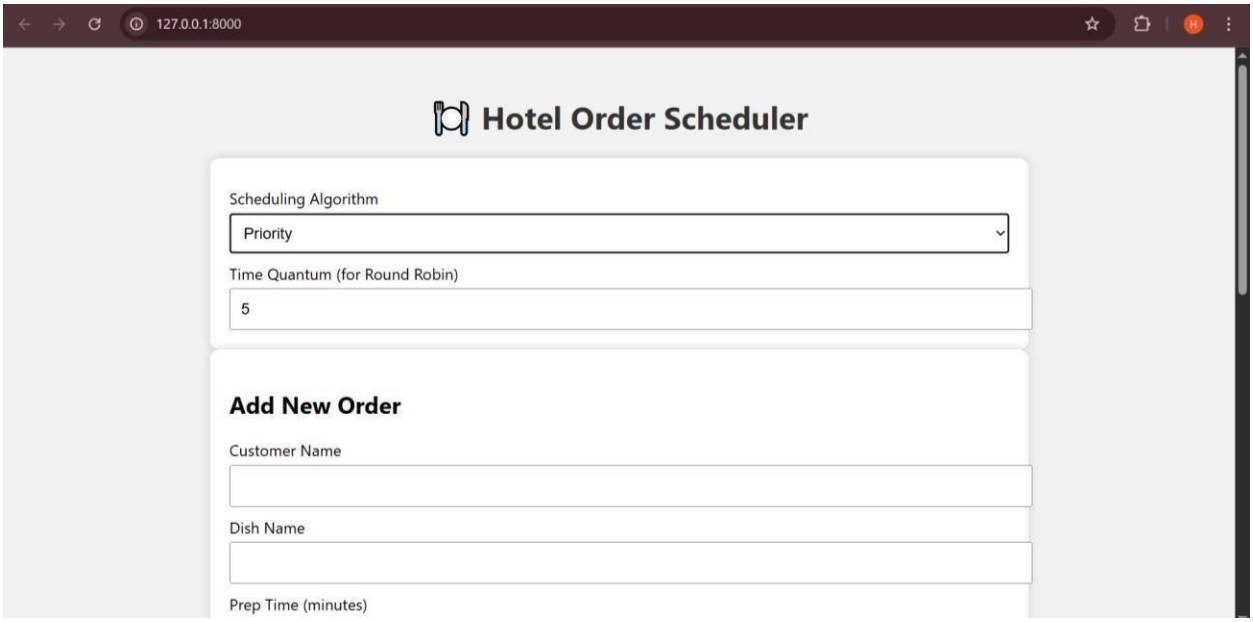

Fig 5.1 Hotel Order Scheduler

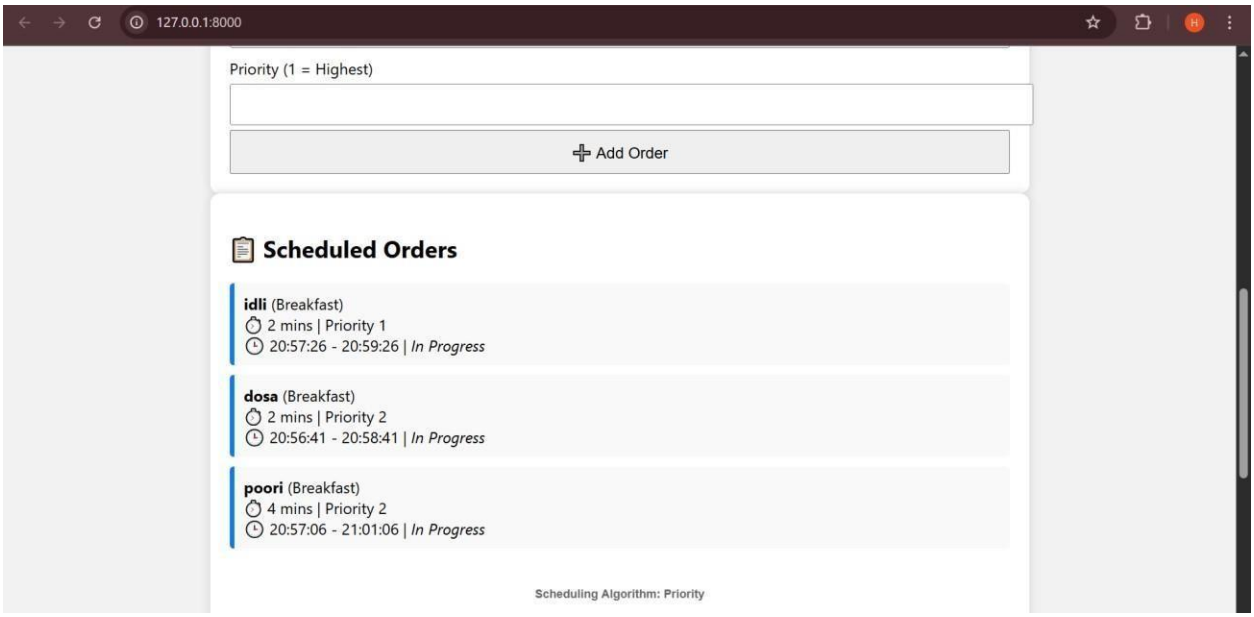

Fig 5.2 Scheduled Orders

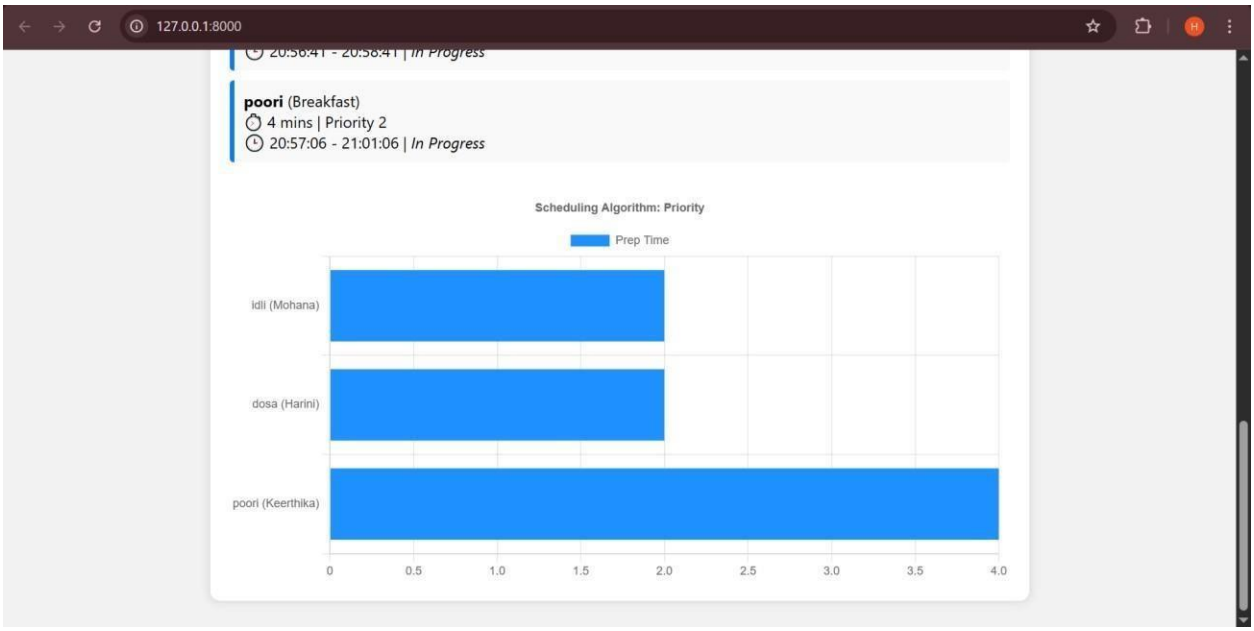

Fig 5.3 Priority Scheduling Graph

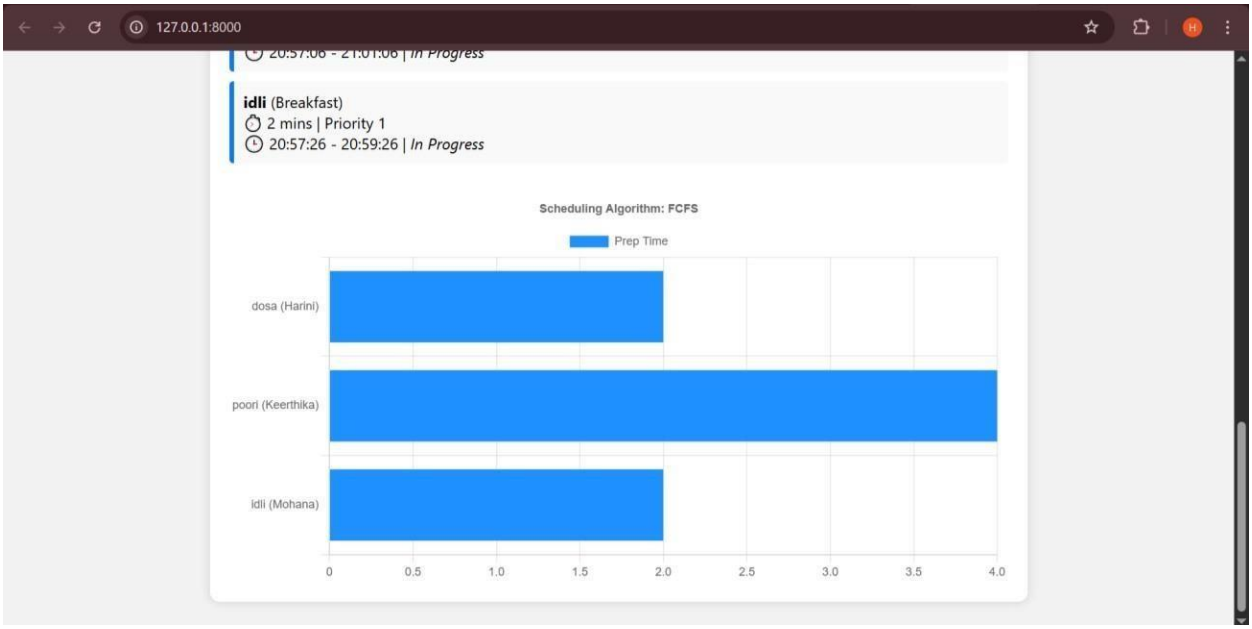

Fig 5.4 FCFS Scheduling Graph

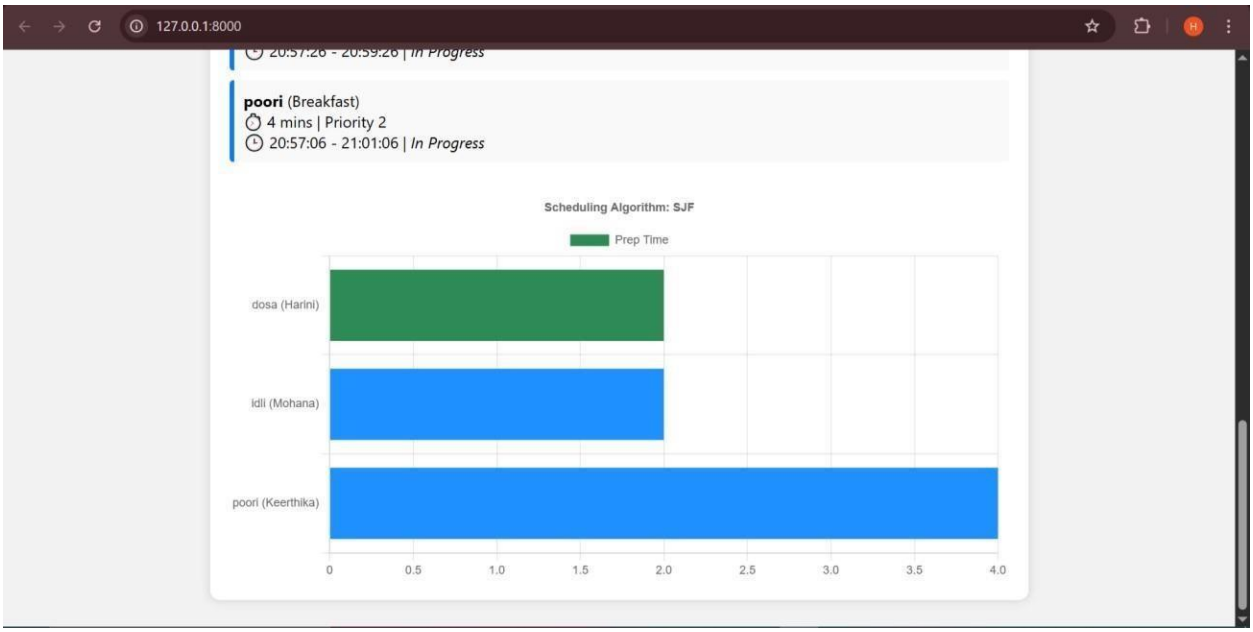

Fig 5.5 SJF Scheduling Graph

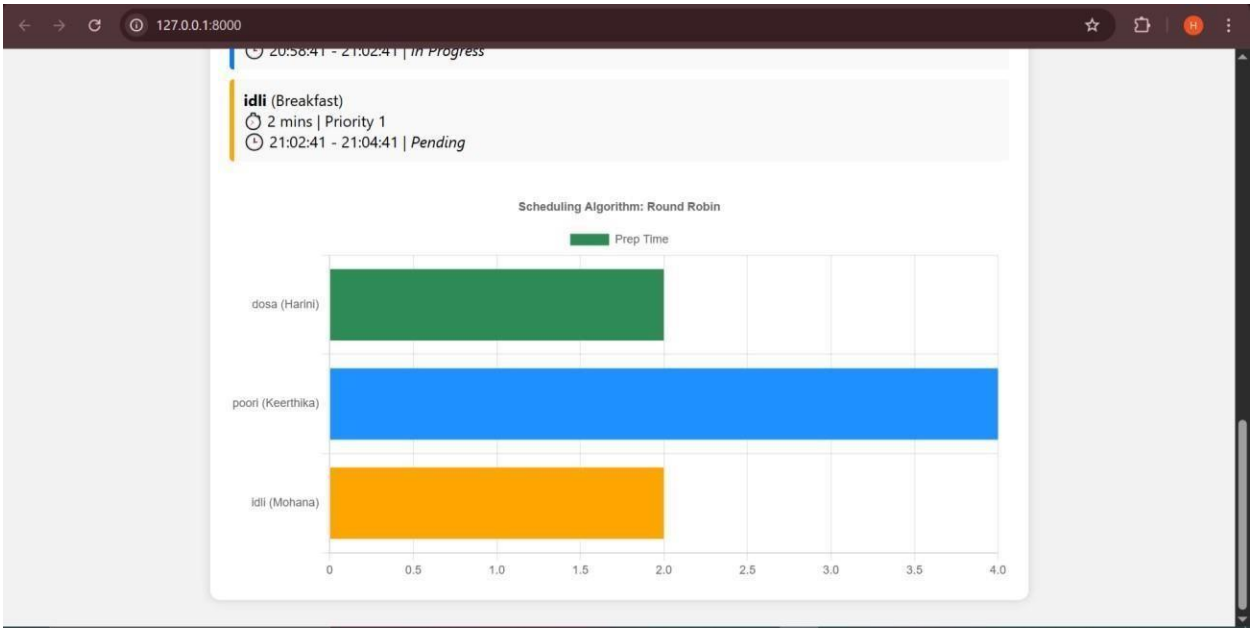

Fig 5.6 Round Robin Scheduling Graph

CHAPTER 6

CONCLUSION AND FUTURE ENHANCEMENT

The Hotel Order Scheduling System efficiently handles food orders using classic scheduling strategies. It allows hotel kitchens to manage tasks more effectively and improve customer service timelines.

Future Enhancements:

- Integrate user authentication for staff
- Shift to SQL database (e.g., SQLite, MySQL)
- Export reports in PDF
- Add mobile compatibility
- Support more scheduling algorithms (e.g., Multilevel Queue)

CHAPTER 7

REFERENCES

- FastAPI Documentation – https://fastapi.tiangolo.com
- Python Official Docs – https://docs.python.org
- Chart.js – https://www.chartjs.org
- GeeksforGeeks – CPU Scheduling Tutorials
- W3Schools – HTML/CSS/JS Reference