| au-logo.png | MESS MANAGEMENTINTERNSHIP REPORT |  |
| --- | --- | --- |

### *Submitted by*

| **PAVITHRA N**  **2303717610422106** |
| --- |

***in partial fulfillment for the award of the degree***

***of***

### BACHELOR OF ENGINEERING

### in

### COMPUTER SCIECNE AND ENGINEERING

### COIMBATORE INSTITUTE OF TECHNOLOGY

### (Government Aided Autonomous Institution Affiliated to Anna University)

### COIMBATORE – 641014

### ANNA UNIVERSITY:CHENNAI 600 025

### JUNE 2025

**COIMBATORE INSTITUTE OF TECHNOLOGY**

**(Government aided Autonomous Institution Affiliated to Anna University)**

**COIMBATORE – 641014**

**ANNA UNIVERSITY::CHENNAI 600 025**

**BONAFIDE CERTIFICATE**

Certified that this Internship report is the Bonafide work of **MIRUDHULA R (10422100)**

| **---------------------------------------------**  **Dr.R.NEDUNCHEZHIAN M.E., Ph.D**  **Professor**  Department of Computer Science and Engineering,  Coimbatore Institute of Technology,  Coimbatore – 641 014. |  | **----------------------------------------**  **Dr.A.KUNTHAVAI M.S.,Ph.D**  **Professor and Head**  Department of Computer Science and Engineering,  Coimbatore Institute of Technology,  Coimbatore – 641 014. |
| --- | --- | --- |

Submitted for the evaluation held on \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

------------------- --------------------- ---------------------

Reviewer 1 Reviewer 2 Reviewer 3

**DETAILS OF INTERNSHIP UNDERGONE**

Reg. No.: 2303717610422106

Name of the Student: Pavithra N

Degree: B.E.

Branch: Computer Science and Engineering

Semester: V

Project Title : MESS MANAGEMENT

| **S. No.** | **Name and Address of the Company** | **Period of Internship undergone** | | **No. of Days** |
| --- | --- | --- | --- | --- |
| **From** | **To** |
| 1. | PROJECT DEVELOPMENT CELL (PDC),  DEPARTMENT OF CSE , COIMBATORE INSTITUTE OF TECHNOLOGY, COIMBATORE | 19.06.2025 | 02.07.2025 | 15 |
| **Total Number of Days:** | | | | 15 |

### Signature of Student Signature of Tutor

**COPY OF CERTIFICATE:**

**ACKNOWLEDGEMENT**

I extend my heartfelt gratitude to **Project Development Cell (PDC)** for permitting me to undergo internship their industries/organizations and their guidance to upgrade my theoretical and practical skills in engineering.

I am sincerely grateful to the Management of Coimbatore Institute of Technology, Coimbatore for supporting me in every possible way, for undergoing internship in industries.

I sincerely thank our Principal **Dr. A. Rajeswari, M.E., Ph.D.,** Coimbatore Institute of Technology, Coimbatoreand Head of the Department **Dr Kunthavai A, M.S., Ph.D.,** Department of Computer Science and Engineering, Coimbatore Institute of Technology, Coimbatore for including industrial training pattern in the curriculum to enrich students with a sound knowledge in engineering and make them industry ready. I am also obliged for permitting me to undergo the training.

I also extend my thanks to the Senior Tutor, Tutors, Teaching faculty and all other supporting staff members of the department for their support and encouragement to take up the internship in industries.

**PAVITHRA N**

**TABLE OF CONTENTS**

| **CHAPTER**  **No.** | **TITLE** | **PAGE**  **No.** |
| --- | --- | --- |
|  | **LIST OF ABBREVIATION** |  |
| **1** | INTRODUCTION |  |
|  | 1.1 PROJECT SCOPE AND OBJECTIVE |  |
|  | 1.2 MODULES |  |
| **2** | SOFTWARE REQUIREMENTS SPECIFICATIONS |  |
| **3** | SYSTEM ARCHITECTURE AND DESIGN |  |
| **4** | TECHNOLOGY USED |  |
|  | 4.1 FRONT END TECHNOLOGY |  |
|  | 4.2 BACK END TECHNOLOGY |  |
| **5** | CODING |  |
| **6** | SCREENSHOTS / RESULTS AND DISCUSSION |  |
| **7** | CONCLUSION |  |
| **8** | BIBILOGRAPHY |  |

**LIST OF ABBREVIATION**

| **ABBREVIATION** | **EXPANSION** |
| --- | --- |
| DB | DATABASE |
| HTML | HYPER TEXT MARKUP LANGUAGE |
| CSS | CASCADING STYLE SHEET |
| JS | JAVASCRIPT |
| API | APPLICATION PROGRAMMING INTERFACE |
| HTTP | HYPERTEXT TRANSFER PROTOCOL |

**WEEKLY OVERVIEW OF INTERNSHIP ACTIVITIES**

|  |  |  |  |
| --- | --- | --- | --- |
| **1st WEEK** | **DATE** | **DAY** | **NAME OF THE TOPIC/MODULE COMPLETED** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **2nd WEEK** | **DATE** | **DAY** | **NAME OF THE TOPIC/MODULE COMPLETED** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

CHAPTER 1

INTRODUCTION

This project titled **Digital Hostel Mess Management System** is a web-based application designed to streamline the day-to-day operations of a hostel mess. It offers students and administrators a user-friendly platform to manage meal skipping, daily menus, feedback, special meal tokens, and billing efficiently. The system eliminates the need for traditional paper-based tracking and allows real-time updates, secure access, and role-based operations.

1.1 PROJECT SCOPE:

**System** that efficiently automates daily mess operations and ensures a seamless user experience for both students and administrators.

* **Digital Meal Management:** Allows students to submit meal skip requests and view updated menus in real time, helping mess staff reduce food wastage and manage quantities accurately.
* **Token-Based Special Meal System:** Integrates a flexible token system where admins can issue daily special meal tokens, and students can opt in, ensuring the correct amount is reflected in their mess bill.
* **Student Feedback & Admin Control Panel:** Enables students to submit feedback about food or services, while administrators have exclusive access to view, analyze, and respond to feedback with student details.
* **Dynamic Monthly Billing:** Automatically calculates monthly bills for each student by adjusting the base mess fee with deductions for skipped meals and additions for token purchases, ensuring fair and transparent accounting.
* **User-Friendly Session-Based Access:** Features a secure, session-managed login system with role-based navigation, ensuring students and admins access only their respective functionalities in an intuitive, user-friendly interface.

OBJECTIVES

The main objectives of the project are:

* To design and develop a session-based, role-driven hostel mess management system using FastAPI and PostgreSQL.
* To enable students to digitally submit meal skip requests, access daily menus, and manage special token-based meal participation.
* To ensure accurate monthly mess billing by integrating skip logic and token purchases dynamically.
* To provide a secure feedback system where students can submit opinions while only admins can view and review them.
* To offer a responsive, user-friendly interface that allows both admins and students to interact with the system efficiently through Jinja2-rendered HTML pages.
* To reduce paper-based mess operations, streamline food quantity estimation, and promote transparency in billing and feedback handling.

**1.2 MODULES**

**1.2.1 User Authentication Module**

**Description:**  
This module handles secure login and session-based authentication for students and admins. It ensures that users access only the features permitted to their role.

**Functionality:**

* Provide login form with username and password.
* Validate credentials and assign session with role (student or admin).
* Restrict or allow access to routes based on session role.

**1.2.2 Meal Skip Module**

**Description:**  
This module allows students to submit skip requests for breakfast, lunch, or dinner on specific dates. Admins can view and track total skipped meals for better meal planning.

**Functionality:**

* Display a form for students to skip meals on selected days.
* Store skip data with timestamps in the database.
* Admin can view consolidated meal skip counts for each day.

**1.2.3 Daily Menu Management Module**

**Description:**  
This module enables admins to create, update, and publish the daily menu for each day of the week. Students can view the current and upcoming menu.

**Functionality:**

* Admin interface to add/edit meals per day.
* Store menu items in the database.
* Students can view menus for all days in a clean layout.

**1.2.4 Feedback Module**

**Description:**  
This module allows students to submit anonymous or named feedback about food, service, or cleanliness. Admins can review all submissions with student details.

**Functionality:**

* Student feedback form with text input.
* Store feedback with timestamps and user info.
* Admin panel to view all feedback entries by name.

**1.2.5 Special Token and Billing Module**

**Description:**  
This module allows the admin to issue special token-based meals for a day. Students can opt in by submitting their name and roll number. Token cost is added to their bill.

**Functionality:**

* Admin assigns a special item and its price for the day.
* Students submit a simple token form.
* If submitted, token cost is added to monthly bill.

**1.2.6 Monthly Billing Module**

**Description:**  
This module calculates and displays each student’s total mess fee based on skipped meals and special tokens. It automates billing and ensures transparency.

**Functionality:**

* Deduct fixed amount for each skipped meal.
* Add price of submitted tokens to bill.
* Generate monthly totals and display them in student dashboard.

**1.2.7 Admin Dashboard Module**

**Description:**  
This module serves as the control panel for administrators to manage all backend operations such as menu updates, feedback review, and billing overview.

**Functionality:**

* Display links to all admin-only features.
* Provide consolidated view of user activity and data.
* Ensure secure access only for authenticated admin sessions.

**CHAPTER - 2**

**SRS DOCUMENT**

**SOFTWARE REQUIREMENTS SPECIFICATION**

**For**

**DIGITAL MESS MANAGEMENT SYSTEM**

**Prepared by Pavithra**

**1. INTRODUCTION**

**1.1 Purpose**

**The purpose of this project is to design and implement a web-based Mess Management System that simplifies hostel mess operations. It allows students to manage meal skipping, view menus, submit feedback, and track monthly bills. It provides administrators with tools to publish menus, monitor participation, view feedback, and handle billing efficiently — all within a secure, session-based application.**

**1.2 Intended Audience**

**This system is intended for:**

* **Students – to view menus, skip meals, submit feedback, and track their monthly mess charges.**
* **Mess Administrators – to manage menus, meals, feedback, and billing.**
* **Developers – for extending the system in the future with new features like digital payments or analytics.**

**1.3 Project Scope**

**The Digital Mess Management System includes:**

* **Meal skip functionality to help reduce wastage.**
* **Daily menu upload and view features.**
* **A feedback system visible only to admins.**
* **Special token meal management linked to billing.**
* **Monthly bill tracking for each student.**
* **A secure, role-based login system using FastAPI sessions.**

**The system will run in a web browser and will use PostgreSQL for persistent data storage. Admin and student dashboards are dynamically rendered using Jinja2 templates.**

**1.4 Definitions, Acronyms, and Abbreviations**

| **Acronym** | **Definition** |
| --- | --- |
| **CRUD** | **Create, Read, Update, Delete** |
| **UI** | **User Interface** |
| **DB** | **Database** |
| **HTTP** | **Hypertext Transfer Protocol** |
| **HTML** | **HyperText Markup Language** |
| **SRS** | **Software Requirements Specification** |

**2. OVERALL DESCRIPTION**

**2.1 Product Perspective**

**The system is a full-stack web application that integrates front-end Jinja2 templates, FastAPI for backend logic, and PostgreSQL for data storage. It operates on a client-server model with session-based access control.**

**2.2 Product Features**

**2.2.1 User Authentication**

**Login system for both admins and students using sessions to control access.**

**2.2.2 Meal Skip System**

**Allows students to opt-out of meals in advance. Admins can view meal participation daily.**

**2.2.3 Daily Menu Module**

**Admins can post or update menus. Students view them via their dashboard.**

**2.2.4 Feedback System**

**Students provide feedback. Admins access the collected responses with user details.**

**2.2.5 Special Token Module**

**Admins publish a special meal item. Students request it via a form, and the cost is added to their monthly bill.**

**2.2.6 Billing Module**

**Automatically adjusts bills based on meals skipped and special tokens used.**

**2.2.7 Admin Dashboard**

**Centralized access to all modules like feedback viewer, token manager, bill tracker, and menu editor.**

**2.3 User Characteristics**

* **Admin – tech-comfortable user managing menus, feedback, and finances.**
* **Student – casual user needing a friendly interface for simple daily tasks.**

**2.4 General Constraints**

* **System must support standard web browsers (Chrome, Firefox).**
* **PostgreSQL should be properly configured and connected.**
* **Sessions should expire after logout or timeout.**
* **Internet access is required for multi-user access in a networked deployment.**

**2.5 Assumptions and Dependencies**

* **Basic web usage knowledge is expected from users.**
* **App assumes valid login credentials.**
* **Database connection is reliable and secure.**
* **Runs on devices with modern browsers.**

**3. SYSTEM FEATURES**

**3.1 Use Case Diagram**

**Actors:**

* **Student**
* **Admin**

**Use Cases:**

* **Login**
* **Skip Meals**
* **View Menu**
* **Submit Feedback**
* **View Feedback (admin only)**
* **Publish Menu (admin)**
* **View Monthly Bill**
* **Manage Token System**

**////////////////DIAGRAM**

**3.2 Use Case Scenarios**

**3.2.1 Use Case: Submit Meal Skip**

**Actor: Student  
Preconditions: Student is logged in  
Main Scenario:**

1. **Student selects the date and meals to skip.**
2. **Form is submitted.**
3. **Confirmation message appears.**
4. **Admin views total meals skipped per day.**

**3.2.2 Use Case: View Monthly Bill**

**Actor: Student  
Preconditions: Student is logged in  
Main Scenario:**

1. **Student clicks “View Bill”.**
2. **System displays monthly summary.**
3. **Includes skips and special token charges.**

**3.2.3 Use Case: Publish Menu**

**Actor: Admin  
Preconditions: Admin is logged in  
Main Scenario:**

1. **Admin selects a day and enters meal info.**
2. **Menu is saved and visible to all students.**

**4. EXTERNAL INTERFACE REQUIREMENTS**

**4.1 User Interfaces**

* **Login Page: Student/Admin authentication**
* **Student Dashboard: Menu, Skip Meal, Token, Feedback**
* **Admin Dashboard: Menu Editor, Feedback Viewer, Token Manager, Bill Tracker**

**4.2 Hardware Interfaces**

* **Standard Web Browser**
* **Hosting machine for FastAPI backend (Linux/Windows)**
* **PostgreSQL-compatible database server**

**4.3 Software Interfaces**

* **FastAPI for backend routes**
* **PostgreSQL for data storage**
* **Jinja2 for HTML rendering**
* **HTML/CSS for frontend**
* **Session Middleware for user sessions**

**5. OTHER NONFUNCTIONAL REQUIREMENTS**

**5.1 Performance Requirements**

* **Pages should load under 2 seconds.**
* **Login/logout must be secure and session-based.**
* **Bill updates must reflect real-time actions (token/skip).**

**5.2 Security**

* **Admin-only access to feedback and billing.**
* **Session-based login to prevent unauthorized access.**
* **User data should be validated and sanitized.**

**5.3 Usability**

* **Interfaces should be responsive and mobile-friendly.**
* **Forms should include validation and confirmation alerts.**

**5.4 Maintainability**

* **Code should be modular and reusable.**
* **Easy to plug in future features like payment gateway.**

**5.5 Scalability**

* **Can support more users, messes, or hostels with minimal configuration.**

CHAPTER – 3

SYSTEM ARCHITECTURE AND DESIGN

3.1 SYSTEM DESIGN:

The design of the Mess Management System involves several interrelated modules that work together to streamline and digitize hostel mess operations. Each module focuses on improving efficiency, transparency, and user convenience.

Key components of the design include:

1. User Authentication and Role Management:  
   This module handles secure login for students and admins. It manages session-based authentication and redirects users based on their roles.
2. Meal Skip Management:  
   Allows students to skip meals on specific dates, helping reduce food wastage and enabling fair billing based on meal consumption.
3. Feedback System:  
   Enables students to submit feedback regarding food quality, cleanliness, or complaints. Only admins have permission to view all feedback along with student identities.
4. Daily Menu Display:  
   Admins can update and manage daily menus. Students can view menus for each day to make informed decisions about meal participation.
5. Special Token System:  
   When special meals are provided, admins can issue a token form. Students opting in are charged additionally, and the cost is automatically added to their monthly bill.
6. Billing Module:  
   Calculates the monthly mess fee per student, adjusting for skipped meals and additional token charges.

3.2 SYSTEM ARCHITECTURE:

The architecture of the Mess Management System is modular and follows a client-server model with clear separation of frontend, backend, and database operations.

Key components include:

1. User Interface (Frontend):
   * Built using HTML, CSS, and Jinja2 templating
   * Provides interfaces for login, feedback, meal skip, menu view, billing, and token forms
   * Communicates with the backend via HTTP requests
2. Backend (FastAPI):
   * Handles routing, form validation, session management, and logic for all modules
   * Uses Python and FastAPI for efficient, asynchronous operations
   * Separates logic based on user role (student/admin)
3. Database Layer (PostgreSQL):
   * Stores user data, skipped meals, feedback entries, menus, token records, and billing history
   * Accessed using psycopg2 and raw SQL queries for performance and clarity
4. Authentication System:
   * Login verifies credentials and sets session variables
   * Middleware ensures protected routes and role-based redirection
5. Bill Generation Engine:
   * Automatically updates monthly bills based on skip and token logs
   * Ensures fair fee calculation with transparent breakdowns for each student
6. Admin Dashboard:
   * Admins can update menus, view feedback, issue tokens, and monitor billing
   * Accessible only to users with admin roles

**CHAPTER – 4**

**TECHNOLOGY USED**

**4.1 FRONTEND TECHNOLOGY / INTERFACE DESCRIPTION**

**4.1.1 Frontend Technology:**

The project utilizes **HTML**, **CSS**, and **Jinja2 templates** to create a responsive and user-friendly interface for managing various hostel mess operations. The interface is integrated with **FastAPI** to enable dynamic rendering and smooth navigation for both students and administrators.

This interface includes:

* **Login and Role-Based Access**  
  A single login page determines the user’s role (student or admin) and redirects them to their respective dashboards using session-based access control.
* **Meal Skip Form**  
  Students can skip breakfast, lunch, or dinner for selected dates using a clean, structured form. This helps in reducing food wastage and enables precise billing.
* **Feedback Submission**  
  Students can send feedback related to food quality, hygiene, or other concerns. The form is simple and allows multiline inputs, and the data is stored for admin viewing.
* **Daily Menu Viewer and Editor**  
  Students can view the mess menu day-wise. Admins are provided with an editable interface to update and manage the daily meal offerings.
* **Token Form for Special Meals**  
  A special token page displays the day’s special item with price. Students can choose to opt in, and the corresponding cost is automatically added to their bill.
* **Mess Bill Display**  
  Students can view their total monthly mess bill, which is auto-calculated based on their meal skips and token selections.
* **Admin Dashboard**  
  Includes links to manage menus, view feedback, handle token forms, and monitor billing data for all students.

**Styling and User Experience:**

* All forms and pages are designed with:
  + **Linear gradient backgrounds**
  + **White, rounded containers**
  + **Gradient headings**
  + **Orange theme buttons with hover effect**
  + **Responsive layout for better usability**

**4.2 BACKEND TECHNOLOGY / SYSTEM LOGIC**

**4.2.1 Backend Technology:**

The backend system is developed using **FastAPI**, a modern Python web framework known for its speed and simplicity. It handles form submissions, session authentication, database communication, and dynamic HTML rendering using Jinja2 templates.

**Libraries and Tools Used:**

* FastAPI – for backend API development
* Jinja2 – for rendering dynamic HTML templates
* psycopg2 – for interacting with PostgreSQL using raw SQL queries
* starlette.sessions – for managing user login sessions
* uvicorn – as the ASGI server to run the application
* datetime – for date-based billing and form entries

**4.2.2 Backend Modules and Logic:**

1. **User Authentication Module**
   * Validates login credentials from the users table
   * Establishes session with user\_id and role
   * Redirects based on role (admin/student)
2. **Meal Skip Module**
   * Allows students to skip meals by selecting date and meal type
   * Entries are saved into the skip\_meal table
   * Monthly billing is adjusted accordingly
3. **Feedback Module**
   * Captures feedback text and links it with student’s session ID
   * Admins can view feedback with student names
   * Stored securely in feedback table
4. **Daily Menu Module**
   * Admins can add or edit daily meal entries (breakfast, lunch, dinner)
   * Menu is displayed to students based on the selected day
   * Data managed in menu table
5. **Token System Module**
   * Admins issue tokens with special item and price for the day
   * Students opt in via a token form
   * If selected, price is added to their monthly\_bill table
6. **Billing Module**
   * Automatically calculates monthly bills using:
     + Base mess fee
     + Deduction for skipped meals
     + Addition for special tokens
   * Students can view bills anytime from their panel

**4.3 DATABASE TECHNOLOGY**

The database layer uses **PostgreSQL** and is accessed via **psycopg2** with raw SQL queries. This allows full control over schema, joins, and data manipulation for high performance.

**Key Tables Used:**

* users – Stores login credentials and roles
* skip\_meal – Logs skipped meals with date and user ID
* feedback – Stores feedback entries
* menu – Contains daily menu data
* token – Tracks issued tokens and student responses
* monthly\_bill – Maintains real-time monthly mess bills for each student

**Data Integrity and Security:**

* Foreign key constraints ensure valid references between tables
* Sessions ensure that only logged-in users can access sensitive pages
* Admin-only routes prevent unauthorized access to critical modules

**CHAPTER-5**

**CODING**

**#main.py**

from fastapi import FastAPI

from fastapi.staticfiles import StaticFiles

from fastapi.middleware.cors import CORSMiddleware

from app.routes import meal\_skip

from app.routes import feedback,auth,menu,tokens,monthly\_bill,home,panels

from starlette.middleware.sessions import SessionMiddleware

import os

from fastapi.staticfiles import StaticFiles

app = FastAPI()

app.add\_middleware(SessionMiddleware, secret\_key="superkey123")

app.mount("/static", StaticFiles(directory="venv/app/static"), name="static") #bg img

# Include the meal skip router

app.include\_router(meal\_skip.router)

app.include\_router(feedback.router)

app.include\_router(auth.router)

app.include\_router(menu.router)

app.include\_router(tokens.router)

app.include\_router(monthly\_bill.router)

app.include\_router(home.router)

app.include\_router(panels.router)

# Calculate full static directory path

static\_dir = os.path.join(os.path.dirname(\_\_file\_\_), "static")

app.mount("/static", StaticFiles(directory=static\_dir), name="static")

#@app.get("/")

#def read\_root():

    #return {"message": "Mess Manager is running!"}

**#auth.py**

from fastapi import APIRouter

import os

import psycopg2

router = APIRouter()

# Database connection

conn = psycopg2.connect(

    database="messManager",

    user="postgres",

    password="pavi",

    host="localhost",

    port="5432"

)

**#panels.py**

from fastapi import APIRouter, Request, Depends, Form, status, HTTPException

from fastapi.templating import Jinja2Templates

from fastapi.responses import RedirectResponse, HTMLResponse

import os

import psycopg2

router = APIRouter()

templates = Jinja2Templates(directory=os.path.join(os.path.dirname(\_\_file\_\_), "..", "templates"))

def get\_current\_user(request: Request):

    user\_id = request.session.get("user\_id")

    is\_admin = request.session.get("is\_admin")

    if not user\_id:

        return None

    return {"user\_id": user\_id, "is\_admin": is\_admin}

@router.get("/adminpanel")

def admin\_panel(request: Request, user=Depends(get\_current\_user)):

    if not user:

        return RedirectResponse(url="/")

    if not user["is\_admin"]:

        return RedirectResponse(url="/studentpanel")

    return templates.TemplateResponse("adminpanel.html", {"request": request, "user\_id": user["user\_id"]})

@router.get("/studentpanel")

def student\_panel(request: Request, user=Depends(get\_current\_user)):

    if not user:

        return RedirectResponse(url="/")

    if user["is\_admin"]:

        return RedirectResponse(url="/adminpanel")

    return templates.TemplateResponse("studentpanel.html", {"request": request, "user\_id": user["user\_id"]})

@router.get("/logout")

def logout(request: Request):

    request.session.clear()

    return RedirectResponse(url="/")

@router.get("/change\_password", response\_class=HTMLResponse)

def change\_password\_form(request: Request, user=Depends(get\_current\_user)):

    if not user:

        return RedirectResponse(url="/")

    return templates.TemplateResponse("changepassword.html", {"request": request, "user\_id": user["user\_id"]})

@router.post("/change\_password")

def update\_password(

    request: Request,

    current\_password: str = Form(...),

    new\_password: str = Form(...),

    user=Depends(get\_current\_user)

):

    if not user:

        return RedirectResponse(url="/")

    conn = psycopg2.connect(

        dbname="messManager",

        user="postgres",

        password="pavi",

        host="localhost",

        port="5432"

    )

    cur = conn.cursor()    cur.execute("SELECT password FROM users WHERE user\_id = %s", (user["user\_id"],))

    result = cur.fetchone()

    if not result or result[0] != current\_password:

        cur.close()

        conn.close()

        raise HTTPException(status\_code=status.HTTP\_400\_BAD\_REQUEST, detail="Incorrect current password.")

    cur.execute("UPDATE users SET password = %s WHERE user\_id = %s", (new\_password, user["user\_id"]))

    conn.commit()

    cur.close()

    conn.close()

    if user["is\_admin"]:

        return RedirectResponse(url="/adminpanel", status\_code=status.HTTP\_303\_SEE\_OTHER)

    else:

        return RedirectResponse(url="/studentpanel", status\_code=status.HTTP\_303\_SEE\_OTHER)

**#home.py**

from fastapi import APIRouter, Request, Form, status

from fastapi.responses import RedirectResponse

from fastapi.templating import Jinja2Templates

import os

import psycopg2

router = APIRouter()

templates = Jinja2Templates(directory=os.path.join(os.path.dirname(\_\_file\_\_), "..", "templates"))

# Database connection

conn = psycopg2.connect(

    database="messManager",

    user="postgres",

    password="pavi",

    host="localhost",

    port="5432"

)

@router.get("/")

async def home(request: Request):

    user\_id = request.session.get("user\_id")

    is\_admin = request.session.get("is\_admin")

    if user\_id and is\_admin is not None:

        if is\_admin:

            return RedirectResponse(url="/adminpanel", status\_code=status.HTTP\_302\_FOUND)

        else:

            return RedirectResponse(url="/studentpanel", status\_code=status.HTTP\_302\_FOUND)

    return templates.TemplateResponse("home.html", {"request": request, "error": None})

@router.post("/login")

async def login(

    request: Request,

    user\_id: str = Form(...),

    password: str = Form(...)

):

    if not user\_id or not password:

        return templates.TemplateResponse("home.html", {

            "request": request,

            "error": "User ID and Password are required"

        })

    try:

        cur = conn.cursor()

        cur.execute("SELECT is\_admin FROM users WHERE user\_id = %s AND password = %s", (user\_id, password))

        result = cur.fetchone()

        cur.close()

    except Exception as e:

        return templates.TemplateResponse("home.html", {

            "request": request,

            "error": "Database error: " + str(e)

        })

    if not result:

        return templates.TemplateResponse("home.html", {

            "request": request,

            "error": "Invalid User ID or Password"

        })

    is\_admin = result[0]

    request.session["user\_id"] = user\_id

    request.session["is\_admin"] = is\_admin

    if is\_admin:

        return RedirectResponse(url="/adminpanel", status\_code=status.HTTP\_302\_FOUND)

    else:

        return RedirectResponse(url="/studentpanel", status\_code=status.HTTP\_302\_FOUND)