Conceptual Framework

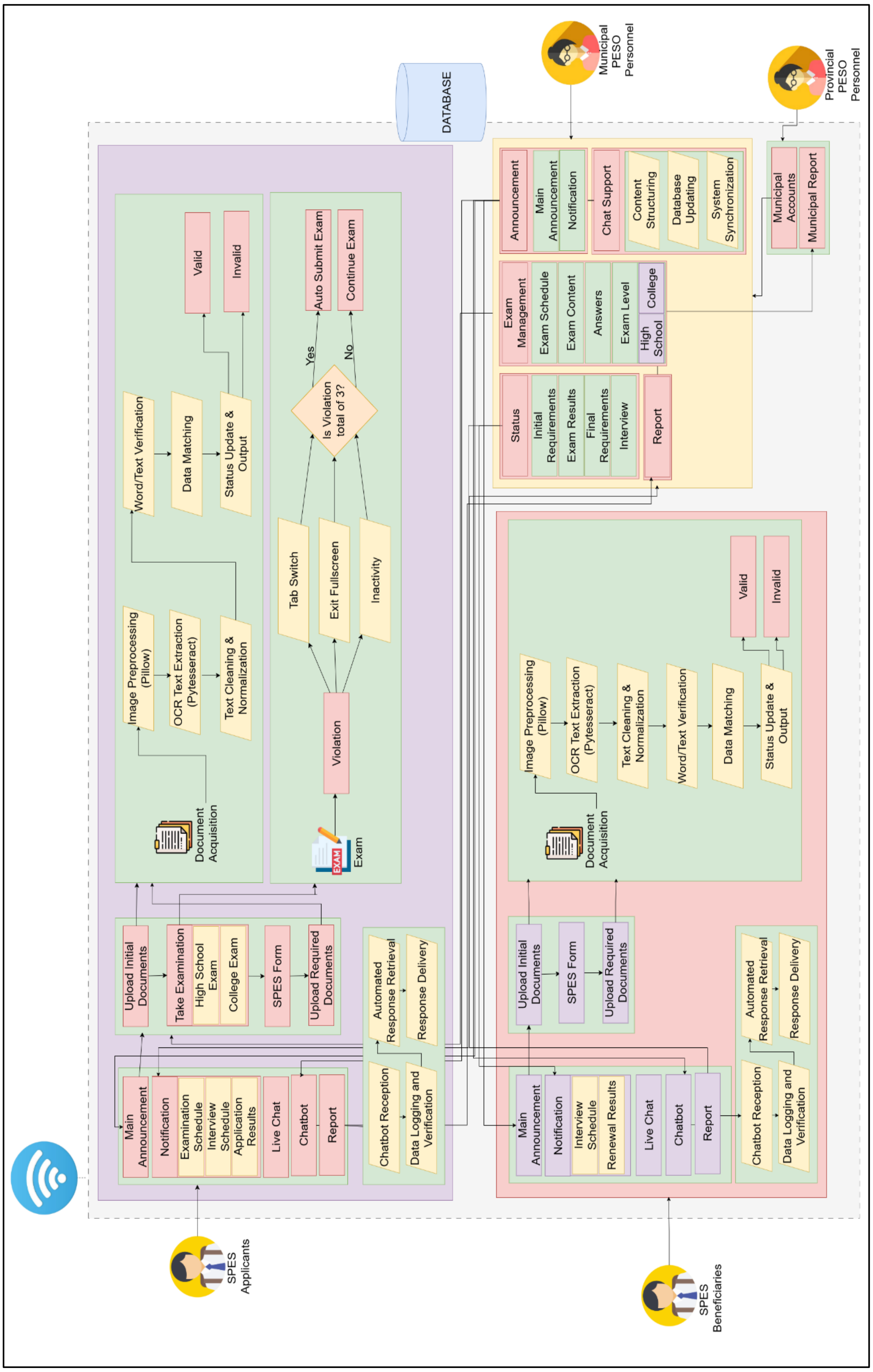
As shown in Figure 1, the DigiSPES Conceptual Framework illustrates the optimized processes of the Special Program for Employment of Students (SPES) through a web-based platform designed to streamline applications, examinations, and data analytics for improved management. The framework integrates various stakeholders, namely SPES applicants, SPES beneficiaries, municipal PESO personnel, and provincial PESO personnel, ensuring a seamless flow of information and efficient coordination. Each module within the system has a specific role, from application and document handling to examination monitoring and centralized data management, all of which contribute to a transparent and effective program implementation.

The **SPES Applicants** module provides students with a digital platform to manage their application process conveniently. Applicants can access announcements, schedules, interview details, and results through an interactive dashboard. They can also upload initial and required documents, take examinations, and complete the SPES form online. To enhance engagement and support, applicants are given access to chatbot reception, live chat, and automated response retrieval for inquiries. The module also includes intelligent monitoring of examination activity, such as detecting tab switches or inactivity, ensuring exam integrity by automatically submitting the test after repeated violations. This creates a fair and accountable examination process for all applicants.

The **SPES Beneficiaries** module is designed for returning participants or those renewing their SPES status. Similar to the applicants, beneficiaries can upload renewal documents, complete SPES forms, and check announcements, notifications, and results of renewal applications. Additionally, they are supported by live chat and automated chatbot responses for faster communication. Document verification is streamlined using image preprocessing, text extraction, and data matching to validate uploaded documents quickly. By digitizing the renewal process, beneficiaries experience reduced paperwork and faster processing times, encouraging smoother participation in the program year after year.

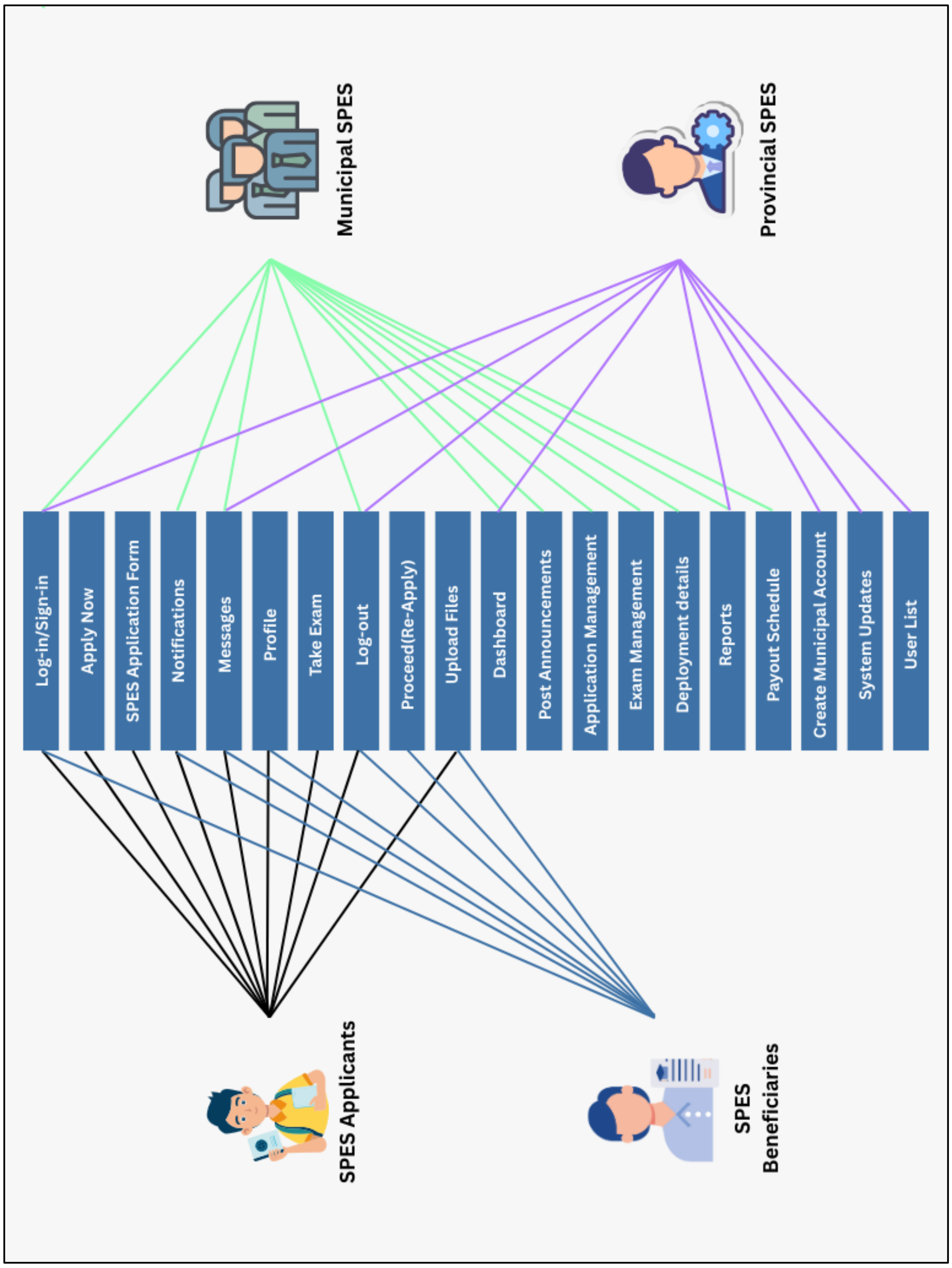
The **Municipal PESO Personnel** module focuses on the local-level management of SPES. Personnel are provided with tools for managing applicant and beneficiary data, including announcements, exam scheduling, content preparation, and interview results. They can monitor exam levels (high school or college), validate requirements, and report the municipal stage. Furthermore, the module provides database updating, chat support, and structured content management, ensuring that information is accurate and accessible. Reports are generated at the municipal level, giving personnel a clear overview of passers and program participants under their jurisdiction for more effective monitoring.

Lastly, the **Provincial PESO Personnel** module integrates higher-level oversight and ensures synchronization across multiple municipalities. This module allows provincial personnel to access consolidated reports, oversee system synchronization, and validate municipal-level outputs. By reviewing municipal level passers, provincial PESO personnel ensure that program implementation is standardized and consistent. This centralized management strengthens accountability and enhances decision-making by using real-time data analytics.

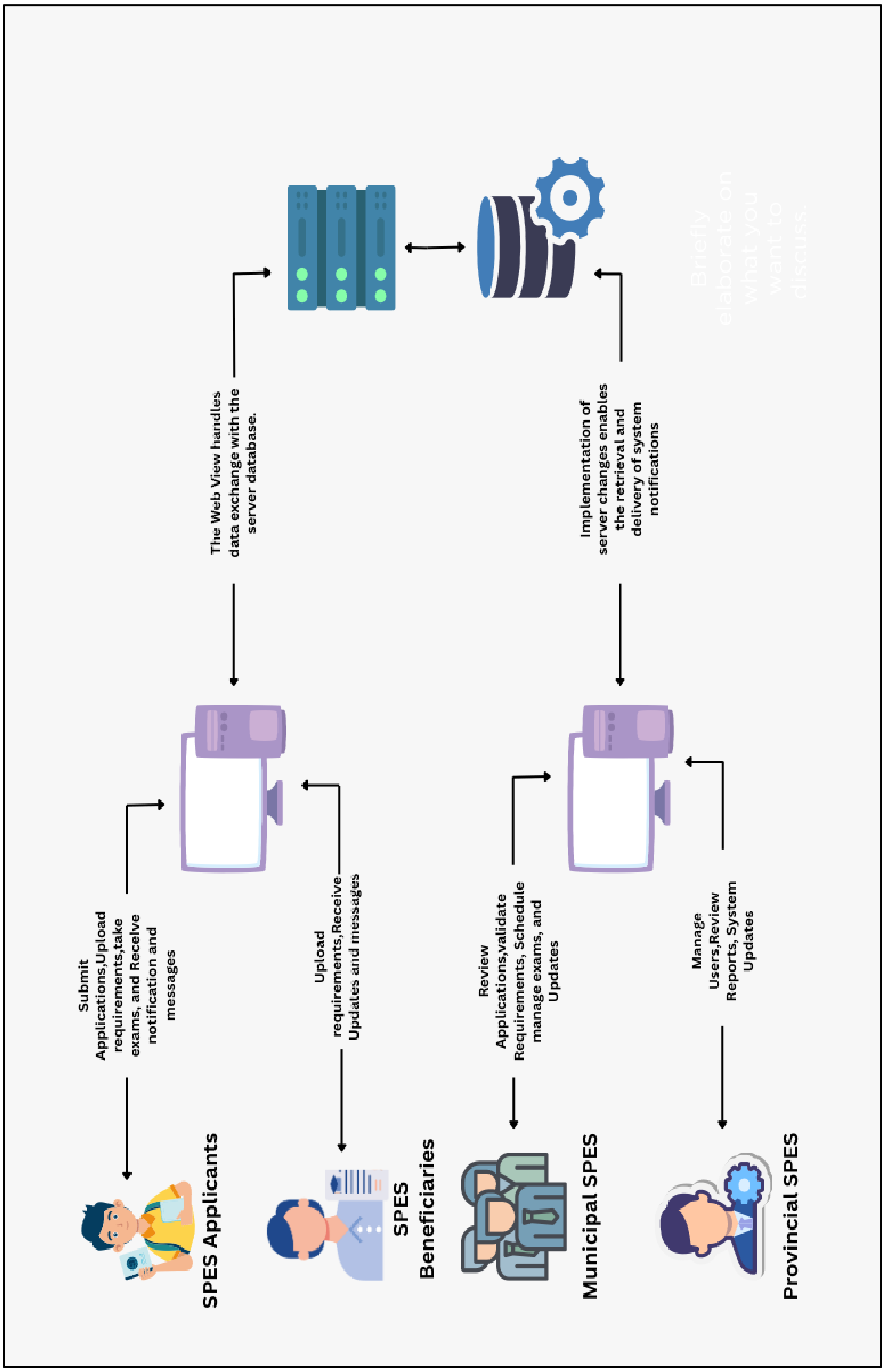


SPES Applicants
SPES Beneficiaries
Municipal PESO Personnel
Provincial PESO Personnel
DATABASE
Main Announcement
Notification
Examination Schedule
Interview Schedule
Application Results
Live Chat
Chatbot
Report
Upload Initial Documents
Take Examination
High School Exam
College Exam
SPES Form
Upload Required Documents
Chatbot Reception
Data Logging and Verification
Automated Response Retrieval
Response Delivery
Document Acquisition
Image Preprocessing (Pillow)
OCR Text Extraction (Pytesseract)
Text Cleaning & Normalization
Word/Text Verification
Data Matching
Status Update & Output
Valid
Invalid
Exam
Violation
Tab Switch
Exit Fullscreen
Inactivity
Is Violation total of 3?
Yes
No
Auto Submit Exam
Continue Exam
Interview Schedule
Renewal Results
Announcement
Main Announcement
Notification
Chat Support
Content Structuring
Database Updating
System Synchronization
Exam Management
Exam Schedule
Exam Content
Answers
Exam Level
High School
College
Status
Initial Requirements
Exam Results
Final Requirements
Interview
Report
Municipal Accounts
Municipal Report

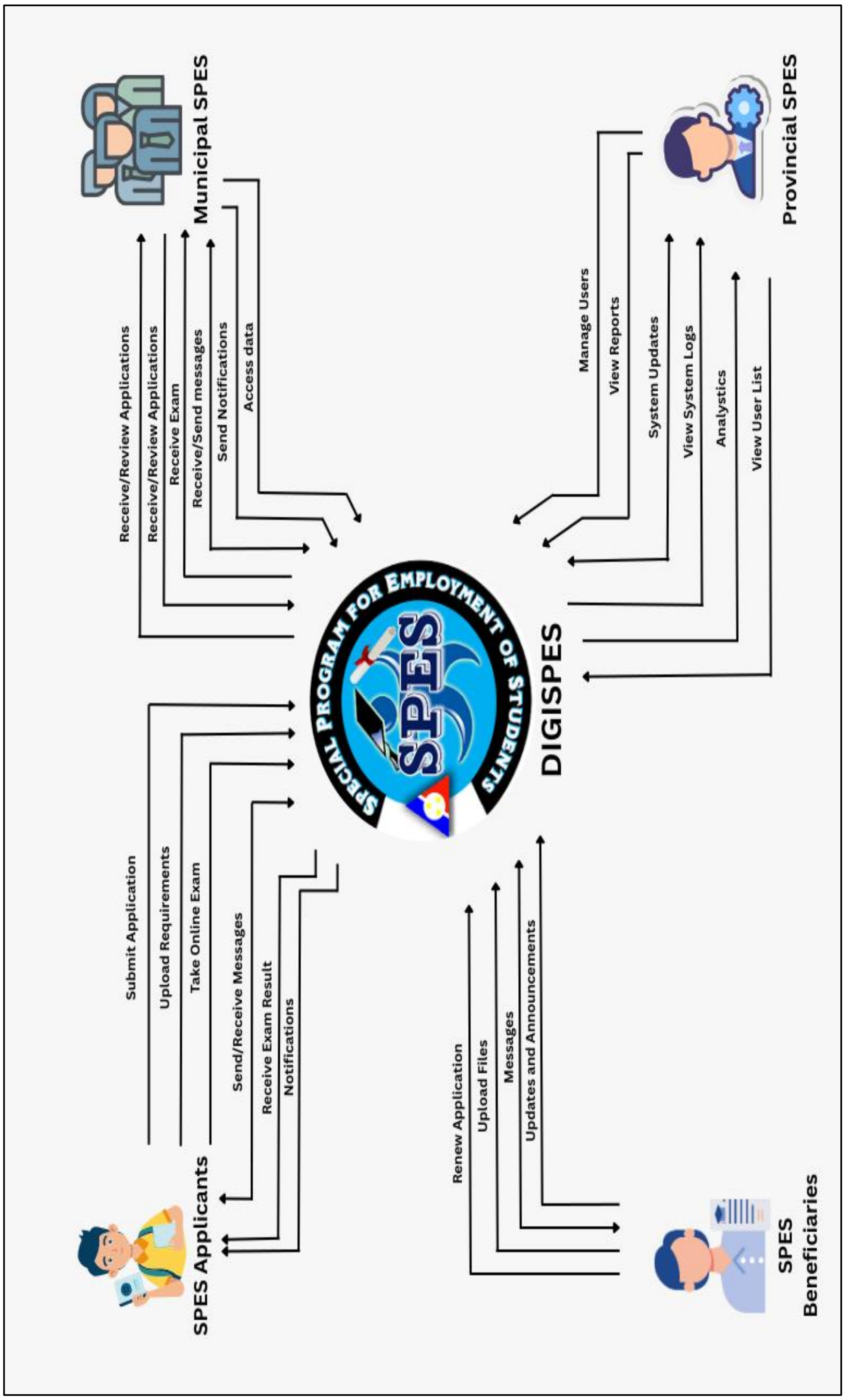
Use Case Diagram



Module Specification

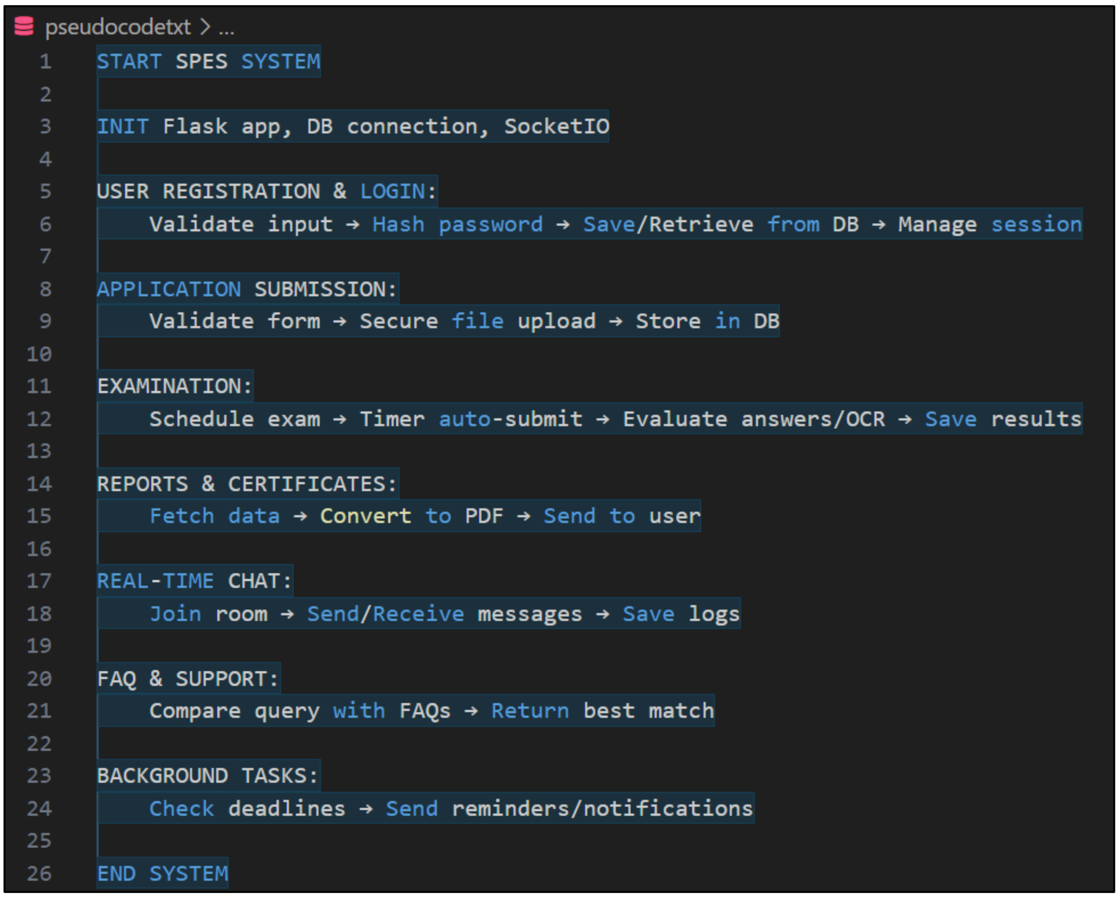

SPES Applicants
SPES Beneficiaries
Municipal SPES
Provincial SPES
Submit Applications,Upload requirements,take exams, and Receive notification and messages
Upload requirements,Receive Updates and messages
Review Applications,validate Requirements, Schedule manage exams, and Updates
Manage Users,Review Reports, System Updates
The Web View handles data exchange with the server database.
Implementation of server changes enables the retrieval and delivery of system notifications
Briefly elaborate on what you want to discuss.

System Context Diagram



Development (Prototyping and Early System Development)

During this phase, the team also developed a pseudocode that outlined the core logic and flow of the system, serving as a bridge between planning and actual coding. This pseudocode defined key processes such as user registration, login authentication, application submission, and automated result generation. By mapping out these components early, the team ensured a clear understanding of the system's functionality, which streamlined both design and development. It also facilitated more efficient communication among developers and allowed potential issues to be identified and addressed before full-scale implementation began.



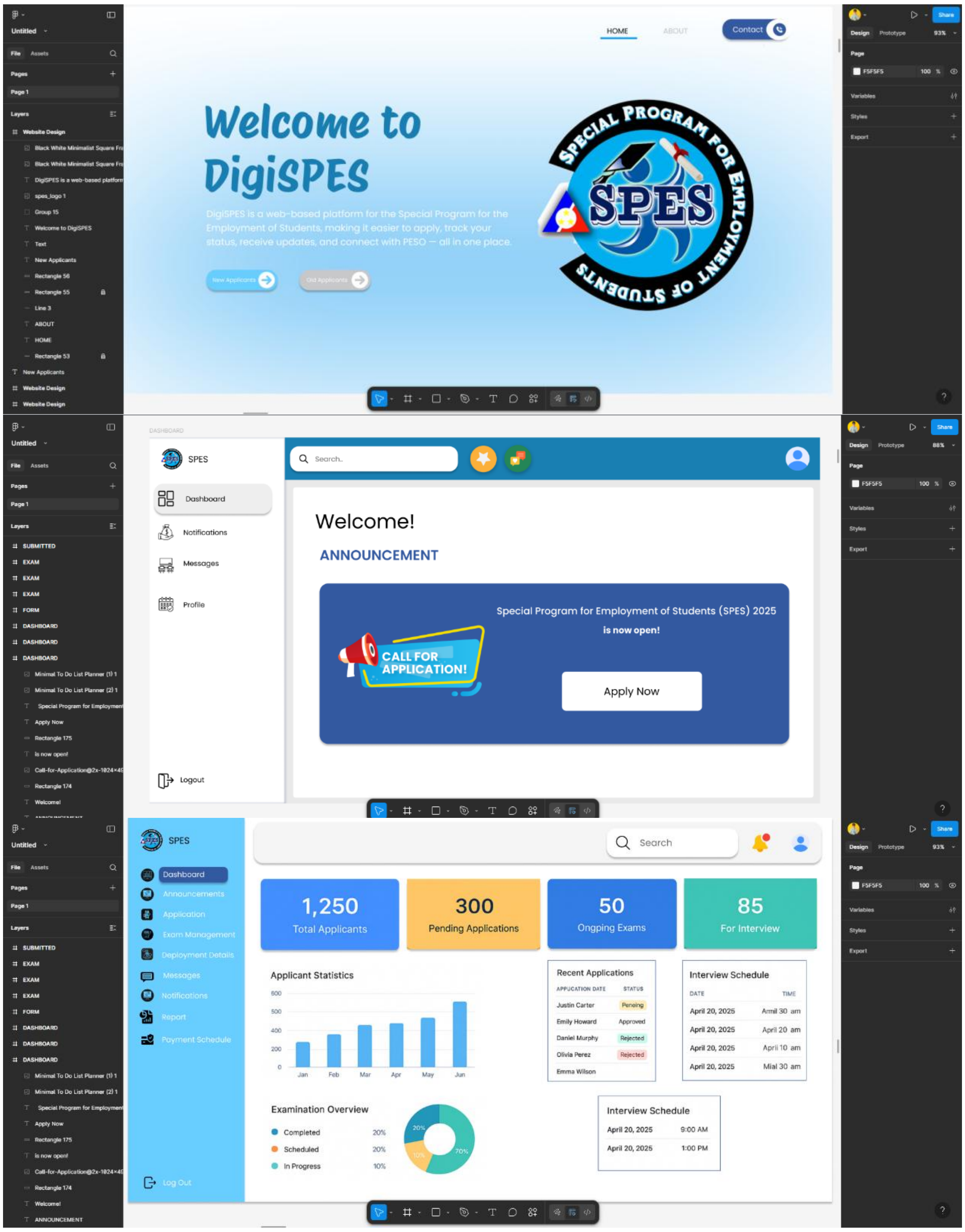
pseudocodetxt > ...

```
START SPES SYSTEM

INIT Flask app, DB connection, SocketIO

USER REGISTRATION & LOGIN:
    Validate input → Hash password → Save/Retrieve from DB → Manage session

APPLICATION SUBMISSION:
    Validate form → Secure file upload → Store in DB

EXAMINATION:
    Schedule exam → Timer auto-submit → Evaluate answers/OCR → Save results

REPORTS & CERTIFICATES:
    Fetch data → Convert to PDF → Send to user

REAL-TIME CHAT:
    Join room → Send/Receive messages → Save logs

FAQ & SUPPORT:
    Compare query with FAQs → Return best match

BACKGROUND TASKS:
    Check deadlines → Send reminders/notifications

END SYSTEM
```

Figure 13. *Pseudocode*

Also, the team conducted prototyping using Figma, transitioning from low-fidelity wireframes to high-fidelity interactive prototypes that closely resembled the final product in both design and functionality. These prototypes were designed with consistent branding, typography, and color schemes to ensure they were visually appealing and easy to use. Interactive elements such as dropdowns, buttons, and navigation flows were connected to mimic real user interactions, allowing stakeholders to explore and experience the application before actual development began.

The researchers held regular usability checking sessions with selected users to identify design flaws, usability problems, or navigation errors. The feedback gathered played a significant role in refining the layout and ensuring that the user experience remained intuitive and easy to navigate. Using Figma for this iterative design process enabled the team to detect and resolve issues early, keeping the platform aligned with user expectations and the project objectives.



Welcome to DigiSPES

DigiSPES is a web-based platform for the Special Program for the Employment of Students, making it easier to apply, track your status, receive updates, and connect with PESO — all in one place.

Welcome!

ANNOUNCEMENT

Special Program for Employment of Students (SPES) 2025 is now open!

1,250 Total Applicants

300 Pending Applications

50 Ongping Exams

85 For Interview

Recent Applications

APPUCATION DATE	STATUS
Justin Carter	Peneing
Emily Howard	Approved
Daniel Murphy	Rejected
Olivia Perez	Rejected
Emma Wilson	

Interview Schedule

DATE	TIME
April 20, 2025	Amil 30 am
April 20, 2025	April 20 am
April 20, 2025	April 10 am
April 20, 2025	Mial 30 am

Interview Schedule

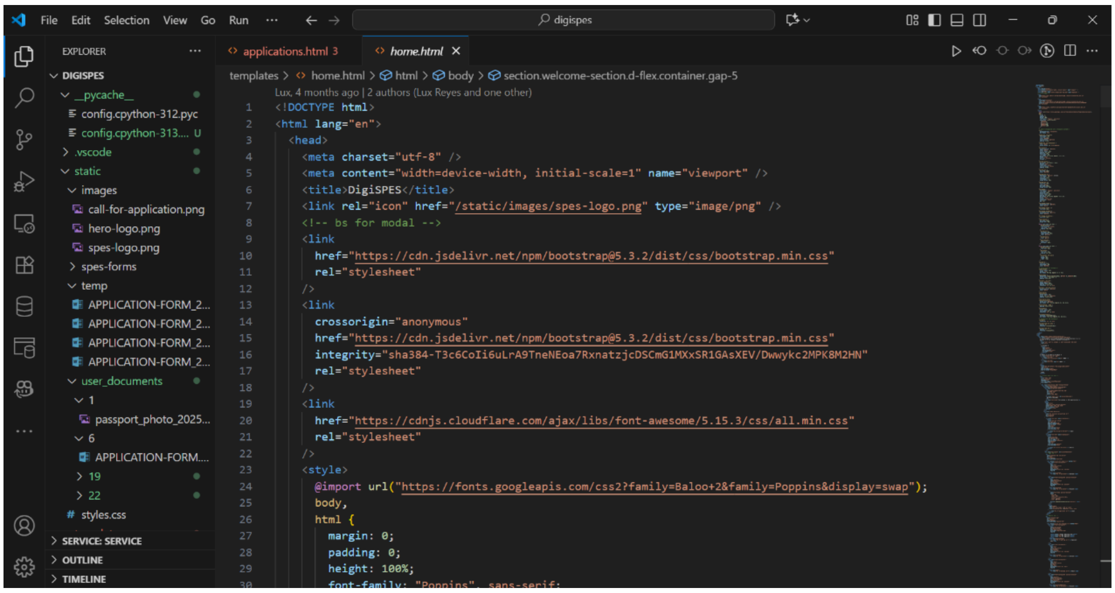
April 20, 2025	9:00 AM
April 20, 2025	1:00 PM

● Completed	20%
● Scheduled	20%
● In Progress	10%

Figure 14. *DigiSPES Prototype*

For development, VS Code was used as the Integrated Development Environment (IDE). It provided a powerful and customizable environment for writing and debugging code, with features like syntax highlighting, auto-completion, and extensions tailored for both front-end and back-end development. The IDE streamlined the coding process, helping the team maintain consistency and efficiency across the platform. With VS Code's built-in terminal and version control integration, the development process was smooth and well-organized, allowing the team to easily manage and track changes throughout the project.

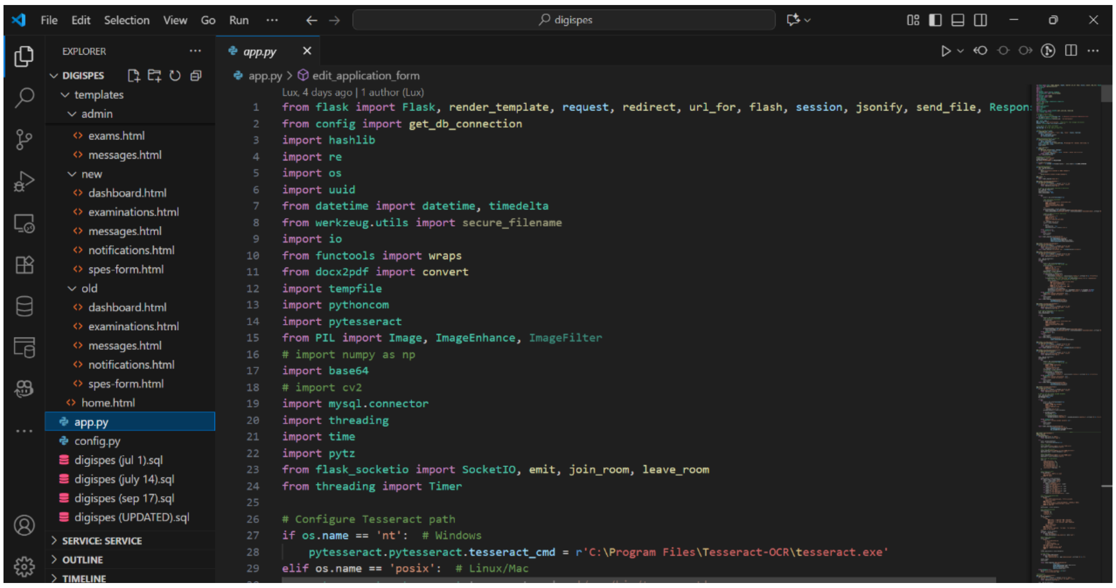
For the front-end, the team used HTML, CSS, and JavaScript to develop a user-friendly interface. These technologies ensured that the platform was responsive and compatible across different devices and browsers. The front-end was designed to align with the prototypes created in Figma, maintaining a consistent and visually appealing design while providing intuitive navigation for users.



```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta content="width=device-width, initial-scale=1" name="viewport" />
    <title>DigiSPES</title>
    <link rel="icon" href="/static/images/spes-logo.png" type="image/png" />
    <!-- bs for modal -->
    <link
      href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css"
      rel="stylesheet"
    />
    <link
      crossorigin="anonymous"
      href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css"
      integrity="sha384-T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/Dwwykc2MPK8M2HN"
      rel="stylesheet"
    />
    <link
      href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.3/css/all.min.css"
      rel="stylesheet"
    />
    <style>
      @import url("https://fonts.googleapis.com/css2?family=Baloo+2&family=Poppins&display=swap");
      body,
      html {
        margin: 0;
        padding: 0;
        height: 100%;
        font-family: "Poppins", sans-serif;
```

Figure 15. *DigiSPES Frontend in VS Code*

On the back-end, Python was used to handle server-side operations, including data processing and form management. Python enabled the efficient handling of logic and database interaction, ensuring smooth operation and quick response times. The combination of Python for back-end processing and XAMPP for database management allowed the system to run seamlessly, providing the necessary functionality for a robust platform.



```python
from flask import Flask, render_template, request, redirect, url_for, flash, session, jsonify, send_file, Respon
from config import get_db_connection
import hashlib
import re
import os
import uuid
from datetime import datetime, timedelta
from werkzeug.utils import secure_filename
import io
from functools import wraps
from docx2pdf import convert
import tempfile
import pythoncom
import pytesseract
from PIL import Image, ImageEnhance, ImageFilter
# import numpy as np
import base64
# import cv2
import mysql.connector
import threading
import time
import pytz
from flask_socketio import SocketIO, emit, join_room, leave_room
from threading import Timer

# Configure Tesseract path
if os.name == 'nt':  # Windows
    pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'
elif os.name == 'posix':  # Linux/Mac
```

Figure 16. *DigiSPES Backend in VS Code*

After it, the team began building the actual platform. XAMPP was used as the local server environment, providing the necessary tools for developing and testing the system. It served as the DBMS (Database Management System), using MySQL to securely store user information, application data, and administrative records. XAMPP ensured smooth development by enabling easy configuration and management of the database, streamlining the process of setting up the environment for testing.

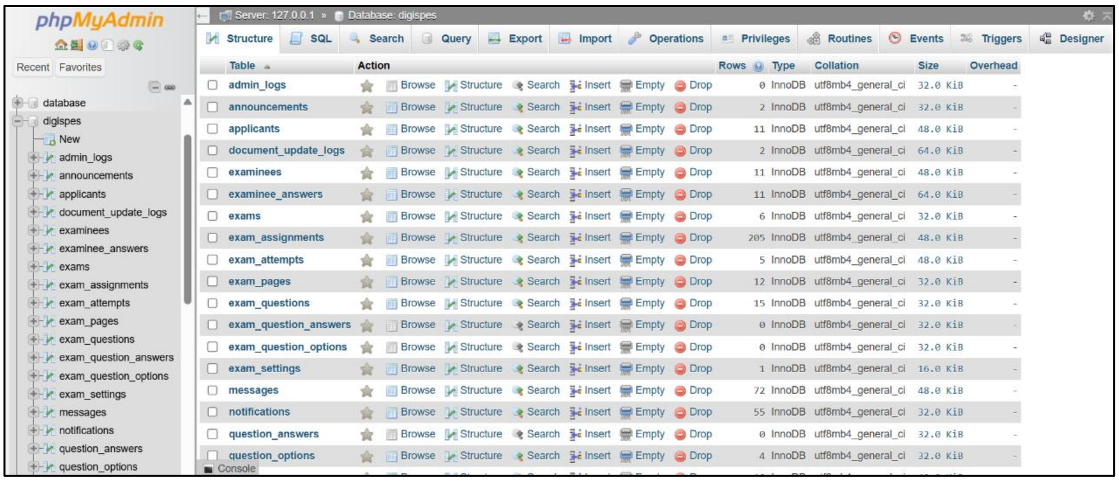
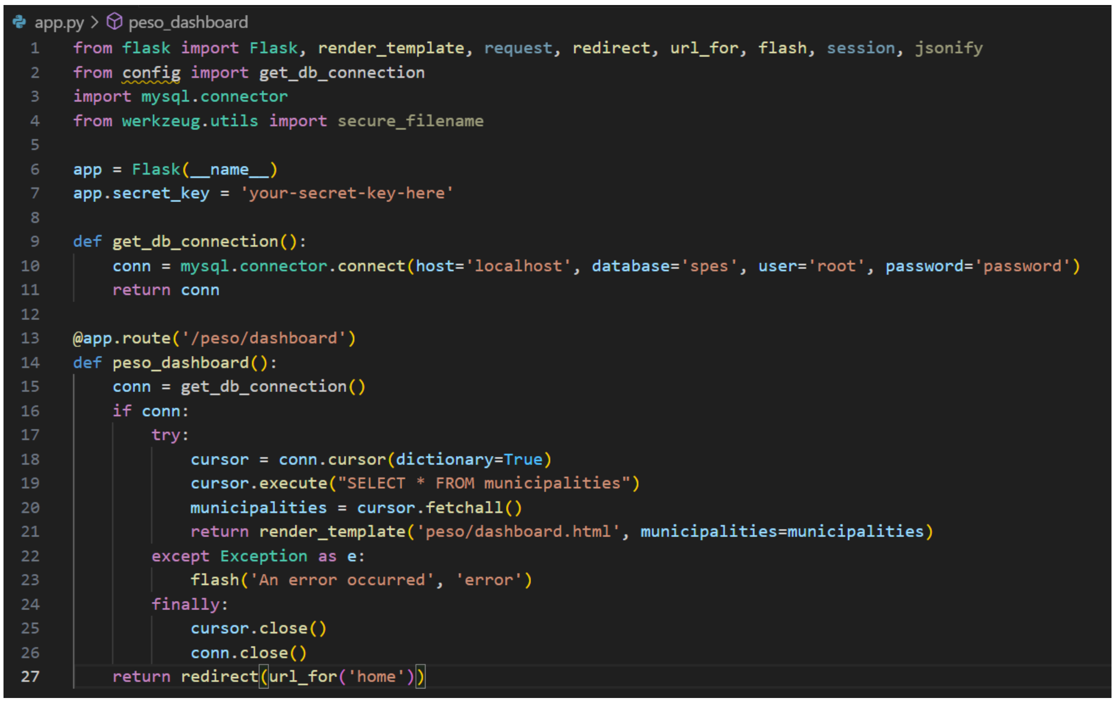


Table	Action	Rows	Type	Collation	Size	Overhead
admin_logs	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	32.0 KiB	-
announcements	Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_general_ci	32.0 KiB	-
applicants	Browse Structure Search Insert Empty Drop	11	InnoDB	utf8mb4_general_ci	48.0 KiB	-
document_update_logs	Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_general_ci	64.0 KiB	-
examinees	Browse Structure Search Insert Empty Drop	11	InnoDB	utf8mb4_general_ci	48.0 KiB	-
examinee_answers	Browse Structure Search Insert Empty Drop	11	InnoDB	utf8mb4_general_ci	64.0 KiB	-
exams	Browse Structure Search Insert Empty Drop	6	InnoDB	utf8mb4_general_ci	32.0 KiB	-
exam_assignments	Browse Structure Search Insert Empty Drop	205	InnoDB	utf8mb4_general_ci	48.0 KiB	-
exam_attempts	Browse Structure Search Insert Empty Drop	5	InnoDB	utf8mb4_general_ci	48.0 KiB	-
exam_pages	Browse Structure Search Insert Empty Drop	12	InnoDB	utf8mb4_general_ci	32.0 KiB	-
exam_questions	Browse Structure Search Insert Empty Drop	15	InnoDB	utf8mb4_general_ci	32.0 KiB	-
exam_question_answers	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	32.0 KiB	-
exam_question_options	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	32.0 KiB	-
exam_settings	Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_general_ci	16.0 KiB	-
messages	Browse Structure Search Insert Empty Drop	72	InnoDB	utf8mb4_general_ci	48.0 KiB	-
notifications	Browse Structure Search Insert Empty Drop	55	InnoDB	utf8mb4_general_ci	32.0 KiB	-
question_answers	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	32.0 KiB	-
question_options	Browse Structure Search Insert Empty Drop	4	InnoDB	utf8mb4_general_ci	32.0 KiB	-

Figure 17. *DigiSPES XAMPP as DBMS*

Implementation (System Implementation and Feature Development)

Once a clear plan had been established, the process of building the system began. The backend was developed using Python and the Flask framework, chosen for its lightweight yet flexible nature, allowing efficient management of routes, user sessions, form handling, and dynamic HTML template rendering. For security and proper platform management, user authentication and access control were implemented, ensuring that only authorized individuals could access or modify sensitive data. All records, including user details, applications, and examination data, were securely stored in a MySQL database, connected through mysql.connector, ensuring a stable and reliable data management layer.

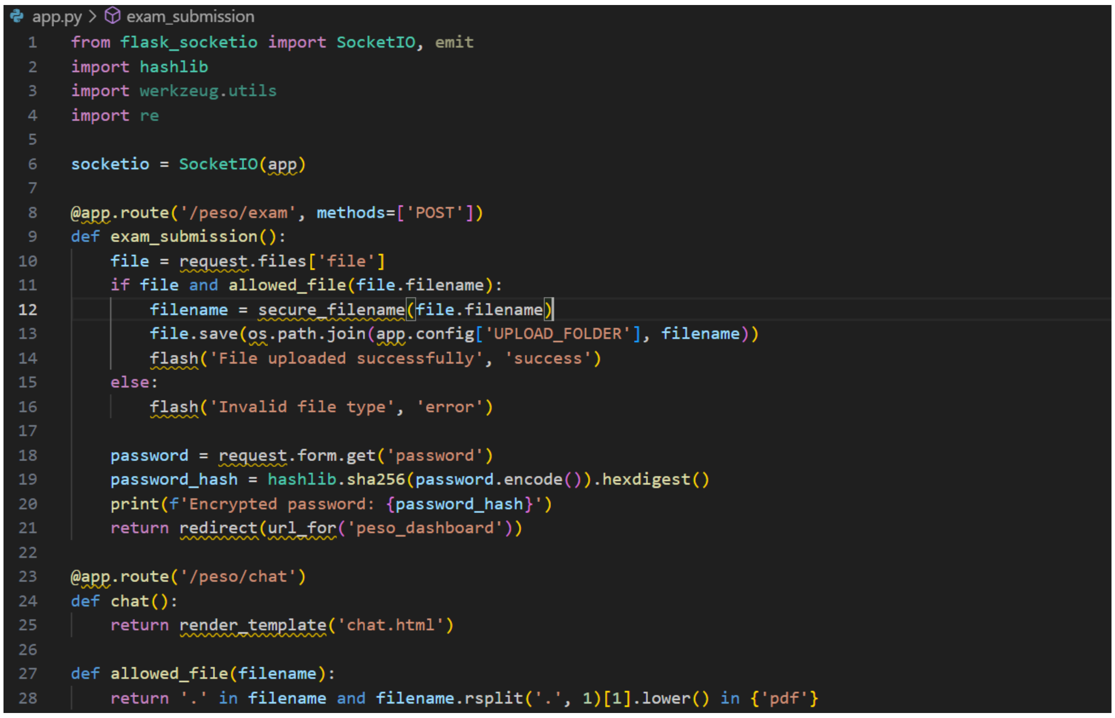


app.py > peso_dashboard

```python
from flask import Flask, render_template, request, redirect, url_for, flash, session, jsonify
from config import get_db_connection
import mysql.connector
from werkzeug.utils import secure_filename

app = Flask(__name__)
app.secret_key = 'your-secret-key-here'

def get_db_connection():
    conn = mysql.connector.connect(host='localhost', database='spes', user='root', password='password')
    return conn

@app.route('/peso/dashboard')
def peso_dashboard():
    conn = get_db_connection()
    if conn:
        try:
            cursor = conn.cursor(dictionary=True)
            cursor.execute("SELECT * FROM municipalities")
            municipalities = cursor.fetchall()
            return render_template('peso/dashboard.html', municipalities=municipalities)
        except Exception as e:
            flash('An error occurred', 'error')
        finally:
            cursor.close()
            conn.close()
    return redirect(url_for('home'))
```

Figure 16. *Python and the Flask framework*

For real-time functionality, flask_socketio was integrated to enable two-way communication, supporting features like live chat assistance and synchronized exam timers. werkzeug.utils.secure_filename was used to ensure secure file handling, while hashlib was used for password hashing and encryption. The re module (Regular Expressions) was applied to validate inputs and match text patterns effectively, preventing invalid or malicious data from entering the system.



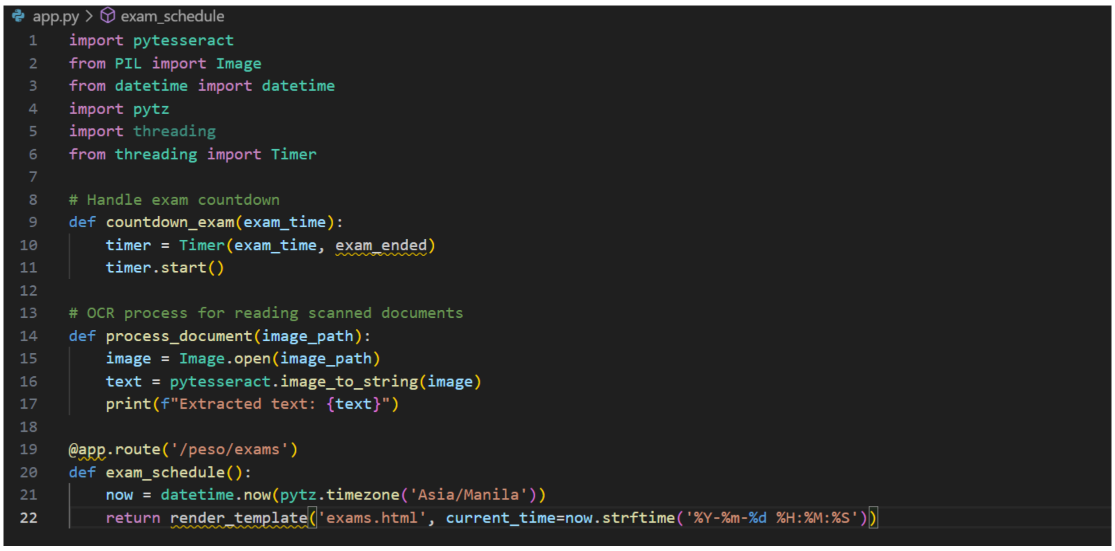
app.py > exam_submission

```python
from flask_socketio import SocketIO, emit
import hashlib
import werkzeug.utils
import re

socketio = SocketIO(app)

@app.route('/peso/exam', methods=['POST'])
def exam_submission():
    file = request.files['file']
    if file and allowed_file(file.filename):
        filename = secure_filename(file.filename)
        file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
        flash('File uploaded successfully', 'success')
    else:
        flash('Invalid file type', 'error')

    password = request.form.get('password')
    password_hash = hashlib.sha256(password.encode()).hexdigest()
    print(f'Encrypted password: {password_hash}')
    return redirect(url_for('peso_dashboard'))

@app.route('/peso/chat')
def chat():
    return render_template('chat.html')

def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in {'pdf'}
```

Figure 17. *Real-Time Functionality*

The system incorporated datetime and pytz for accurate time management, especially when managing exam schedules and deadlines across different regions. Timed events such as countdown timers for exams or automatic session logouts were handled with threading and Timer. Additionally, OCR support was integrated using pytesseract and PIL (Pillow) for the automatic reading and verification of scanned documents. docx2pdf was used to convert Word files into PDFs for uniformity and easy distribution.

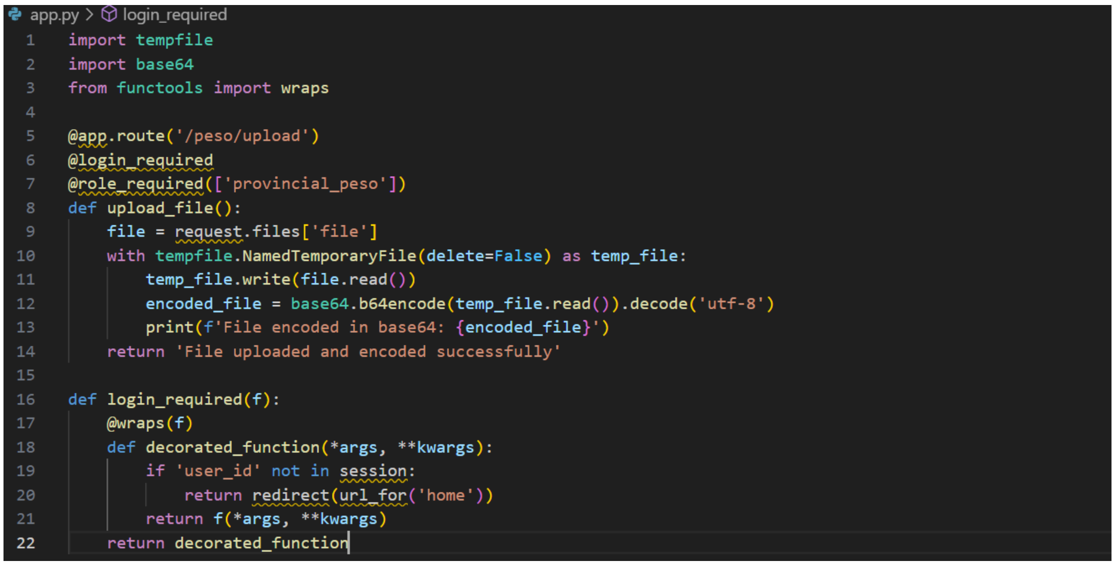
Figure 18. *OCR Process Flow Diagram and Time Management*



app.py > exam_schedule

```python
import pytesseract
from PIL import Image
from datetime import datetime
import pytz
import threading
from threading import Timer

# Handle exam countdown
def countdown_exam(exam_time):
    timer = Timer(exam_time, exam_ended)
    timer.start()

# OCR process for reading scanned documents
def process_document(image_path):
    image = Image.open(image_path)
    text = pytesseract.image_to_string(image)
    print(f"Extracted text: {text}")

@app.route('/peso/exams')
def exam_schedule():
    now = datetime.now(pytz.timezone('Asia/Manila'))
    return render_template('exams.html', current_time=now.strftime('%Y-%m-%d %H:%M:%S'))
```

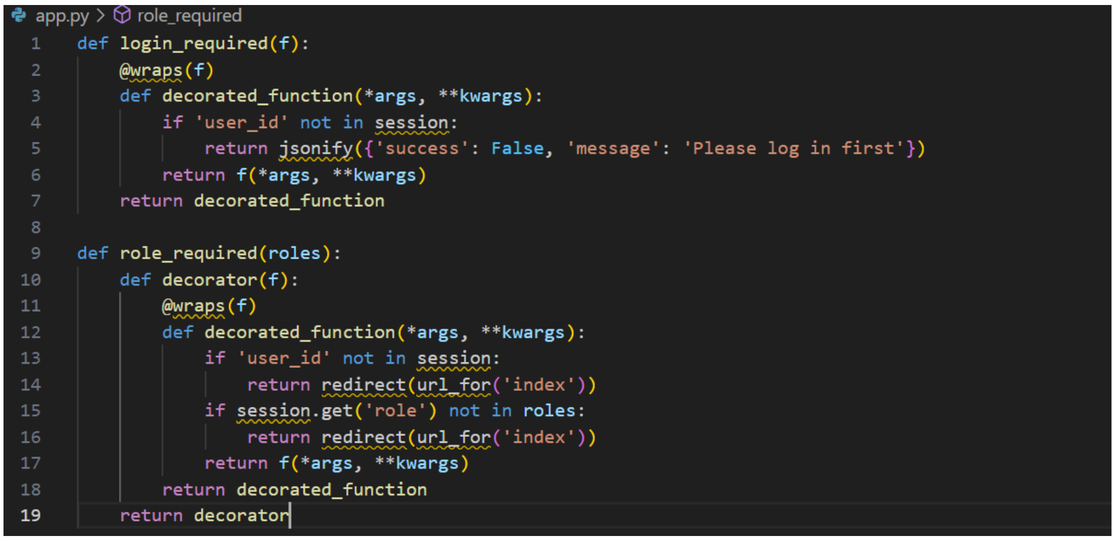
Also, tempfile, base64, and functools.wraps were employed for efficient file handling and route protection. tempfile was used to handle temporary file storage, base64 was used to encode files securely, and functools.wraps helped implement decorators for login authentication and access control to sensitive routes.



```
app.py > login_required
import tempfile
import base64
from functools import wraps

@app.route('/peso/upload')
@login_required
@role_required(['provincial_peso'])
def upload_file():
    file = request.files['file']
    with tempfile.NamedTemporaryFile(delete=False) as temp_file:
        temp_file.write(file.read())
        encoded_file = base64.b64encode(temp_file.read()).decode('utf-8')
        print(f'File encoded in base64: {encoded_file}')
    return 'File uploaded and encoded successfully'

def login_required(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        if 'user_id' not in session:
            return redirect(url_for('home'))
        return f(*args, **kwargs)
    return decorated_function
```

Figure 17. *File Handling*

Lastly, to ensure the protection of sensitive information, hashlib and werkzeug were employed for encrypting user passwords and handling file uploads securely. werkzeug.utils.secure_filename was used to safely store uploaded files, preventing potential security risks from malicious uploads. The platform also integrated login_required and role_required decorators to enforce authentication and ensure that users can only access pages relevant to their roles. This strict access control system significantly improves the security and integrity of the platform, ensuring that only authorized personnel can access sensitive data.



app.py > role_required

```python
def login_required(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        if 'user_id' not in session:
            return jsonify({'success': False, 'message': 'Please log in first'})
        return f(*args, **kwargs)
    return decorated_function

def role_required(roles):
    def decorator(f):
        @wraps(f)
        def decorated_function(*args, **kwargs):
            if 'user_id' not in session:
                return redirect(url_for('index'))
            if session.get('role') not in roles:
                return redirect(url_for('index'))
            return f(*args, **kwargs)
        return decorated_function
    return decorator
```

Figure 18. *Security Data*