TECHNICAL DOCUMENTATION

mddconstantino@mymail.mapua.edu.ph (09628493593)

radcordero@mymail.mapua.edu.ph (09088654210)

dmcflores@mymail.mapua.edu.ph (09062654515)

smzamoras@mymail.mapua.edu.ph (09626712840)

Table of Contents

Technical Manual

Prepared By: WeatherWise

1. System Overview

1.1. Architecture

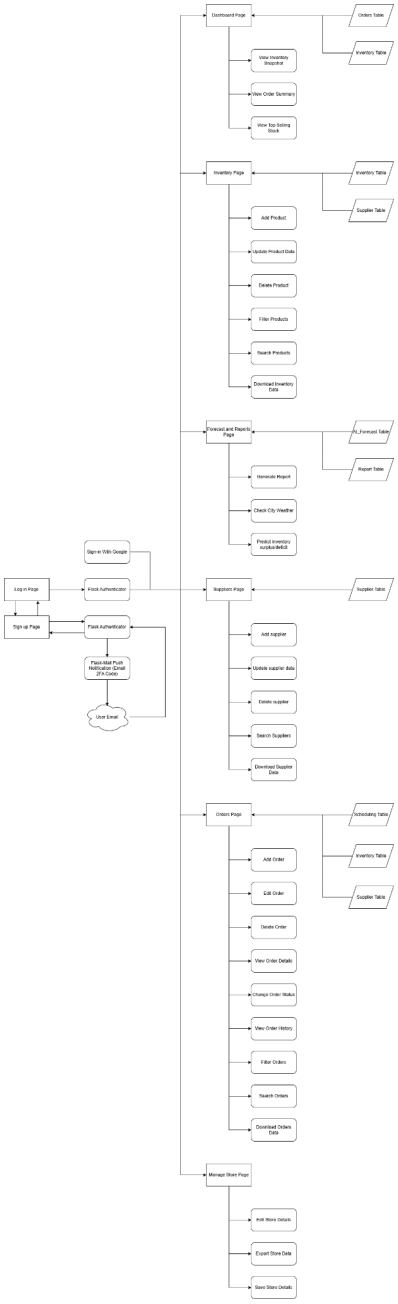


Figure 1.1 System Architecture

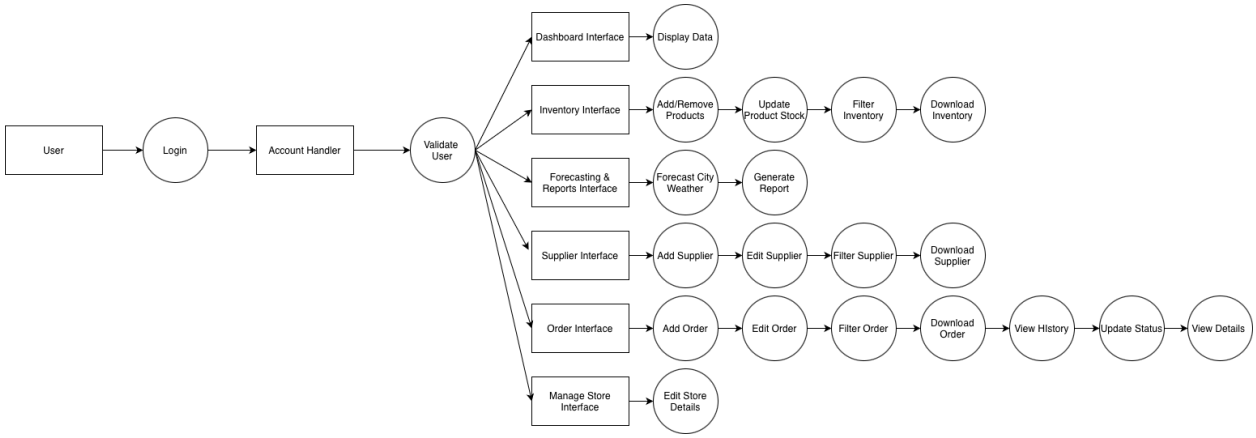


Figure 1.2 System Data Flow

1.2. Technologies Used and Dependencies

1.2.1. Frontend Framework

- HTML5 and CSS3**:** Core technologies used for layout and styling. Provide structure, responsive grids, and visual consistency across all browsers.
- JavaScript (Vanilla)**:** Handles client-side interactivity and asynchronous data fetching using the Fetch API.
- Chart.js**:** Used for rendering graphical representations of sales, orders, and forecasts within the dashboard.

1.2.2. Build and Tooling

- Visual Studio Code: The main IDE for writing and testing frontend and backend scripts.
- GitHub: Used for version control, issue tracking, and collaborative development.
- Flask Development Server: The built-in Flask server is used during local testing at http://127.0.0.1:5000/.
- PythonAnywhere: Used for deploying the final version of the WeatherWise application with live MySQL integration.

1.2.3. Web Application and Porting

- Browser Compatibility: WeatherWise runs seamlessly on modern browsers such as Google Chrome, Microsoft Edge, and Mozilla Firefox.
- Responsive Design: Optimized using flexible CSS layouts to adjust across screen sizes.
- Mobile Accessibility: The system can be accessed from mobile browsers without layout breakage.

1.2.4. Backend and Storage

- Flask Framework (Python 3.10): Core backend handling REST API endpoints, routing, authentication, and AI module integration.
- MySQL Database: Stores all structured data including weather forecasts, inventory, user credentials, supplier data, and scheduling details.
- Connection Layer: Flask connects to MySQL using mysql.connector or SQLAlchemy (depending on configuration).
- AI Integration Flow:
    - Weather data → AI forecasting script → Predicted demand output → Stored in MySQL → Displayed on dashboard.
    - Hosting: PythonAnywhere hosts both the Flask app and database, providing stable uptime and cloud accessibility.

1.2.5. Data Processing and File Formats

- Weather Data Input: Data may originate from API responses
- Format Types:
    - JSON: Used for web API communication and frontend-backend data exchange.

- CSV: Used for manual weather data import or export.
    - AI Model: Reads structured temperature, humidity, and product trend data, and outputs predictions as JSON or table results displayed in the dashboard.

2. Installation Guide

2.1. System Requirements

- Before installation, ensure that the following minimum requirements are met:

2.1.1. Hardware Requirements

- Processor: Dual-Core CPU (2.0 GHz or higher)
- Memory: Minimum 4 GB RAM (8 GB recommended for production)
- Storage: At least 2 GB of free disk space

- Network: Stable internet connection for API and data updates

2.1.2. Software Requirements

- Operating System: Windows 10 / 11, Ubuntu 20.04+, or compatible Linux distribution
- Python: Version 3.10 or higher
- MySQL Server: Version 8.0 or higher
- pip (Python Package Installer): Latest version recommended
- Web Browser: Google Chrome, Mozilla Firefox, or Microsoft Edge (latest version)

2.2. Installation Steps (Local Environment)

- Prerequisites:
  - Python: Version 3.10 or newer recommended
  - MySQL: Version 8.0+ recommended
  - Git: Version 2.20+ recommended
  - Browser: Latest versions of Chrome, Firefox, or Edge
  - Visual Studio Code: Recommended IDE for development
  - Hardware: At least 4-8 GB RAM, 10 GB available storage (see System Requirements)

- Steps:
    - *Clone the repository:*
        i. Open the terminal in Visual Studio Code
        ii. Navigate to the directory where you want to clone the project
        iii. Run the following commands:
        git clone https://github.com/aynaynayn/WeatherWise.git
        cd ITS120L-WEB/
    - *Create the virtual environment:*
        i. Run this command next to create the virtual environment for python:
        python -m venv venv
        ii. Then this to activate it:
        .\venv\Scripts\Activate.ps1
    - *Install Dependencies:*
        i. Ensure you are in the ITS120L-WEB project directory
        ii. Install required Python packages with the command: pip install -r requirements.txt
    - *Start Development Server:*
        i. Run the flask app with:
        flask run
        OR
        python app.py
        ii. The app will start at: http://127.0.0.1:5000

3. Configuration Guide

3.1. Configuration Parameters

3.1.1. Application Build

- Folder Structure:
  - app.py: Main Flask application entry point
  - templates/: HTML templates for the frontend UI
  - static/: Static assets like CSS, JavaScript, and images

3.1.2. Environment Variables

- Uses dotenv and os in order to securely load environment variables from a .env file (not shown in results)

3.1.3. Runtime Parameters

- Optional flags or command-line arguments can be added when starting the Flask app:
  - --debug: Enables debug logging
  - --port=<port_number>: Overrides the default Flask port (5000)
- Example:
  - python app.py --debug --port=8000

3.2. Environment Setup

3.2.1. Python and Dependencies

- Virtual Environment:
  - Create isolated Python environment to prevent dependency conflicts:
    - python -m venv venv
    - venv\Scripts\activate
- Install Libraries:
  - pip install -r requirements.txt

3.2.2. Running the Application (Development)

- Start the flask server with either of the two:
  - flask run
  - python app.py
- Open browser to the given localhost address (usually http://127.0.0.1:5000).

4. Usage Guide

4.1. User Interface Overview

- WeatherWise features a responsive web-based interface designed with HTML, CSS, JavaScript, and Flask (Python). The layout provides intuitive navigation between features for Inventory, Forecasting, Suppliers, Orders, and Reports.

    - Login/Register Page: Allows users to access the website using email credentials or Google Sign-In authentication.
    - Dashboard: Displays system summaries such as total products, sales trends, and AI weather forecasts.
    - Forecast View: Shows weather-based insights on product demand and inventory levels.
    - Inventory and Scheduling Panels: Manages stock quantities, reorder levels, and supplier schedules.
    - Reports Section: Generates performance metrics and sales summaries with real-time data.

4.2. Core Functionality

4.2.1. Core Functions (Business Logic)

Where to Find: app.py, weather_api.py

app.py

login_user(username, password)

- Purpose: Authenticate users against stored credentials in the MySQL database.
- Parameters:
    - username (string) — inputted username from login form.
    - password (string) — inputted password.
- Returns: User object on success; redirects to login page on failure.
- Side Effects: Creates a Flask session, sets session['user_id'], and redirects to dashboard.

signup_user(form_data)

- Purpose: Register a new user account.
- Parameters:
    - form_data (dict) — includes username, email, password, and role.
- Returns: Redirects to login page on success.
- Side Effects: Inserts new record into user table in MySQL.

get_dashboard_data()

- Purpose: Retrieve summarized weather, inventory, and forecast data for dashboard display.
- Returns: Dictionary with aggregated values for key metrics.
- Side Effects: Executes multiple database SELECT queries.

weather_api.py

get_weather_data(location)

- Purpose: Fetch real-time weather data (temperature, humidity, rainfall) using external weather APIs.
- Parameters:
    - location (string) — name of city or region.
- Returns: JSON object containing current and forecasted weather metrics.
- Side Effects: Performs external API request; may raise network-related exceptions.

4.2.2. Interface Component Summary

Where to Find: templates/, static/js/, static/css/

Login and Signup Pages

Files: templates/login.html, templates/signup.html

Purpose: Handle user authentication and account creation.

Navigation Flow:

Signup → Login → Dashboard

Dashboard

Files: templates/dashboard.html, static/js/dashboard.js

Purpose: Display key insights including forecast summary, weather overview, and alerts.

Navigation Flow:

Dashboard → Inventory / Reports / Suppliers / Orders

Inventory

Files: templates/inventory.html, static/js/inventory.js

Purpose: Display product list, categories, and stock levels; allows updates and deletions.

Navigation Flow:

Dashboard → Inventory → Edit / Update Stock → Save

Forecasting & Reports

Files: templates/forecasting_report.html, static/js/report.js

Purpose: Visualize AI demand forecasts, trends, and weather impact reports.

Navigation Flow:

Dashboard → Forecasting & Reports → Export / Download

Suppliers

Files: templates/suppliers.html, static/js/supplier.js

Purpose: Manage supplier records and contact details; supports add, edit, and delete.

Navigation Flow:

Dashboard → Suppliers → Add Supplier Modal → Save

Orders

Files: templates/orders.html, static/js/orders.js

Purpose: Manage scheduled product deliveries; shows order history and forecast integration.

Navigation Flow:

Dashboard → Orders → Create Schedule → View History

Manage Store

Files: templates/mngstore.html

Purpose: Customize store information, product configurations, and operational details.

Navigation Flow:

Dashboard → Manage Store → Save Changes

4.3. Troubleshooting (Localhost)

Issue	Symptoms	Possible Cause(s)	Solution(s)
1. App Fails to Start	Flask app does not run; error like ModuleNotFoundError or ImportError	Missing dependencies or virtual environment not activated	1. Run pip install -r requirements.txt.2. Activate virtual environment: • venv\Scripts\activate (Windows) • source venv/bin/activate (Linux/macOS)3. Ensure you are in the WeatherWise project directory.
2. Database Connection Error	Error: *"Cannot connect to MySQL server on 'localhost'"* or *"Access denied for user"*	MySQL not running, wrong credentials, or database not created	1. Start MySQL service: • net start mysql (Windows) • sudo service mysql start (Linux/macOS)2. Verify credentials in app.py.3. Run SQL schema to recreate required tables.
3. Template or Static Files Not Loading	CSS or JS not applied when viewing pages	Incorrect file paths or missing Flask config	1. Ensure Flask setup: app = Flask(__name__, static_folder='static', template_folder='templates').2. Use {{ url_for('static', filename='css/style.css') }}.3. Restart Flask after file changes.
4. Changes Not Appearing After Save	Updates in code not visible in browser	Flask debug mode off or cached content	1. Run app in debug mode: app.run(debug=True).2. Use **Ctrl + Shift + R** for hard refresh.
5. Forecast or AI Features Not Working	Forecast not generated; AI error messages	Invalid API key, missing model, or dependency issue	1. Verify API key and model path.2. Reinstall packages: pip

			install numpy pandas.3. Check console logs for detailed errors.
6. “Port Already in Use” Error	Error: *“Address already in use”*	Flask or another app still using port 5000	1. Kill existing process: taskkill /f /im python.exe (Windows) sudo kill -9 $(lsof -t -i:5000) (Linux/macOS)2. Restart Flask.
7. 404 or 500 Errors	404: Page not found500: Server error	Missing route, incorrect URL, or SQL error	1. Check @app.route() names.2. View Flask terminal for traceback.3. Verify table/column names in queries.
8. Browser Doesn’t Update After Database Change	Updated MySQL data not showing in app	Cached data or missing page refresh	1. Manually refresh browser.2. Ensure Flask route re-fetches latest data.3. If using JS fetch, confirm API endpoint updates correctly.

5. API Documentation

5.1. Endpoints

- The WeatherWise API offers RESTful endpoints for core system functions. The common endpoints include:

- /forecast – retrieve weather forecasts based on location and date
- /inventory – manage inventory items, including stock updates and queries
- /suppliers – manage supplier information
- /orders – create, update, and track orders
- /users – handle user administration and roles

Each endpoint supports conventional HTTP methods (GET, POST, PUT, DELETE) for performing CRUD operations.

5.2. Request and Response Formats

- JSON is used for all requests and responses.
- Standardized responses include success indicators, messages, and relevant data payloads.

6. Database Schema

6.1. Entity-Relationship Diagram

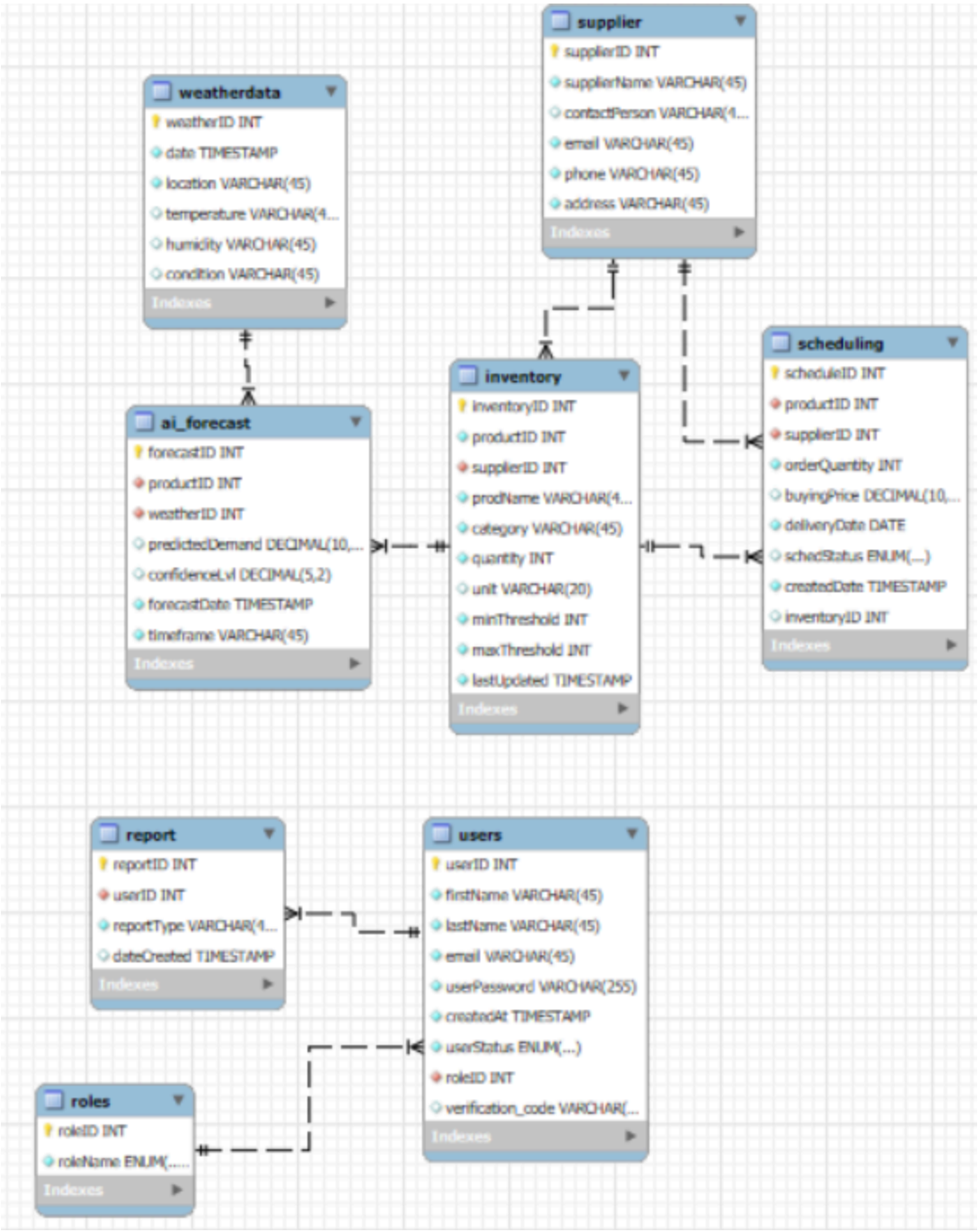


Figure 6.1 ERD

6.2. Table Definitions

- Users: table used to store user accounts in the system. Contains attributes such as username, password, and role. Links to the Role table for access control.

- Roles: table used to store the different user roles (e.g., Admin, Supplier, User). Provides role references for the Users table.

- Supplier: table used to store supplier details such as name, email, and contact number. Linked to inventory for inventory sourcing.

- Inventory: table used to store product stock levels. Tracks quantity and last update time of each product. Has a product key for each product in inventory.

- Scheduling: table used to store staff schedules. Includes user assignment, shift date, start and end times. Linked to the User table.

- WeatherData: table used to store weather-related information. Attributes include date, temperature, humidity, and rainfall. Used for forecasting purposes and report generation.

- AI_Forecast: table used to store predicted demand for products. Includes forecast date, predicted demand, and confidence level. Linked to the Inventory table.

- Report: table used to store generated reports from the system. Includes type, generation timestamp, and report content.

7. Testing

7.1. Test Plan

Module Name	Applicable Roles	Description
Login & Authentication	All users.	Ensures that users can securely enter and exit the application using valid credentials or Google Sign-In.
User Management	Admin.	Allows the administrator to create, modify, and delete user accounts with specific roles.
AI Forecast Integration	Admin, Manager.	Validates that weather data is properly accurate and used to generate product demand predictions.
Inventory Management	Admin, Manager.	Tests CRUD operations and real-time updates.
Supplier Tracking	Admin, Manager.	Verifies supplier list management, contact details, and import/export data.
Order Monitoring	Admin, Manager.	Confirms that orders are correct in notifications (logged, displayed, and summarized).

Report Generation	Admin.	Checks if the information is accurate when the PDF or CSV is downloaded.
Alert & Notification System	All users.	Ensures that low-stock items, weather alerts, and update notifications display correctly.
Backup and Restore	Admin.	Tests the ability to back up and restore key data tables.
Multi-User Session	All users.	Confirms that multiple users can access the system without data error.

Table 7.1 Test Plan

7.2. Test Cases

7.2.1. Login

Test Case Description	Procedure & Test Data	Expected Output	Result
Users enter a valid email and password.	1. Open the WeatherWise application.	The user is redirected to the dashboard.	Pass
Users enter invalid credentials.	1. Enter the wrong password. 2. Click “Login”.	Cannot enter the website.	Pass
Forgot password	1. Click “Forgot	The email link	Pass

feature.	Password?”. 2. Enter email. 3. Reset link sent.	was received, and the new password works.	

Figure 7.2 Test Case (Login)

7.2.2. Signup

Test Case Description	Procedure & Test Data	Expected Output	Result
Create new account	1. Click “Sign Up”. 2. Fill the information fields. 3. Click “Register”.	The user is redirected to the dashboard.	Pass
Duplicate email	1. Use existing email. 2. Attempt registration.	The system blocks duplicate accounts.	Pass

Figure 7.3 Test Case (Signup)

7.2.3. Inventory Management

ID	Test Case Description	Procedure & Test Data	Expected Output	Result
TC1	Add new product	1. Go to inventory. 2. Click	Product added.	Pass

		“Add Item”. 3. Enter item details.		
TC2	Edit existing product	1. Select a product. 2. Change details, 3. Save.	Details updated.	Pass
TC3	Delete product	1. Select product. 2. Click “Delete”.	Product gone.	Pass
TC4	Low-stock notification	1. Reduce stock to less than minimum threshold.	Alert displays.	Pass

Figure 7.4 Test Case (Inventory Management)

7.2.4. AI Forecast Integration

ID	Test Case Description	Procedure & Test Data	Expected Output	Result
TC1	Retrieve weather data	1. Access Forecast feature. 2. Trigger	Weather forecast data displayed.	Pass

		weathe r fetch.		
TC2	Generate stock recommendation	1. Click “Gener ate Foreca st”.	Surplus and deficit are describing the items.	Pass

Figure 7.5 Test Case (AI Forecast Integration)

7.2.5. Reports and Analytics

ID	Test Case Description	Procedure & Test Data	Expected Output	Result
TC1	Generate daily report	1. Choose a date. 2. Genera te summa ry.	Daily reports are downloaded and corrected.	Pass
TC2	Generate monthly report.	1. Choose a month. 2. Genera te summa ry.	Monthly reports are downloaded and correct.	Pass

Figure 7.6 Test Case (Reports and Analytics)

7.2.6. Multi-User and Session Management

ID	Test Case	Procedure	Expected	Result

	Description	& Test Data	Output	
TC1	Two users log in simultaneously	1. Open two browsers. 2. Log in with different accounts.	Both sessions run.	Pass
TC2	Concurrent data updates	1. Two users edit the same inventory.	No data conflict.	Pass

Figure 7.7 Test Case (Multi-User and Session Management)

7.2.7. Backup and Restore

ID	Test Case Description	Procedure & Test Data	Expected Output	Result
TC1	Create backup file	1. Click “Backup” from settings.	Backup file generated.	Pass
TC2	Restore backup	1. Upload backup. 2. Click “Restore”.	System restored successfully.	Pass

Figure 7.8 Test Case (Backup and Restore)

7.3. Test Results

No.	Module	Pass	Fail	Pending
1	Login	3	0	0
2	Signup	2	0	0
3	Inventory	4	0	0
4	AI Forecast	2	0	0
5	Reports	2	0	0
6	Multi-User Session	2	0	0
7	Backup and Restore	2	0	0
Total:	**-**	**17**	**0**	**0**
Test Coverage:	**100%**	**Test Success Rate:**	**100%**	

Figure 7.9 Test Results

8. Deployment

8.1. Overview

- This section outlines the deployment procedures for the WeatherWise web application through the PythonAnywhere hosting service.

8.2. Prerequisites

- PythonAnywhere account
- Local Flask app ready (app.py, requirements.txt, templates, static files)
- MySQL database structure (tables and users)
- Virtual environment (optional but recommended)

8.3. Web Application Deployment

- Uploading the app files:
  i. Log in to PythonAnywhere
  ii. Go to the Files tab
  iii. Upload the flask app files (app.py, templates/, static/, any modules like weather_api.py).
  iv. Upload requirements.txt for easy virtualenv setup.

- Virtual Environment Setup:
  i. Open the bash console on PythonAnywhere
  ii. Create a virtualenv by entering this in the console:
  python3.10 -m venv ~/WeatherWise/venv

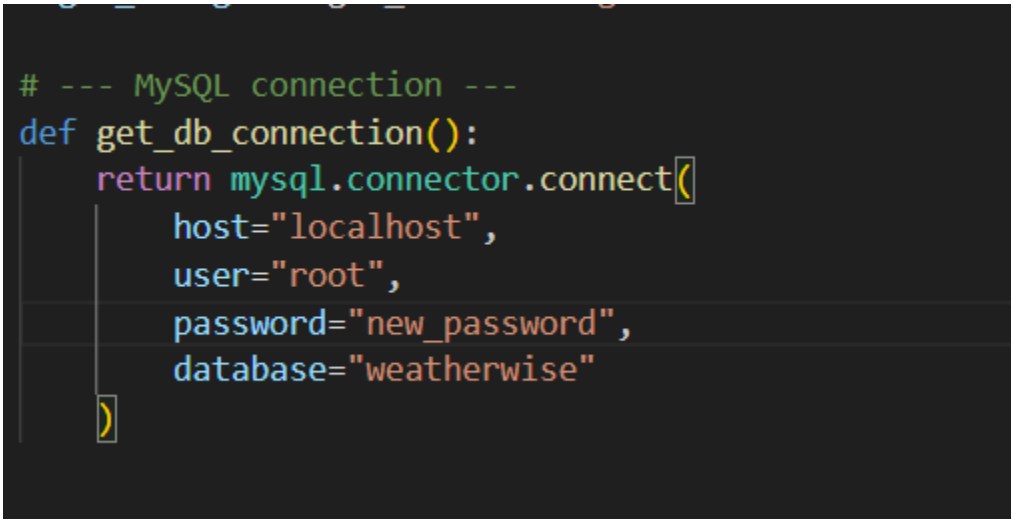
iii. Activate the virtual environment:
source ~/WeatherWise/venv/bin/activate
iv. Install required packages:
pip install –upgrade pip
pip install -r ~/WeatherWise/requirements.txt

v. No errors should surface when testing locally in the bash console with these commands:
cd ~/WeatherWise
python app.py

- MySQL Database Setup:
  - Go to Databases tab in PythonAnywhere
  - Create a new MySQL database:
    - Name: yourusername$weatherwise
    - User: automatically created with same prefix
    - Password: yourownpassword
  - Connect to MySQL via console:
    mysql -u yourusername -p
    Then enter password when prompted
  - Copy the SQL Create script and paste into the console. It will automatically run and create the tables along with their foreign key relationships and initial insert data.

- Configure app.py backend:

- Update the database connection from the local credentials:



```
# --- MySQL connection ---
def get_db_connection():
    return mysql.connector.connect(
        host="localhost",
        user="root",
        password="new_password",
        database="weatherwise"
    )
```

- To the credentials PythonAnywhere uses:

```
def get_db_connection():
    return mysql.connector.connect(
        host="yourusername.mysql.pythonanywhere-services.com
        user="yourusername",
        password="your_mysql_password",
        database="yourusername$weatherwise"
    )
```

- Make sure your static and template paths are correct (templates/ and static/ folders).

- Web App setup in PythonAnywhere:
  - Go to the **Web** tab → Add a new web app
  - Choose Manual Configuration → Python 3.10
  - Enter the path to your Flask app:
    - WSGI file: /var/www/yourusername_pythonanywhere_com_wsgi.py

- In the WSGI file, change it from:

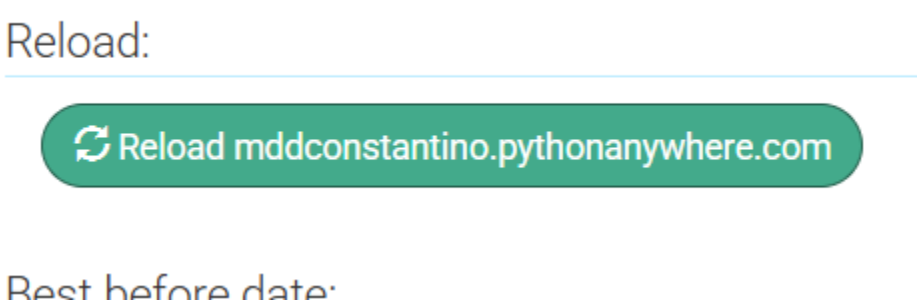
```
# This file contains the WSGI configuration required to serve up
# web application at http://yourusername.pythonanywhere.com/

import os
import sys

# The path to your project directory
path = '/home/yourusername/mysite'
if path not in sys.path:
    sys.path.append(path)

# The environment variable for Flask
os.environ['FLASK_APP'] = 'flask_app.py'

from flask_app import app as application
```

- To:

```
import sys
import os

# Add project folder to the path
project_home = '/home/mddconstantino/WeatherWise'
if project_home not in sys.path:
    sys.path = [project_home] + sys.path

# Set the working directory
os.chdir(project_home)

# Import the Flask app
from app import app as application
```

- Reload the Web app for it to go live:

- Go to the **Web** tab then click the green button that says reload



- The web app would now be accessible from the given domain.

8.4. Security Considerations

- Input Sanitization and Secure Configuration:
  - Sanitize all user inputs before processing or storing them in the database. Use prepared statements or ORM methods: cursor.execute("INSERT INTO suppliers (name, email) VALUES (%s, %s)", (name, email))
  - Validate uploaded files (e.g., CSVs or images) for correct type and size to prevent malicious uploads.
  - Escape dynamic HTML content in Jinja2 templates using {{ variable | e }} to avoid XSS.
  - Enforce HTTPS via PythonAnywhere's web tab to encrypt all traffic.
  - Regularly update dependencies (pip install --upgrade -r requirements.txt) to patch known vulnerabilities.

- Database Credential Protection:

- To protect sensitive database credentials, environment variables must be used rather than hardcoding credentials into app.py.
- Restrict file permissions on sensitive configuration files (e.g., .env, config.py) to avoid exposure.
- Disable public database access and ensure the MySQL instance only accepts connections from your web app.

8.5. Maintenance

- Daily log monitoring by checking PythonAnywhere's error and server logs for runtime issues or failed requests.
- Monthly retraining of the AI forecast model with new weather and sales data to maintain accuracy.
- Weekly backing up of the database by regularly performing SQL dumps and securing them in a secure backup folder or cloud storage.
  Example:
  mysqldump -u weather_admin -p weatherwise_db > backups/weatherwise_$(date +%F).sql

9. Support and Maintenance

9.1. Troubleshooting Guide (Deployed)

Issue	Possible Cause	Resolution
App not loading	Incorrect WSGI path or syntax error in app.py	Check /var/log/uwsgi.log and verify WSGI configuration. Reload app in Web tab.
Database connection error	Wrong credentials or missing environment variable	Recheck .env or MySQL settings on PythonAnywhere.
CSS or JS not loading	Static file path misconfigured	Verify static file mapping in the Web tab: /static/ → /home/mddconstantino/WeatherWise/static/
Forecast not updating	Old AI model or missing API key	Retrain model and reconfigure weather_api.py key.
“Broken pipe” in logs	User closed connection mid-response	Safe to ignore; not an actual server error.

Table 9.1 Troubleshooting (Deployed)

9.2. Frequently Asked Questions (FAQs)

- Why are my forecasts inaccurate?
  - Ensure the AI model is retrained with recent data and that weather API responses are

up-to-date.

- The dashboard data doesn't refresh. What should I do?
    - Clear browser cache or reload the app via the PythonAnywhere Web tab to apply the latest changes.

- How do I restore my database after a crash?
    - Upload the latest .sql backup and run: mysql -u weather_admin -p weatherwise_db < backups/weatherwise_latest.sql
- Why can't I access the app after code changes?
    - Check for syntax errors in app.py and confirm that the virtual environment (venv) is correctly configured in the Web tab.
- How can I monitor usage and performance?
    - Use PythonAnywhere's CPU and memory graphs, and inspect the server logs for long-running queries.

9.3. Contact Information

- **Scrum Master, Ricci Cordero**:
    - radcordero@mymail.mapua.edu.ph (09088654210)
- **Front-end Developer, Stephanie Zamoras**:
    - smzamoras@mymail.mapua.edu.ph (09626712840)
- **Back-end Developer, Marc Constantino:**
    - mddconstantino@mymail.mapua.edu.ph (09628493593)

- **QA Tester, Dean Flores:**
  - dmcflores@mymail.mapua.edu.ph (09062654515)

10. Change Log

10.1. Version History (System)

Effective Date	Version	Location	Change Description	Reference
October 14, 2025	1.0	Mapúa Makati	Frontend Creation	HTML & CSS(Design), JavaScript (Logic)
October 14, 2025	1.1	Mapúa Makati	Backend Implementation	MySQL Workbench (Backend); Report Generator
October 27, 2025	1.2	Mapúa Makati	Peer Review	Dashboard, Inventory, Forecasting & Reports, Suppliers, Orders, Manage

				Store
October 29, 2025	1.3	Mapúa Makati	Functionalities Testing	Functional requirements.
October 30, 2025	1.4	Mapúa Makati	Deployment	Accessible link.

Table 10.1 Version History (System)

10.2. Version History (Technical Document)

Effective Date	Version	Location	Change Description
November 6, 2025	1.0	Mapúa Makati	Creation of the full technical document.

Table 10.2 Version History (Technical Document)

11. Glossary

11.1. Terms and Definitions

Term	Definition
AI Forecasting	The process of using artificial

	intelligence or machine learning algorithms to predict future events, such as weather conditions or product demand, based on historical data.
API (Application Programming Interface)	A set of rules that allows software systems to communicate. In WeatherWise, it is used to fetch weather data from external services.
Backup	A copy of data stored separately to prevent loss in case of system failure. WeatherWise uses MySQL dumps for database backups.
Database	A structured collection of data managed by a database management system (DBMS). WeatherWise uses MySQL to store users, suppliers, products, schedules, and forecasts.
Deployment	The process of making an application available for public or production use — in this case, hosting the Flask app on PythonAnywhere.
Environment Variables	Secure key-value pairs that store configuration data (e.g., database credentials, API keys) without exposing them in the source code.

Flask	A lightweight Python web framework used to build and serve the WeatherWise application.
Frontend	The part of the application the user interacts with — implemented using HTML, CSS, and JavaScript.
Forecast Model	A trained AI or statistical model used to generate predictions about future weather or demand patterns.
HTTPS (Hypertext Transfer Protocol Secure)	A secure version of HTTP that encrypts data transferred between the client and the server.
Jinja2	A template engine used by Flask to dynamically generate HTML pages using variables from Python code.
JSON (JavaScript Object Notation)	A lightweight data format used for exchanging data between the client (browser) and server. Flask uses it for API responses.
Log File	A record of events, errors, and server activity. PythonAnywhere provides access to error and access logs for debugging.
Machine Learning (ML)	A subset of AI that enables computers to learn from data and improve predictions over

	time without explicit programming.
MySQL	A relational database management system used in WeatherWise to store and query application data.
PythonAnywhere	A cloud platform that hosts Python-based applications and provides tools for managing web apps, databases, and virtual environments.
Query	A command used to retrieve or manipulate data in a database, written in SQL (Structured Query Language).
Scheduler	A system feature that manages product orders, delivery dates, and supplier assignments within WeatherWise.
Static Files	Non-dynamic resources such as images, CSS, and JavaScript files served by Flask for frontend rendering.
Template	A pre-defined HTML layout that Flask fills with data before sending it to the user's browser.
uWSGI	A web server gateway interface used by PythonAnywhere to run Flask applications efficiently.
Virtual Environment (venv)	An isolated Python environment

	where dependencies are installed, ensuring no conflicts between projects.
Web Tab	A PythonAnywhere configuration interface where the user sets up WSGI paths, environment variables, and static file directories.
WSGI (Web Server Gateway Interface)	A Python standard that defines how web servers communicate with web applications like Flask.

Table 11.1 Glossary