COMP4920 Senior Design Project II, Spring 2024
Advisor: Assoc. Prof. Dr. Ömer ÇETİN

GLOWG: Personalized Skincare Powered by AI
Product Manual

Revision 1.0
04.05.2025

By:
Ceren Sude Yetim 21070001045
İrem Demir, 20070001029
Gizem Tanış, 20070001047
Ece Topuz, 21070001057

Revision History

Revision	Date	Explanation
1.0	04.05.2025	Initial Product Manual

Table of Contents

1. Introduction

This document presents the Product Manual for **GlowGenie**, an AI-powered skincare recommendation system developed within the scope of the **COMP4910 Senior Design Project**. The primary objective of this manual is to comprehensively document the final implementation of the software, including its architectural structure, development environment, testing strategy, installation procedure, and operational guidelines.

GlowGenie is designed to offer users a personalized skincare experience by utilizing artificial intelligence to identify their skin type, evaluate cosmetic product ingredients, and generate suitable product recommendations. The platform gathers user input through a structured questionnaire, analyzes it with machine learning models, and assesses ingredient compatibility using natural language processing models such as DeepSeek. Based on this analysis, the system delivers customized product suggestions aligned with the user's skin characteristics and possible sensitivities.

This document builds upon the prior **Requirements Specification Document (RSD)** and **Detailed Software Design Document (DSD)**. While the RSD and DSD focus on the conceptual and architectural design of the system, this manual presents a detailed overview of the implemented features and operational logic of the GlowGenie application.

The application enables secure user registration and login, real-time profile updates, AI-based skin analysis, and compatibility checks for skincare product ingredients. It supports users in navigating the product catalog through category-based filtering and enables informed decision-making by identifying which ingredients are suitable or potentially harmful. The system also provides mechanisms for password recovery and user verification via email-based confirmation.

In the sections that follow, the software subsystem, development tools, testing procedures, and installation steps will be discussed in detail. This manual serves as a reference for developers, testers, and stakeholders evaluating or extending the GlowGenie system.

2. GlowGenie Software Subsystem Implementation

2.1. Source Code and Executable Organization

The source code of the GlowGenie system is organized into a modular and layered structure, separating concerns across the backend, machine learning models, and frontend components. The backend is developed using Python with the Flask framework and implements a RESTful API that handles HTTP requests and responses. Core functionalities such as user authentication, profile management, product retrieval, and ingredient evaluation are encapsulated within dedicated service modules (e.g., AuthService, RecommendationService, OpenAIRoutes). Each endpoint is routed through a centralized controller (AppController) that delegates tasks to these underlying services.

The machine learning subsystem is composed of two main model classes: SkinTypePredictor, which classifies users' skin types based on questionnaire inputs using a regression model, and ProductSuitabilityModel, which utilizes a random forest algorithm to determine product compatibility based on binary-encoded ingredient data. These models are trained offline and loaded during runtime for inference.

The frontend is implemented using React.js and communicates with the backend through secure API calls. It presents user-friendly interfaces for skin type testing, product browsing, feedback submission, and personalized recommendations. All forms and inputs are validated both client-side and server-side to ensure data integrity and security.

The codebase is divided into clearly defined folders such as /backend, /ml_models, and /frontend, each containing relevant modules, configuration files, and static assets. Database interactions are abstracted within service layers, ensuring a clean separation between application logic and data persistence. Configuration settings, such as database connection strings, API keys, and environment-specific variables, are stored in .env and config.py files to facilitate portability and deployment flexibility.

All code is maintained in a version-controlled repository on GitHub, with branches used for development, testing, and production integration. While the design largely adheres to the structure proposed in the DSD document, minor adjustments were made during implementation to improve modularity and maintainability. The source code includes properly structured comments and inline documentation, written in accordance with professional software engineering practices.

2.2. Software DevelopmentTools

The implementation of the GlowGenie application utilized a comprehensive set of software tools distributed across macOS and Windows environments. This distributed setup was adopted to align each development task with the most compatible and efficient platform. Tools and frameworks were carefully selected based on their suitability for modular full-stack development, AI integration, and real-time user interaction.

The backend and database subsystems were developed on **macOS Ventura**, using **Python 3.10.6** and the **Flask 2.2.2** microframework, both installed via the official python.org and pip. The Flask backend served as a RESTful API layer, facilitating communication between the user interface, the PostgreSQL database, and integrated machine learning modules. A virtual environment was configured using Python's built-in venv module to ensure dependency isolation and reproducibility. Backend code was written and managed using **Visual Studio Code 1.87.2** for macOS, with relevant extensions for Python and database interaction.

The database layer was implemented using **PostgreSQL 15**, also on macOS, and managed with **pgAdmin 4**, which enabled schema design, data inspection, and administrative control over stored procedures and backup operations. The backend and database were hosted locally on the same macOS machine to ensure low-latency and secure data exchange during development.

The machine learning components were developed in a **Windows-based environment**, using **Jupyter Notebook 6.5.4** for model prototyping. **scikit-learn 1.2.2** was used to implement the SkinTypePredictor and ProductSuitabilityModel classes, and **SHAP 0.41.0** was incorporated to provide interpretability into feature influence on model outcomes. Trained models were exported via joblib and transferred to the backend system on macOS for deployment.

Frontend development was carried out in a separate **Windows environment** using **React 18.2.0**, initialized with create-react-app. HTTP communication with the backend API was handled using **Axios 1.3.4**, and interface styling was achieved through **Bootstrap 5.2.3**. The frontend environment was managed using **Node.js 18.15.0** and **npm 9.5.0**, both installed from nodejs.org. Visual Studio Code (Windows edition) served as the primary IDE for interface development, and browser-based testing was conducted locally over specified ports.

API endpoints were tested and monitored using **Swagger 0.9.7.1**, while version control was maintained throughout the project using **Git 2.42.0**. All source code was hosted on **GitHub** within a private repository, and branching strategies were adopted to organize feature development, testing, and stable release workflows.

Although no custom-built third-party tools were developed, all core service modules — such as UserService, RecommendationService, and OpenAIIngredientAnalyzer — were implemented from scratch by the team, following principles of modularity, reusability, and separation of concerns.

This platform-aware toolchain enabled efficient collaboration across systems while maintaining a cohesive and testable architecture throughout development and integration phases.

2.3. Hardware and System Software Platform

The development and testing of the GlowGenie system were carried out on both macOS and Windows platforms, with different subsystems implemented on each based on performance needs and developer environment compatibility. This hybrid approach allowed for optimized development across all components of the system, including backend logic, database operations, machine learning models, and frontend interfaces. The backend and database subsystems were developed on macOS Ventura 13.x, running on a MacBook Air with an Apple M1 chip, 8 GB of unified memory, and 256 GB SSD storage. The backend was implemented using Python 3.10.6 and Flask 2.2.2, and was hosted locally for testing purposes. PostgreSQL 15 was installed natively on macOS and managed via pgAdmin 4, which provided a graphical interface for schema design, data queries, and administrative operations. The macOS Terminal, using the zsh shell, was used for environment activation, package installation via pip, and execution of backend services. This configuration ensured smooth integration between the backend logic and the local database service. The frontend and machine learning components were developed and tested on a Windows 11 environment running on a laptop equipped with an 11th Gen Intel Core i7-1165G7 processor (2.80 GHz), 16 GB RAM, and 1 TB SSD storage, using a 64-bit architecture. React 18.2.0 and supporting libraries such as Axios and Bootstrap were used to build and test the user interface. The machine learning models — including SkinTypePredictor and ProductSuitabilityModel — were developed in PyCharm using scikit-learn 1.2.2, and model interpretability was supported by SHAP 0.41.0. The trained models were exported using joblib and transferred to the backend on macOS for integration and runtime prediction. All code development was performed using Visual Studio Code 1.87.2 on both platforms, with relevant extensions installed for Python, JavaScript/React, and SQL. API development and endpoint validation were performed using Postman 10.19.5, ensuring accurate testing across environments. The project's hardware and software setup enabled distributed development while maintaining consistent behavior across platforms. The system remains platform-independent at the architectural level, and can be deployed on cloud-based services or containerized environments such as Docker in future versions.

3. GlowGenie Software Subsystem Testing

Throughout the development of the GlowGenie application, a systematic and layered testing approach was followed to validate both individual modules and the complete system workflow. Testing activities were carried out across backend services, machine learning models, frontend components, and database interactions to ensure that all functionalities operated as expected.

The core backend endpoints—such as user registration, login, logout, profile updates, product listing, and personalized recommendation generation—were tested using **Swagger**, where each API response was verified for both valid and invalid input scenarios. These endpoints were connected to

a PostgreSQL database running locally, and backend–database integration was validated through successful data insertions, updates, and queries. For instance, the profile update endpoint was tested by submitting user inputs from the frontend interface and confirming changes via database queries in **pgAdmin 4**.

The machine learning models (SkinTypePredictor and ProductSuitabilityModel) were evaluated in an offline environment using **Py Charm**, where prediction accuracy was measured using test datasets. The trained models were later integrated into the Flask backend and accessed through API endpoints, which were also tested via **Swagger** to confirm their runtime behavior. Predictions were confirmed to match expected outcomes, and SHAP values were analyzed to verify model interpretability.

The user interface developed in React was manually tested in a browser environment. Frontend forms for login, profile updates, and skin test submission were evaluated for layout integrity, input handling, and communication with the backend. Browser developer tools were used to observe network requests and verify that the frontend correctly displayed backend responses and handled errors gracefully.

Some functionalities, such as the **password reset and email confirmation system**, were partially implemented but not fully tested due to time constraints. These modules currently have placeholder logic in place and are intended for further testing in future versions. Additionally, features related to administrative controls or internal data management were not implemented in this version of the project and are scheduled for later phases.

During testing, logs and error traces were monitored directly from the terminal and development environment to capture any anomalies. Overall, the majority of critical features—especially those visible to the end-user—were tested and confirmed to work correctly under normal usage conditions.

4. GlowGenie Installation, Configuration and Operation

This section outlines the complete installation procedure, configuration steps, and operational flow of the GlowGenie system, which consists of a Python-based backend server, a React-based frontend client, a PostgreSQL relational database, and integrated machine learning components. The system is designed to run on a local development environment but remains flexible for future deployment to remote or containerized platforms.

4.1. Backend Installation

The backend was developed using **Python 3.10.6** and the **Flask** framework. To install and run the backend:

1.Clone the Project Repository

```
git clone https://github.com/your-org/glowgenie.git
cd glowgenie/backend
```

2.Set up Virtual Environment

Create and activate a virtual environment to isolate dependencies:

```
python3 -m venv venv
source venv/bin/activate  # macOS/Linux
.\venv\Scripts\activate    # Windows
```

3.Install Backend Dependencies

```
pip install -r requirements.txt
```

4.Prepare the Environment Variables

Create a .env file in the root of the backend/ folder with the following content:

```
FLASK_APP=app.py
FLASK_ENV=development
SECRET_KEY=your-secret-key
DATABASE_URL=postgresql://username:password@localhost:5432/glowgenie
```

5.Start the Backend Server

After setting up the environment and database (see 5.3), run:

```
flask run
```

The backend will start at http://127.0.0.1:5000.

4.2. Frontend Installation

The frontend was developed using **React 18.2.0**, **Node.js 18.15.0**, and **npm 9.5.0**.

1. **Navigate to the Frontend Directory**

   ```
   cd ../frontend
   ```

2. **Install Node.js Dependencies**

   ```
   npm install
   ```

3. **Start the React Development Server**

   ```
   npm start
   ```

The UI will be accessible at http://localhost:3000. It communicates with the backend through RESTful API calls defined in Axios-based service modules.

4. **Update Axios Base URLs if Needed**

In frontend/src/config/api.js or equivalent, ensure the backend URL matches:

```
const BASE_URL = "http://localhost:5000";
```

4.3. Database Setup and Configuration

The application uses **PostgreSQL 15** to manage user data, skin test results, product information, and recommendation records.

1. **Install PostgreSQL (via postgresql.org)**
2. **Create a Database**
   Open **pgAdmin 4**, create a new database named `glowgenie`.
3. **Run Schema Scripts**

   Use SQL tab in pgAdmin to run provided schema creation script:

```
-- Example: users table

CREATE TABLE users (
    user_id SERIAL PRIMARY KEY,
    email VARCHAR(255) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    skin_type_id INT,
    skin_tone_id INT
);
```

4. **Connect to the Database**

   Ensure that the connection string in .env matches:

   postgresql://postgres:yourpassword@localhost:5432/glowgenie

4.4. Operation Flow

Once the backend, frontend, and database are all running correctly:

- The user launches the UI at localhost:3000
- They register a new account and log in
- The system directs them to a skin test form
- Their responses are analyzed by the integrated ML model (SkinTypePredictor)
- Their skin type is stored, and product recommendations are fetched via /recommend endpoint
- Users can view ingredient compatibility for any product and filter by category
- Their actions, preferences, and feedback are stored in the database
- Incompatible products (due to allergens or skin mismatch) are flagged in real time

Throughout usage, form validation, input sanitization, and error handling ensure a smooth and secure interaction. If users enter invalid data, they receive contextual feedback both from frontend and backend validators.

4.5. Developer and Maintenance Notes

- Ensure .env and database credentials are correct before running flask run
- Check PostgreSQL service is active and listening on default port (5432)
- Use browser dev tools (Console & Network tab) to debug frontend–backend communication
- Backend logs (terminal) will display request traces and error stacks
- Database records can be inspected and modified via pgAdmin 4

The system is currently optimized for local development but remains structurally compatible with future cloud deployment (e.g., Render, Heroku) or Docker-based containerization.

References

1. Python Software Foundation. (2023). *Python 3.10.6 Documentation*. Retrieved from https://docs.python.org/3.10/
2. Flask Documentation. (2023). *Flask 2.2.x*. Retrieved from https://flask.palletsprojects.com/
3. PostgreSQL Global Development Group. (2023). *PostgreSQL 15 Documentation*. Retrieved from https://www.postgresql.org/docs/15/
4. pgAdmin Team. (2023). *pgAdmin 4 Documentation*. Retrieved from https://www.pgadmin.org/docs/pgadmin4/latest/
5. React Contributors. (2023). *React 18 Documentation*. Retrieved from https://reactjs.org/docs/getting-started.html
6. Node.js Foundation. (2023). *Node.js 18 Documentation*. Retrieved from https://nodejs.org/en/docs/
7. scikit-learn Developers. (2023). *scikit-learn 1.2.2 Documentation*. Retrieved from https://scikit-learn.org/stable/
8. Lundberg, S. M., & Lee, S. I. (2017). *A Unified Approach to Interpreting Model Predictions*. In Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17). SHAP Library Documentation: https://shap.readthedocs.io/
9. OpenAI. (2023). *DeepSeek API Documentation*. Retrieved from https://api-docs.deepseek.com/
10. Git Documentation. (2023). *Git 2.42 User Manual*. Retrieved from https://git-scm.com/docs
11. Visual Studio Code. (2023). *VS Code Docs*. Retrieved from https://code.visualstudio.com/docs
12. Swagger API Platform. *Swagger API Testing*. Retrieved from https://swagger.io/

13. GlowGenie-RSD-2025-01-12.doc (Requirements Specification Document).

14. GlowGenie-DSD-2025-05-01.doc (Detailed Software Design Document).