COMP4910 Senior Design Project 1, Fall 2024
Advisor: Assoc. Prof. Dr. Ömer ÇETİN

GLOWG: Personalized Skincare Powered by AI
High Level Design
Design Specifications Document

Revision 1.0
20.01.2025

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

Revision History

Revision	Date	Explanation
1.0	20.01.2025	Initial high level design

Table of Contents

1. Introduction

Purpose:

The purpose of this project is to develop the **GlowGenie Application** based on the foundation laid in the **GlowGenie Requirements Specification Document (RSD)**. GlowGenie is designed to provide a user-friendly, AI-powered skincare solution to address challenges in determining skin types, evaluating product suitability, and providing personalized recommendations. This Detailed Software Design (DSD) document outlines the design and implementation details of the GlowGenie application, utilizing **Python for backend, React.js for frontend**, and **PostgreSQL** for database management.

Main Functions of GlowGenie System:

1. **User Account Operations:**
    - Login, registration, and logout functionalities.
    - Ability to update user profiles, including skin type, skin tone, and allergens.
    - Password reset functionality with secure email-based verification.
2. **Skin Type Test:**
    - An AI-based interactive questionnaire to determine the user's skin type (Dry, Oily, Combination, Normal).
    - Results are saved in the user's profile for future use.
3. **Product Suitability Feedback:**
    - Analyze skincare products to determine their compatibility with the user's skin type, tone, and allergens.
    - Provide ingredient-based feedback with visual indicators (green/red framing) for suitability.
4. **Generate Product Recommendations:**
    - Suggest personalized products tailored to the user's preferences and skin profile.
    - Allow filtering by product categories like cleansers, moisturizers, sunscreens, toners, and serums.
5. **Update Product Ingredients:**
    - Periodically update product ingredient data via GPT integration.
    - Re-evaluate product suitability based on updated information.
6. **List All Products:**
    - Display a comprehensive list of products with filtering options by category.
    - Provide detailed information for each product, including ingredients and suitability feedback.

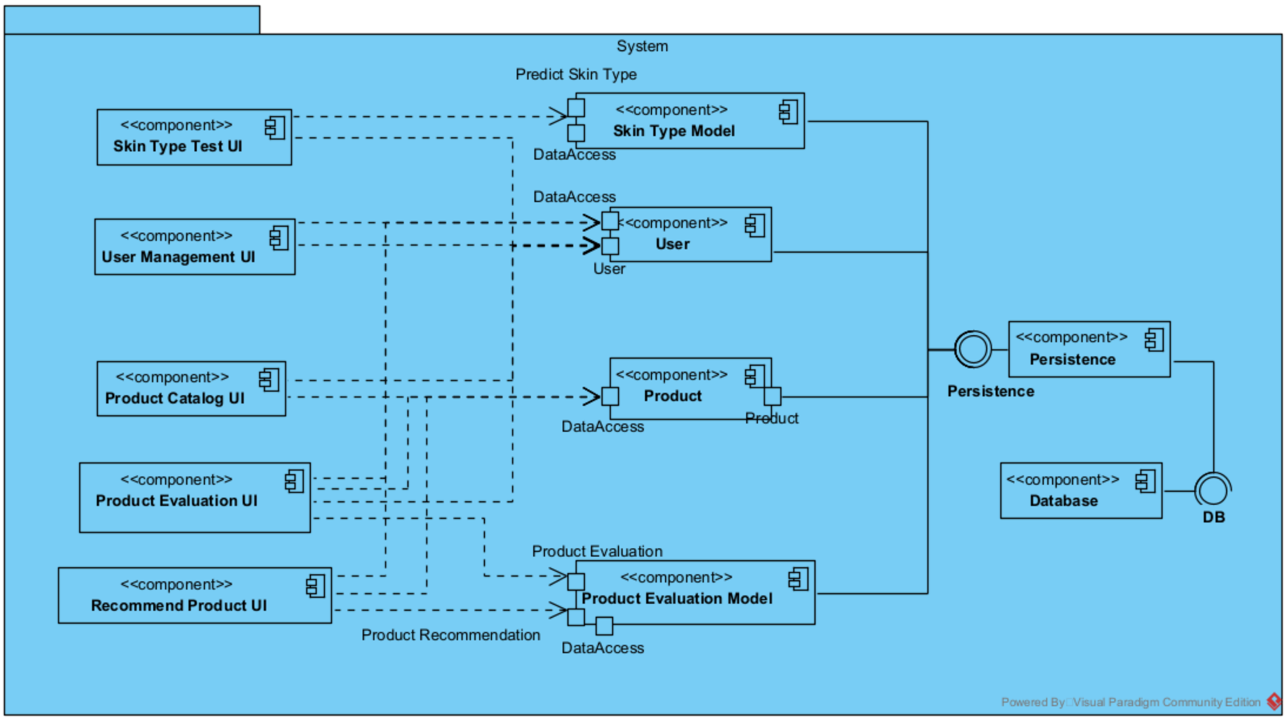
Design Basis and Methodology

The design of GlowGenie is based on the **GlowGenie Requirements Specification Document (RSD), Revision 2.0**, in the file `GlowGenie-RSD-2025-01-12.doc`[1]. The design process used to produce this document conforms to the organizational specifications provided in [2]. This DSD document serves as a continuation of the RSD, detailing the system's architectural choices, structural breakdown, and implementation specifics. The notation used to describe the design of the GlowGenie Application is primarily **UML**, which adheres to the organizational standards outlined in [3].

The design methodology adheres to **ISO/IEC 9001 software quality standards**, ensuring a structured and reliable process for developing the GlowGenie application. This standard emphasizes **usability**, **performance efficiency**, and **security**, aligning with internationally recognized best practices for quality management systems. To ensure clarity and consistency, **Unified Modeling Language (UML)** is extensively used throughout the document, detailing the architecture, component interactions, and class designs. This approach guarantees that the system meets user requirements while maintaining high quality and scalability.

2. GLOWG System Design

The GLOWG system design focuses entirely on the **software subsystem**, as no hardware components are required for the development and implementation of this project. This section provides an overview of the system, identifying its components and their interactions, and includes a UML Component Diagram to visualize the relationships among the components including **Skin Type Test UI**, **User Management UI, Product Catalog UI, Recommend Product UI, User, Product, Product Evaluation Model**, **Skin Type Model, Persistence** and **Database**. These components work together to provide a personalized and user-friendly experience for evaluating skincare products. The design decisions ensure scalability, maintainability, and extensibility, making the system adaptable to future requirements.


System
Predict Skin Type
<<component>> Skin Type Test UI
<<component>> Skin Type Model
DataAccess
DataAccess
<<component>> User Management UI
<<component>> User
User
<<component>> Persistence
Persistence
<<component>> Product Catalog UI
<<component>> Product
Product
DataAccess
<<component>> Database
DB
<<component>> Product Evaluation UI
Product Evaluation
<<component>> Recommend Product UI
<<component>> Product Evaluation Model
Product Recommendation
DataAccess
Powered By: Visual Paradigm Community Edition

3. GLOWG Software Subsystem Design

Since the GLOWG project is a software-only system, this section focuses on the design of the **software subsystem**, which encompasses the entire system. Below is the detailed explanation of the architectural style chosen for the GLOWG system, along with a justification of the design decisions.

3.1. GLOWG Software System Architecture

The GlowGenie system employs a **Layered Architecture**, a widely used architectural style in software engineering. It structures the application into three primary layers, ensuring modularity, scalability, and separation of concerns.

Architecture Layers:

1. **Presentation Layer**:
   - **Role**: Responsible for the user interface and interaction.
   - **Technology**:
   - Developed using React.js (with Node.js for Server-Side Rendering) and Angular for building dynamic and responsive user interfaces. Styled with Bootstrap for consistent and responsive design.
   - **Features**:
     - Handles user interactions (e.g., login, product filtering, and test submissions).

        - Displays data from the backend using RESTful APIs.
2. **Application Layer**:
    - **Role**: Serves as the bridge between the presentation and data layers.
    - **Technology**: Implemented in Python, using Flask, RESTful API and GPT API.
    - **Features**:
        - Processes user inputs and applies business logic.
        - Implements AI-powered features, such as skin type analysis, compatibility feedback, and product ingredient evaluation via GPT API.
        - Facilitates secure and seamless communication between the frontend, database, and external APIs.
3. **Data Layer**:
    - **Role**: Manages persistent data storage and retrieval.
    - **Technology**: Uses **PostgreSQL** for relational database management.
    - **Features**:
        - Stores user profiles, product information, test results, and recommendation data.
        - Ensures data integrity and security through proper indexing, constraints, and role-based access.

Justification for Layered Architecture:

- **Scalability**: Each layer can be scaled independently to handle increased traffic or data.
- **Maintainability**: The separation of concerns allows developers to modify one layer without affecting others.
- **Security**: Sensitive data is handled securely at the application and data layers, reducing exposure risks.
- **Flexibility**: New features can be added or existing features updated within their respective layers.

Design Decisions and Justification

The design decisions for the GLOWG system are guided by the need for scalability, modularity, and user-centric design. Below is a detailed justification for the selected architectural style:

1. **Layered Architecture:**
    - **Why Chosen:** Layered architecture ensures clear separation of responsibilities, making the system easier to extend, debug, and maintain. It also allows independent development of each layer by different teams.
    - **Benefits:** Facilitates the integration of new features (e.g., additional ML models or external APIs) without impacting other layers.
2. **Integration of Machine Learning Models:**
    - **Why Chosen:** Machine learning provides the ability to classify and evaluate product suitability dynamically based on user attributes and product data. It enhances the system's intelligence and adaptability.
    - **Benefits:** Provides transparent and explainable feedback to users, ensuring trust and personalization.
3. **Use of External APIs:**
    - **Why Chosen:** Leveraging external APIs enables the system to access up-to-date product data and enhance analysis without duplicating existing data sources.
    - **Benefits:** Reduces the need for extensive data entry and maintenance, while ensuring accuracy and comprehensiveness.
4. **Scalable Database Design:**
    - **Why Chosen:** PostgreSQL was selected for its support of complex queries and scalability. It handles the relational data of users, products, and feedback effectively.
    - **Benefits:** Ensures efficient data storage and retrieval, supporting future growth in the number of users and products.
5. **Frontend Technology (React.js):**
    - **Why Chosen:** React.js was selected for its ability to build highly interactive user interfaces with reusable components.

    - **Benefits:** Provides a responsive and dynamic user experience, which is essential for attracting and retaining users.

3.2. GLOWG Software System Structure

The system structure of GlowGenie is organized into several primary components, each fulfilling specific roles within a layered architecture. These components have been designed with a focus on modularity, scalability, and maintainability, ensuring that the system is adaptable to future enhancements. The core components include the **Frontend**, **Backend**, **Machine Learning Models**, and **Database** packages, with the integration of external APIs handled within the backend.

Key Components and Their Roles:

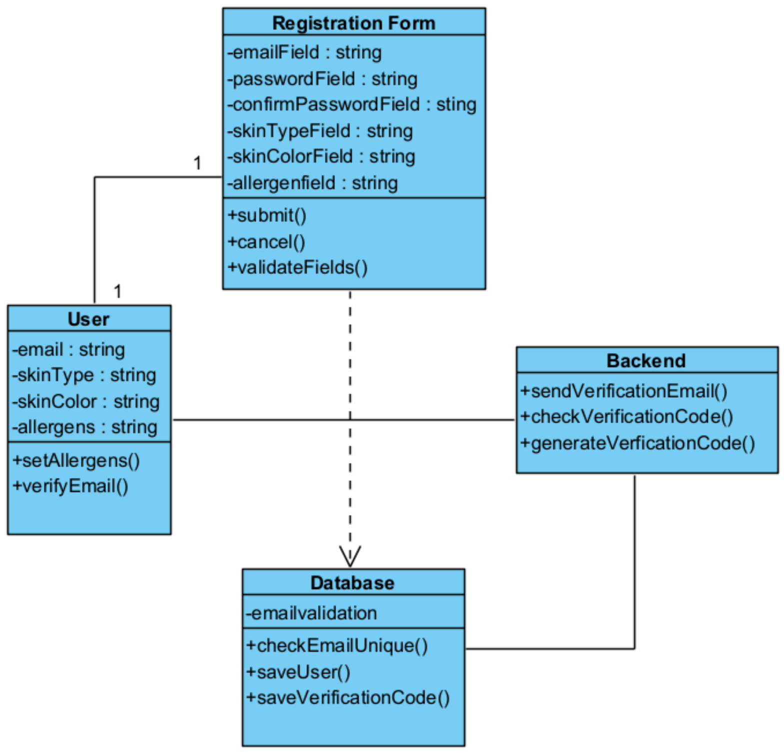
Frontend Package:

- **Responsibilities:** Handles user interactions, including skin type tests, product browsing, and feedback display.
- **Features:**
    - User-friendly interface for personalized skincare recommendations.
    - Real-time filtering of products based on categories, skin types, and preferences.
    - Visual cues for product suitability (e.g., green/red framing).
- **Sub-components:**
    - **User Management UI:** Manages registration, login, profile updates, and skin type test interactions.
    - **Product Catalog UI:** Displays products, allows filtering, and shows detailed product information.

Backend Package:

- **Responsibilities:** Manages core business logic, API services, and interactions with the database and external APIs.
- **Features:**
    - Secure user authentication and profile management.
    - Product data management, including ingredient-based filtering.
    - Integration with external APIs for ingredient data (e.g., GPT APIs).
- **Sub-components:**
    - **User Service:** Implements logic for user registration, authentication, and profile management.
    - **Product Service:** Handles product categorization, filtering, and compatibility analysis.
    - **GPT API Integration:** Manages communication with external APIs to fetch updated ingredient data.
    - **Logging & Security:** Ensures secure operations and logs key activities for debugging and monitoring.

Registration Class Diagram:

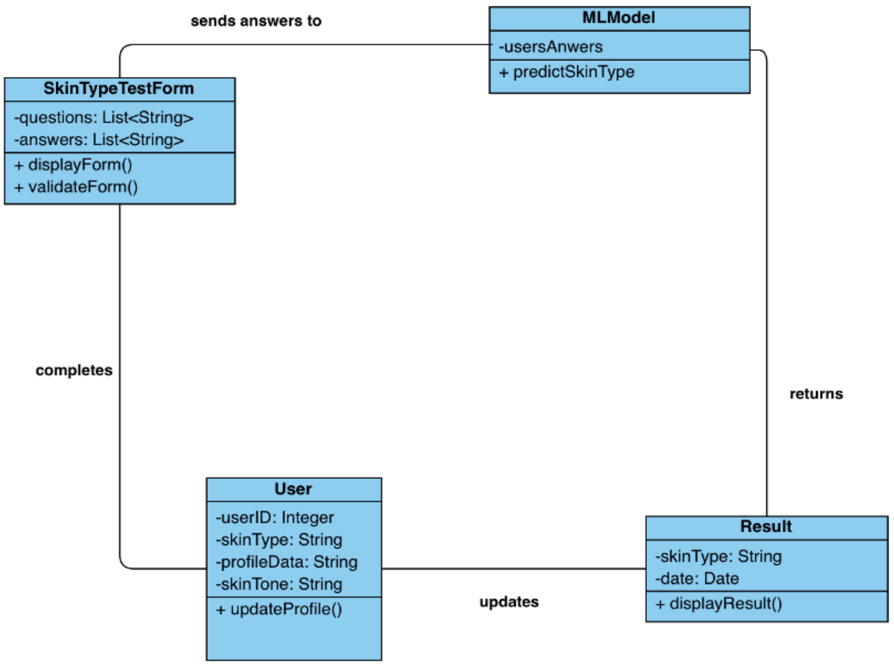


Machine Learning Models Package
Responsibilities:
Manages product compatibility evaluations and updates, ensuring accurate skin type classification and suitability labeling for skincare products based on AI models.

Features:

- **Skin Type Prediction:** Determines users' skin types through a questionnaire.

Skin Type Prediction Class Diagram:



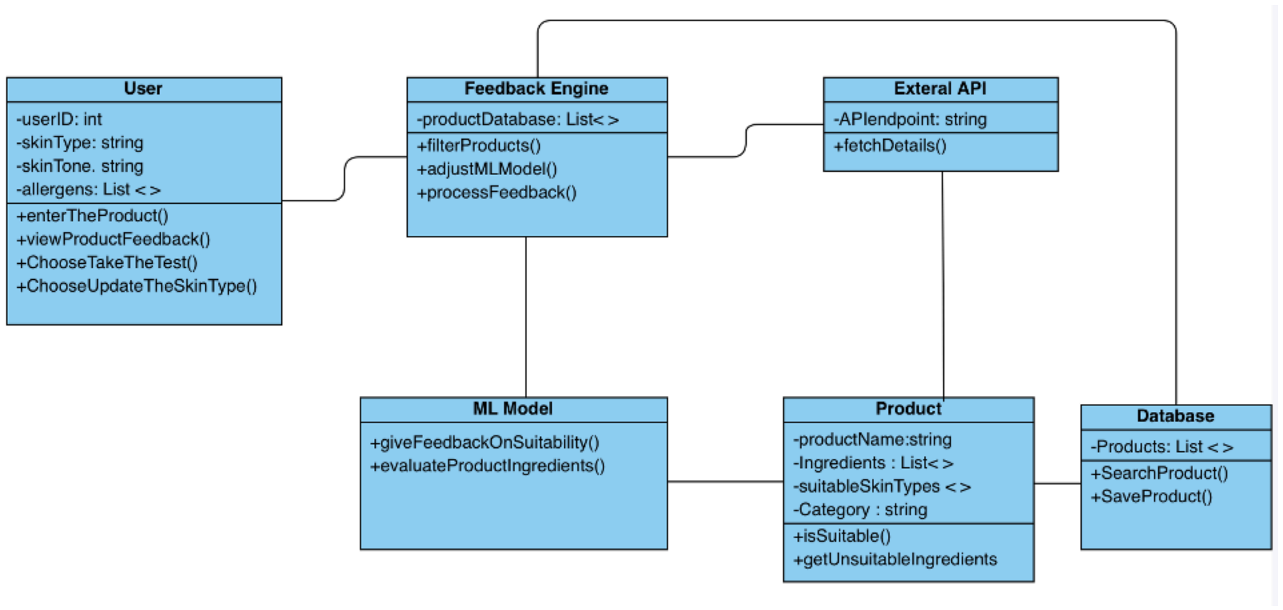
This Class Diagram illustrates the core components and relationships within a skin type detection system. The diagram represents the flow where a user completes a form, submits answers, and the machine learning model processes these answers to predict the skin type, updating the user's profile with the results.

Classes and Attributes:

1. **SkinTypeTestForm:**
   - **Attributes:**
     - questions: List<String> – A list of questions presented to the user.
     - answers: List<String> – A list of responses provided by the user.
   - **Methods:**
     - displayForm() – Displays the test form to the user.
     - validateForm() – Validates user responses for completeness and accuracy.
2. **MLModel:**
   - **Attributes:**
     - usersAnswers – Answers submitted by the user.
   - **Methods:**
     - predictSkinType() – Analyzes responses and predicts the user's skin type.
3. **Result:**
   - **Attributes:**
     - skinType: String – The detected skin type.
     - date: Date – The date the test was conducted.
   - **Methods:**
     - displayResult() – Displays the test results to the user.
4. **User:**
   - **Attributes:**
     - userID: Integer – Unique identifier for the user.
     - skinType: String – The user's skin type.
     - profileData: String – Additional profile information.
     - skinTone: String – The user's skin tone.
   - **Methods:**
     - updateProfile() – Updates the user's profile based on test results.

- **Product Suitability Evaluation:** Classifies products based on ingredient data and predicts their compatibility with various skin types.
- **Ingredient Analysis Feedback:** Identifies specific ingredients contributing to a product's suitability or unsuitability for a given skin type.

Ingredient Analysis Feedback Class Diagram:



This Class Diagram illustrates the architecture and interactions within the product suitability feedback system. The system evaluates skincare products based on user attributes (skin type, skin tone(for sunscreen) and allergens) and provides feedback using machine learning models and external product databases.

Classes and Attributes:

1. **User:**
   - **Attributes:**
     - userID: int – Unique identifier for the user.
     - skinType: string – User's skin type (e.g., oily, dry, sensitive).
     - skinTone: string – User's skin tone (for evaluating products like sunscreen).
     - allergens: List<> – List of allergens to avoid.
   - **Methods:**
     - enterTheProduct() – Allows the user to input a product name.
     - viewProductFeedback() – Displays the feedback on product compatibility.
     - ChooseTakeTheTest() – Directs the user to take a skin type test.
     - ChooseUpdateTheSkinType() – Updates user's skin type profile.
2. **Feedback Engine:**
   - **Attributes:**
     - productDatabase: List<> – A list of products available for feedback and analysis.
   - **Methods:**
     - filterProducts() – Filters products based on user attributes and queries.
     - adjustMLModel() – Adjusts machine learning parameters for better predictions.
     - processFeedback() – Processes and refines product suitability feedback.
3. **ML Model:**
   - **Methods:**
     - giveFeedbackOnSuitability() – Generates feedback on whether a product is suitable for the user.
     - evaluateProductIngredients() – Evaluates the ingredients of a product to determine its compatibility.

4. **Product:**
   - **Attributes:**
     - productName: string – The name of the skincare product.

        - Ingredients: List<> – A list of ingredients present in the product.
        - suitableSkinTypes<> – Skin types for which the product is suitable.
        - Category: string – Product category (e.g., moisturizer, sunscreen).
    - **Methods:**
        - isSuitable() – Determines if the product is suitable for the user's skin type.
        - getUnsuitableIngredients() – Lists ingredients that may cause issues for the user.

5. **External API:**
    - **Attributes:**
        - APIendpoint: string – The endpoint used to fetch external product details.
    - **Methods:**
        - fetchDetails() – Fetches product data from external databases.
6. **Database:**
    - **Attributes:**
        - Products: List<> – A list of stored products.
    - **Methods:**
        - SearchProduct() – Searches for products in the database.
        - SaveProduct() – Saves new product entries.

- **Retraining and Updating Models:** Keeps models up-to-date by incorporating new user data, product information, and feedback.

Sub-components:

1. **Skin Type Test:**
    - AI-based questionnaire designed to predict users' skin types using a pre-trained model.
2. **AI Model Integration:**
    - Implements the logic for integrating machine learning models to evaluate and label products for skin type suitability.
3. **Model Retraining Service:**
    - Handles retraining of the skin type prediction and product suitability models, incorporating updated ingredient and user feedback data to improve accuracy.
4. **Recommendation Engine:**
    - Generates personalized skincare recommendations based on the compatibility between user skin types and labeled product data.
    - Provides ingredient-based feedback to users for transparency and informed decision-making.

Skin Type Prediction Model

1. **Input Collection:**
    - The user completes a skin type test, where each question represents a feature in the model (e.g., frequency of dryness, oiliness, sensitivity, etc.).
    - At first, all questions will directly contribute as features, ensuring a straightforward and comprehensive representation of the user's skin characteristics.
2. **Model Processing:**
    - The responses are processed and fed into the pre-trained skin type prediction model.
    - The model is trained to classify skin types (e.g., dry, oily, combination, sensitive) based on labeled training data.
3. **Prediction:**
    - The model outputs a single skin type as the prediction for the user.
    - The result is saved in the user's profile and used for product recommendations.

Evaluating and Labeling Products for Skin Types Model

1. **Product Data Preparation:**

    - Ingredient data for each product is retrieved from the database.
    - This data serves as input to the model for suitability evaluation.
    - No additional conversion into feature vectors is considered as Random Forest can directly handle structured input.
2. **Model Processing:**
    - A Random Forest model is used to predict product suitability.
    - The model evaluates how suitable a product is for each skin type (e.g., 85% suitable for dry skin, 20% for oily skin, etc.).
    - Softmax is applied to the output to provide a percentage for each skin type.
3. **Labeling:**
    - The product is labeled for each skin type as either **suitable** or **unsuitable** based on the highest suitability percentage or a threshold.
    - Using SHAP (SHapley Additive exPlanations), the model identifies which ingredients contributed to the suitability or unsuitability of the product for each skin type.
    - The reasons for labeling (i.e., the specific ingredients responsible for suitability or unsuitability) are saved to the database alongside the product.
4. **Database Update:**
    - The product data is updated with its suitability percentages, suitability labels for each skin type, and SHAP analysis results (ingredient-level explanations).

Database Package:

- **Responsibilities:** Manages user data, product information, and feedback in a structured database.
- **Features:**
    - Persistent storage of user profiles, products, and recommendations.
    - Efficient data retrieval and updates.
- **Sub-components:**
    - **User Repository:** Handles CRUD operations for user-related data.
    - **Product Repository:** Manages product-related data in the database.
    - **Recommendation Repository:** Stores and retrieves generated recommendations and feedback.

3.3. GLOWG Software System Environment

The GLOWG software subsystem is designed to operate in a robust and scalable environment, leveraging modern hardware, system software, and middleware to ensure optimal performance and reliability. This section provides a detailed description of the target environment, programming tools, and supporting software.

3.3.1 System Software Environment

The system software environment comprises the backend, frontend, database, and middleware technologies that collectively run the GLOWG application.

1. **Backend**:
    - **Programming Language**: Python 3.10.
    - **Frameworks**:
        - Flask: Lightweight web framework for RESTful API implementation.
    - **APIs**: Integration with GPT APIs for ingredient data updates and external AI enhancements.

    Machine Learning Integration:

    - **Scikit-learn:** For implementing the Random Forest model used for product suitability evaluation.
    - **Numpy:** For numerical computations and efficient matrix operations.

    - **Pandas:** For data manipulation and preprocessing, particularly for handling datasets of ingredients and skin type responses.
    - **Softmax (via Scikit-learn/Numpy):** To compute suitability percentages for products across different skin types.
    - **SHAP (SHapley Additive exPlanations):** For generating interpretability insights for product suitability and storing ingredient-based explanations in the database.
2. **Frontend**:
    - **Programming Language**: JavaScript/TypeScript, HTML, CSS
    - **Frameworks**:
        - React.js 18: For building responsive and dynamic user interfaces.
        - Node.js: For server-side rendering and improved performance.
    - **Styling**: Bootstrap 5 for consistent and responsive UI design.
3. **Database**:
    - **Database Management System**: PostgreSQL 15.
    - **Features**:
        - Support for advanced queries and indexing for faster data retrieval.

3.3.2 Development Tools

The development environment for the GLOWG application uses the following tools and platforms:

1. **Integrated Development Environment (IDE)**:
    - Visual Studio Code for coding, debugging, and integrating frontend and backend services.
2. **Version Control**:
    - Git for source code management and collaborative development.
    - GitHub as the central repository for version control and issue tracking.
3. **Testing Tools**:
    - **Postman**: For API testing and verification.
    - **Selenium**: For automated testing of the user interface.
    - **Pytest**: For unit and integration testing in the backend.

3.3.3 Justification for Environment Choices

1. **Modern Frameworks and Tools**:
    - React.js and Flask ensure efficient development and user-friendly interfaces.
    - PostgreSQL provides robust data management capabilities.
2. **Middleware**:

- **Authentication**:
    - **JWT** (jsonwebtoken)
- **Body Parsing**:
    - **express.json()** and **express.urlencoded()** (Built-in in Express)
- **Input Validation**:
    - **express-validator**
- **Error Handling**:
    - Custom middleware in Express
- **Caching**:
    - **cache-manager** (in-memory, Redis, etc.)

4. GLOWG Software System Detailed Design:

Detailed design will be carried out in the context of the COMP 4920 course next semester.

5. Testing Design

We will use **unit testing**, **integration testing**, and **user acceptance testing (UAT)** to ensure the application is reliable and user-friendly. Unit testing will validate individual components, integration testing will ensure seamless interactions between modules, and UAT will confirm the application meets user expectations. Together, these methods ensure functionality, reliability, and user satisfaction.

References

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