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 2.0
17.04.2025

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
2.0	17.04.2025	Detailed design of software system revised in Section 4.

Table of Contents

1. Introduction

This section provides an overview of the purpose, main functionalities, and design methodology of the GlowGenie application, including the system's goals, core features, and adherence to quality standards.

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 DeepSeek 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 as seen below on *Figure 20* 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.

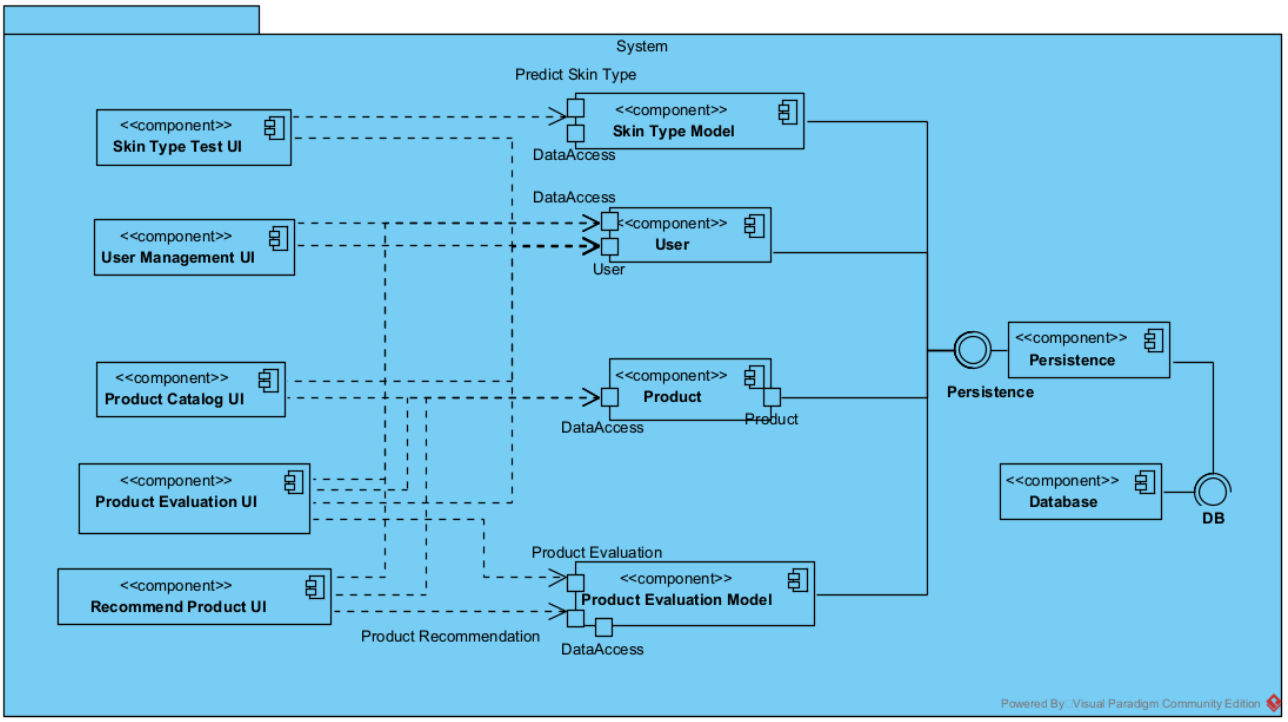


Figure 20 Component Diagram of Glow Genie Software System

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 DeepSeek 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 DeepSeek 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 as seen in *Figure 21*, login as seen in *Figure 22* below, profile updates *Figure 23*, and skin type test interactions.
    - Product Catalog UI: Displays products, allows filtering, and shows detailed product information.

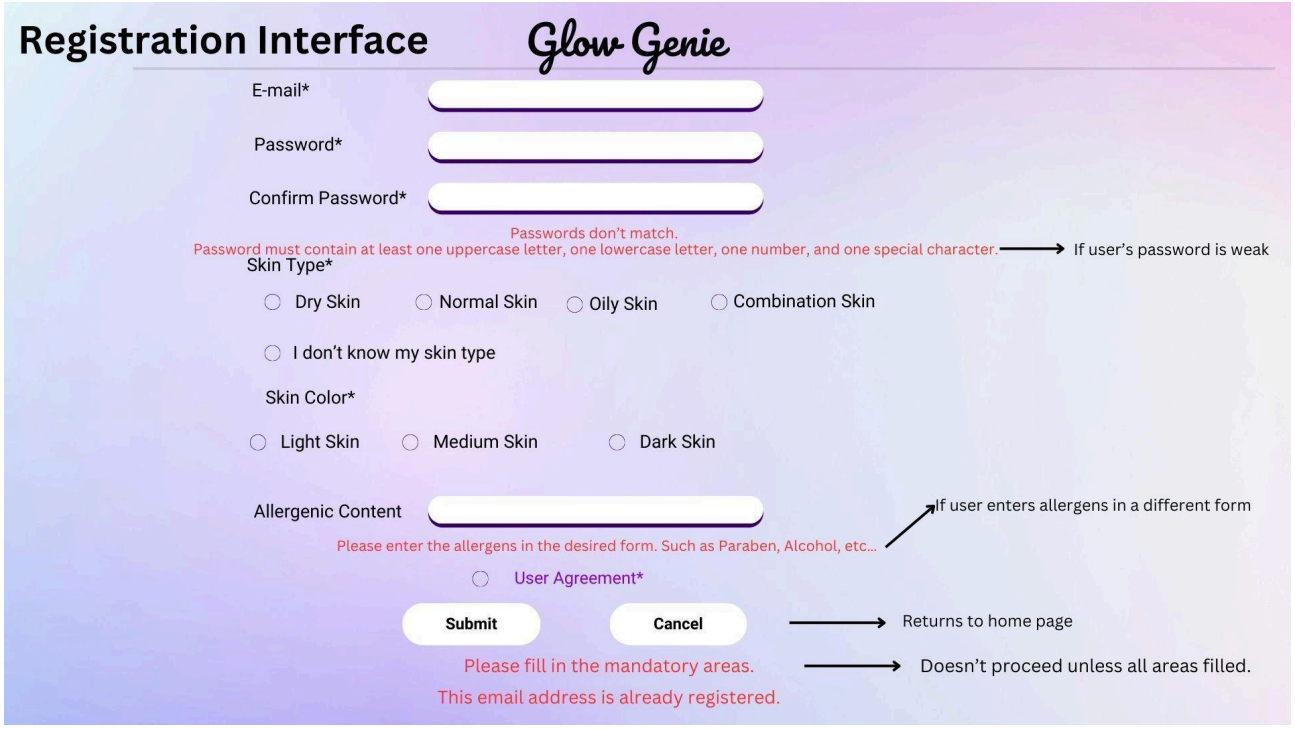


Figure 21 Registration Interface with error messages

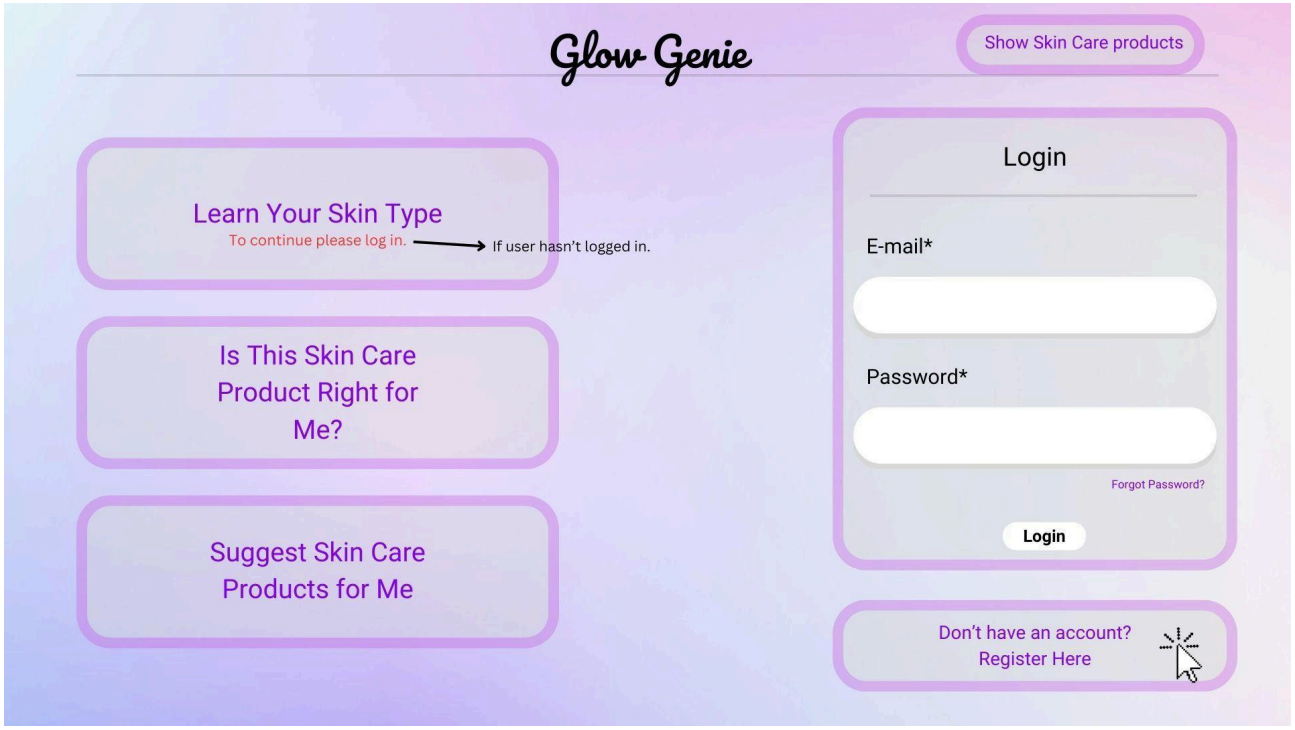


Figure 22 Log In Interface from GUI Demo

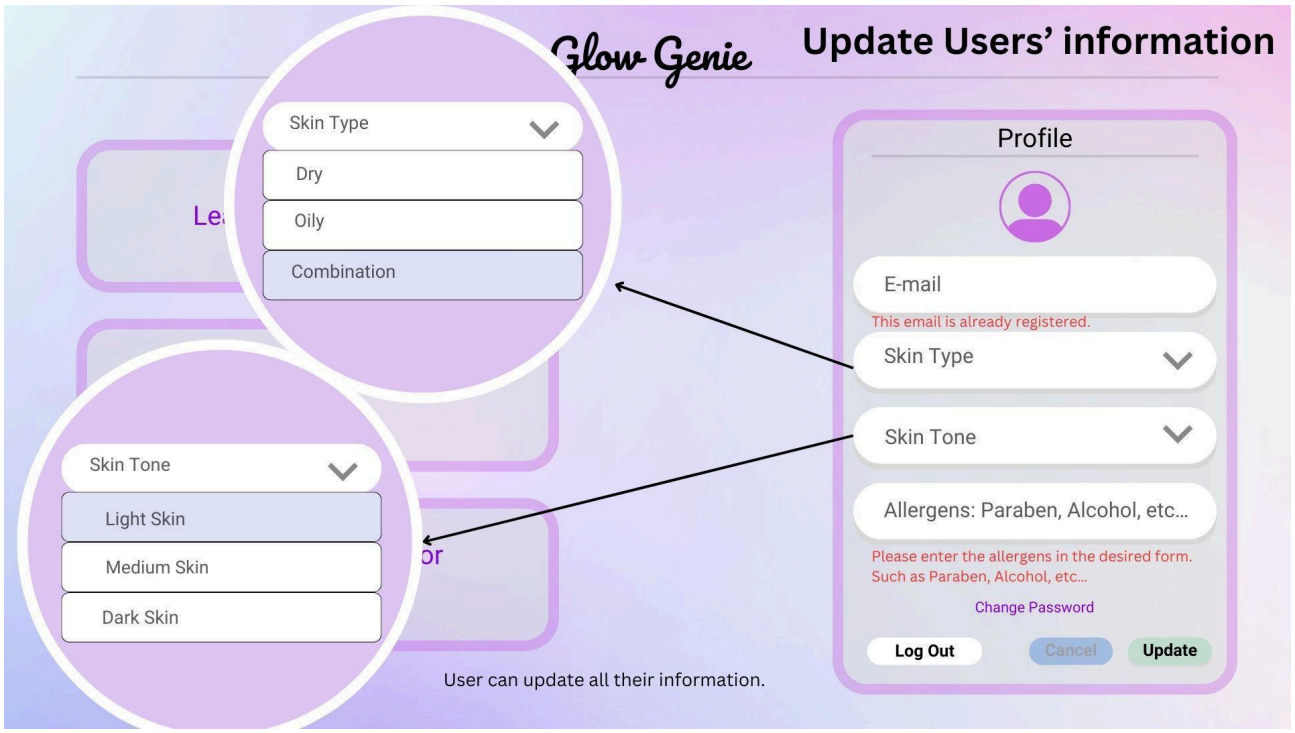


Figure 23 Update Profile Interface with error messages from GUI Demo

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., DeepSeek APIs).
- Sub-components:
  - User Service: Implements logic for user registration, authentication, and profile management.
  - Product Service: Handles product categorization, filtering, and compatibility analysis.
  - DeepSeek 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.

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.

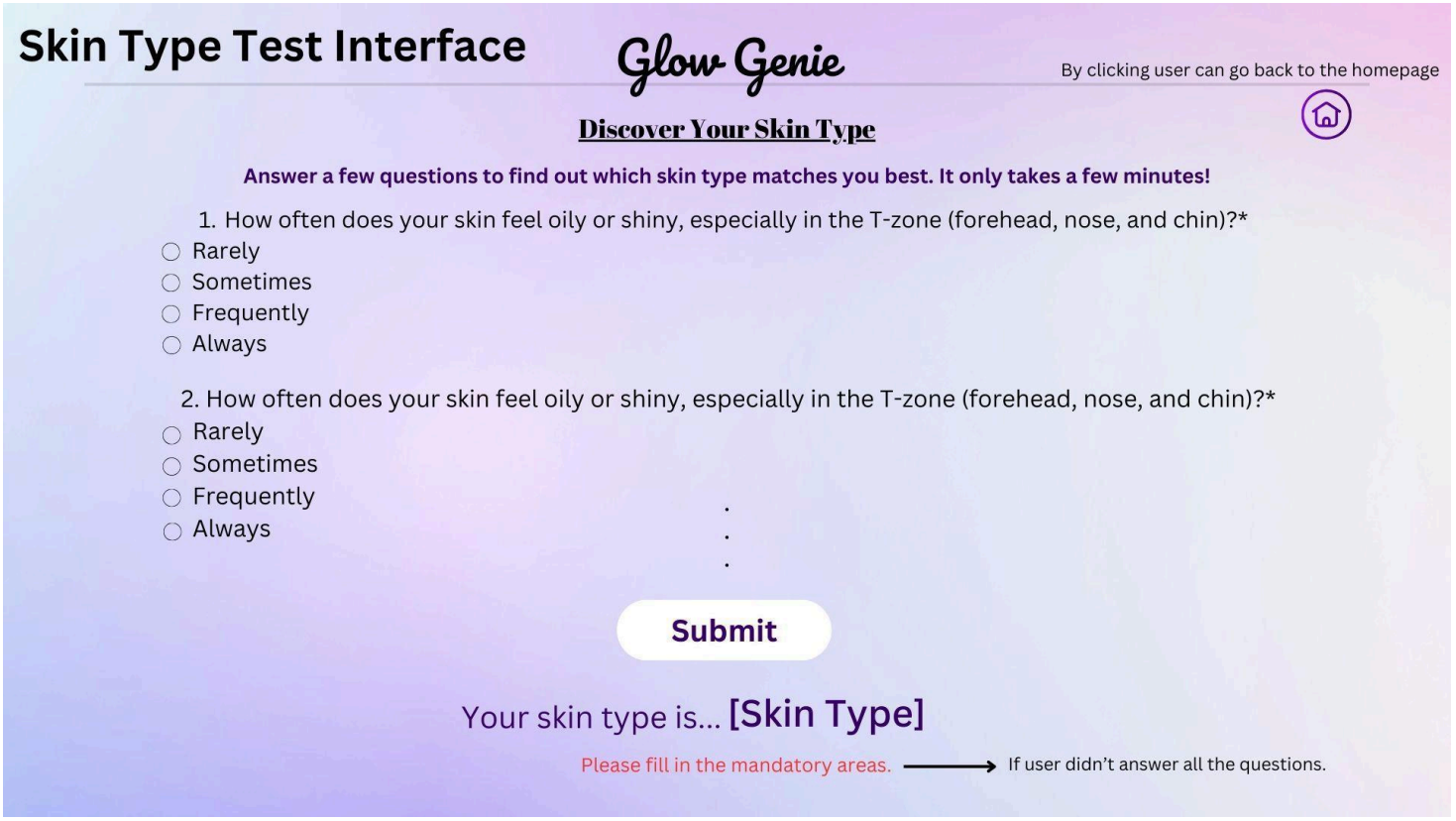


Figure 24 Skin type test and prediction interface from GUI Demo

- 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 as seen in *Figure 25*.

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

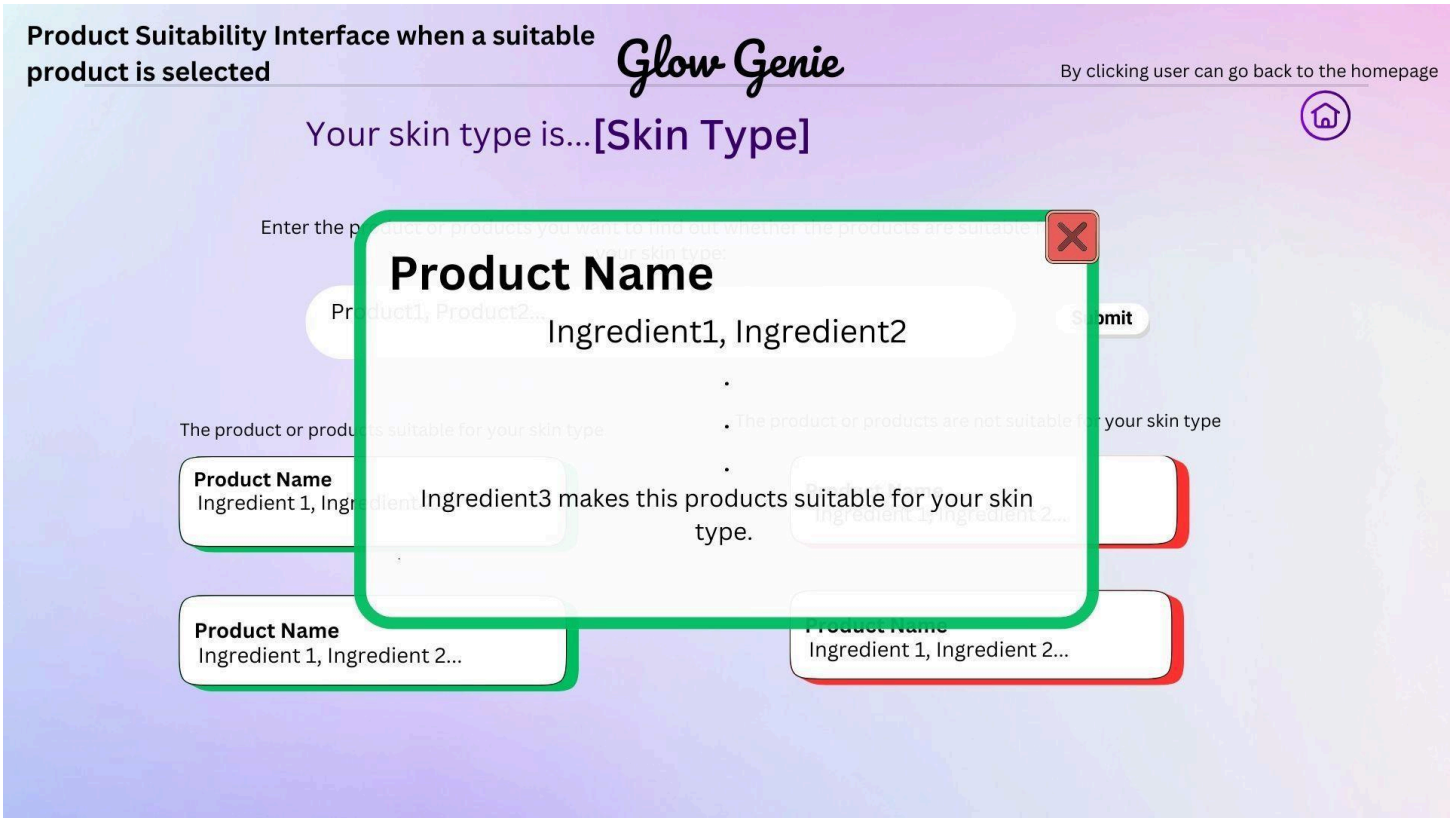


Figure 25 Product suitability interface with given feedback from GUI Demo

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 seen in *Figure 26*.
   - 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.

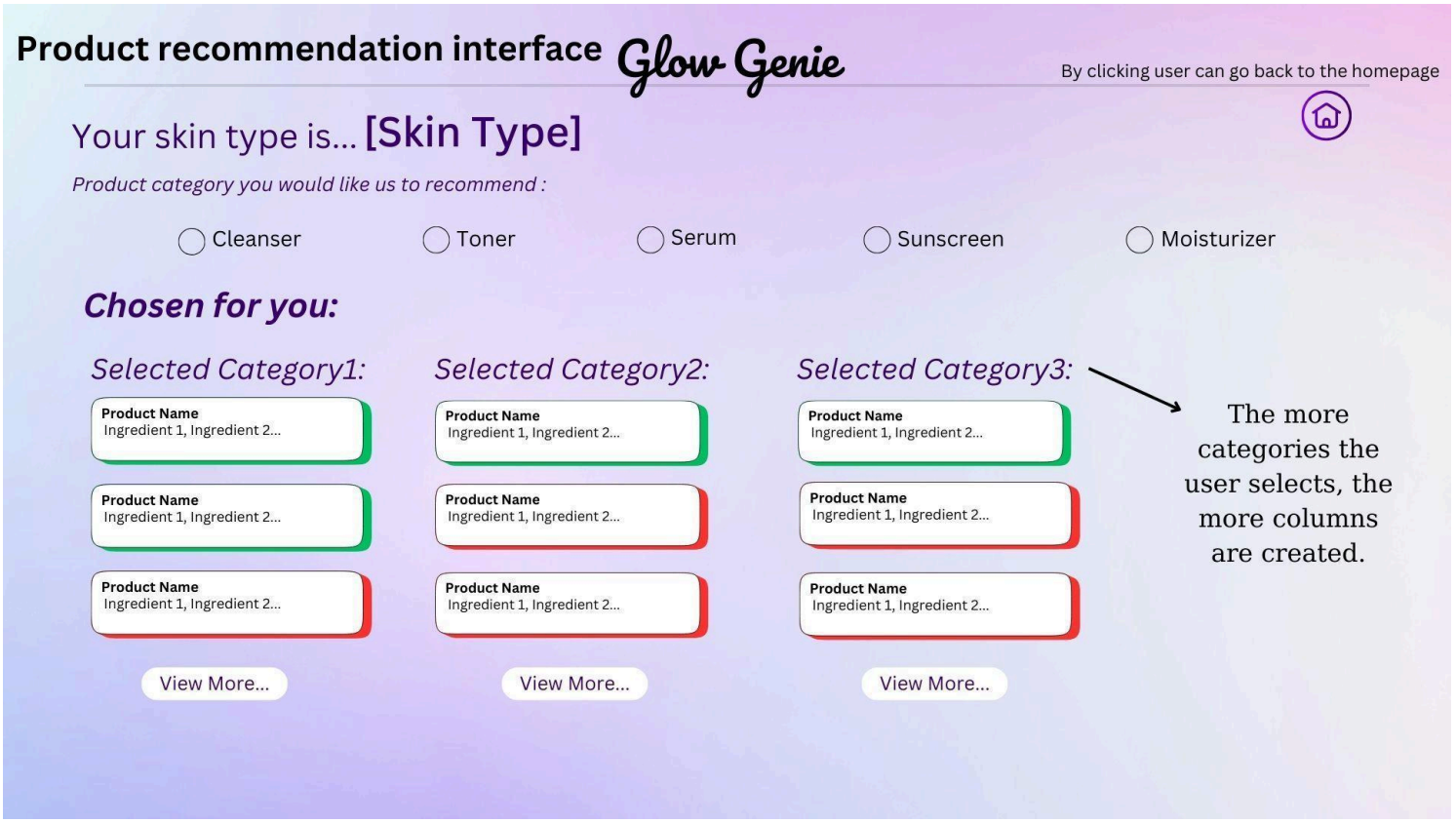


Figure 26 Product recommendation interface from GUI Demo

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 DeepSeek 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:

4.1.GlowGenie Main Module: Backend Main Control Class

The AppController class is the main control class of the GlowGenie backend system, implemented in Python using Flask, a microframework that supports RESTful API development. In object-oriented terms, AppController acts as the central coordinator of the application, responsible for:

- Defining and routing API endpoints
- Connecting frontend requests (React.js) to backend logic and data
- Handling user sessions, authentication, and form validation
- Delegating business logic to specialized services such as:
    - UserService
    - SkinTestService
    - RecommendationService
    - OpenAIIngredientAnalyzer

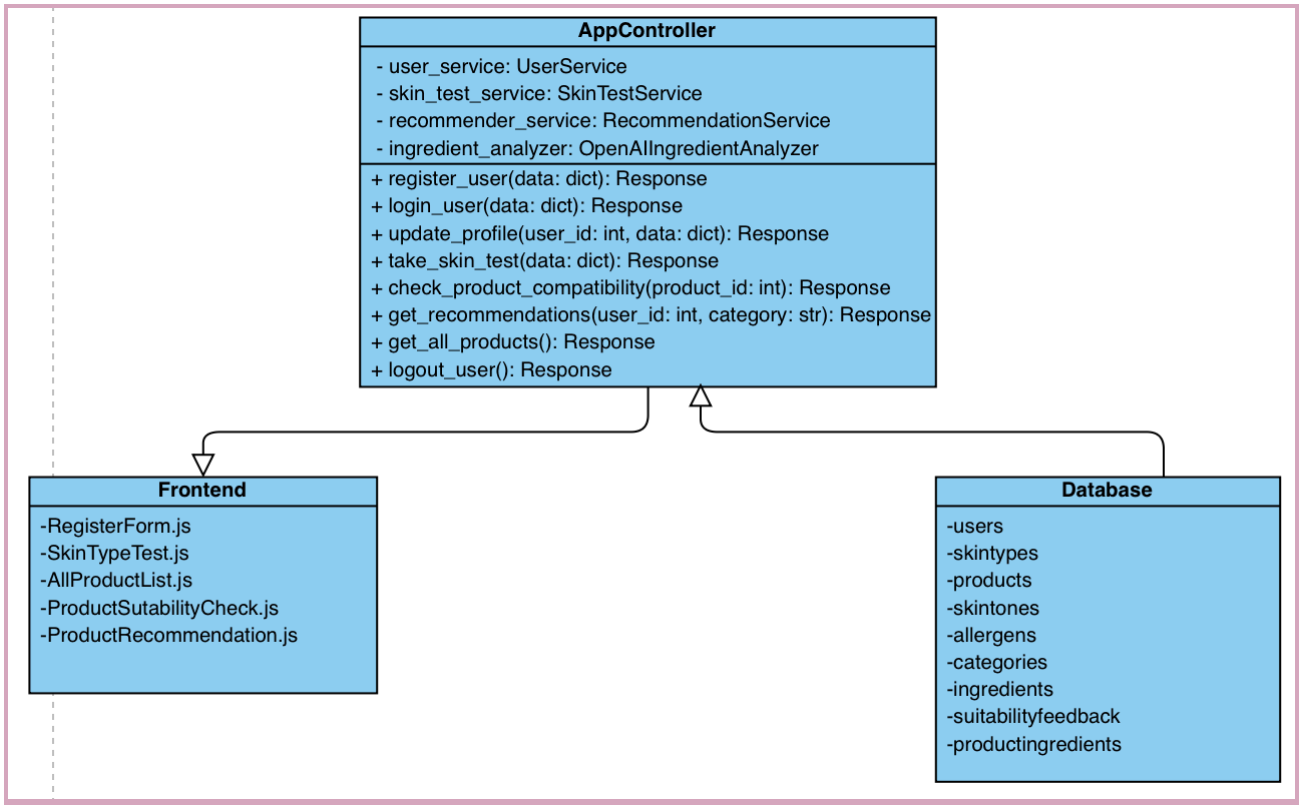
This class adheres to the Single Responsibility Principle by remaining focused on request handling and response dispatching. All heavy processing is delegated to imported services.

Responsibilities of AppController:

- Defining and routing API endpoints that handle user registration, authentication, profile management, skin testing, product compatibility checks, and personalized recommendations.
- Connecting frontend requests (React.js) with backend logic and data resources.
- Managing user sessions, authentication, and input validation to ensure secure and robust data flow.
- Delegating business logic to specialized service classes, including:
    - UserService – for user account management
    - SkinTestService – for skin type prediction via ML
    - RecommendationService – for generating personalized product suggestions
    - OpenAIIngredientAnalyzer – for ingredient analysis using DeepSeek API

This class adheres to the Single Responsibility Principle (SRP) by focusing solely on request handling and response dispatching. All complex logic is offloaded to helper services, ensuring modularity and clarity.

UML Class Diagram – AppController



Sample Method Logic: get_recommendations(user_id, category)

Purpose:

Returns a personalized list of skincare product recommendations using:

- Stored user profile
- Skin type + preferences
- AI (DeepSeek/ML) ingredient filtering

Internal Methods and Responsibilities

The AppController class contains several key methods, each handling a specific responsibility in the GlowGenie backend system:

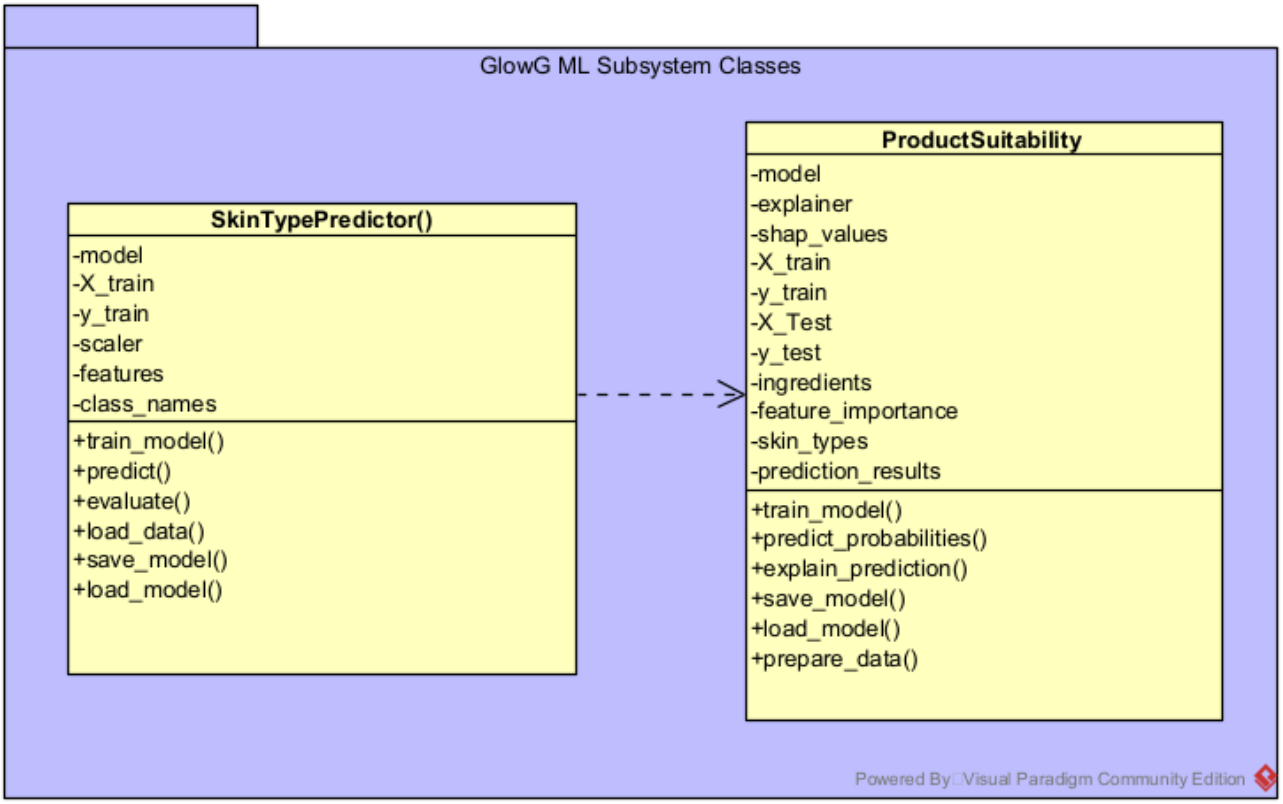
- register_user(): This method is responsible for user registration. It first validates the input data (such as email format, password strength, and required fields), then checks for email uniqueness in the database. Upon successful validation, it creates a new user record and sends a verification email to activate the account.
- login_user(): This method authenticates users by verifying the provided email and password against the database. If the credentials are correct, it generates and returns an access token or session information to authorize the user for further actions.
- update_profile(): This method allows users to update their profile information, including their skin type, skin tone, and known allergens. It ensures all updated values are valid and securely stores them in the database.
- take_skin_test(): When a user completes the skin type questionnaire, this method receives the form results and sends them to the pre-trained machine learning model. The predicted skin type is then stored in the user's profile for future use in recommendations.
- check_product_compatibility(): This method analyzes the ingredient list of a selected product using GPT-based ingredient evaluation. It determines whether the product is suitable for the user's skin type and returns feedback including compatible/incompatible ingredients.

- get_recommendations(): This method combines data from the user's skin profile, filtering logic from the ML-based recommendation system, and ingredient insights from GPT analysis. It returns a personalized list of skincare product suggestions tailored to the user.
- get_all_products(): This method queries the product database and retrieves the entire product list. It is used for browsing or debugging purposes and supports filtering on the frontend.
- logout_user(): This method handles session termination. It invalidates the user's access token or session data and clears any temporary user context stored during the session.

Design Justification

- Separation of Concerns: All business logic is delegated to helper services (UserService, RecommendationService, etc.). This keeps the main controller clean, readable, and maintainable.
- Modularity: Each method handles a single responsibility (register, login, test, recommend), which aligns with RESTful API principles.
- Scalability: AppController is extensible; new endpoints can be added without affecting existing logic.
- Security & Validation: Built-in decorators validate incoming requests, check for authentication tokens, and sanitize input, reducing attack surface.

4.2. GlowGenie Subsystem Machine Learning Classes



4.2.1. ML-SkinTypePredictor

Purpose:
Predicts the user's skin type based on their answers to a fixed questionnaire.

Design Decisions:
Linear regression is selected for its simplicity and interpretability since the dataset includes clearly structured numerical/categorical answers to 9 questions. Each question is treated as a feature in the model.

ML-SkinTypePredictor Method: train_model()

- Input: Training dataset with question responses and known skin types
- Output: Trained linear regression model
- Activity Diagram Logic: Load data → Preprocess → Train linear regression → Save model

ML-SkinTypePredictor Method: predict_skin_type()

- Input: New user's answers to the questionnaire
- Output: Predicted skin type
- Activity Diagram Logic: Load model → Preprocess input → Predict using model → Return prediction

ML-SkinTypePredictor Method: evaluate_model()

- Input: Validation data
- Output: Accuracy metrics
- Activity Diagram Logic: Run predictions → Compare with actual → Calculate accuracy

4.2.2. ML-ProductSuitabilityModel

Purpose:
Classifies skincare products based on their ingredient lists, predicting the suitability percentage for each skin type.

Design Decisions:
Random Forest is chosen due to its high performance on categorical and binary feature sets. SHAP (SHapley Additive exPlanations) is used for post-model interpretability to log which ingredients contribute most to suitability/unsuitability.

ML-ProductSuitabilityModel Method: train_model()

- Input: Product dataset with binary ingredient vectors and known suitable skin types
- Output: Trained Random Forest model
- Activity Diagram Logic: Load data → Convert ingredients to binary vectors → Train random forest classifier → Save model

ML-ProductSuitabilityModel Method: predict_suitability()

- Input: New product's ingredient list
- Output: Suitability percentages for each skin type (via softmax)
- Activity Diagram Logic: Vectorize input → Predict raw scores → Convert to percentages via softmax → Return results

ML-ProductSuitabilityModel Method: explain_prediction()

- Input: Product's ingredient vector
- Output: SHAP explanation values for each skin type
- Activity Diagram Logic: Run SHAP on input → Identify contributing ingredients → Save explanation to database

4.3. GLOWG Subsystem Frontend Classes

4.3.1. Front-End: RegistrationFormComponent Class

RegistrationFormComponent is a frontend class responsible for rendering the user interface for new user registration in the GlowGenie platform. It provides form fields for email, password, skin type, skin tone, and allergens. The class handles field validation, displays error messages, and interacts with the backend API for email verification.

RegistrationFormComponent Class Method: handleSubmit():

This method is triggered when the user clicks the "Submit" button on the registration form. It performs field validation (e.g., password complexity, required fields, email uniqueness) and initiates the email verification process by sending the data to the backend.

RegistrationFormComponent Class Method: validateForm():

This method checks whether all required fields are filled and whether input formats are correct. If validation fails, appropriate error messages are shown (e.g., "Please fill the mandatory areas." or "Passwords must contain at least one uppercase letter…").

RegistrationFormComponent Class Method: requestVerificationCode():

If the email is valid and unique, this method sends a request to the backend to dispatch a 4-digit verification code to the user's email. Rate limits are enforced (e.g., 2 requests per 10 minutes) as defined in the RSD.

RegistrationFormComponent Class Method: handleVerificationCodeSubmit():

Handles the code entry part of registration. If the entered code matches the one sent, registration proceeds. Otherwise, appropriate error messages are displayed. If the wrong code is entered 3 times, the system prompts the user to request a new code.

4.3.2. Front-End: LoginComponent Class

LoginComponent is a class that handles user authentication and password recovery functionalities. It provides error-handling and interaction with the backend for secure login operations.

LoginComponent Class Method: handleLogin():

Validates login credentials entered by the user. If valid, starts a session and redirects to the main dashboard. Otherwise, it displays an error: “Login information is not correct. Try again.”

LoginComponent Class Method: initiateForgotPassword():

When users click "Forgot Password", this method sends a 4-digit verification code to their registered email. Ensures email is registered and not rate-limited.

LoginComponent Class Method: resetPassword():
After entering the verification code, the user sets a new password. This method validates the code and password format, confirms password match, and updates it on the backend.

4.3.3. Front-End: ProfileUpdateComponent Class

ProfileUpdateComponent is the class that provides a user interface for modifying user profile information such as email, skin type, tone, and allergens. It also handles password changes and related email verification steps.

ProfileUpdateComponent Class Method: handleInputChange():

Tracks and updates form input values in real time as the user edits fields. Ensures dropdowns (skin type, skin tone) and text inputs (email, allergens) are synchronized with state variables.

ProfileUpdateComponent Class Method: submitUpdatedProfile():

Sends updated profile information to the backend. Includes input validation (e.g., “Email already exists”, “Please enter allergens in the correct format”) and displays relevant messages accordingly.

ProfileUpdateComponent Class Method: initiatePasswordChange():

Starts the password reset process by sending a verification code to the user’s email. Displays message: “A 4-digit confirmation code has been sent to your email.”

ProfileUpdateComponent Class Method: confirmNewPassword():

Handles code verification and password update logic. If the verification code is correct and new password meets format requirements, saves the password and notifies the user. Handles all errors (e.g., mismatched passwords, invalid code, request limits) gracefully.

4.3.4. Front-End: SkinTypeTestComponent Class

SkinTypeTestComponent is a frontend class that manages the skin type prediction interface. It renders a dynamic questionnaire, collects user input, and communicates with the backend to predict the skin type using an ML model.

SkinTypeTestComponent Class Method: renderQuestions():

Dynamically displays multiple-choice questions about the user's skin behavior based on the predefined dataset (see Appendix D). Each question is a mandatory field.

SkinTypeTestComponent Class Method: handleTestSubmit():

Triggered when the user submits the test. It checks if all questions are answered and sends the data to the backend for skin type prediction. If any question is unanswered, it displays the message: “Please fill in the mandatory areas.”

SkinTypeTestComponent Class Method: displayResult():

After receiving the prediction result from the backend (Dry, Oily, Combination, Normal), this method updates the UI to show the determined skin type and updates the user profile accordingly.

4.3.4. Front-End: ProductSuitabilityCheckComponent Class

ProductSuitabilityCheckComponent is a class responsible for analyzing whether a specific skincare product is suitable for the user's skin type, tone, and allergens. It provides feedback on the product's ingredients using GPT-based analysis.

ProductSuitabilityCheckComponent Class Method: handleProductSearch():
Allows the user to enter a product name or select from a dropdown. Sends a request to the backend to retrieve product details including ingredient list.

ProductSuitabilityCheckComponent Class Method: evaluateSuitability():
Analyzes each ingredient of the selected product by comparing it with the user’s profile data and GPT-inferred ingredient tags (e.g., comedogenicity, hydration properties, irritants). Displays visual cues and explanations for each compatibility result.

ProductSuitabilityCheckComponent Class Method: renderFeedback():
Presents a comprehensive feedback section, showing why the product is suitable or not. Uses clear visual indicators like green suitable, red not suitable alongside ingredient explanations.

4.3.5. Front-End: ProductRecommendationComponent Class

ProductRecommendationComponent is a class responsible for generating and displaying personalized skincare product recommendations. Based on the user's profile data (skin type, tone, allergens, preferences), it interacts with the backend AI model and presents category-based recommendations (e.g., cleanser, toner, serum, etc.).

ProductRecommendationComponent Class Method: fetchRecommendations():
Sends a request to the backend with the user’s current skin type and category preferences. The response contains a list of suitable products based on the content-based filtering model trained using ingredient suitability.

ProductRecommendationComponent Class Method: renderRecommendations():

Renders the recommended product cards, each displaying the product name, category, ingredients, and visual cues indicating suitability (e.g., green or red for each ingredient).

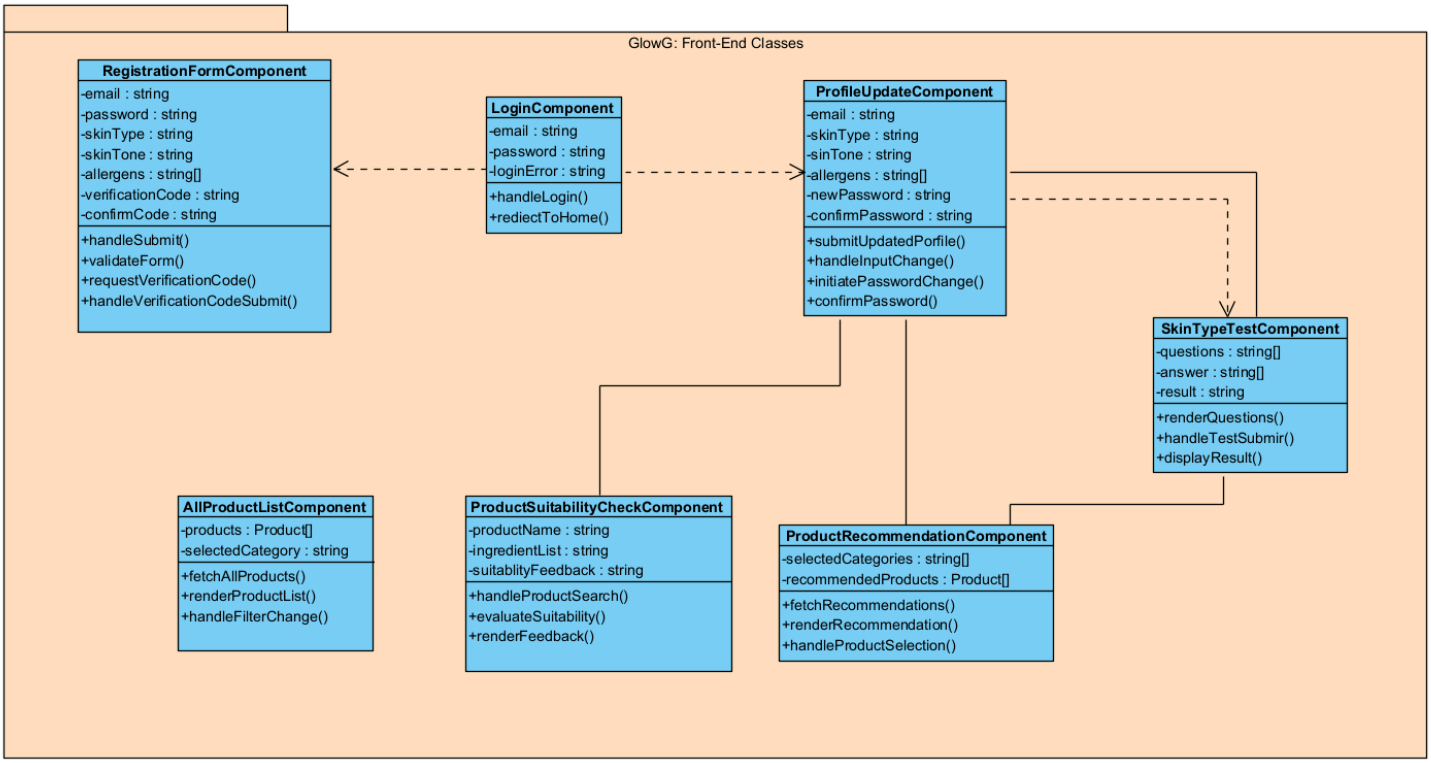
ProductRecommendationComponent Class Method: handleProductSelection():
Allows users to select 1 product for each step (except serums — where up to 3 can be chosen). Selected products are stored in the user's routine and optionally added to a “favorites” list.

4.3.6. Front-End: AllProductsListComponent Class
AllProductsListComponent provides a complete list of all skincare products in the system’s database. It allows filtering by product category, compatibility status, and keyword-based search.

AllProductsListComponent Class: Method: fetchAllProducts():
Sends a request to the backend to fetch the full list of skincare products, optionally applying filters like category or compatibility level.

AllProductsListComponent Class: Method:renderProductList():
Displays the list of products in a responsive grid format. Each product card includes name, brand, category, brief description, and a link to check its suitability.

AllProductsListComponent Class: Method: handleFilterChange():
Handles user-selected filters and re-fetches the product list accordingly (e.g., selecting "only moisturizers", or "only suitable products").



The diagram above illustrates the front-end class structure of the GlowG skincare application. It captures the primary UI components involved in the user journey from registration and login to personalized product recommendations and feedback.

Each blue box represents a frontend component, designed as a class with attributes and operations. These components are grouped within the GlowG:Frontend package, reflecting their shared purpose within the user interface layer.

LoginComponent → RegistrationFormComponent: User cannot log in without registering. Therefore, a dependency relationship was defined between them.

RegistrationFormComponent, LoginComponent, ProfileUpdateComponent: Since they all manage user information, they are grouped as classes with similar responsibilities.

ProfileUpdateComponent → SkinTypeTestComponent: Since the user can repeat the skin test through this component, a dependency relationship was established.

SkinTypeTestComponent → ProductRecommendationComponent: The skin test result is used to determine product recommendations. Therefore, there is a data flow connection between them.

ProductSuitabilityCheckComponent → ProductRecommendationComponent: The suitability of the recommended products is checked with this component. It is an optional supporting component.

AllProductListComponent: It works independently of other components. It lists all products and provides a filtering function.

4.4. GLOWG Subsystem Database Classes

The Database Subsystem of GlowGenie is designed to ensure structured, secure, and scalable data management for all core functionalities of the platform. This component is responsible for storing user information, skincare product data, test results, ingredient analyses, recommendation metrics, and user feedback. It acts as the backbone of the system, enabling seamless communication between the backend logic, machine learning modules, and the user-facing frontend.

The system utilizes PostgreSQL as its relational database management system. Data access is handled through a well-structured data layer that provides efficient methods for querying, updating, and maintaining the information used by other subsystems. The PostgreSQL database is managed via pgAdmin 4, which provides a visual and reliable interface for inspection, query execution, user management, and backup operations.

Design Overview

The database structure includes several main data groups:

- User Management: Handles registration, login, profile updates, and secure credential storage.
- Skin Test Records: Stores questionnaire results and machine learning predictions for skin type.
- Product Catalog: Maintains product information, ingredient lists, and expiration tracking.
- Ingredient Evaluation: Stores ingredient compatibility results based on AI analysis.
- Recommendation Scores: Keeps suitability scores for products based on user skin type.
- User Feedback: Collects feedback and ratings used to improve future recommendations.
- Security & Verification: Manages codes for password reset and account verification processes.

Each of these sections is managed through reusable functions that allow backend services to interact with the database without directly performing low-level operations.

Main Operations

Some core examples of database functionality include:

- Creating and verifying user accounts
- Storing and retrieving skin test data
- Listing and filtering skincare products
- Updating product ingredients and checking for expiration
- Storing AI-generated evaluations of ingredients
- Saving and using user feedback for future recommendations
- Managing password reset and verification processes

Backup and Maintenance Strategy

To protect user data and ensure system reliability, the following strategy is implemented:

- Weekly full database backups
- Daily incremental backups
- Encrypted storage of backups in a secure cloud environment
- Monthly restore simulations to verify disaster recovery processes

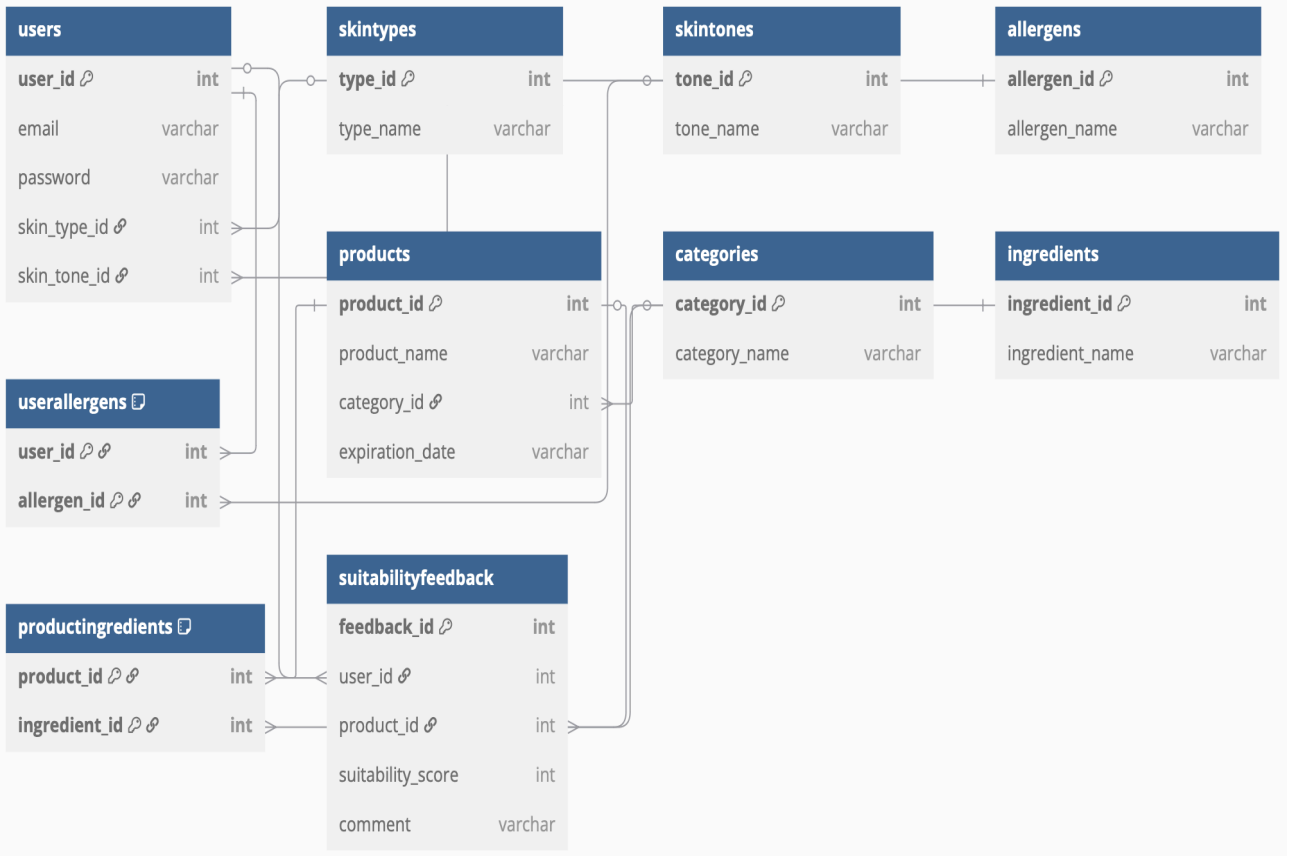
Database operations and maintenance tasks are performed using pgAdmin 4, a visual database management tool that allows administrators to manage data securely and effectively.

Security and Compliance Considerations

- Passwords are encrypted and never stored in plain text.
- Input validation is enforced to prevent harmful queries.
- Different access levels are applied based on user roles.
- Temporary codes for account verification and password reset are time-limited.
- Sensitive actions (like profile updates) are recorded for accountability.

Design Justification

The database subsystem is structured to provide clarity, maintainability, and future flexibility. It supports the overall system architecture by offering centralized, secure data management. As new features are added, the database design allows for smooth integration without affecting core functionality.



The ER diagram presented in this section illustrates the core database structure of the GlowGenie skincare recommendation platform.The design supports key operations such as user registration, product management, skin type analysis, and AI-based product recommendation.

The schema consists of 10 main tables, each aligned with the functional modules and user flows defined in the RSD and DSD documents.

Entity Descriptions and Relationships:

- users: Stores user account and profile information, including references to their skin type and skin tone.
- skintypes and skintones: Reference tables that define the possible skin type and tone categories. Each user is linked to one skin type and one skin tone.
- allergens: Lists possible allergen types, allowing the system to avoid recommending products containing them.
- userallergens: A join table representing a many-to-many relationship between users and allergens.
- products: Contains core product data and references the product's category and expiration information.
- categories: Stores product categories (e.g., cleanser, serum, sunscreen).
- ingredients: Represents individual ingredients used in skincare products.
- productingredients: A join table defining the many-to-many relationship between products and ingredients.
- suitabilityfeedback: Captures feedback given by users about product suitability, supporting machine learning-based updates.

Relationship Logic:

- A user has one skin type and tone (Users → SkinTypes,SkinTones)
- A user can have many allergens (via UserAllergens)
- A product belongs to one category (Products → Categories)
- A product has many ingredients (ProductIngredients)
- A user can give feedback on many products (SuitabilityFeedback)
- All join tables use composite primary keys to uniquely identify many-to-many connections

By adhering to database design best practices—such as normalization, clear foreign key constraints, and appropriate many-to-many structures—the schema ensures data integrity, scalability, and efficient querying, forming the backbone of AI-supported skincare personalization within GlowGenie.

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).