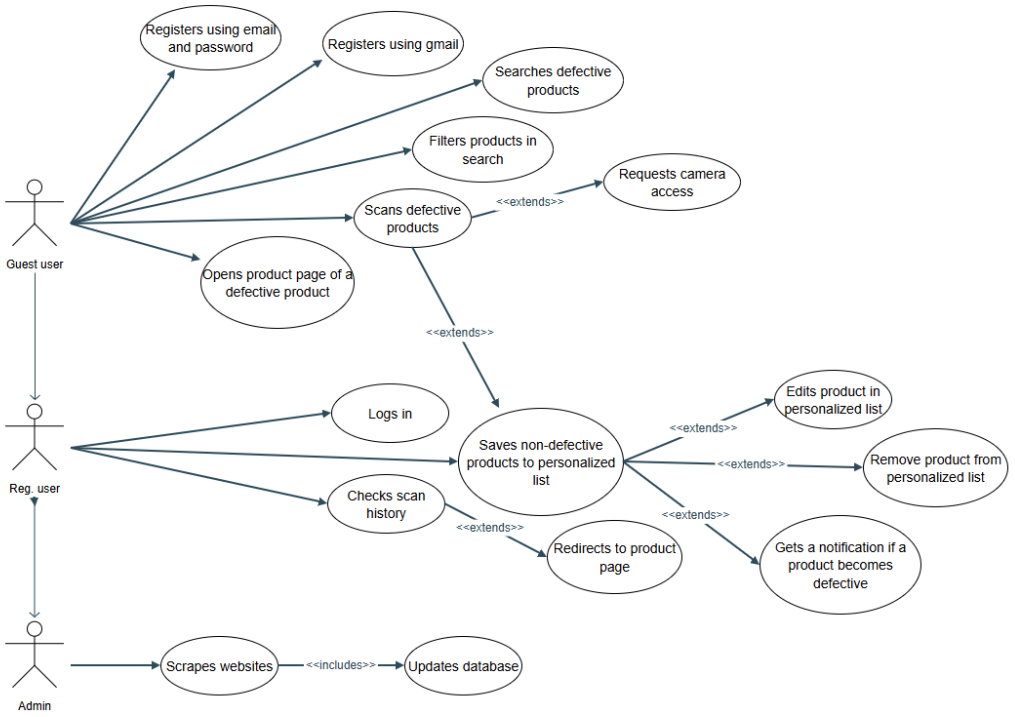
Safety inspection

Martin Demovič, Heorhii Rudyi, Liliia Orlova, Richard Macko

1. Document Introduction

This document presents a proposal for the implementation of a system for product safety verification through a barcode scanning application. It includes a database analysis using a data model, the technologies utilized, a graphical design of the user interfaces, diagrams illustrating the system's functionalities and interactions, the processing of requirements in the code, and additional adjustments to ensure the system operates efficiently.

1.1. Use-case diagram



This **Use Case Diagram** shows the main interactions among three types of users—**Guest User**, **Registered User**, and **Admin**—and the system's features for scanning and tracking product defects:

- **Guest User** can:
  - **Register** using either a standard email/password flow or a Google account to gain access to additional features.
  - **Search and filter** products to see if they're defective.

    - **Scan barcodes** (with a required **camera access** request) to identify defective products and view their details.
- **Registered User** inherits all Guest capabilities and can also:
    - **Log in** to access personal features.
    - **Check scan history** of previously scanned products.
    - **Save non‑defective products** to a personalized list, then **edit** or **remove** them.
    - **Be notified** if any saved product becomes defective (e.g., via email).
- **Admin** can:
    - **Scrape websites** for updated product data and
    - **Update the database**, ensuring new or changed product defect information is always up to date.

2. Data model

Data Model Overview

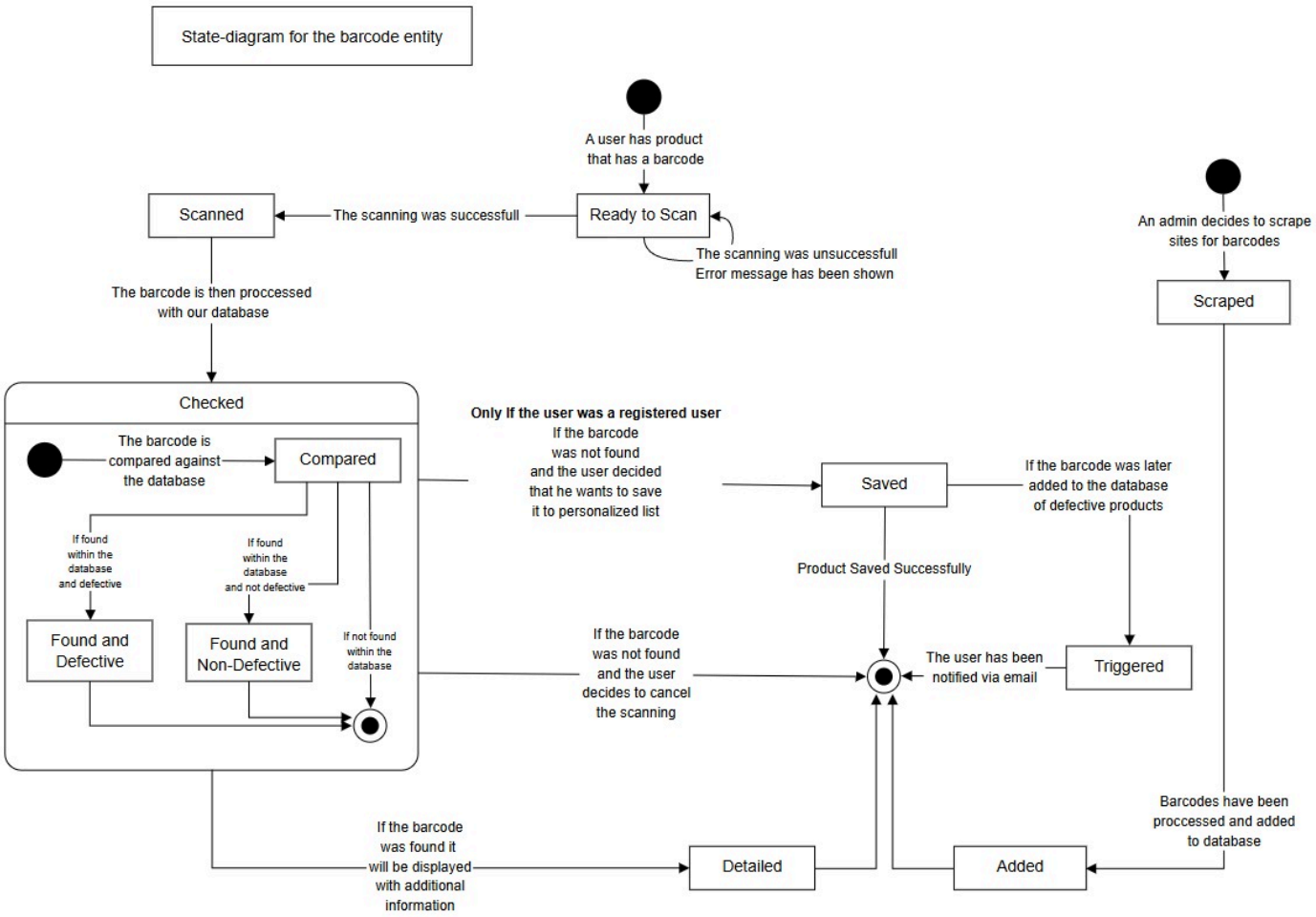
The schema comprises four main tables—**defective_products**, **product_history**, **user_submitted_products**, and **users**—which together store product data, user accounts, and scanning history:

- **defective_products** holds official recall and safety‑alert information (e.g., alert types, product details, brand, barcode, date of publication), typically pulled from external sources via scraping.

- **product_history** logs user scans, linking a **user_id** to a **product_id** (from the defective_products table), as well as storing the date, time, and barcode details for each scan.
- **user_submitted_products** captures custom product entries added by users themselves, including the product's name, brand, barcode, and a brief description, with a **user_id** indicating who submitted it.
- **users** contains user account information such as **username**, **email**, hashed passwords, **google_id** (for Google sign‑in), and **role** (e.g., admin or standard user).
- **password_reset_tokens** manages password reset requests by storing unique tokens associated with user accounts, their expiration times, and creation timestamps to ensure secure password reset processes.

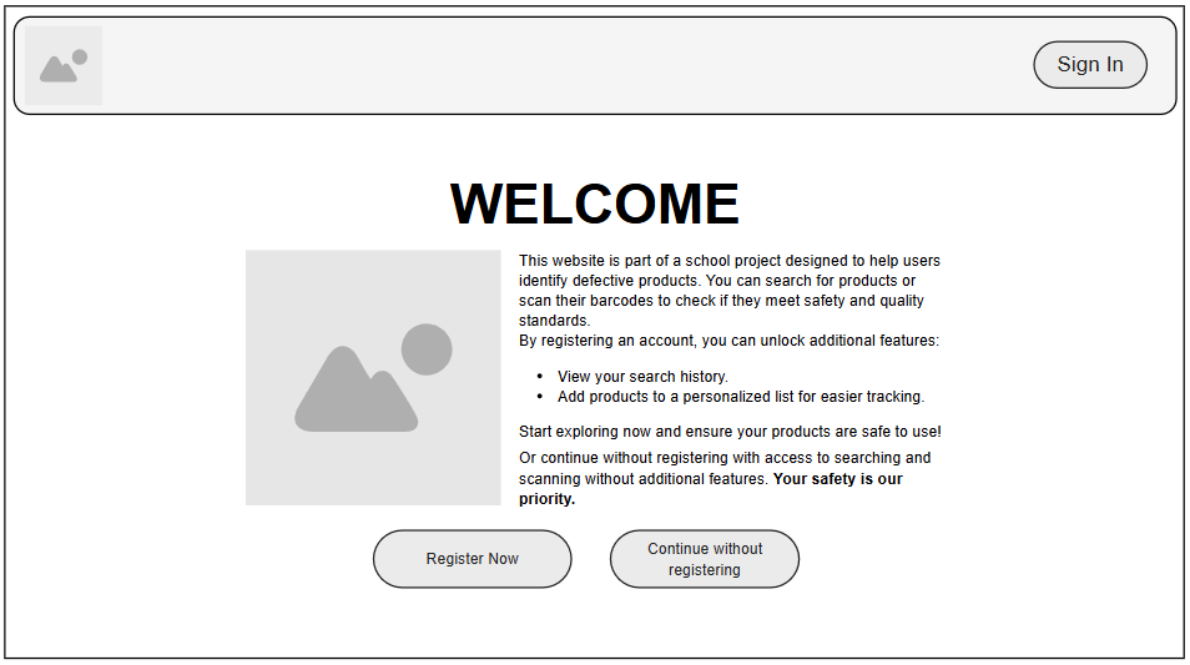
This design provides clear relationships between scanned products, user‑added products, and the users who perform or submit data, ensuring traceable history and easy retrieval of both defective and non‑defective product records.

2.1 State diagram

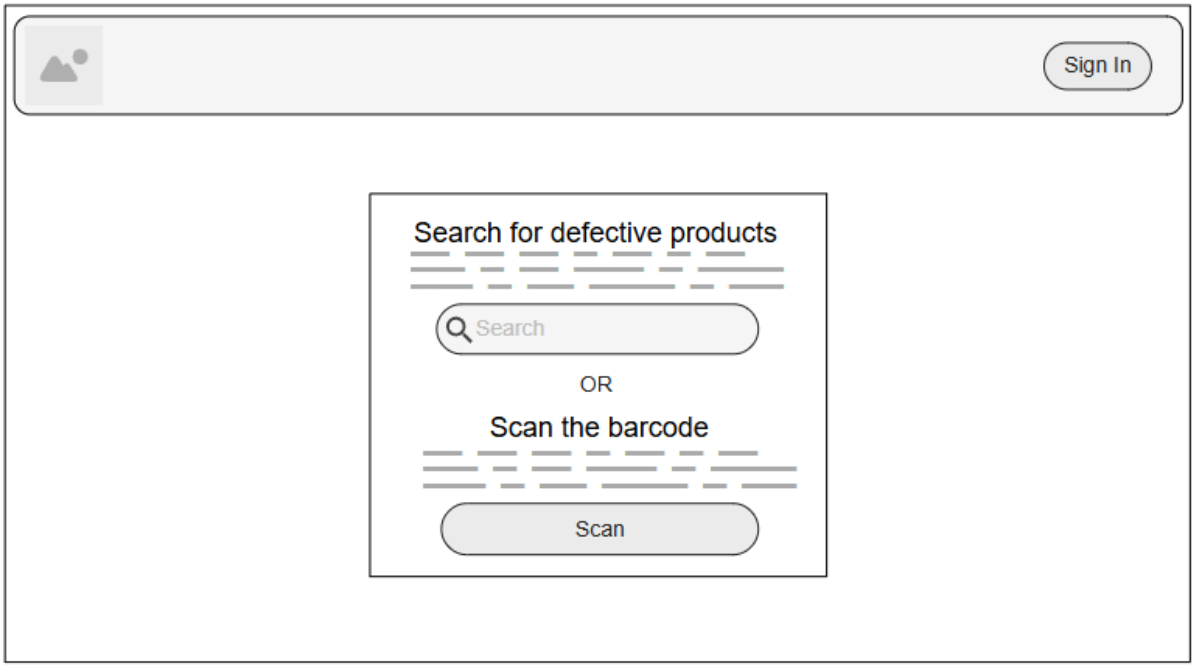


3. User interface design

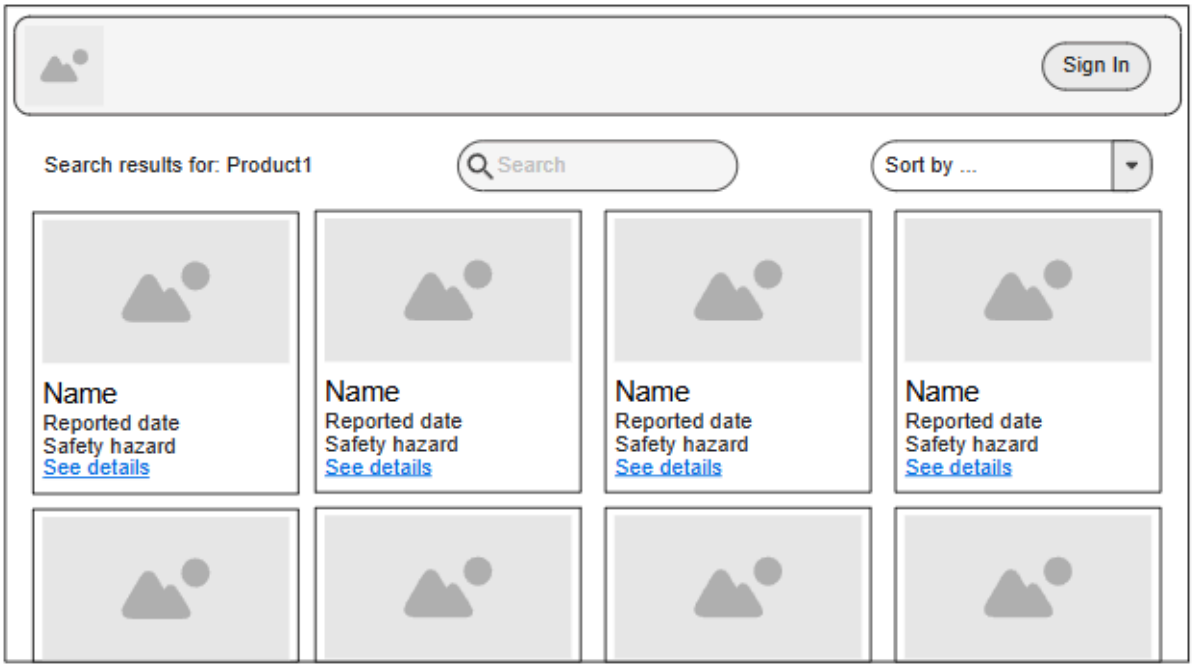
3.0 Introductory page



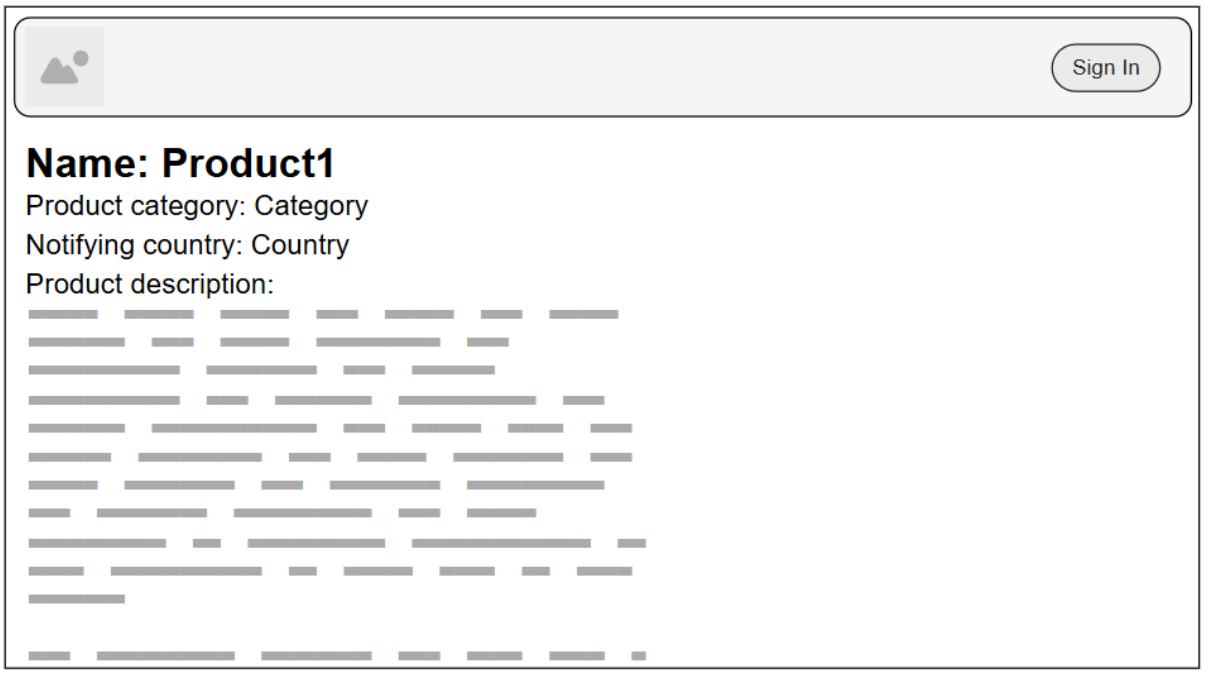
3.1 Homepage of a guest user



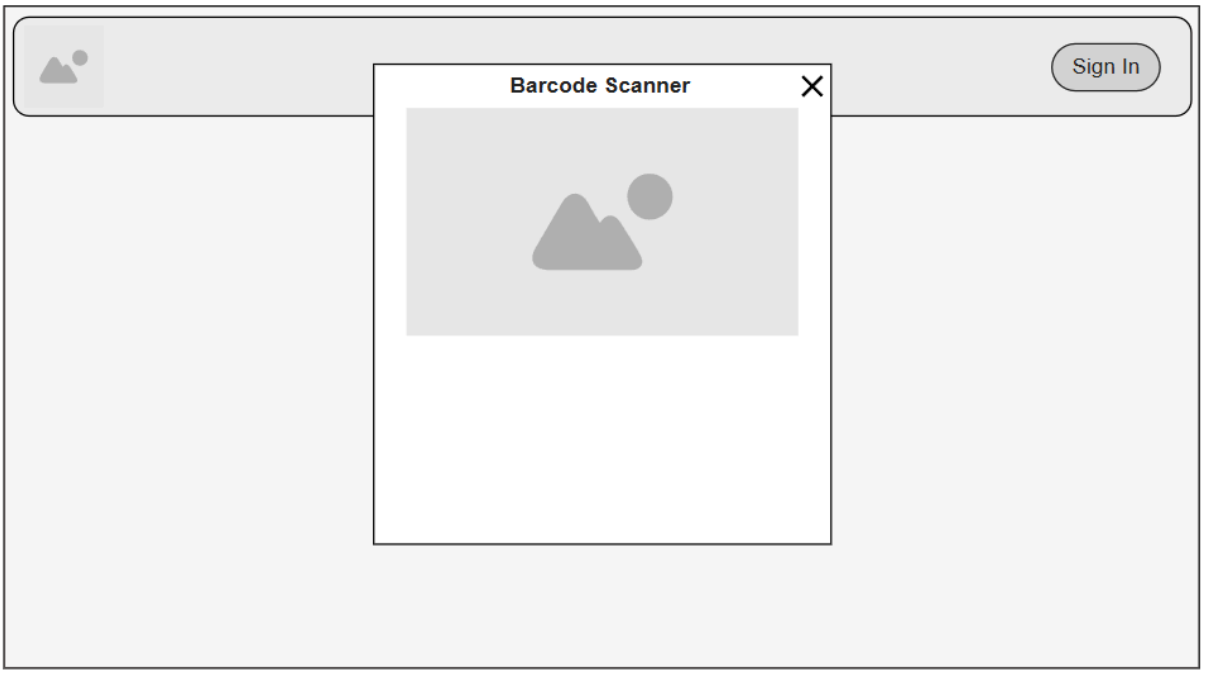
3.2 Search page



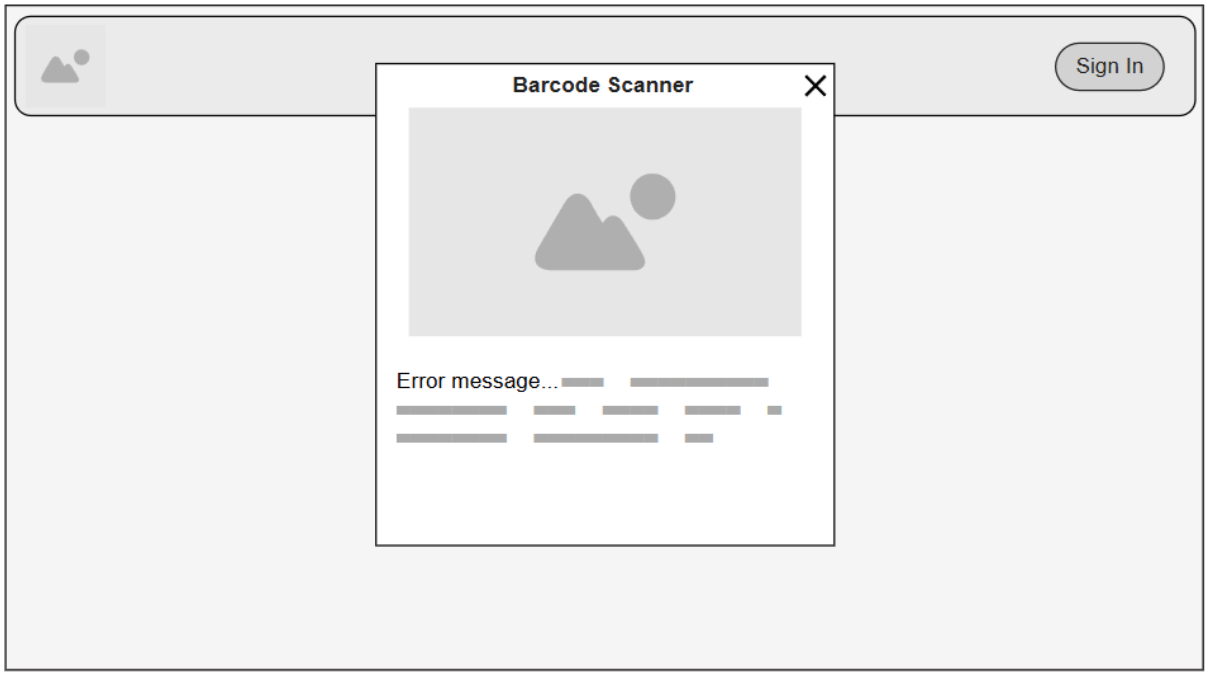
3.3 Product page



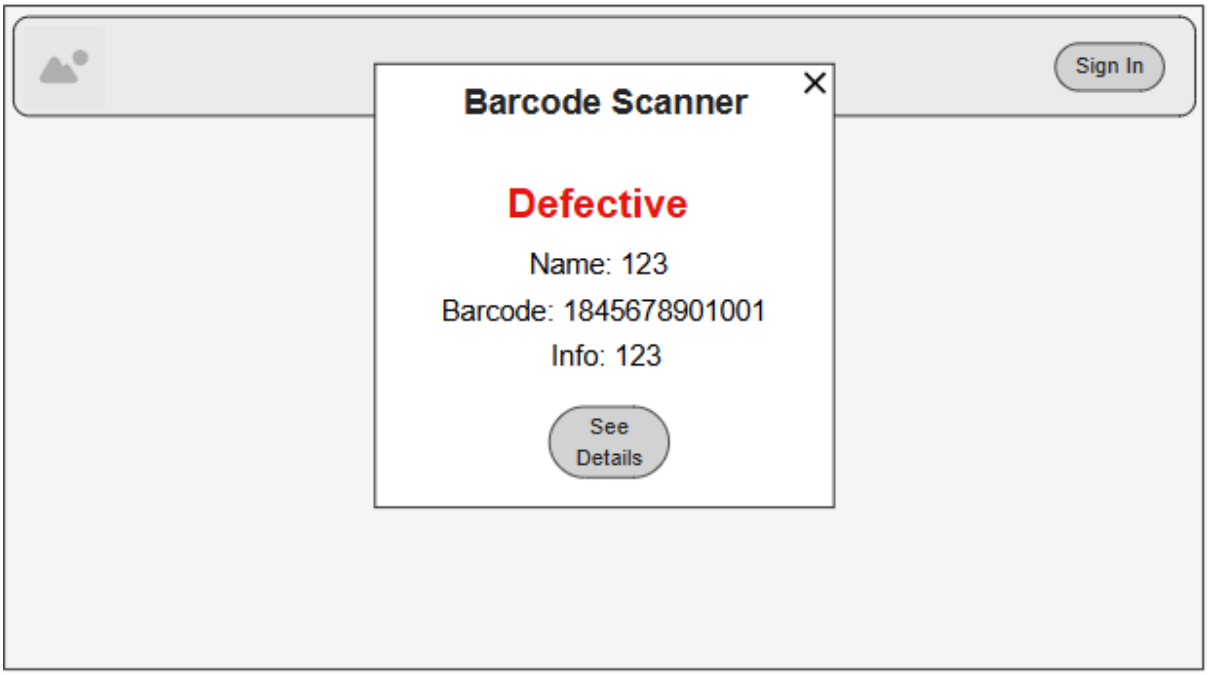
3.4.1 Barcode scanning (Modal)



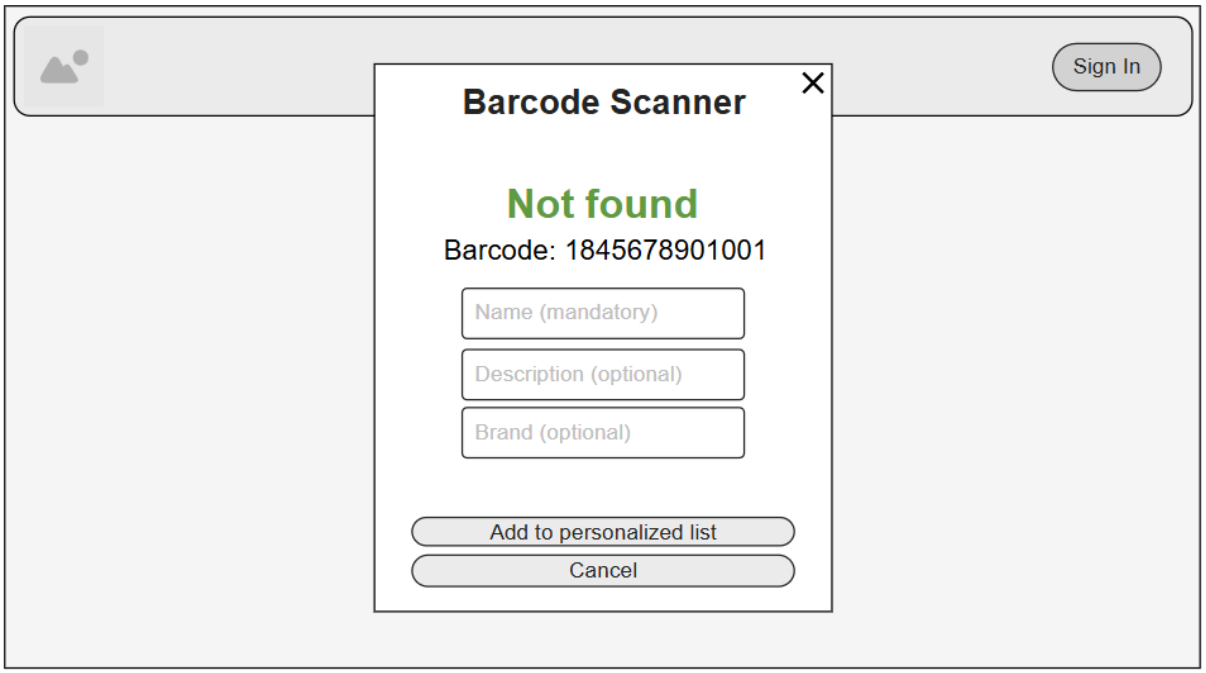
3.4.2 Barcode scanning - Error (Modal)



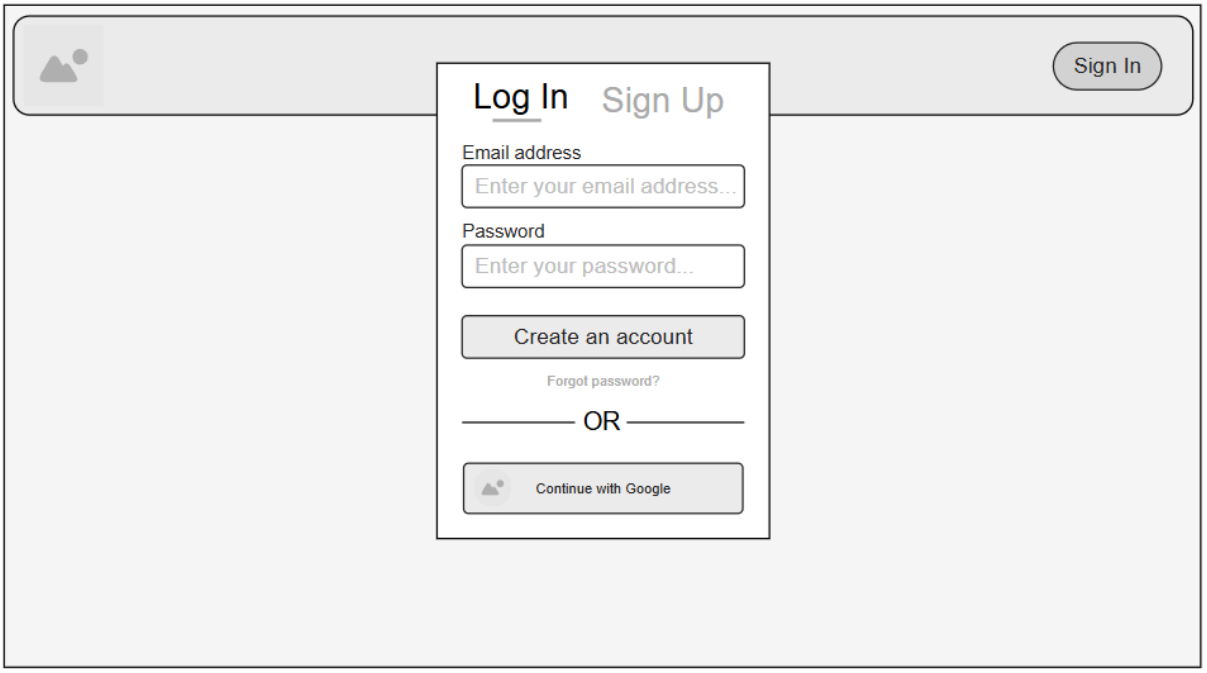
3.4.3 Barcode scanning - Found (Modal)



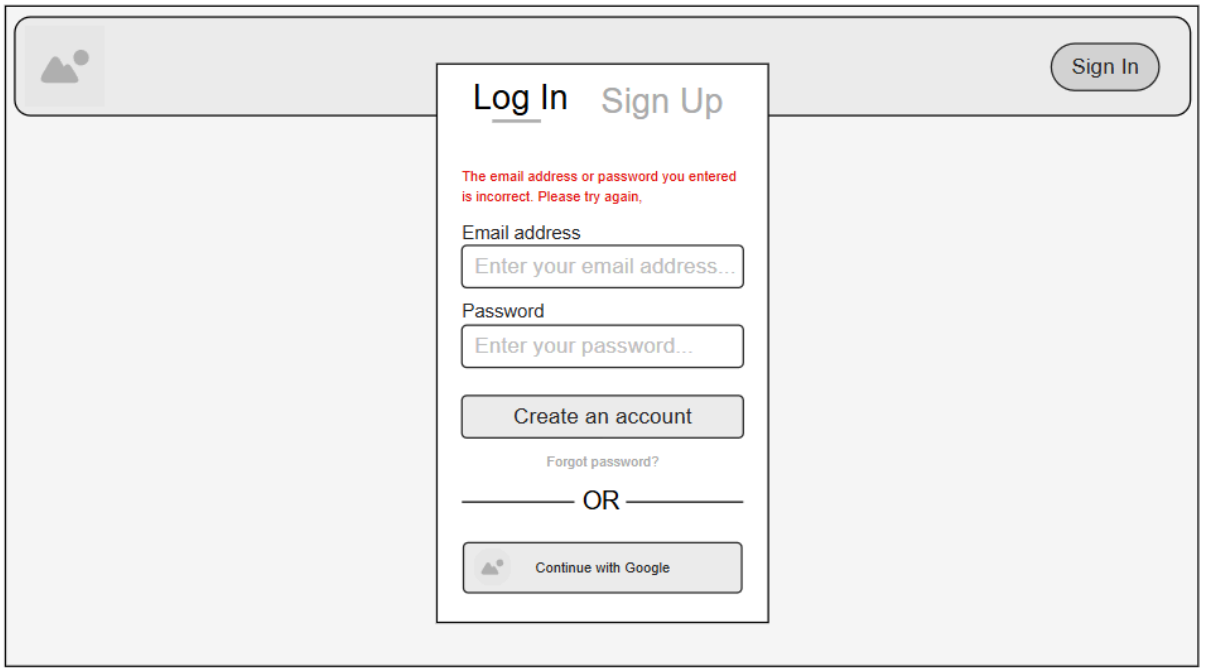
3.4.2 Barcode scanning - Not Found (Modal)



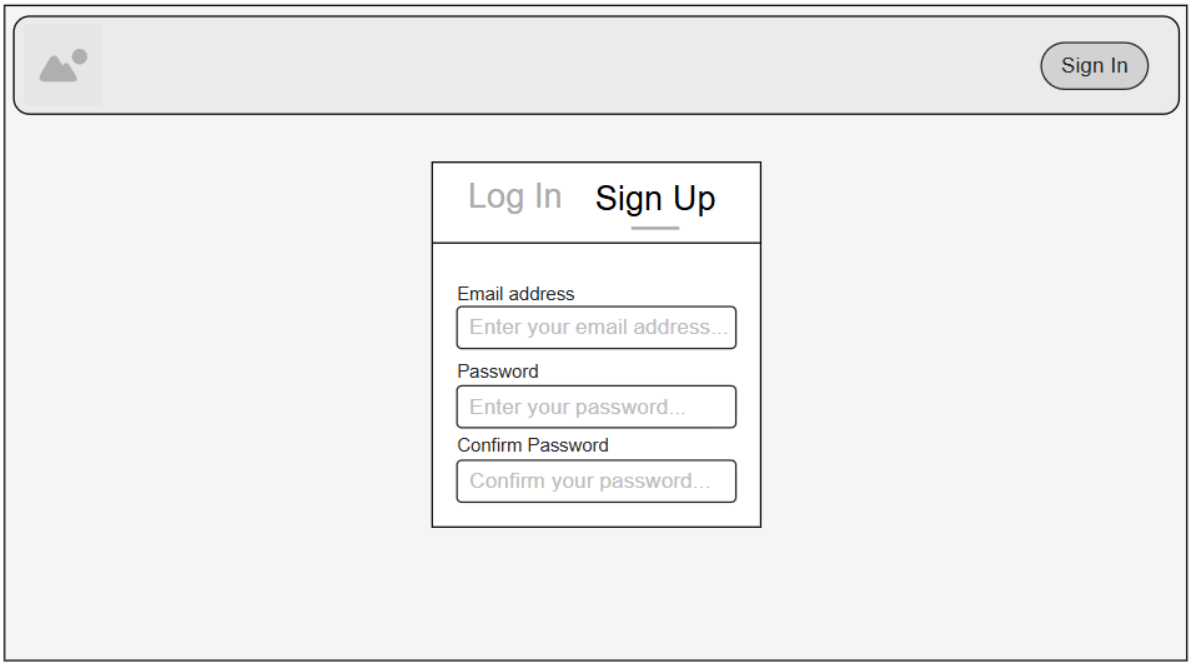
3.5.1 Login page (Modal)



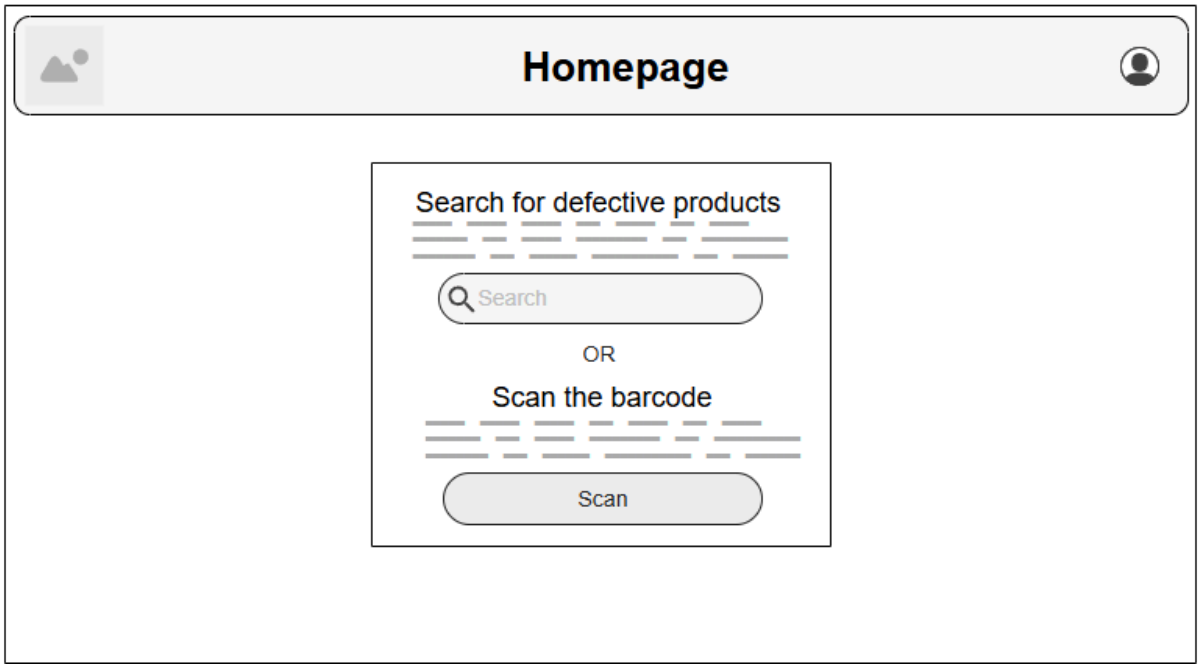
3.5.2 Error in Log In/Sign In (Modal)



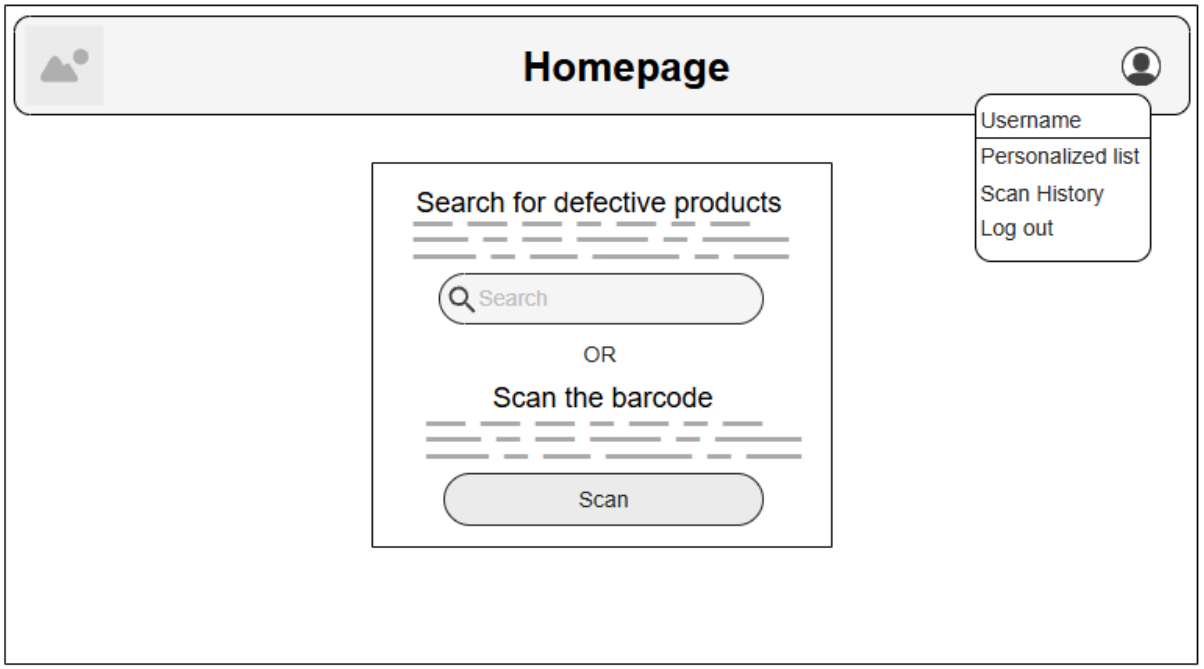
3.6 Registration page (Modal)



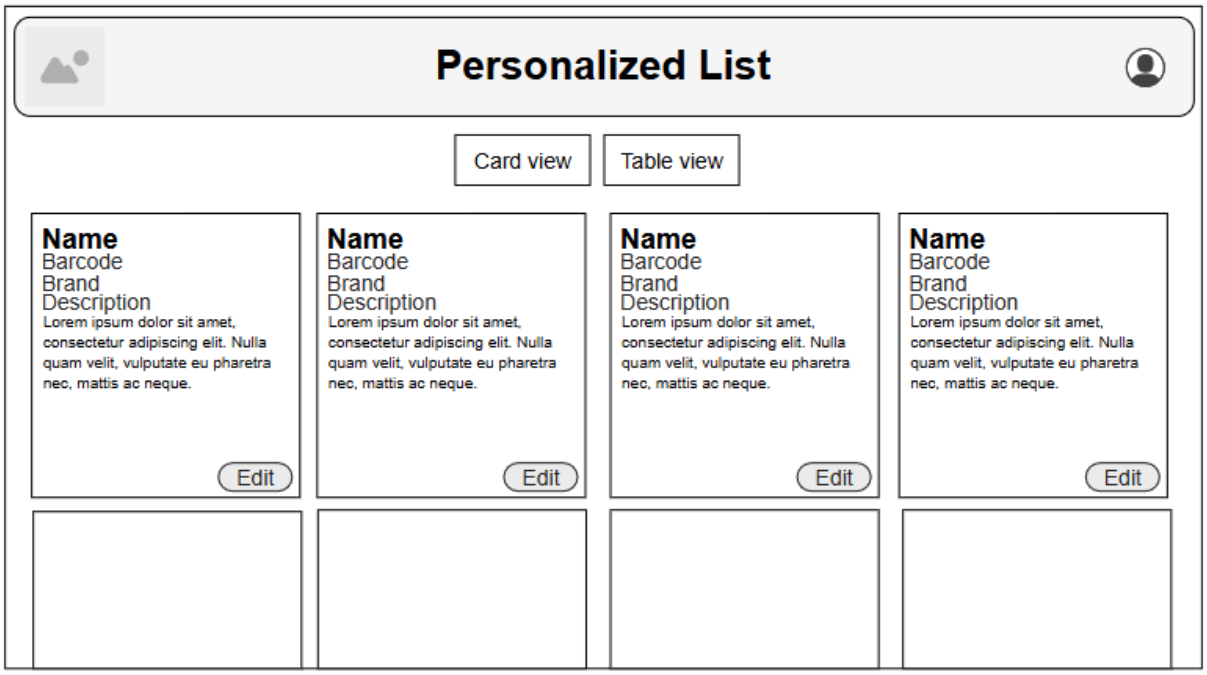
3.7.1 Homepage of a registered user



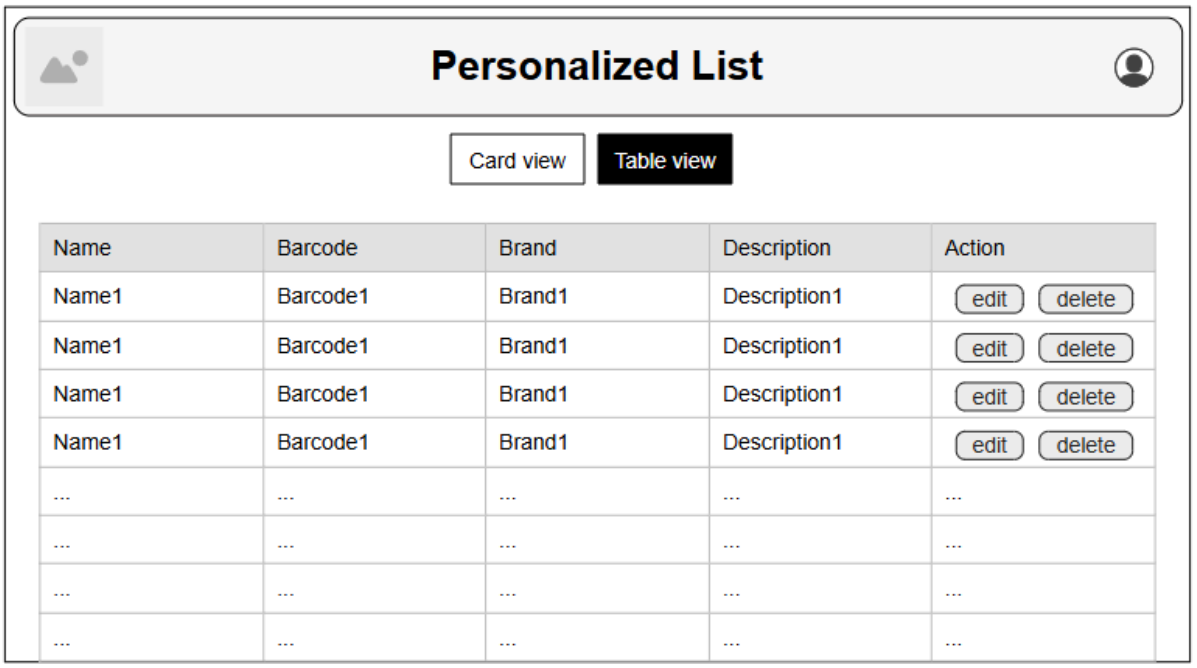
3.7.2 Homepage of a registered user (with user modal open)



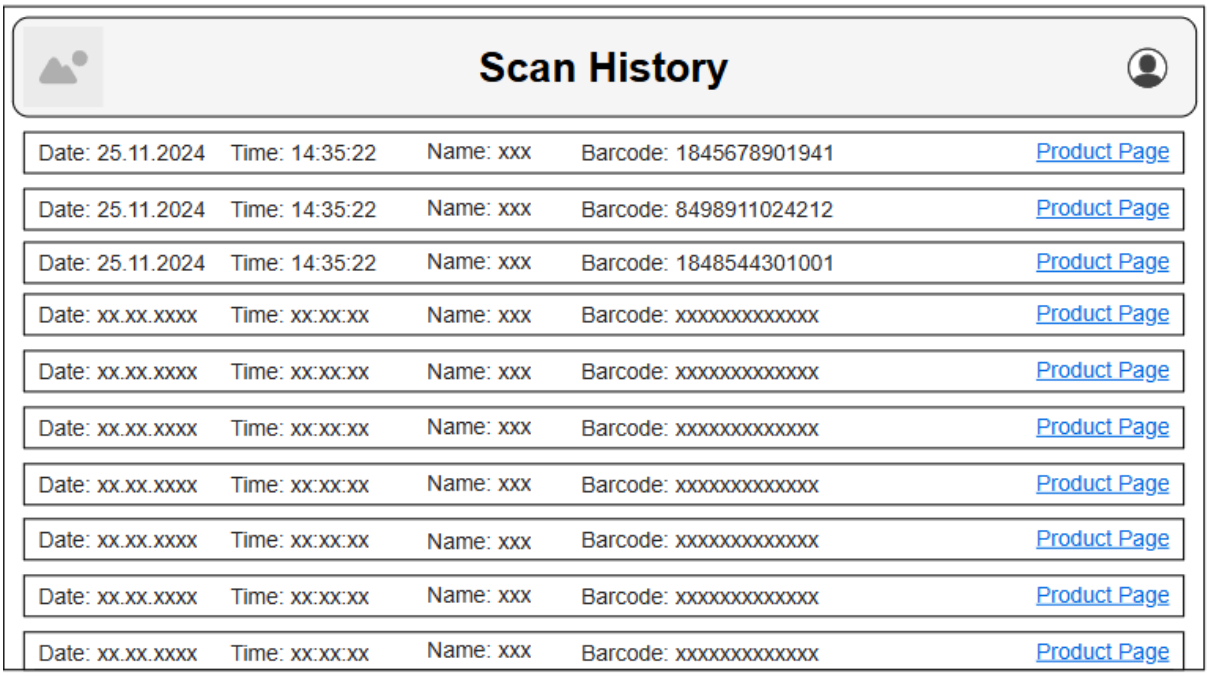
3.8.1 Personalized list page (Card view)



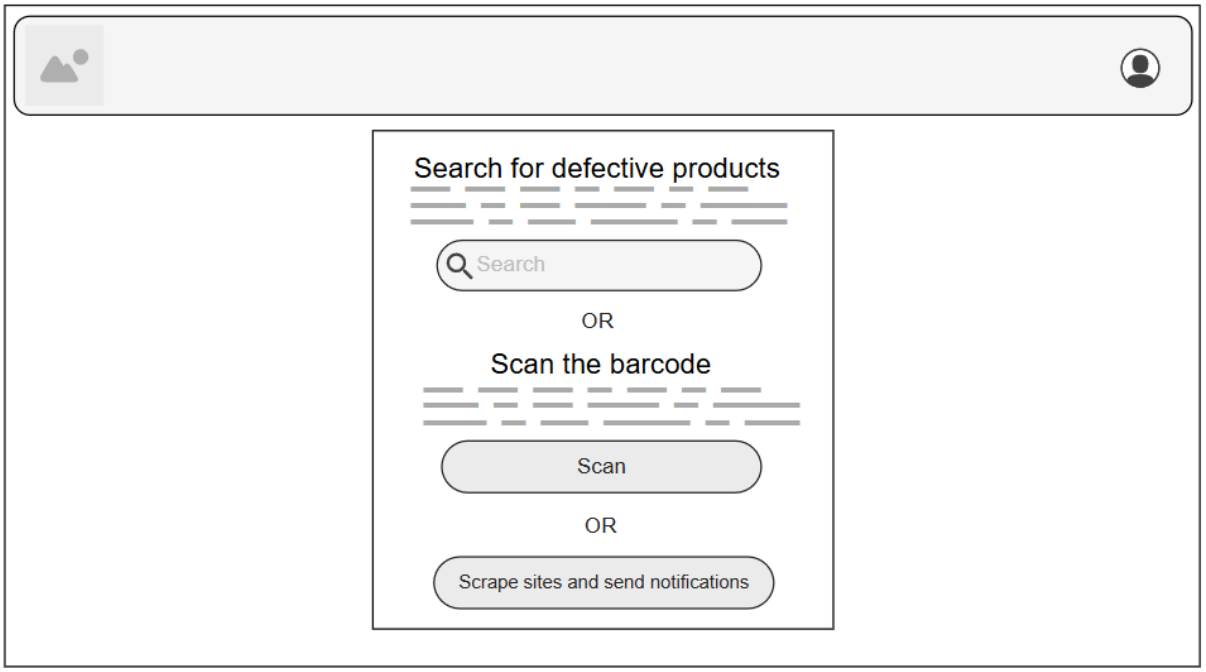
3.8.2 Personalized list page (Table view)



3.9 Scan History page

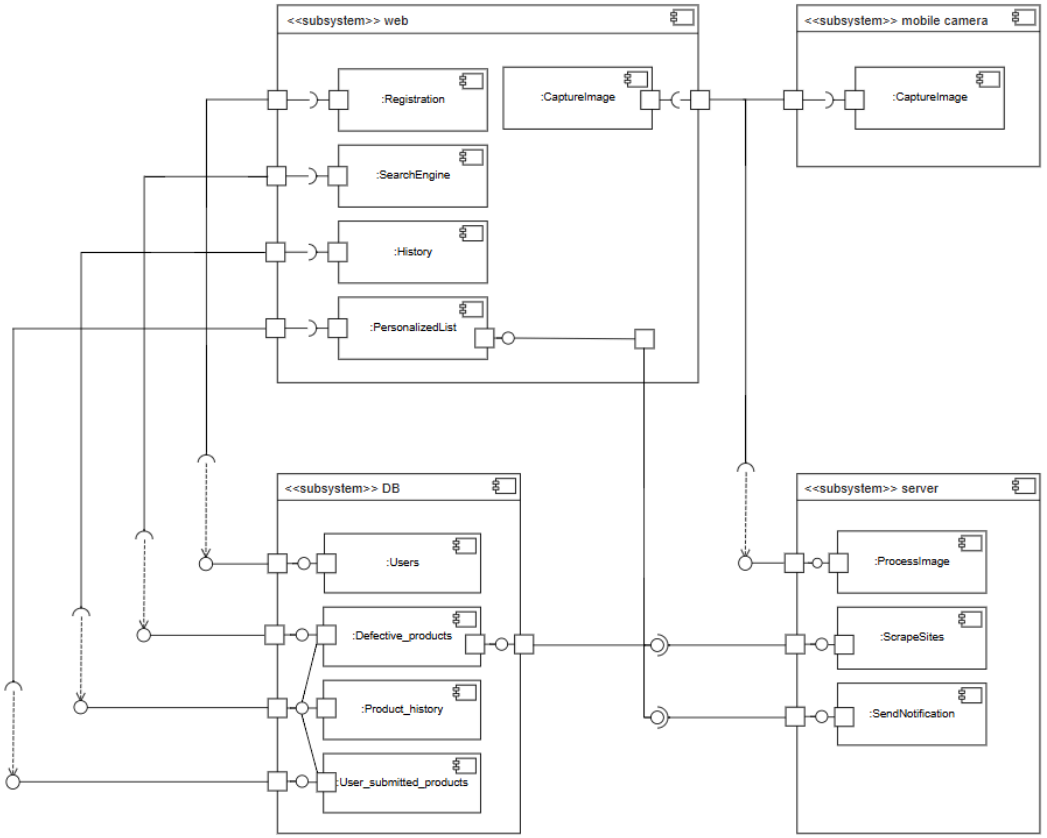


3.10 Homepage of an admin user



4. Draft of implementation

4.1 Component diagram

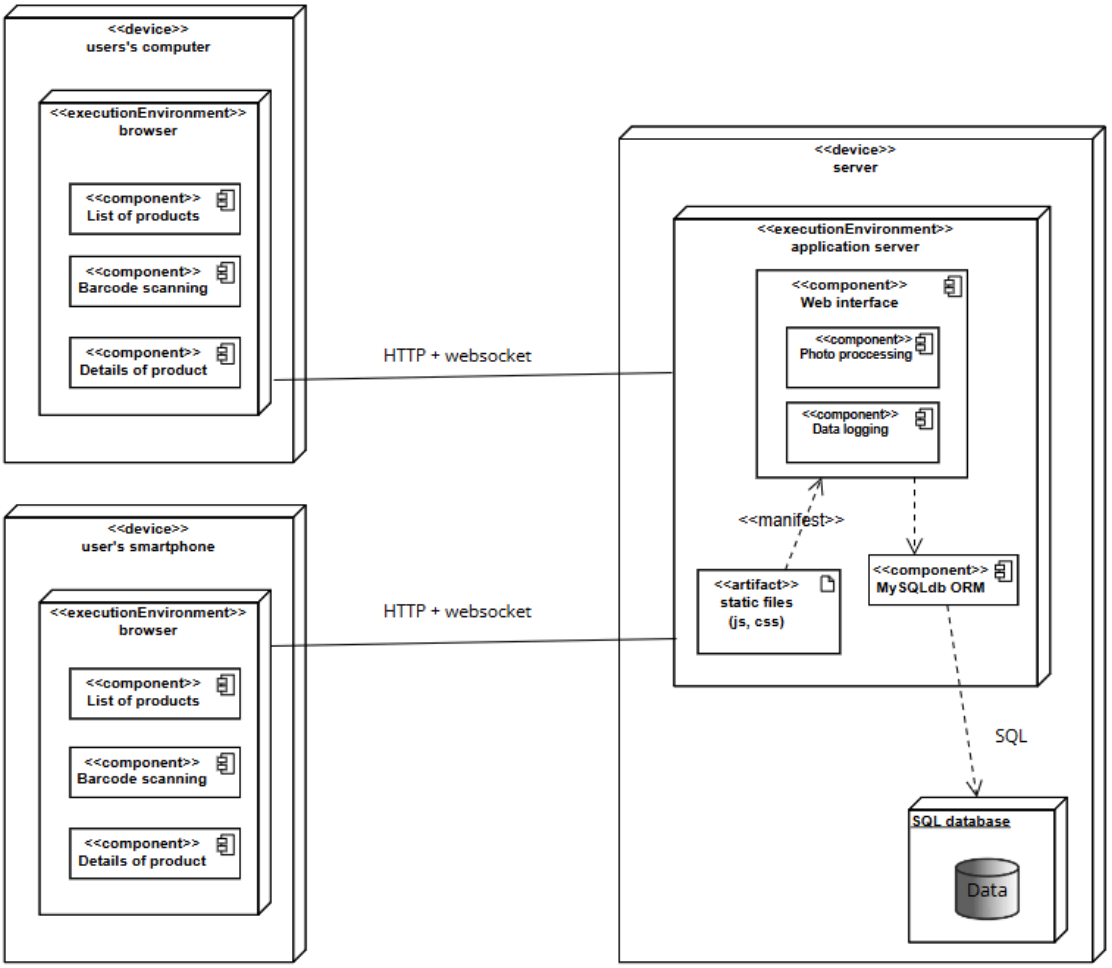


Web Subsystem: Houses components such as *Registration*, *SearchEngine*, *History*, and *PersonalizedList*, representing the main features available in a standard browser environment. It also includes a *CaptureImage* component for barcode scanning via a desktop webcam.
Mobile Camera Subsystem: Contains a dedicated *CaptureImage* component to handle image capture (e.g., barcode scanning) from a smartphone's camera.
Server Subsystem: Encompasses key backend processes:

- **ProcessImage**: Handles the logic for analyzing and interpreting captured images or barcodes.
- **ScrapeSites**: Retrieves product data from external sources and updates the database.
- **SendNotification**: Sends email or other alerts when products in a user's personalized list become defective.

DB Subsystem: Stores data in tables such as *Users*, *Defective_products*, *Product_history*, and *User_submitted_products*, maintaining all user info, product details, scan history, and user‑generated entries.

5. Deployment



1. **User Devices (Computer or Smartphone)**
    - Runs a **browser** environment that hosts three main UI components: *List of Products*, *Barcode Scanning*, and *Product Details*.
    - Communicates with the server via **HTTP** and **WebSockets**.
2. **Application Server**
    - Hosts the **Web Interface**, which includes *Photo Processing* (for scanning) and *Data Logging*.
    - Serves **static files** (JavaScript, CSS) and uses a **MySQL ORM** component to interact with the database.
3. **SQL Database**
    - Stores persistent product, user, and scan data.
    - The ORM on the server side handles queries and updates.

All client-side requests (including barcode scanning) flow to the server over HTTP/WebSocket connections; the server then processes data and performs database operations as needed.

6. Technologies used

In this project, we will be utilizing a combination of these specific technologies and frameworks:

1. PHP - Powers the server‑side logic, handles requests from the client, and manages interactions with the database.
2. HTML, CSS, JavaScript - Form the foundation of the front‑end interface, rendering pages, styles, and running interactive features (like barcode scanning) in the browser.
3. MySQL - Serves as the relational database for storing product details, user data, and scan histories.
4. Python - Used primarily for scraping tasks and data processing, running scripts that fetch product information from external websites.
5. BeautifulSoup4 (Python Lib) - Facilitates HTML parsing and data extraction from web pages (e.g., for finding product information within scraped content).
6. Selenium (Python Lib) - Automates browser actions for dynamic sites that require JavaScript execution, ensuring complete, accurate scraping of product data.

7. Plan of Implementation

1. **Set Up Project & Basic Homepage**
    - Initialize the project structure (folders, initial codebase).
    - Implement a simple landing or "main" page to ensure basic UI frameworks (HTML/CSS/JS) are in place.
2. **Design & Build the Database**
    - Define tables for products, users, personalized lists, etc.
    - Establish relationships and any necessary indexing.
3. **Implement Scraping Service**
    - Develop Python scripts (or equivalent) to scrape the two target sites (SK, EU).
    - Store results in the database.
    - (Optional) Provide a command‑line or minimal UI trigger to confirm it works.
4. **Create Welcome Page & User Management**
    - Implement registration (email + Google) and login flows.
    - Set up roles: admin vs. regular user.
    - Provide a simple "profile" or "welcome" screen after login.
5. **Add Admin Functionality to Trigger Scraping**
    - Give admins a UI button or dashboard section to manually kick off scraping.
    - Show feedback for scraping progress or results.
6. **Implement Personalized List & History**
    - Allow registered users to create a personalized list of products.
    - Track scanning history so users can view products they've scanned.
    - Basic add/remove functionality for products on the list.
7. **Develop Product Search & Product Details Pages**
    - Let guests and registered users search for products by name.
    - Show product details (including defect status) on a dedicated "product info" page.
    - Incorporate any relevant filters or sorting.
8. **Integrate Camera Access & Barcode Scanning**
    - Implement code for requesting camera permissions (webcam/mobile).
    - Use a suitable barcode scanning library for real‑time detection.

    - Handle edge cases (denied permission, invalid format, etc.).
9. **Tie Scanning to Personalized List & Notifications**
    - On successful scan of a *non-defective* product, give the option to add it to a personalized list.
    - If a product later becomes defective (via scraping updates), trigger notification emails to the user.
10. **Final Testing & Quality Assurance**

- Conduct end-to-end testing on both desktop and mobile devices.
- Verify all requirements: scanning, searching, personalized lists, scraping, etc.
- Gather feedback, fix bugs, and refine the user experience.