Makeup Product Recommendation System

Using Arrays and Bitwise Operators in C Language

Made by:
Alisha Naushad CT-169
Aimen Aslam CT-152
Syeda Zainab CT-171

The Problem

Consumers struggle to identify suitable product shades. The purchasing decision is confusing, especially without professional knowledge.

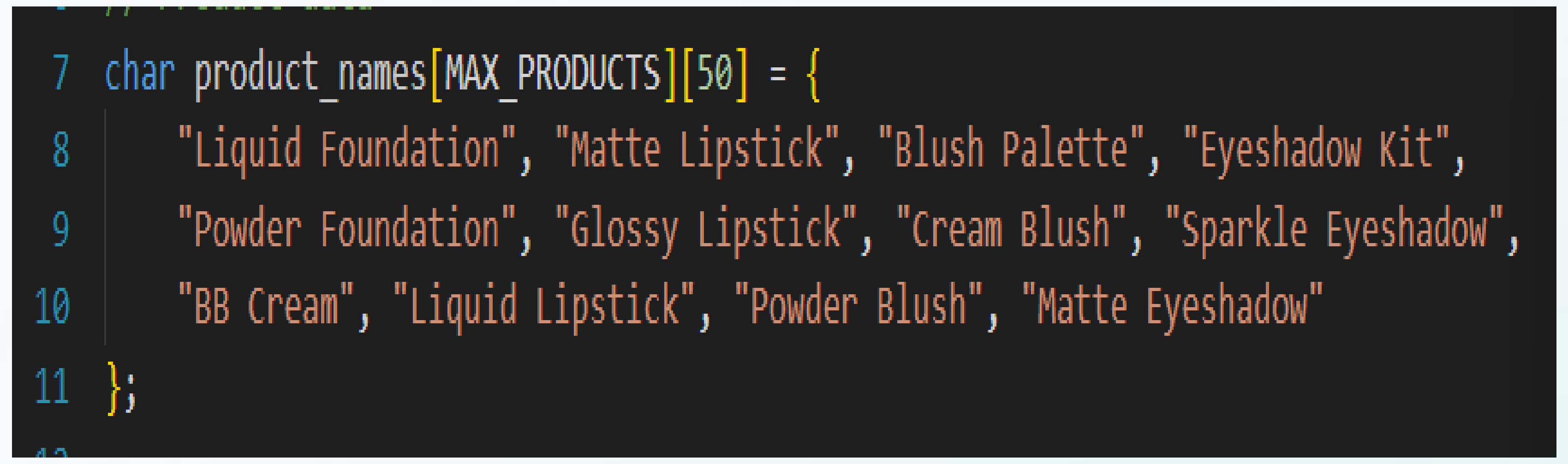
Why is this a Problem?

- **Expert Advice Required:** Traditional selection relies on beauticians or makeup artists.
- **Trial-and-Error:** Users must physically test multiple products to find a match.
- **Physical Cost & Time Wastage:** Buying multiple shades to test is expensive and inefficient.
- **Emotional Dissatisfaction:** A poor match leads to an unnatural look and frustration.

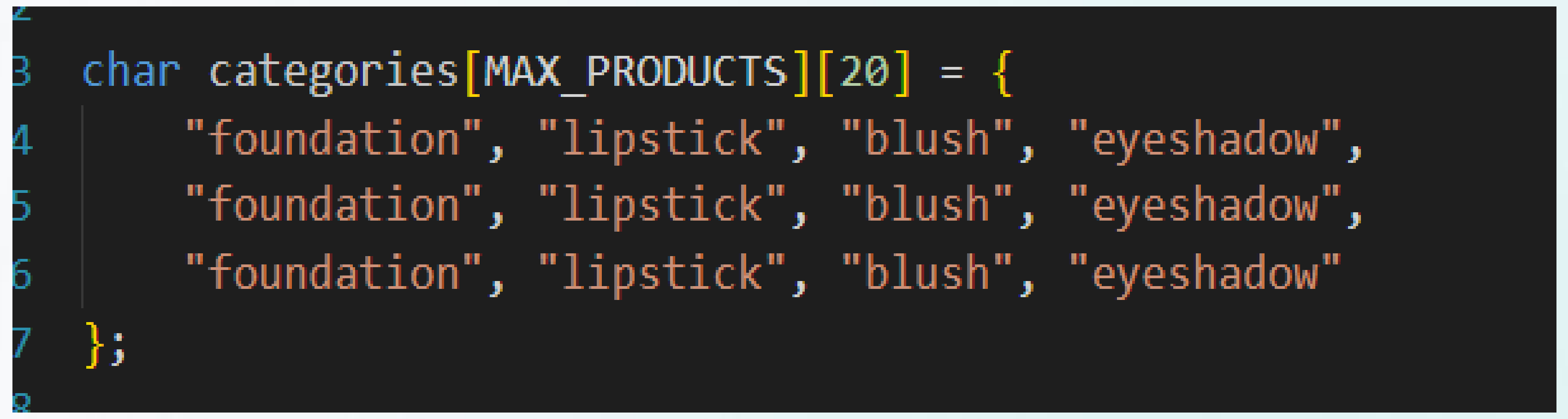
The Solution: Algorithmic Support

This project transforms subjective human judgment into algorithmic decision support. It demonstrates how simple data structures and logic can solve a practical, real-world problem.

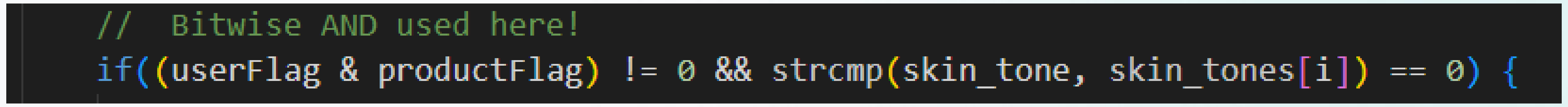
- **Structured Data Storage:** Using Arrays to hold all product information.
- **Conditional Evaluation:** Using `if` statements and user input to filter data.
- **Bitwise Logic:** A complex, efficient, and scalable method for category matching.



```c
char product_names[MAX_PRODUCTS][50] = {
    "Liquid Foundation", "Matte Lipstick", "Blush Palette", "Eyeshadow Kit",
    "Powder Foundation", "Glossy Lipstick", "Cream Blush", "Sparkle Eyeshadow",
    "BB Cream", "Liquid Lipstick", "Powder Blush", "Matte Eyeshadow"
};
```



```c
char categories[MAX_PRODUCTS][20] = {
    "foundation", "lipstick", "blush", "eyeshadow",
    "foundation", "lipstick", "blush", "eyeshadow",
    "foundation", "lipstick", "blush", "eyeshadow"
};
```



```c
//  Bitwise AND used here!
if((userFlag & productFlag) != 0 && strcmp(skin_tone, skin_tones[i]) == 0) {
```

Core Technical Concepts

Arrays

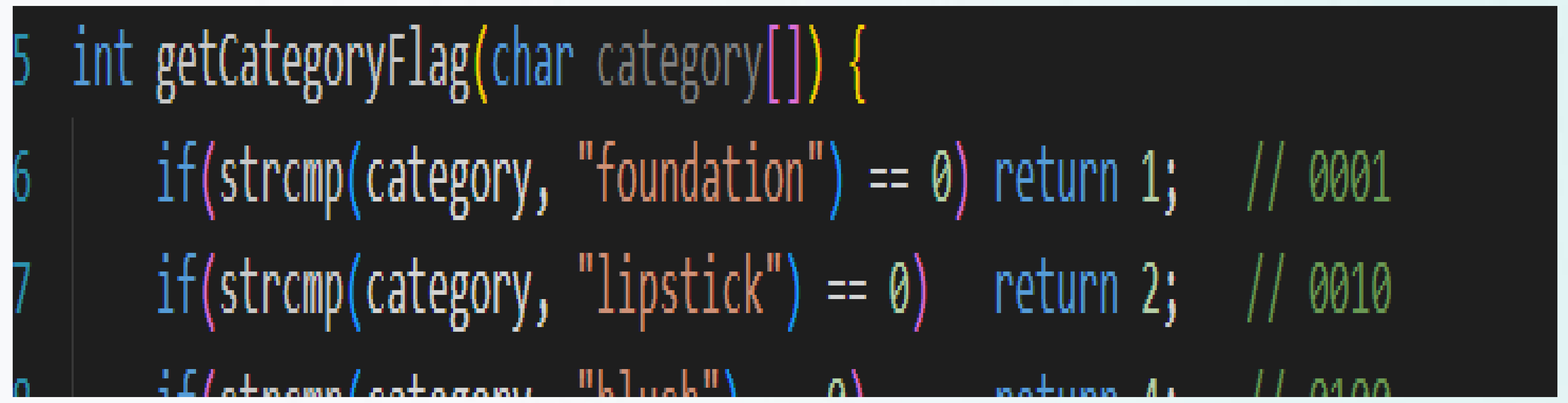
Stores 12+ product names, categories, shades, and tones. Allows for bulk storage and fast index-based access.

Functions

Creates modular, reusable code. Key functions: `display_menu()`, `get_user_preferences()`, `find_recommendations()`.

Logic & Input

Conditional logic (if/else) and user input (scanf) create a dynamic, interactive system that filters results in real-time.



```
int getCategoryFlag(char category[]) {
    if(strcmp(category, "foundation") == 0) return 1;   // 0001
    if(strcmp(category, "lipstick") == 0)   return 2;   // 0010
    if(strcmp(category, "blush") == 0)      return 4;   // 0100
```

The Complex Part: Bitwise Operators

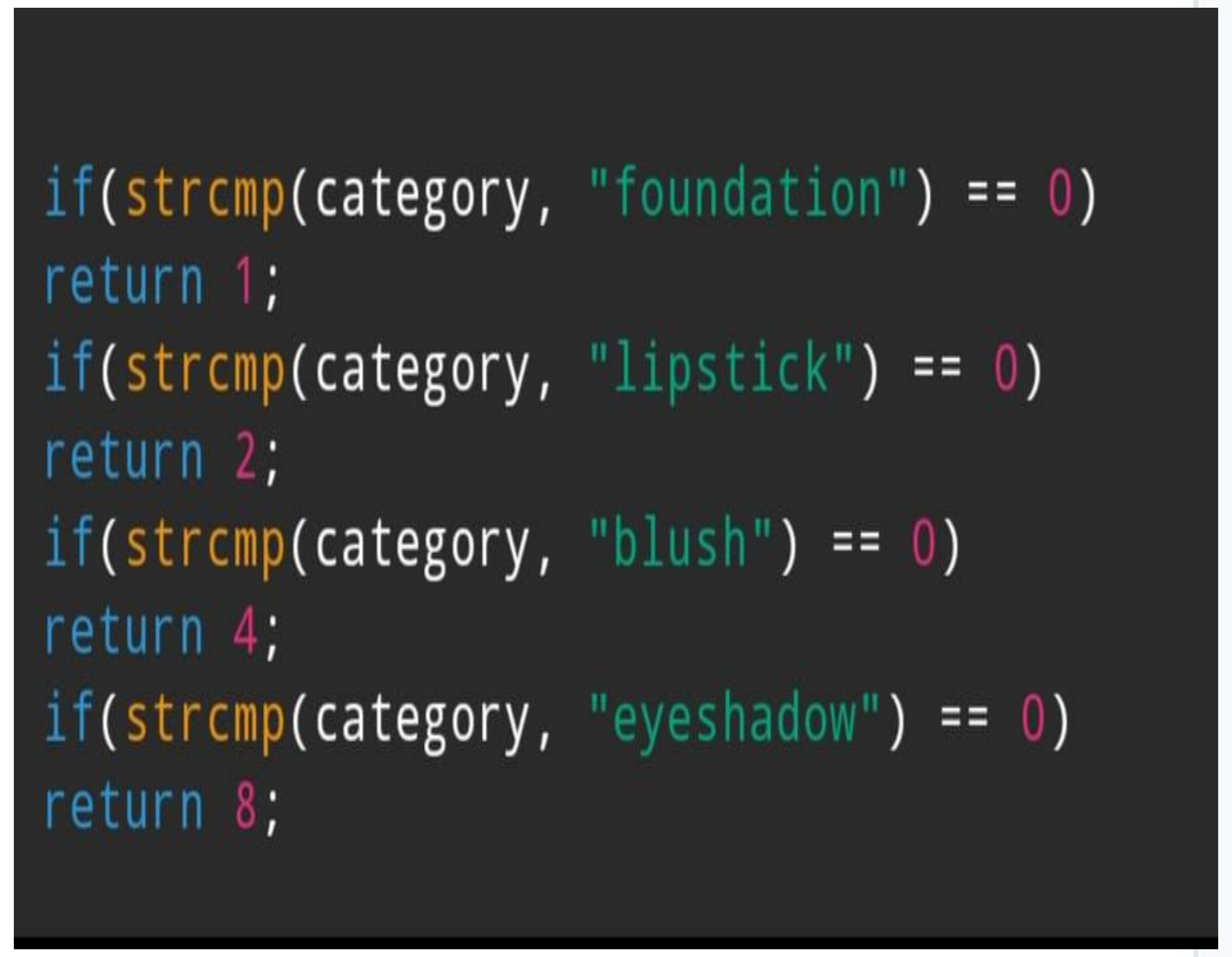
This is the core of the "Complex Computing Problem." It goes beyond simple string matching and uses advanced computing logic.

How Bitwise Operators Work

The "Flag" System

Each category is assigned a unique power-of-two, which represents a single "on" bit in binary.

- `Foundation` : 1 (Binary: `0001`)
- `Lipstick` : 2 (Binary: `0010`)
- `Blush` : 4 (Binary: `0100`)
- `Eyeshadow` : 8 (Binary: `1000`)



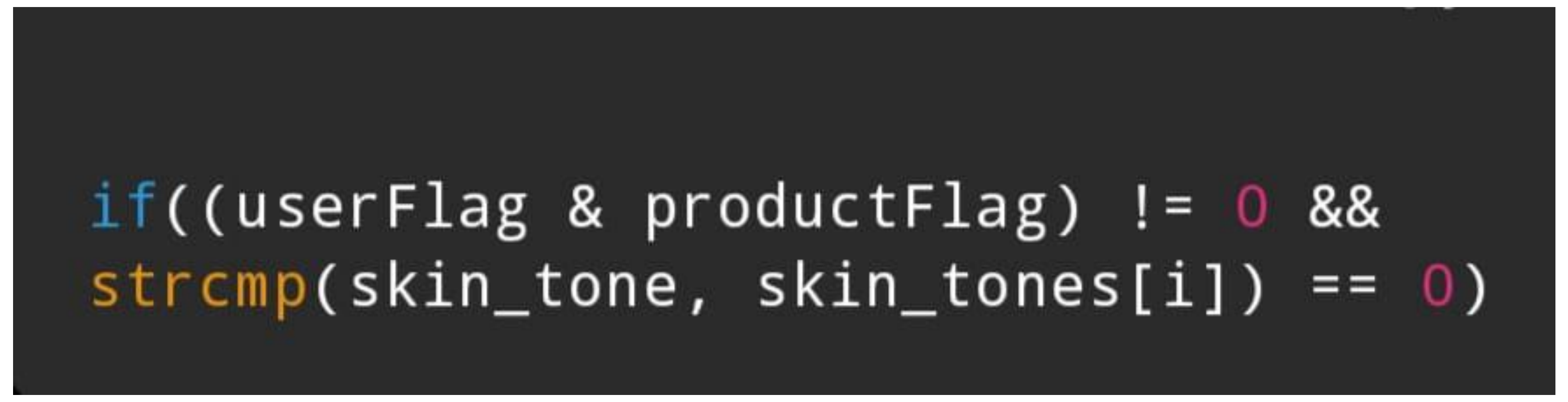
```
if(strcmp(category, "foundation") == 0)
return 1;
if(strcmp(category, "lipstick") == 0)
return 2;
if(strcmp(category, "blush") == 0)
return 4;
if(strcmp(category, "eyeshadow") == 0)
return 8;
```

The Logic

We check if the user's flag and the product's flag have a match using the bitwise AND (`&`) operator.

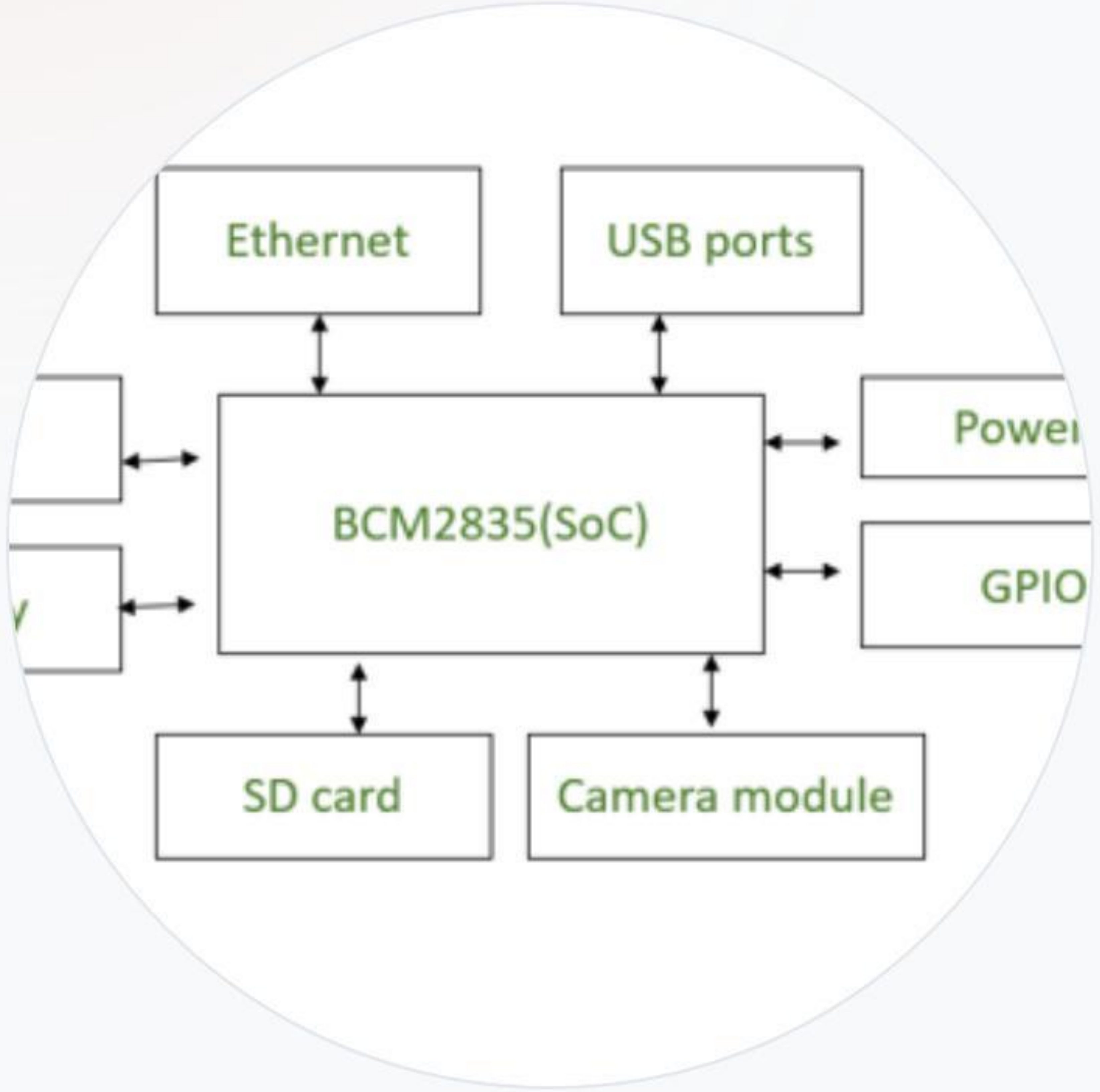
```
if (userFlag & productFlag) != 0
```

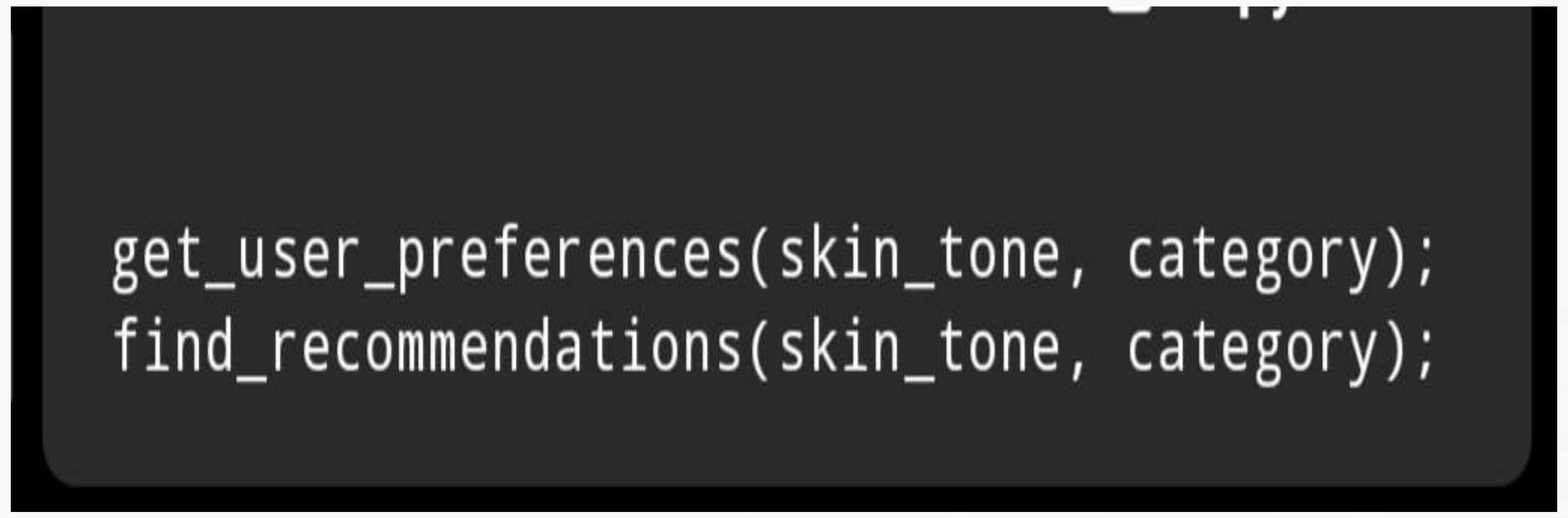
This logic is faster, more scalable, and more professional than multiple `if` statements.



```
if((userFlag & productFlag) != 0 &&
strcmp(skin_tone, skin_tones[i]) == 0)
```

System Architecture





```
get_user_preferences(skin_tone, category);
find_recommendations(skin_tone, category);
```

Layered Logic

The system's architecture is simple and layered for clear data flow.

1. **Input Module:** Takes user preferences.
2. **Data Storage:** Arrays hold all product data.
3. **Processing Module:** Bitwise filtering & string comparison.
4. **Output Module:** Displays recommended items.

Built-In product list

In-Memory Catalog

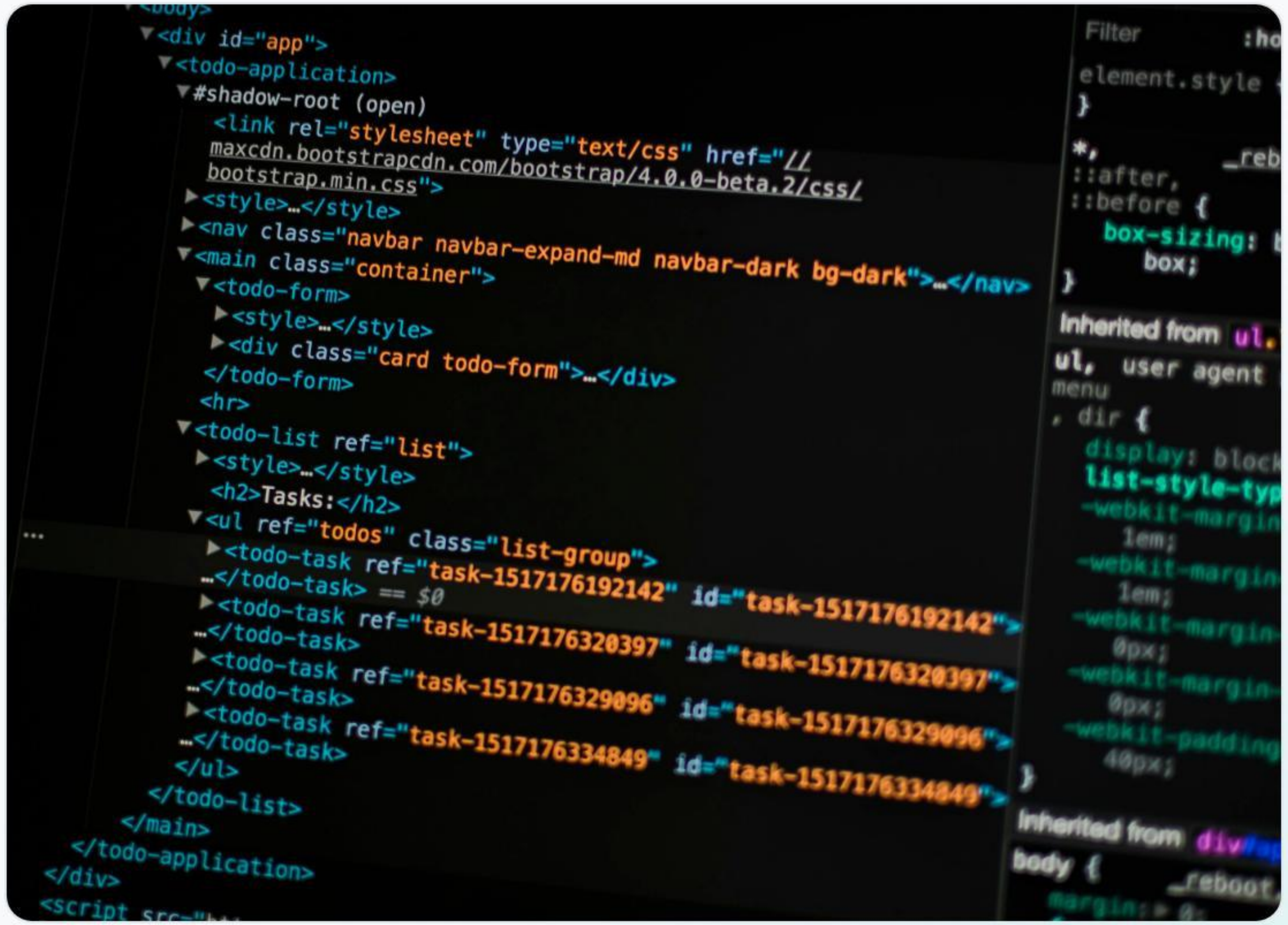
The dataset is statically embedded inside the C code. This makes the program self-contained.

Four parallel arrays store all the data:

- `char product_names[]`
- `char categories[]`
- `char shades[]`
- `char skin_tones[]`

```c
char product_names[MAX_PRODUCTS][50] =
{...};
char categories[MAX_PRODUCTS][20] = {...};
char skin_tones[MAX_PRODUCTS][20] = {...};
```

This acts as an efficient, in-memory database for the program.



```html
▼<div id="app">
  ▼<todo-application>
    ▼#shadow-root (open)
      <link rel="stylesheet" type="text/css" href="//maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta.2/css/bootstrap.min.css">
      ▶<style>…</style>
      ▶<nav class="navbar navbar-expand-md navbar-dark bg-dark">…</nav>
      ▼<main class="container">
        ▼<todo-form>
          ▶<style>…</style>
          ▶<div class="card todo-form">…</div>
        </todo-form>
        <hr>
        ▼<todo-list ref="list">
          ▶<style>…</style>
          <h2>Tasks:</h2>
          ▼<ul ref="todos" class="list-group">
            ▶<todo-task ref="task-1517176192142" id="task-1517176192142">
            …</todo-task> == $0
            ▶<todo-task ref="task-1517176320397" id="task-1517176320397">
            …</todo-task>
            ▶<todo-task ref="task-1517176329096" id="task-1517176329096">
            …</todo-task>
            ▶<todo-task ref="task-1517176334849" id="task-1517176334849">
            …</todo-task>
          </ul>
        </todo-list>
      </main>
  </todo-application>
</div>
```

Program Flow (do-while loop)

```c
do {
    display_menu();
    scanf("%d", &choice);
} while(choice != 3);
```

1. display_menu()

Shows options 1, 2, 3 to the user.

2. User Selects

User inputs a choice (e.g., "1").

3. switch(choice)

Program executes the correct functions, like `find_recommendations()`.

4. Loop Repeats

The menu is displayed again until user presses "3" (Exit).

Code Demonstration

A look at the key C functions that power the system.

Code: Key Functions Explained

getCategoryFlag()

Converts the user's string input into its integer bitwise flag.

```
if(strcmp(cat, "lipstick") == 0)
  return 2;
```

find_recommendations()

The main loop that iterates through all products.

```
int userFlag = getCategoryFlag(...);
for(int i = 0; i < product_count; i++) {
  ...
}
```

The Match Logic

The core conditional check combining both logic types.

```
if((userFlag &
productFlag) != 0
&& strcmp(skin,
skin_tones[i]) ==
0)
{ ... // Match
Found!
```

System Results: In Action

Scenario 1: Match Found

Input: Skin Tone = Fair, Category = Foundation

```
--- RECOMMENDATIONS ---
✅ Liquid Foundation
  Shade: Light Beige

✅ BB Cream
  Shade: Fair Ivory
```

Scenario 4: No Match Found

Input: Skin Tone = Dark, Category = Foundation

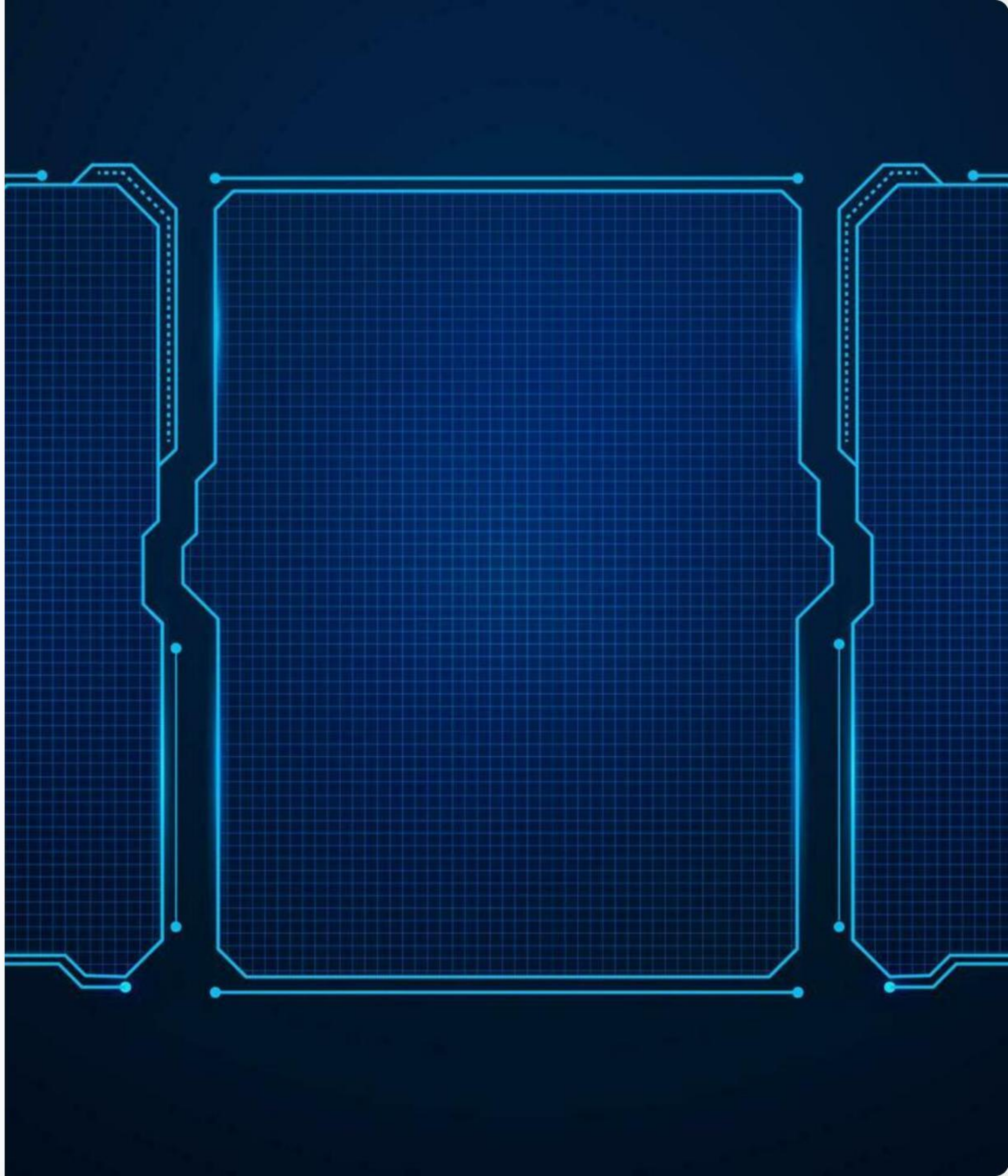
```
--- RECOMMENDATIONS ---
❌ No matching products found.
```

This demonstrates robust handling for cases where no product in the dataset matches the user's query.

Future Work

The current system is a strong foundation. Future versions could be expanded to enhance usability and commercial relevance.

- ✓ Expanding the dataset with more products
- ✓ Developing a Graphical User Interface (GUI)
- ✓ Implementing advanced filtering (price, brand)
- ✓ Integrating a real external database (e.g., SQL)
- ✓ Deploying as a web or mobile application



Conclusion

"This project transforms the subjective decision-making process in cosmetic selection into a reproducible, computationally efficient system."

Key Takeaways:

Successfully used Arrays for data storage.

Proved Bitwise Operators are efficient for filtering.

Created a practical, user-centered application.