SkinSync

Kayla Thompson

SkinSync

A secure client-server application that helps users explore skincare and makeup products for all skin types, tone, and acne concerns.

➔ **Socket Programming**
Establishes an encrypted communication between clients and a server that stores all SkinSync Data.

➔ **System supports**
User Authentication, encryption, and personalized product recommendations.

SkinSync Mission & Goals

Our mission and goals at SkinSync hope to foster an inclusive environment by creating a secure, educational, confident, and transparent skincare program that empowers women to make informed beauty and skincare choices.

Educational:
- Teaching users about ingredients, wash routines, and product management!

Security-driven:
- Promotes and achieves secure data transmission and user authentication!

Inclusive:
- Supporting all skin types, tones, and acne severity levels!

Why SkinSync

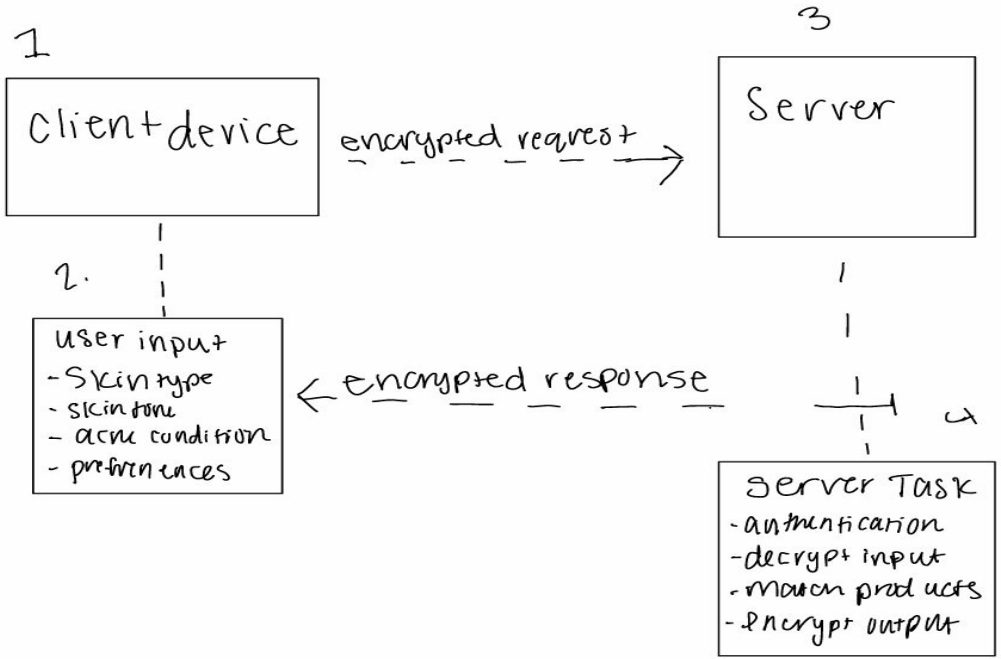
Most Online platform programs do not prioritize data security, leading to customer data exploitation and potential breaches.

The urgency for a secure and intelligent program that helps users explore skincare and makeup products customized to their needs while protecting their data through encrypted communication and authenticated access.

Example

Brands like Ulta and Sephora have programs like this, but it is based off marketing rather than effectiveness.

General Function



Encryption

```python
# ---------------- Encryption ---------------- #
# need help with encryption dont know what im doing
def generate_or_load_key():
    """
    Generate a new Fernet key if not present; otherwise load existing.
    """
    if not os.path.exists(KEY_FILE):
        key = Fernet.generate_key()
        with open(KEY_FILE, "wb") as f:
            f.write(key)
        print("Encryption key generated and saved to key.key (copy this to the client).")
    else:
        with open(KEY_FILE, "rb") as f:
            key = f.read()
    return key

def encrypt(cipher, message: str) -> bytes:
    return cipher.encrypt(message.encode())

def decrypt(cipher, token: bytes) -> str:
    return cipher.decrypt(token).decode()

```

Fernet

The same key is used to encrypt and decrypt and authenticated encryption. Secure communication and protects sensitive data. Prevents attackers from intercepting login success or failure.

Data and Storage

```
34
35  # ---------------- Data storage ---------------- #
36
37  def load_users():
38      if not os.path.exists(USER_DB):
39          return {}
40      try:
41          with open(USER_DB, "r") as f:
42              return json.load(f)
43      except json.JSONDecodeError:
44          return {}
45
46  def save_users(users):
47      with open(USER_DB, "w") as f:
48          json.dump(users, f, indent=2)
49
50  # ---------------- Skin types and recommendations ---------------- #
51
52  skin_types_definitions = {
53      "oily": "Excess sebum, shiny, and more likely to develop acne.",
54      "dry": "Tight, flaky, and low moisture; can feel rough or dull.",
55      "combination": "Oily T-zone (forehead, nose, chin) with dry or normal cheeks.",
56      "sensitive": "Easily irritated, redness or stinging with fragrances or actives.",
57      "normal": "Balanced sebum and hydration; minimal visible concerns."
58  }
```

JSON

Users.json is simple and automatically sees usernames, passwords, etc. server verifies credentials. Supports login and personalization. I didn't encrypt users.json for simplicity, but real-world would use hash.

Formatting

```
# ---------------- Formatting ---------------- #

def banner():
    return (
        "****************************************\n"
        "***            SkinSync                ***\n"
        "****************************************\n"
    )

def build_skin_type_menu():
    lines = []
    lines.append("Please select your skin type (1-5):")
    lines.append("1. Oily - " + skin_types_definitions["oily"])
    lines.append("2. Dry - " + skin_types_definitions["dry"])
    lines.append("3. Combination - " + skin_types_definitions["combination"])
    lines.append("4. Sensitive - " + skin_types_definitions["sensitive"])
    lines.append("5. Normal - " + skin_types_definitions["normal"])
    return "\n".join(lines)

def format_recommendations(skin_type_key):
    if skin_type_key not in recommendations:
        return "Invalid skin type selection."
    data = recommendations[skin_type_key]
    output = []
```

Readability for users, clear section headers, consistency, technical simplicity, and professional appeal. It's like plating food!

The SkinSync Data

```
# ---------------- Skin types and recommendations ---------------- #

skin_types_definitions = {
    "oily": "Excess sebum, shiny, and more likely to develop acne.",
    "dry": "Tight, flaky, and low moisture; can feel rough or dull.",
    "combination": "Oily T-zone (forehead, nose, chin) with dry or normal cheeks.",
    "sensitive": "Easily irritated, redness or stinging with fragrances or actives.",
    "normal": "Balanced sebum and hydration; minimal visible concerns."
}

recommendations = {
    "oily": {
        "general": "Oil free moisturizers/non-comedogenic and salicylic acid cleanser",
        "products": {
            "Cleansers": ["CeraVe foaming cleanser", "Skinmedica face cleanser", "CosRX gel cleanser"],
            "Face wash": ["Neutrogena oil free acne face wash", "Cetaphil oil removing face wash"],
            "Face mask": ["Paula's choice BHA liquid exfoliant", "La Roche Posay shine control face mask"],
            "Serums": ["The Ordinary niacinamide oil control serum", "La Roche Posay hyaluronic acid", "The Inkey List niacinamide serum"],
            "Moisturizers": ["Tacha water cream", "Shani Darden hydration moisturizer", "Dr Jart ceramidin skin barrier moisturizer"],
            "Sunscreen": ["Neutrogena oil free SPF 50", "Aveeno protect and hydrate"],
            "Sunscreen for darker skin": ["Black Girl Sunscreen", "Supergoop unseen SPF 40"]
        },
        "makeup": ["Nars oily skin foundation/concealer", "IT Cosmetics", "Estee Lauder", "Bare Minerals"]
    },
```

Authentication

```python
# ---------------- Authentication ---------------- #

def authenticate(users, username, password):
    return username in users and users[username]["password"] == password

def create_account(users, username, password):
    if username in users:
        return False, "Username already exists."
    users[username] = {"password": password, "skin_type": None, "last_results": None}
    save_users(users)
    return True, "Account created."

def set_user_profile(users, username, skin_type_key, results_text):
    users[username]["skin_type"] = skin_type_key
    users[username]["last_results"] = results_text
    save_users(users)
```

Verification

Verifying a users identity. Matches it with the users.json file.

Looping

```
184
185  # ---------------- Server loop ---------------- #
186
187  def handle_client(conn, cipher):
188      users = load_users()
189
190      # Send cute banner
191      conn.send(encrypt(cipher, banner()))
192
193      conn.send(encrypt(cipher, "Do you have an account? (yes/no): "))
194      account_status = decrypt(cipher, conn.recv(4096)).strip().lower()
195
196      # login or create account
197      if account_status == "yes":
198          conn.send(encrypt(cipher, "Username: "))
199          username = decrypt(cipher, conn.recv(4096)).strip()
200          conn.send(encrypt(cipher, "Password: "))
201          password = decrypt(cipher, conn.recv(4096)).strip()
202
```

I used lists inside dictionaries, loop lets me display each item clearly. No constant print statements. Handling multiple users. Different menu options when showing the skin type options. Automatically formats the menu and easy to add new skin types.

References

Alvarez, G. V., Kang, B. Y., Richmond, A. M., Hoss, E., Sulewski, R., Minkis, K., Rozenberg, S. S., Antonovich, D., Boucher, A., Bernstein, E. F., Bertucci, V., Chapas, A. M., Cohen, J. L., Council, M. L., Dover, J. S., Geronemus, R., Given, K. M. L., Goldbach, H. S., Goldman, M. P., Hooper, D., Kaufman, J., Munavalli, G., Pacheco, T. R., Rossi, A. M., Wilson, S., & Alam, M. (2025). Skincare ingredients recommended by cosmetic dermatologists: A Delphi consensus study. *Journal of the American Academy of Dermatology, 93*(6), 1509–1525. Skincare ingredients recommended by cosmetic dermatologists: A Delphi consensus study - PubMed

CeraVe. (n.d.). *CeraVe: Skincare developed with dermatologists*. Retrieved December 2, 2025, from https://www.cerave.com/

La Roche-Posay. (n.d.). *La Roche-Posay: Dermatologist recommended skincare*. Retrieved December 2, 2025, from https://www.laroche-posay.us/

Neutrogena. (n.d.). *Neutrogena: Skincare products for healthier skin*. Retrieved December 2, 2025, from https://www.neutrogena.com/

Northwestern University. (2025, July 8). *Best skin care ingredients revealed in thorough, national review*. Northwestern Now. https://news.northwestern.edu/stories/2025/07/best-skin-care-ingredients-revealed-in-thorough-national-review?fj=1