Final Project Report

AuraSkin

[AC3] AuraSkin | AI 7993 | Section W01 | Fall 2025

Kennesaw State University

Website | GitHub

Prepared By: Nyah Robinson

Date: December 8, 2025

Table of Contents

1. Introduction

1.1 Purpose

AuraSkin is an AI-powered mobile prototype designed to classify images of human skin into three categories, acne, dark spots, and normal skin, and provide personalized skincare recommendations. The purpose of the project is to demonstrate a complete applied AI development pipeline, including dataset preparation, model training, backend design, mobile app implementation, UI design, system integration, and documentation. AuraSkin represents a realistic scenario in which machine learning is embedded into a consumer-facing application.

1.2 Motivation

The motivation behind AuraSkin comes from the increasing desire for accessible skincare insights. Many individuals struggle with acne or hyperpigmentation but lack access to dermatologists or trustworthy tools to guide their routines. By leveraging lightweight deep learning models and mobile development frameworks, AuraSkin provides an accessible method to obtain skin assessments quickly and efficiently.

AuraSkin was motivated by the question:

> "What would it look like if a user could get a basic skin assessment simply by taking a picture on their phone and letting an AI model help guide them?"

From a technical perspective, the project is also a synthesis of skills learned across the program:

- Training and evaluating CNNs on real image data
- Exposing trained models via a web API
- Building a mobile UI that talks to that API

The goal of this report is to document the full engineering process.

2. Requirements

This section summarizes the functional and non-functional requirements necessary for AuraSkin. The requirements were refined throughout the project and aligned with the project's scope as a working prototype.

2.1 Functional Requirements

AuraSkin must support the essential functions of a mobile AI classifier:

1. **Image Selection**
    - The user must be able to upload a skin image from the device's gallery. This is the primary method used during testing because the iOS simulator restricts camera functionality.

2. **Image Transmission to Backend**
    - The Flutter application must send the selected image to the FastAPI backend using an HTTP multipart request.

3. **Model-Based Classification**
    - The backend must preprocess the image, run inference using the MobileNetV2 model, and determine whether the image depicts acne, dark spots, or normal skin.

4. **Return of Analysis Results**
    - The backend must respond with a JSON object containing:
        - Predicted label
        - Confidence score
        - A set of personalized skincare recommendations

5. **User Interface Display**
    - The Flutter app must parse the backend's response and present results clearly to the user on a dedicated results screen.

6. **Error Handling**
    - If the user fails to select an image or the backend is offline, the application must show a meaningful error instead of crashing.

2.2 Non-Functional Requirements

To ensure reliability, usability, and maintainability, AuraSkin must also satisfy:

- **Performance:** Inference must complete within 1-3 seconds on local hardware.
- **Accuracy:** The model should achieve at least 85% classification accuracy. AuraSkin's final model achieved approximately 88% validation accuracy.
- **Usability:** The UI must be intuitive and aligned with the AuraSkin brand identity (tan and lavender theme).
- **Privacy:** No images should be stored. Processing is transient.
- **Portability:** The system must operate on macOS using the iOS simulator and local backend.
- **Maintainability:** Backend, frontend, and model must be modular and documented.

3. Analysis

The primary problem AuraSkin addresses is the lack of accessible, affordable, and personalized skincare tools. While dermatologists provide expert assessments, many users turn to unreliable online sources or guesswork. Advances in deep learning allow for basic image classification models to offer meaningful analysis, even with relatively small datasets.

AuraSkin is positioned as an early-stage prototype rather than a clinical diagnostic tool. Its purpose is to demonstrate how machine learning can meaningfully augment user decision-making about skincare while also showcasing a fully implemented end-to-end AI system. The system's feasibility was evaluated across technical and practical dimensions:

- **Model Feasibility:** MobileNetV2 is lightweight enough for efficient inference and ideal for mobile-centered applications.
- **Backend Feasibility:** FastAPI provides a fast, modern, and simple interface for serving machine learning predictions.
- **Frontend Feasibility:** Flutter enables cross-platform development with a single codebase and offers extensive UI customization capabilities.

Overall, AuraSkin's architecture is well-suited for the constraints of dataset size, timeline, and prototype-level functionality.

4. System Design

AuraSkin is designed as a modular system consisting of a machine learning model, a backend inference server, and a mobile user interface. The separation of these components allows each to be developed and tested independently while still supporting integration into a cohesive product.

4.1 System Architecture

The architecture follows a straightforward request-response cycle:

> Flutter App → FastAPI Server → MobileNetV2 Model → JSON Response → Flutter Results Screen

This design ensures:

- The **frontend** handles user interaction and display.
- The **backend** performs image preprocessing and inference.
- The **model** runs efficiently and consistently with training procedures.

4.2 Machine Learning Model Design

The core of AuraSkin's intelligence is a MobileNetV2 model fine-tuned on a dataset of 300 labeled skin images. MobileNetV2 was selected for its balance of accuracy and computational efficiency. The model architecture includes:

- Pretrained convolutional layers (frozen during training)
- GlobalAveragePooling2D
- Dense(128, ReLU)
- Dense(3, softmax output)

The model was trained for approximately 20-30 epochs using augmented data to reduce overfitting. The final validation accuracy of around 88% demonstrates strong performance given the dataset size.

4.3 Backend Design (FastAPI)

The backend's responsibility is to serve the trained model and expose a single endpoint: **POST /analyze.** The backend performs the following steps:

1. Reads the uploaded image file
2. Resizes and normalizes it to match model training preprocessing
3. Runs prediction using the loaded MobileNetV2 model
4. Determines the label and probability
5. Generates a set of recommendations
6. Returns results as JSON

By isolating inference in the backend, the system remains flexible and maintainable.

4.4 Frontend Design (Flutter)

The Flutter app contains a simple, polished flow:

1. **Welcome Screen –** AuraSkin logo, gradient background, brand color palette
2. **Analyze Screen –** Button to choose image, preview section, analyze action button
3. **Results Screen –** Displays classification and recommendations

The design emphasizes clarity, femininity, and accessibility. Components are arranged to guide users through the analysis without confusion.

5. Development

This section outlines exactly how the system was built and serves as a blueprint.

5.1 Dataset Preparation

A total of 300 images were collected and manually categorized into acne, dark spot, and normal classes. With the use of Python images were resized to 224x224 pixels and normalized to values between 0 and 1. Data augmentation was applied to enhance model robustness.

5.2 Model Training

Using TensorFlow/Keras, MobileNetV2 was loaded with pretrained ImageNet weights. Only the custom classification head was trained. After achieving ~ 88% accuracy, the trained model was saved as **skin_model.h5**, this file is the core of the FastAPI backend.

5.3 Backend Implementation

The backend was built using:

- **FastAPI** for routing
- **Uvicorn** as the application server

The /analyze endpoint handles image uploads and calls the model for inference. The backend returns the predicted label, confidence score, and a predefined list of skincare recommendations specific to each class.

Critical code patterns:

- img = Image.open(file.file)
- img = img.resize((224, 224)
- arr = np.array(img) / 255.0
- arr = np.expand_dims(arr, 0)
- pred = model.predict (arr) [0]

5.4 Flutter Application Development

The Flutter app was structured into screens and widgets. The **image_picker** package was used for selecting an image, and the **http** package was used to send POST requests to the backend. The UI was refined to reflect the AuraSkin palette and integrate the radial glow gradient background.

5.5 Integration

Once the backend was fully functional, the Flutter app was updated to send images to the server and display the JSON response. Debugging addressed:

- Incorrect YAML formatting
- HTTP errors when backend was not running
- Flutter layout issues (RenderFlex overflow)
- Simulator camera limitations

5.6 Recommendation Engine Implementation

Rather than building a full database-driven engine, AuraSkin uses a simple rule-based mapping inside the backend:

- For **acne**:
    - Gentle cleanser
    - Salicylic acid / benzoyl peroxide
    - Avoid harsh scrubs and heavy oils
- For **dark spots**:
    - Vitamin C serums
    - Niacinamide and azelaic acid

    - Daily broad-spectrum SPF sunscreen

- For **normal skin**:
    - Maintain gentle cleanse–moisturize–protect routine
    - SPF in the morning
    - Occasional exfoliation

These recommendations are encoded as Python dictionaries and included in the JSON response.

5.7 Tools, Frameworks, and Libraries

- **Languages:** Python, Dart
- **ML:** TensorFlow, Keras, NumPy
- **Backend:** FastAPI, Uvicorn
- **Mobile:** Flutter SDK, Dart, Xcode iOS simulator
- **Dev Tools:** VS Code, Terminal, Git, GitHub
- **Design:** Manual mockups + iterative code-based UI

5.8 Development Challenges and Resolutions

1. Backend Not Found / **uvicorn** Command Issues
    - Encountered **"zsh: command not found: uvicorn"**
    - Fixed by installing uvicorn in the correct virtual environment and activating venv before running the command.

2. Flutter **pubspec.yaml**
    - YAML is whitespace-sensitive; duplicate and mis-idented entries caused errors.
    - Fixed by removing duplicate dependencies and respecting indentation.

3. Simulator and Camera Problems
    - iOS simulator camera access is limited; attempts caused error messages and disconnects.
    - Resolution: prioritized gallery-based upload for the demo.

4. Flutter Rendering Errors
    - Occurred when columns and scroll views were nested incorrectly.
    - Fixed by:
        - Using **SingleChildScrollView** appropriately
        - Adding **Expanded/Flexible**
        - Simplifying layout on the camera/results screens.

These issues were important learning experiences in debugging mobile UI, APIs, and dev environments.

6. Testing and Evaluation

Testing focused on ensuring the workflow operated smoothly across model, backend, and UI layers.

6.1 Functional Testing

- Verified image uploads
- Confirmed successful HTTP communication
- Confirmed model returns valid JSON
- Verified correct UI display of prediction and confidence
- Confirmed recommendations mapping was accurate

6.2 Model Performance

- Validation accuracy ≈ **88%**
- Effective at distinguishing acne and dark spots
- Normal class performance dependent on lighting and texture clarity

6.3 Integration Testing

Tested the full flow from image selection to results. Confirmed Flutter renders results correctly and safely handles backend downtime.

- **Hardware:** MacBook
- **Simulator:** Xcode iOS Simulator (e.g., iPhone 16 Pro Max)
- **Backend:** Local FastAPI server running via Uvicorn
- **Tools:** Flutter debug console, FastAPI logs, manual test images.

6.4 Test Results and Bug Fixes

During the early prototype testing phase, I encountered a recurring timeout issue when the Flutter application attempted to communicate with the backend API. The simulator displayed a network error stating that the connection to **http://127.0.0.1:8000 /analyze** had timed out, and the terminal showed no incoming requests from the Flutter client. After investigating the cause, I discovered that the backend server (FastAPI) was not fully initialized due to a missing TensorFlow import and a mismatch in the model file path.

More significant bugs were:

- UI layout crashes due to layout constraints → fixed with scroll views and constraints.
- Missing or wrong JSON key names when the backend changed → updated Flutter parsing logic.
- Network-related errors when backend not started → added error handling and user messages.

After iterative testing and fixes, the system performed reliably for the scope of the prototype.

7. Version Control

The project was maintained using Git and GitHub. The repository includes:

- Flutter app (**auraskin_app/**)
- Backend API (**app_api/**)
- Model training (**model_training/**)
- Documentation (**docs/**)

Commits were made frequently with descriptive messages documenting progress and debugging steps. Version control ensured traceability and reproducibility and supported the final deliverable requirements.

8. Summary

AuraSkin successfully demonstrates a fully integrated applied AI system. The project fulfills all requirements of the AI 7993 final package by delivering:

- A trained MobileNetV2 classifier
- A FastAPI backend with real-time inference
- A Flutter mobile prototype
- A clean and accessible UI
- A fully documented system that can be reproduced

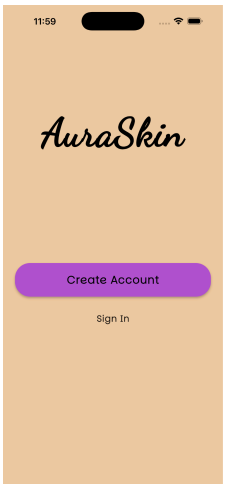
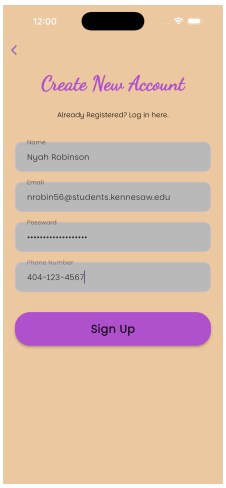
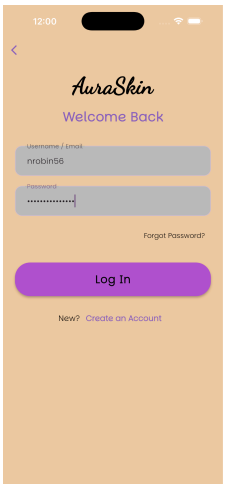
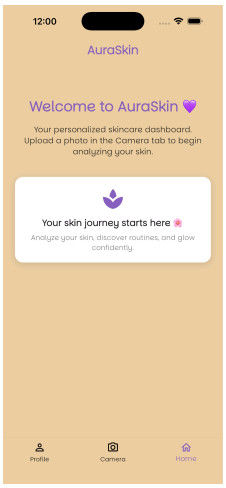
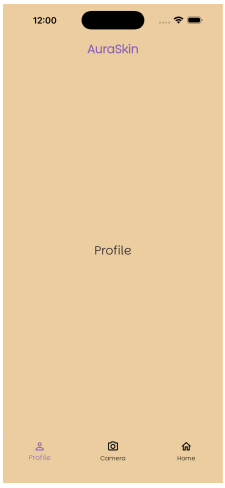
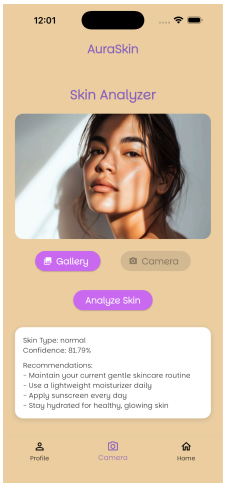
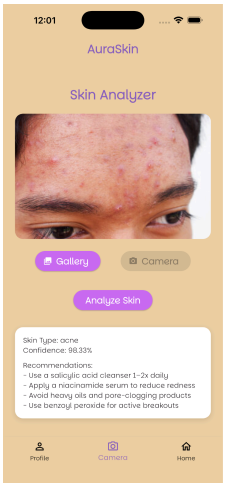
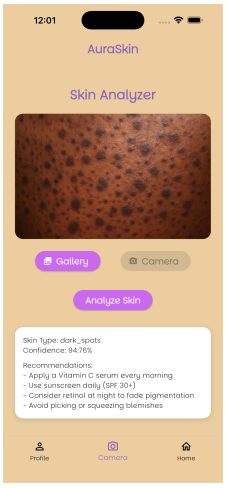
The development process also revealed opportunities for future expansion, including dataset enlargement, more sophisticated recommendations, user accounts, history tracking, and on-device inference using TensorFlow Lite.

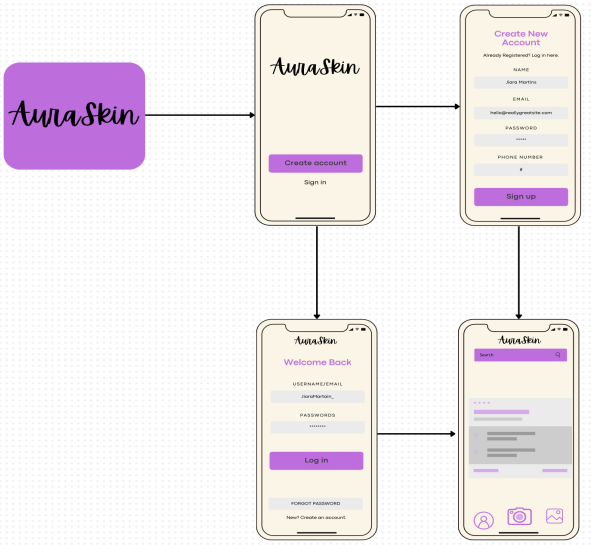
AuraSkin provides a strong foundation for future AI-powered skincare applications and showcases the ability to take an AI concept from a dataset through deployment into a functional mobile prototype.

9. Appendix

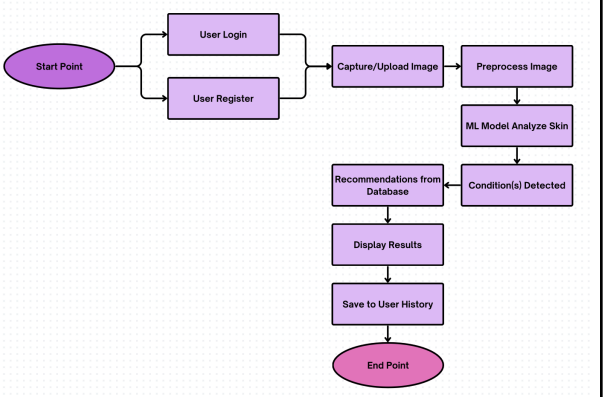
9.1 Mockups and Diagrams

These diagrams visually support the descriptions given in the Design and Development sections.

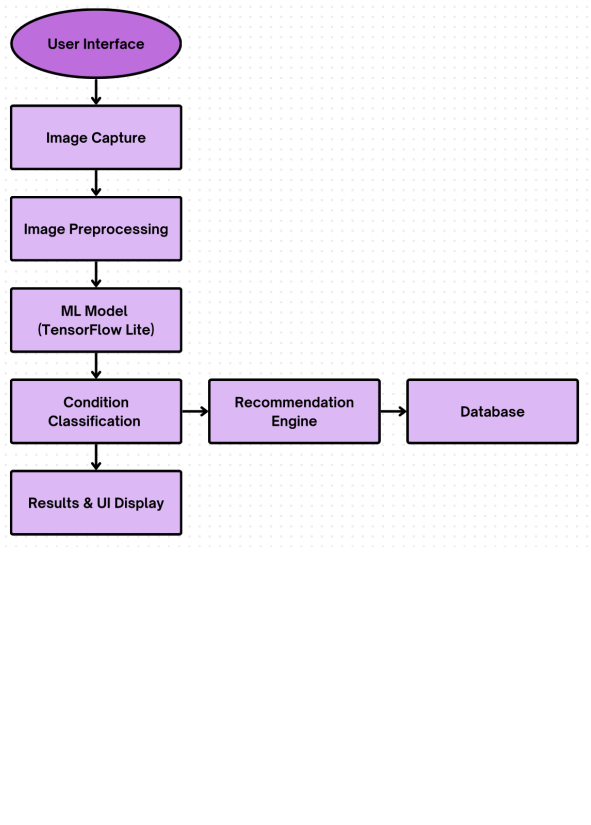
				
Splash Screen	Welcome Screen	SignUp Screen	Login Screen	Home Screen
				
Profile Screen	Camera Screen	Normal Skin Analyzer	Acne Skin Analyzer	Dark Spots Skin Analyzer



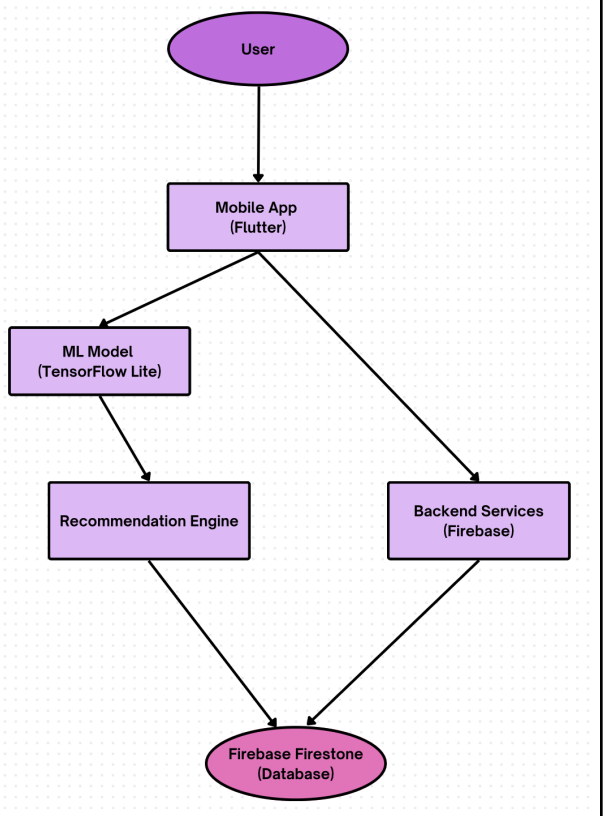
Mobile App Mock-Up



Process Flow Diagram



High-Level Architecture



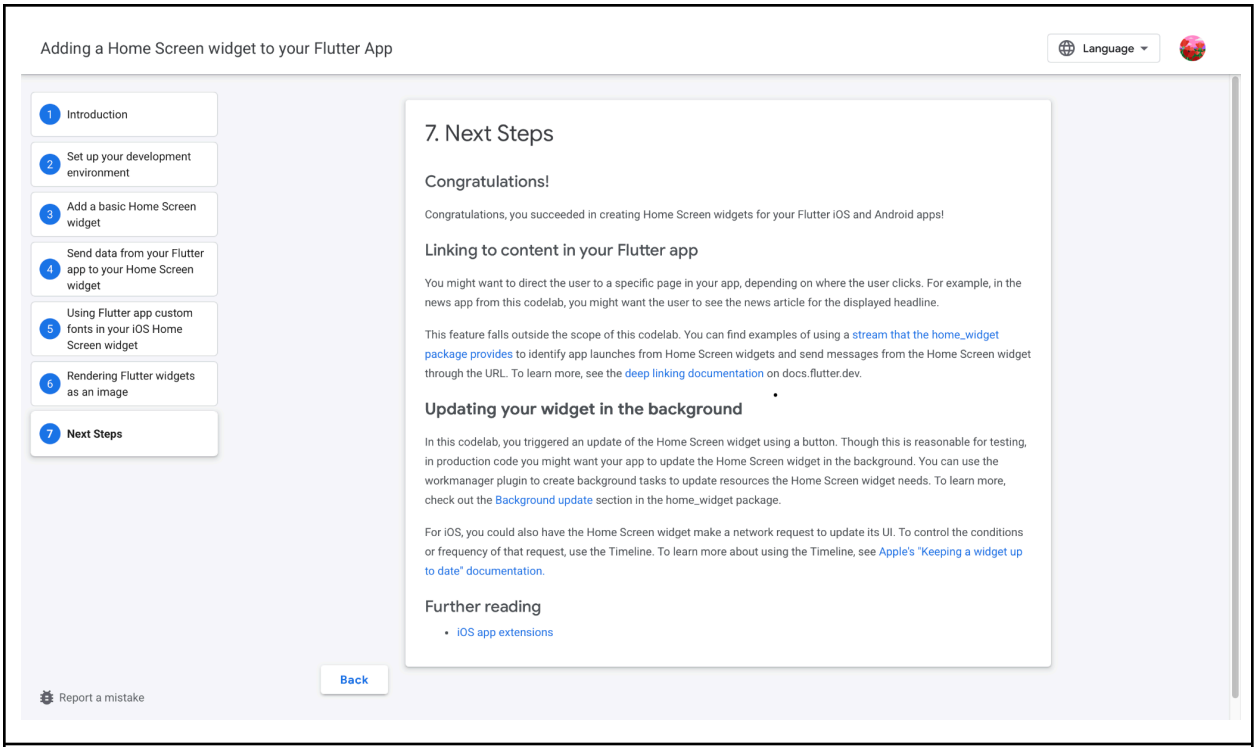
System Architecture

9.2 Training and Tutorial

- FastAPI Tutorial:
  - #1 FastAPI - Python Web Framework
  - FastAPI Tutorial | FastAPI vs Flask
  - https://www.reddit.com/r/FastAPI/comments/1bs889k/why_i_chose_fastapi_how_was_my_experience_and/

- TensorFlow/Keras:
  - How to Train TensorFlow Lite Object Detection Models Using Google Colab | …
  - https://www.reddit.com/r/computervision/comments/1l39yq9/good_reasons_to_prefer_tensorflow_lite_for_mobile/
  - https://keras.io/examples/
  - https://www.youtube.com/watch?v=GnGhI1vKi20

Flutter 'Read and Write Files'



Flutter 'Adding a Home Screen Widget to your Flutter App'

9.3 Links (Website, GitHub, Presentation Video)

All links are also accessible from the project website home page.

- Project Website (Public): https://nrobin56.github.io/AuraSkin/
- GitHub Repository: https://github.com/nrobin56/AuraSkin
- Presentation Video (YouTube): https://www.youtube.com/watch?v=yITVVoOcfWY

9.4 Documentation

- Project Plan: AC3-AuraSkinAI-ProjectPlan.pdf
- Requirements Specification Reference: AC3-AuraSkin-Requirements.pdf
- Software Design Document Reference: AC3-AuraSkin-Design.pdf