Module Guide for Hairesthetics

Team 18
Charlotte Cheng
Marlon Liu
Senni Tan
Qiushi Xu
Hongwei Niu
Bill Song

January 19, 2023

1 Revision History

Date	Version	Notes
Jan 16	0	Rev0 MG

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
DAG	Directed Acyclic Graph
M	Module
OS	Operating System
R	Requirement
SC	Scientific Computing
ProgName	Explanation of program name
UC	Unlikely Change
ML	Machine Learning
UI	User Interface
AI	Artificial Intelligence
AR	Augumented Reality
App	Application
API	Application programming interface
REST	Representational state transfer
RGB	Red, Green, Blue
macOS	Operating system developed by Apple Inc
SRS	Software Requirement Specification
VnVPlan	Validation and Verification Plan
MG	Module Guide
MIS	Module Interface Specification

Contents

1 Revision History i

2 Reference Material ii
  2.1 Abbreviations and Acronyms . . . ii

3 Introduction 1
  3.1 Overview . . . 1
  3.2 Context . . . 1
  3.3 Design Principles . . . 1
  3.4 Design Pattern . . . 1
  3.5 Document Structure . . . 1

4 Anticipated and Unlikely Changes 2
  4.1 Anticipated Changes . . . 2
  4.2 Unlikely Changes . . . 3

5 Module Hierarchy 3

6 Connection Between Requirements and Design 5

7 Module Decomposition 5
  7.1 Hardware Hiding Modules (M13) . . . 5
    7.1.1 Camera Module (M13) . . . 5
  7.2 Behaviour-Hiding Module(M1, M2, M3, M4, M5, M8, M9, M10, M11, M12) . . . 5
    7.2.1 Controller Module (M1) . . . 6
    7.2.2 Facial Recognition Module (M2) . . . 6
    7.2.3 Hair Color Module (M3) . . . 6
    7.2.4 Hair Style Module (M4) . . . 6
    7.2.5 Salon Recommendation Module (M5) . . . 7
    7.2.6 Hair Color View Module (M8) . . . 7
    7.2.7 Hair Style View Module (M9) . . . 7
    7.2.8 Salon Recommendation View Module (M10) . . . 7
    7.2.9 Home View Module (M11) . . . 7
    7.2.10 Error View Module (M12) . . . 8
    7.2.11 Launch View Module (M14) . . . 8
  7.3 Software Decision Module (M6, M7) . . . 8
    7.3.1 ML Model Module (M6) . . . 8
    7.3.2 Utility Module (M7) . . . 9

8 Traceability Matrix 9

9 Use Hierarchy Between Modules **10**
9.1 Use Hierarchy Diagram 10
9.2 UML Diagram 11

List of Tables

1 Module Hierarchy 4
2 Trace Between Requirements and Modules 9
3 Trace Between Anticipated Changes and Modules 10

List of Figures

1 Use hierarchy among modules 10
2 UML Diagram 12

3 Introduction

3.1 Overview

This project is to develop a software application that allows users to virtually simulate hairstyles and hair colors. The goal is to create a user-friendly, functional, and robust application that meets the essential functional requirements and non-functional requirements. The end result will be a software application that helps users easily and effectively experiment with different hairstyles and hair colors, providing an immersive and personalized virtual styling experience.

3.2 Context

This Module Guide(MG) of our project is developed after the Software Requirements Specification(SRS). The purpose of the MG is to explain the structure of modules based on selected design principles and patterns which further helps understand the project's functionalities based on the SRS. It also briefly introduces the services each module provides.

3.3 Design Principles

The design principles which would be used in the MG include information Hiding, high cohesion, low coupling, high fan-in, and low fan-out for module decomposition. Information Hiding includes identifying and encapsulating the expected changes.

3.4 Design Pattern

Our project follows the MVC (model-view-controller) design pattern. MVC allows us to apply the separation of concern principle and breaks the complex problem into smaller subproblems that can be solved in different modules.

3.5 Document Structure

- Anticipated and Unlikely Change
- Module Hierarchy: It places modules related to the functional requirements in the behaviour hiding module.
- Connection Between Requirements And Design: It provides the decisions of design that are needed to be made to realize the requirements
- Module Decomposition: It explains each module by following design principles.
- Traceability Matrices: One matrix is between modules and requirements, the other is between modules and anticipated changes.

- Uses Hierarchy: Diagram explains user hierarchy.
- Project Schedule
- Revision History

4 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 4.1, and unlikely changes are listed in Section 4.2.

4.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

AC1: This project will only result in the development of a MacOS app. However, an android application may also be developed to work with, or in place of, the MacOS application in the future.

AC2: The application may be extended to offer compatibility as a web application.

AC3: The application will not initially pursue functionality for users to "cut" their 3D simulated hair.

AC4: The application may require more data for the facial recognition feature.

AC5: The load capacity of the system is initially set low for the first edition of this application. As demand for the application increases, the load capacity must also increase to support a larger user base.

AC6: The application may be required to include new features as discovered through use of the system in real-world scenarios.

AC7: The format of the input and output data of facial recognition, hair modification, hair salon recommendation system.

AC8: The interface style requirements might change since no metric has been provided for this by stakeholders.

AC9: The speed and latency requirements might change since no metric has been provided for this by stakeholders.

AC10: The precision and accuracy requirements might change since no metric has been provided for this by stakeholders.

4.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decisions should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

ULC1: Since several of the features might require exposing API endpoints, an offline-only version of the application is unlikely to be necessary.

ULC2: As the main demographic of this application is less computer literate, the ease of use of this application must remain a high priority consideration throughout the design of this system.

ULC3: The application is designed to ensure that data will only be modified when necessary.

ULC4: The application is designed to ensure that only admins and supervisors will be able to unlock locked resources.

ULC5: The application is designed to ensure that no inappropriate or offensive language towards any culture will be used.

ULC6: The application is designed to ensure compliance with the Data Privacy Act of Canada. All user personal information will not be stored or used without consent.

ULC7: The application is designed to ensure that users are only able to see data that they have proper authorization for.

ULC8: The application is designed to help users simulate hairstyles and hair colors virtually.

ULC9: The primary user of this project is unlikely going to change since the project purpose is determined already, only certain types of users will be interested in this kind of application.

ULC10: The client of this project is unlikely going to change because this project is a student capstone project and it has limited marketability.

ULC11: The schedule and budget constraints are not likely going to change, because this project is a student capstone project. It needs to strictly follow SE4G06 course constraints.

5 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 1. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Controller Module

M2: Facial Recognition Module

M3: Hair Color Module

M4: Hair Style Module

M5: Salon Recommendation Module

M6: ML Model Module

M7: Utility Module

M8: Hair Color View Module

M9: Hair Style View Module

M10: Salon Recommendation View Module

M11: Home View Module

M12: Error View Module

M13: Camera Module

M14: Launch View Module

Level 1	**Level 2**
Hardware-Hiding Module	M13
Behaviour-Hiding Module	M1 M2 M3 M4 M5 M8 M9 M10 M11 M14
Software Decision Module	M6 M7

Table 1: Module Hierarchy

6 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the SRS. In this stage, the system is decomposed into modules. The connection between requirements and modules is listed in Table 2.

7 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by David Parnas. The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by the operating system or by standard programming language libraries. *ProgName* means the module will be implemented by the ProgName software.

The software architecture we use is the model-view-controller(MVC) architecture, it provides the principle of separation of concerns for our system as well as convenience for future addition and modification to certain components. So we will follow the MVC architecture while decomposing our modules.

7.1 Hardware Hiding Modules (M13)

Secrets: The implementation of the hardware adaptation.

Services: This module would allow the software to change and adapt the changes of hardware environment.

Implemented By: Swift

7.1.1 Camera Module (M13)

Secrets: The implementation of access to camera.

Services: Provide access for the app to the device's camera.

Implemented By: Swift

7.2 Behaviour-Hiding Module(M1, M2, M3, M4, M5, M8, M9, M10, M11, M12)

Secrets: The implementation of the behaviors of our application after processing user input

Services: The behavior-hiding modules includes both user-interface modules and backend logic modules. They serve as a communication layer between the hardware-hiding module and the software decision module.

7.2.1 Controller Module (M1)

Secrets: Controllers to interact with the model and view components after processing the user input.

Services: Provides communication between the user inputs and modules and passes them to the UI.

Implemented By: Python / Swift

Type of Module: Abstract Data Type

7.2.2 Facial Recognition Module (M2)

Secrets: Algorithm for detecting faces in the input image/video.

Services: Detect faces in the input image/video and provide facial landmarks

Implemented By: Python

Type of Module: Abstract Object

7.2.3 Hair Color Module (M3)

Secrets: Algorithm for detecting the hair and changing its color on detected faces

Services: Perform hair re-coloring based on hair segmentation results.

Implemented By: Python

Type of Module: Abstract Object

7.2.4 Hair Style Module (M4)

Secrets: Algorithm for adding new hair on detected faces

Services: Obtain position and rotation degree for adding AR hair model based on input user faces.

Implemented By: Python

Type of Module: Abstract Object

7.2.5 Salon Recommendation Module (M5)

Secrets: The implementation of nearby salon recommendation.

Services: Highlight the nearby salons on the map according to the input address and provide a listing that can be filtered by rankings and distances.

Implemented By: Python, Google Map API

Type of Module: Abstract Object

7.2.6 Hair Color View Module (M8)

Secrets: Algorithms and data structure for the hair color changing screen.

Services: Display the resulted color of hair when user changes the hair color.

Implemented By: Swift

Type of Module: Abstract Object

7.2.7 Hair Style View Module (M9)

Secrets: Algorithms and data structure for hair style changing screen.

Services: Put the selected AR hair model to the detected face when user press the buttons accordingly.

Implemented By: Swift

Type of Module: Abstract Object

7.2.8 Salon Recommendation View Module (M10)

Secrets: Algorithms and data structure for salon recommendation screen.

Services: Display nearby hair salon and display information about a chosen salon.

Implemented By: Python, Google Maps API

Type of Module: Abstract Object

7.2.9 Home View Module (M11)

Secrets: Algorithms and data structure for displaying the home screen.

Services: Provide basic guidance and buttons for the users to start using the app.

Implemented By: Swift

Type of Module: Abstract Object

7.2.10 Error View Module (M12)

Secrets: Algorithms and data structure for displaying the error screen.

Services: Provide details and guidance to errors happening in the application.

Implemented By: Swift

Type of Module: Abstract Object

7.2.11 Launch View Module (M14)

Secrets: Algorithms and data structure for displaying the error screen.

Services: Provide details and status when loading the application.

Implemented By: Swift

Type of Module: Abstract Object

7.3 Software Decision Module (M6, M7)

Secrets: The design decision based on mathematical models, and real-world objects in the form of data structures. It also includes libraries required to support behavioral functionalities.

Services: Includes data structure and algorithms used in the system that do not provide direct interaction with the user.

Implemented By: Python

7.3.1 ML Model Module (M6)

Secrets: Data structure for representing a pre-trained model

Services: This module will provide functions to load the pre-trained model for hair segmentation if it's not already loaded. It also provides functions to interact with the model easily.

Implemented By: Python

Type of Module: Abstract Object

7.3.2 Utility Module (M7)

Secrets: Utility functions for image processing

Services: This module will provide utility services and operations required for performing functional behaviors.

Implemented By: Python

Type of Module: Library

8 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	**Modules**
FR1	M8, M9
FR2	M7
FR3	M2, M6
FR6	M2
FR7	M2, M6
HM1	M8
HM2	M3
HM3	M3
HM4	M4, M6
HM5	M9
HM7	M7
HM8	M4
HM9	M8, M9
HM10	M3, M6
HR1	M10
HR2	M5
HR3	M10
HR4	M5

Table 2: Trace Between Requirements and Modules

AC	Modules
AC1	M8, M9, M10, M11, M12
AC2	Not applicable
AC3	M4, M9
AC4	M6
AC5	Not applicable
AC6	M1
AC7	M2, M4, M5
AC8	M8, M9, M10, M11, M12
AC9	M5, M10
AC10	M4, M9

Table 3: Trace Between Anticipated Changes and Modules

9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. Two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure 2 illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

9.1 Use Hierarchy Diagram

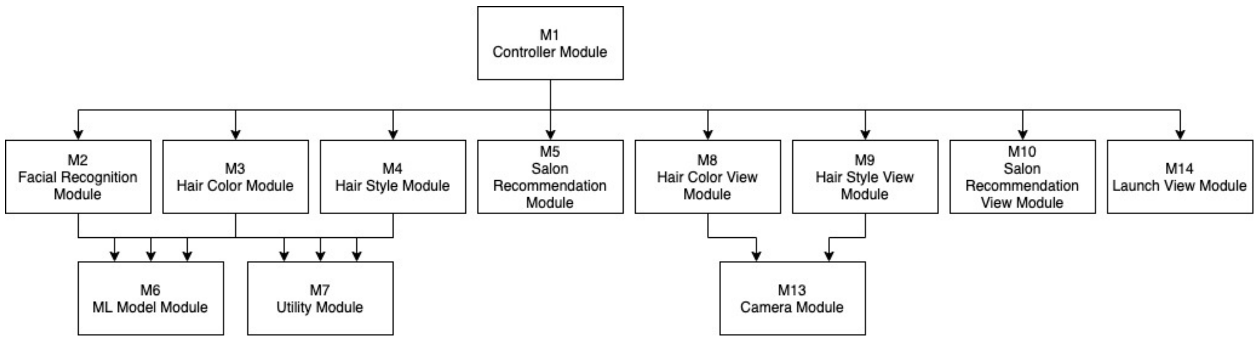
M1 Controller Module
M2 Facial Recognition Module
M3 Hair Color Module
M4 Hair Style Module
M5 Salon Recommendation Module
M8 Hair Color View Module
M9 Hair Style View Module
M10 Salon Recommendation View Module
M14 Launch View Module
M6 ML Model Module
M7 Utility Module
M13 Camera Module

Figure 1: Use hierarchy among modules

9.2 UML Diagram

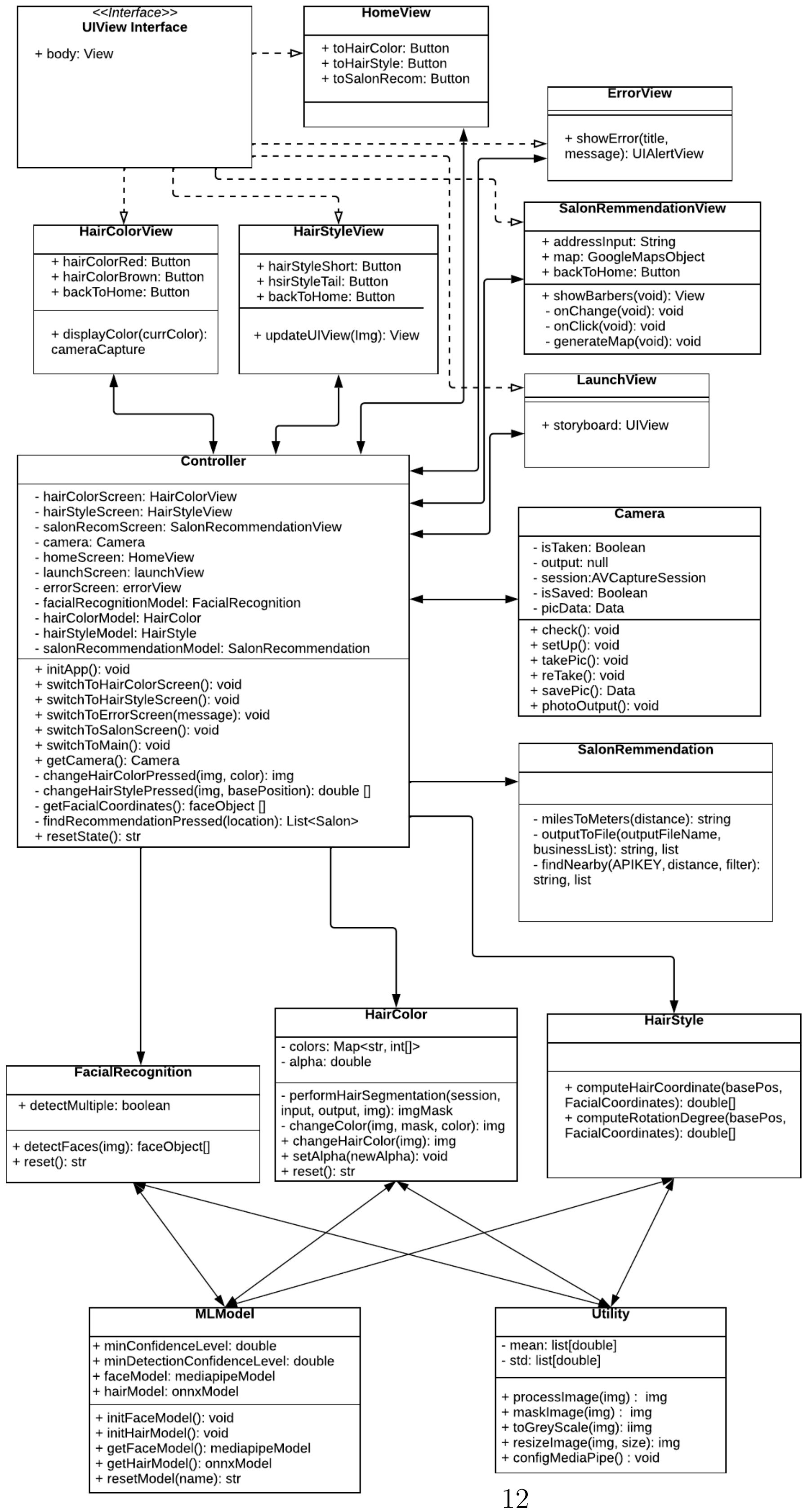
<<Interface>>
UIView Interface
+ body: View
HomeView
+ toHairColor: Button
+ toHairStyle: Button
+ toSalonRecom: Button
ErrorView
+ showError(title, message): UIAlertView
SalonRemmendationView
+ addressInput: String
+ map: GoogleMapsObject
+ backToHome: Button
+ showBarbers(void): View
- onChange(void): void
- onClick(void): void
- generateMap(void): void
HairColorView
+ hairColorRed: Button
+ hairColorBrown: Button
+ backToHome: Button
+ displayColor(currColor): cameraCapture
HairStyleView
+ hairStyleShort: Button
+ hsirStyleTail: Button
+ backToHome: Button
+ updateUIView(Img): View
LaunchView
+ storyboard: UIView
Controller
- hairColorScreen: HairColorView
- hairStyleScreen: HairStyleView
- salonRecomScreen: SalonRecommendationView
- camera: Camera
- homeScreen: HomeView
- launchScreen: launchView
- errorScreen: errorView
- facialRecognitionModel: FacialRecognition
- hairColorModel: HairColor
- hairStyleModel: HairStyle
- salonRecommendationModel: SalonRecommendation
+ initApp(): void
+ switchToHairColorScreen(): void
+ switchToHairStyleScreen(): void
+ switchToErrorScreen(message): void
+ switchToSalonScreen(): void
+ switchToMain(): void
+ getCamera(): Camera
- changeHairColorPressed(img, color): img
- changeHairStylePressed(img, basePosition): double []
- getFacialCoordinates(): faceObject []
- findRecommendationPressed(location): List<Salon>
+ resetState(): str
Camera
- isTaken: Boolean
- output: null
- session:AVCaptureSession
- isSaved: Boolean
- picData: Data
+ check(): void
+ setUp(): void
+ takePic(): void
+ reTake(): void
+ savePic(): Data
+ photoOutput(): void
SalonRemmendation
- milesToMeters(distance): string
- outputToFile(outputFileName, businessList): string, list
- findNearby(APIKEY, distance, filter): string, list
HairColor
- colors: Map<str, int[]>
- alpha: double
- performHairSegmentation(session, input, output, img): imgMask
- changeColor(img, mask, color): img
+ changeHairColor(img): img
+ setAlpha(newAlpha): void
+ reset(): str
HairStyle
+ computeHairCoordinate(basePos, FacialCoordinates): double[]
+ computeRotationDegree(basePos, FacialCoordinates): double[]
FacialRecognition
+ detectMultiple: boolean
+ detectFaces(img): faceObject[]
+ reset(): str
MLModel
+ minConfidenceLevel: double
+ minDetectionConfidenceLevel: double
+ faceModel: mediapipeModel
+ hairModel: onnxModel
+ initFaceModel(): void
+ initHairModel(): void
+ getFaceModel(): mediapipeModel
+ getHairModel(): onnxModel
+ resetModel(name): str
Utility
- mean: list[double]
- std: list[double]
+ processImage(img) : img
+ maskImage(img) : img
+ toGreyScale(img): iimg
+ resizeImage(img, size): img
+ configMediaPipe() : void

Figure 2: UML Diagram