**Low Level Design Document**

Compare Faces Code Challenge

By: Mark Hawkins

Version 1.0

Change History:

|  |  |  |
| --- | --- | --- |
| Author | Date | Changes |
| Mark Hawkins | March 31, 2020 | * Initial |
| Mark Hawkins | April 2, 2020 | * Added Technology * Guidelines |

**Table of Contents**

[Introduction 3](#_Toc36818838)

[Design Overview 3](#_Toc36818839)

[Technology 3](#_Toc36818840)

[Architecture 3](#_Toc36818841)

[Guidelines 4](#_Toc36818842)

[User Stories 4](#_Toc36818843)

[Use Case Diagram 4](#_Toc36818844)

[Deployment Diagram 5](#_Toc36818845)

# Introduction

Compare Faces is an application that allows two images to be compared by having the faces in one photo be matched to another.

# Design Overview

The application is broken into three tiers:

* **Front-End Tier:** Where the User interacts with an interface to send and receive data from.
* **Middleware Tier:** Uses a business layer within the application to gather and process information received from Back End.
* **Back-End Tier:** API that receives information from a server processing the data sent and sending back the compiled result.

The goal of the application is to send and acquire data through an API. Making the core functionality platform agnostic. The application will send data through the API and have the Back-End services process the information. Once the API completes the data, it will send a response with data from the processing for the application to present.

### Why this approach?

By having the main computation on the API, it allows to easily compute data from various platforms without having to rewrite code to produce the same functionality.

## Technology

|  |  |
| --- | --- |
| Front-End / Middleware | |
| Xcode 1.4 | IDE |
| Swift 5 | Language |
| Alamofire | Library |

|  |  |
| --- | --- |
| Back-End | |
| AWS – API Gateway | Service |
| AWS - Lambda | Service |
| Visual Studio Code | IDE |
| Node.js | Language |
| AWS SDK | Library |

## Architecture

The application will utilize an MVC pattern. This allows for separation of the data (Model) and UI (View) layers. With this separation, any changes to the View do not affect the Model and vice-versa. Utilizing this architecture also allows multiple developers to easily work on various sections without compromising other areas of the program.

## Guidelines

Working with iOS, Apple has provided complete documentation on how applications developed on their platform should act and how users should be interacting. The Human Interface Guidelines used can be viewed [here](https://developer.apple.com/design/human-interface-guidelines/ios/overview/themes/).

A small overview of adherence to guidelines:

|  |  |
| --- | --- |
| Section | Guideline |
| App Architecture/Launching | Launch in the appropriate orientation |
| App Architecture/Launching | Don’t encourage rebooting |
| App Architecture/Onboarding | Get to the action quickly |
| App Architecture/Loading | Make it clear when loading is occurring |
| App Architecture/Modality | Reserve alerts for delivering essential — and ideally actionable — information |
| App Architecture/ Requesting Permission | Don’t request location information unnecessarily |
| App Architecture/ Gestures | As a general rule, use standard gestures |

# User Stories

|  |
| --- |
| As a User, I can select an image from either the Photo Library or the Camera. |
| As a User, I can submit two images with Faces in them and see if they match. |

# Use Case Diagram

A close up of a logo

Description automatically generated

# Deployment Diagram

A screenshot of a cell phone

Description automatically generated