Bilkent University

Department of Computer Engineering

CS 492: Senior Design Project II

Fakenstein

Low Level Design Report

Group Members:
Yusuf Ardahan Doğru
Atakan Dönmez
Öykü Irmak Hatipoğlu
Elif Kurtay
Cansu Moran

Website: Fakenstein

Supervisor: Dr. Selim Aksoy
Innovation Expert: Adnan Erdursun
Jury Members: Erhan Dolak and Tağmaç Topal

Low Level Design Report February 28, 2022.

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491/2.

Table of Content

1. Introduction

The concerns about the violation of privacy have become more and more prominent in human lives with the ever growing internet. Every day, people are photographed without their consent and appear in the photographs that are uploaded online. On average, an American is caught on camera 75 times a day without being aware of it [1]. This poses a security threat for people on a daily basis. Examples of people trying to protect other people's privacy on social media is increasing everyday, especially with celebrities covering their children's faces with emojis to protect them from media publicity. Moreover, in social events when a picture is going to be taken, there are always a few people who do not want to appear in the photograph. Therefore, before taking a photograph, permission must be sought from all participants.

With more widespread use of the internet from day to day, personal data protection laws are becoming stricter. When taking pictures, people have to be more sensitive about the privacy of the data holders. For example, from Fig. 1, it can be seen that Google Street View blurs the faces of people appearing in photographs in order to protect their privacy [2]. Apart from blurring the images, another method to protect the privacy rights of the individuals that appear in the photograph is removing them using external tools such as Adobe Photoshop which people should buy a subscription to use [3], and Magic Eraser which is actually not available for anyone who does not own a Google Pixel 6 and thus have a limited user base [4].

Additionally, removing people and filling the remaining space with background is a task that requires advanced knowledge in Adobe Photoshop and would take a significant amount of time and effort. A more user-friendly alternative to remove unwanted people from the photograph is an image manipulation tool named Inpaint designed for the purpose of image restoration [5]. However, this tool requires the user to carefully paint each object to be removed. If the painting is not precise enough or if the photograph has a complicated background, then the Inpaint tool outputs unrealistic photographs. The Inpaint tool does not solely exist to remove humans but it is a general purpose removing algorithm. A complicated background is considered to be a background that does not have consistent lines or has different patterns in close areas.

Our proposed phone application, Fakenstein, aims to realistically replace the faces of unknown people with artificially generated faces in order to protect their anonymity. Unlike Inpaint or Adobe Photoshop, Fakenstein intends to keep user interaction as little as possible while outputting a realistic looking photograph. Today, the style-based GAN algorithms can produce extremely realistic artificially generated faces that do not exist. The examples of such faces can be examined from the website where each time the page is refreshed, the website displays a generated

face [6]. For the task of replacing faces in a picture, the guidance of the infamous “deep fakes” can be used. Mostly originated from the GitHub repository “DeepFaceLab”, deep fakes are used to replace faces in videos to generate realistic fake videos. For example, it is possible to replace Iron Man’s face with Tom Cruise’s and produce a video that looks like Tom Cruise is the actual actor playing Iron Man. Using a similar method we aim to obtain seamlessly replaced faces. Additionally, we are trying to build an application that will be available for everyone unlike Magic Eraser which is only available for Google Pixel 6 owners and Adobe Photoshop which can only be used through an expensive subscription.

This report will start with the discussion of object design trade-offs. Then, we will illustrate the interface documentation guidelines. Then, we will touch upon the engineering standards we used in this project. The definitions, acronyms, and the abbreviations we used in this project will also be given. The high level design for our system included in the High Level Design Report will be detailed into low level design in sections, packages and class interfaces.

1.1. Object Design Trade-Offs

When we made some decisions to make our application better, we also had to consider some trade-offs. The design trade-offs we made were compatibility vs. development time, functionality vs. usability, and time-efficiency vs. cost.

1.1.1. Compatibility vs. Development Time

We would like this application to be present in two different platforms, mobile (Android) and desktop. The mobile version is a lightweight version while the desktop version is for professional use. The fact that we made the program compatible for various platforms is a trade-off in terms of the development time.

1.1.2. Functionality vs. Usability

In the face editing part, if the user does not like the face our program constructs and wants to change it, we only offer a few attributes that can be changed which include gender, age and skin color. This clearly decreases the functionality of the design, however, by this way we increase the usability of our application. If there were a lot of minor attributes that the user could change, (nose size, clothing, etc.) it would be harder for users to use the application as they would have to determine a lot of attributes and spend a lot of time to provide attributes to generate an appropriate face.

1.1.3. Time-Efficiency vs. Cost

To decrease the waiting time for generation of the artificial faces, we decided to store several pre-generated faces from all combinations of the attributes on a database.

The database will be updated with new faces periodically. Since there will be a large variety of faces offered to the user, it will be unlikely that the user tries all the faces stored on the database and still can't pick one, since it would be time-consuming on the user's end. Therefore, we will only be offering the user the faces stored in the database at the moment of using the application. In the runtime, for selecting a suitable face, we will retrieve an appropriate face from the database. However, storing a large number of faces in the database will cost money, which will be our trade-off for time-efficiency.

1.2. Interface Documentation Guidelines

In this report, all the class names are named in the standard -each word starts with capital letter- format, where all of these names are singular. The attribute and method names follow the camel case format as in 'attributeName' and 'methodName()'. In the class description table, the class names appear first with the description of the class underneath. The attributes of the class follow the class description in the following format "<accessModifier> <attributeType> attributeName". The hierarchy ends with the methods of the class alongside information about their returned values in the format "<accessModifier> <returnType> methodName(<Parameters>)". Getter and setter functions for attributes are excluded for simplicity purposes except for when they have a higher significance compared to other methods. In the following table, the detailed outline is presented as:

ClassName	
This is a sample class.	
Attributes	
private String sampleAttribute	
Methods	
public void sampleMethod(int[] parameter)	This is the explanation of the given method on the left.

1.3. Engineering Standards

We have followed the UML guidelines [7] we have learned in CS319 class to construct the diagrams and models on our Requirement Elicitation, Analysis, High Level Design, and Low Level Design reports. We have used IEEE citation method [8] for the references we gave.

1.4. Definitions, Acronyms, and Abbreviations

BMP	Bitmap Image File
JPEG	Joint Photographic Expert Group Image
PNG	Portable Network Graphics
WebP	Web Picture Format
HEIF	High Efficiency Image File Format
JVM	Java Virtual Machine
IDE	Integrated Development Environment
ML	Machine Learning
DL	Deep Learning
SDK	Software Development Kit
HTTP	Hypertext Transfer Protocol
TCP	Transmission Control Protocol
IP	Internet Protocol
API	Application Programming Interface
GAN	Generative Adversarial Network

2. Packages

Fakenstein has two main subsystems in its Low Level system, which are client and server. There are two parts located inside the client-side. These parts are the DesktopFrontend and MobileFrontend. The three parts that represent the Server subsystem are called LogicController, Data and ModelController. In the project, the Client side makes use of its frontend subsystems to make the user able to use the packages on the Server side. The Client side has components like TutorialManager, FaceEditor, ImageManager, FileManager that receive the interaction of users with the application and send them to the server to keep the UI (user interface) running. On the other hand, the server-side is responsible for communication between the Data and the ModelController, which contain necessary items like model parameters and previously generated faces.

2.1. Client

The client side consists of 2 subsystems; DesktopFrontend and MobileFrontend. The DesktopFrontend is responsible for communicating the user’s actions in the desktop application to the server side. Similarly, the MobileFrontend subsystem is responsible for connection between the server and the client while the user is

interacting with the mobile application. DesktopFrontend is an extended version of MobileFrontend, it makes use of all the modules in MobileFrontend. Hence, the shared modules between the two subsystems are only represented inside the MobileFrontend subsystem. If a module is extended with the addition of new abilities, then this module is represented separately in the DesktopFrontend with a prefix “Advanced-” added to the module name in MobileFrontend.

2.1.1. Desktop Frontend

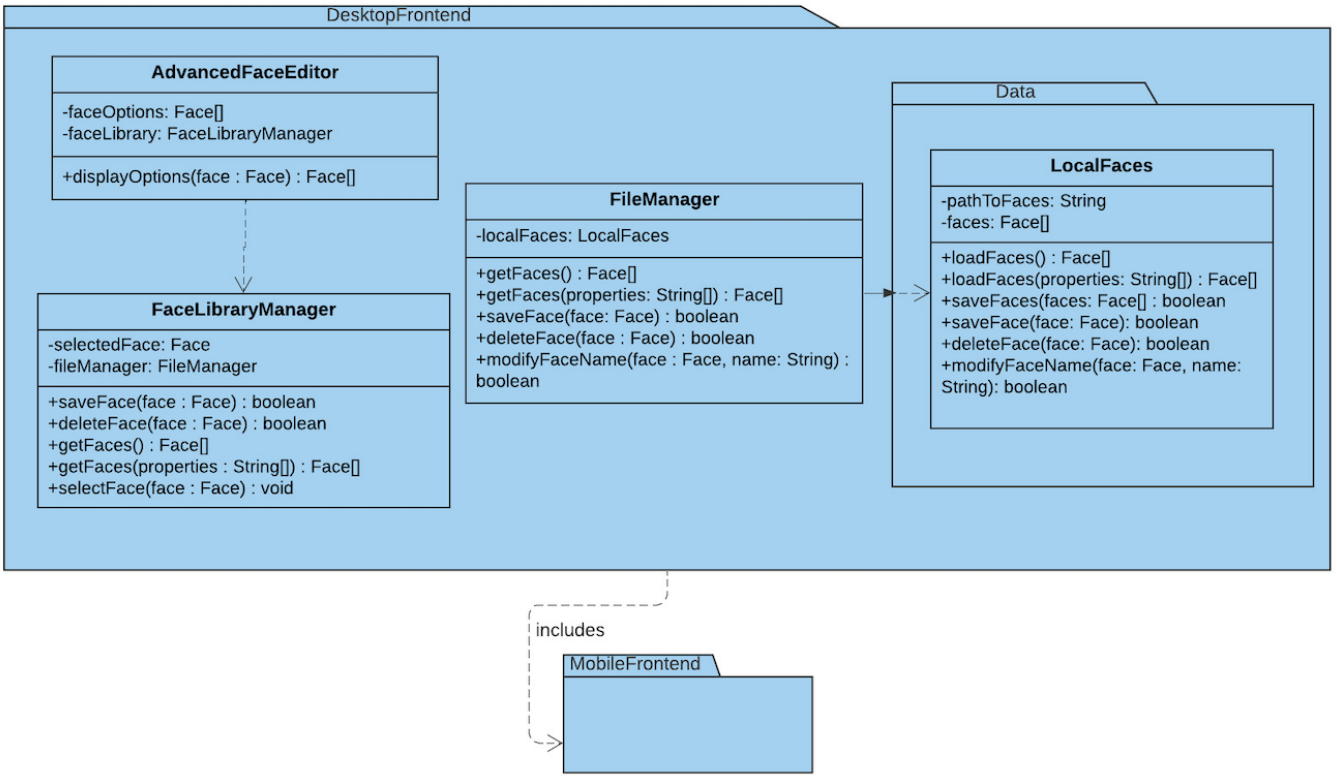

Figure 1: Subsystem Decomposition: DesktopFrontend

The DesktopFrontend subsystem is responsible for handling the user’s interaction with the desktop application by communicating between the server and the client. It contains the AdvancedFaceEditor, FaceLibraryManager, FileManager, and LocalFaces components in addition to the components that are shared with the MobileFrontend subsystem.

AdvancedFaceEditor:

The AdvancedFaceEditor is responsible for handling the user’s requests to manually edit faces in a chosen image. It allows the user to switch a face in the image with another face stored in the face array.

FaceLibraryManager:

The FaceLibraryManager is responsible for showing the user the array, or library of faces that the user can use to edit the image.

FileManager:

The FileManager is responsible for the necessary files needed for the application. It contains the Data folder, which has the LocalFaces in it.

LocalFaces:

The LocalFaces is responsible for holding the application's local faces. The local faces are the faces that the user saves while using the application.

2.1.2. Mobile Frontend

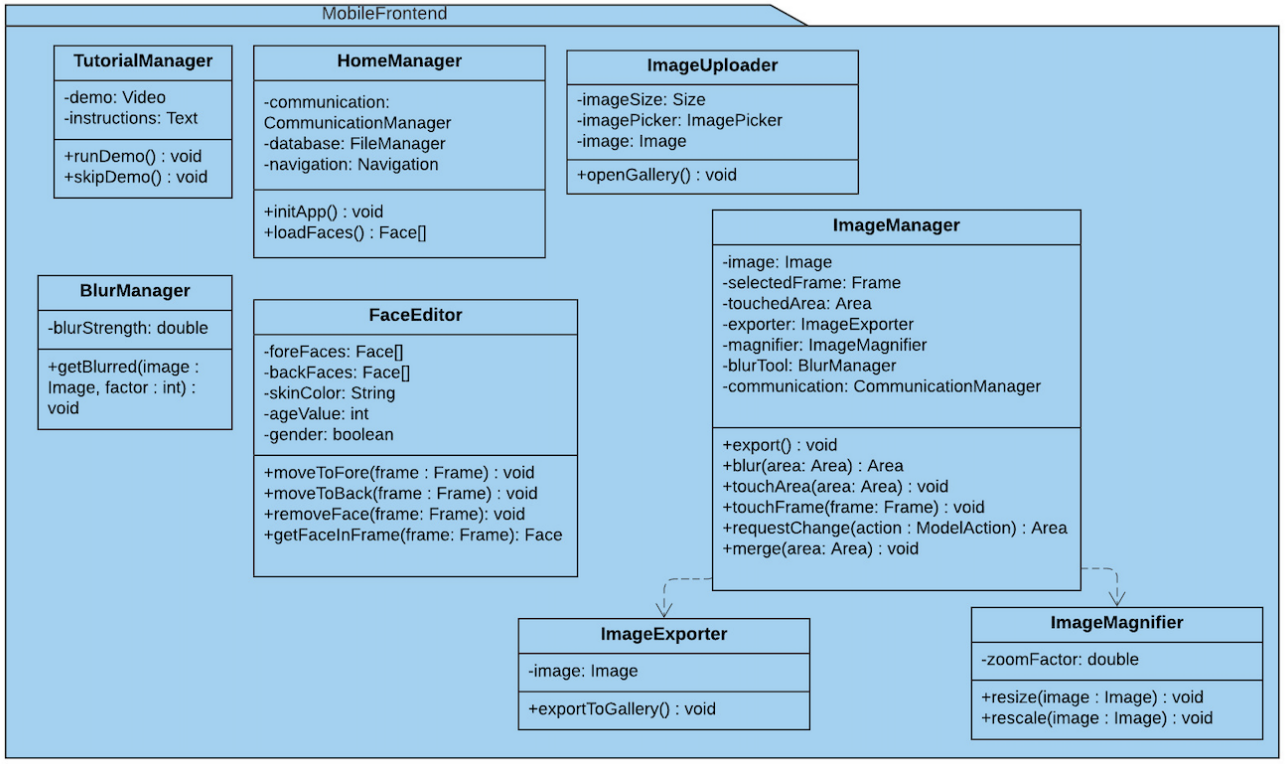

Figure 2: Subsystem Decomposition: MobileFrontend

The MobileFrontend subsystem is mainly responsible for handling the user's interaction with the mobile application by communicating between the server and the client. In addition, all classes in this subsystem are included or extended in the DesktopFrontend system as mentioned in the section 2.1.1.

HomeManager:

HomeManager is responsible for loading the application on the initial startup.The application makes a connection with the server, and the initial screen is loaded.

TutorialManager:

TutorialManager is responsible for guiding the user through the tutorial of the application, which is specifically for the mobile version of the application.

BlurManager:

BlurManager is responsible for helping the user to blur an image that the user has been editing in the application.

ImageManager:

ImageManager is responsible for dealing with the user's actions concerning an image in the application.

ImageUploader:

ImageUploader is responsible for dealing with how the user uploads images to the application.

ImageExporter:

ImageExporter is responsible for the activities of exporting and saving the edited images in the application.

ImageMagnifier:

ImageMagnifier is responsible for helping the user with zooming in and out of the image and to see clearly which the user wants to work on.

FaceEditor:

FaceEditor provides the user an easy-to-use interface for editing the faces in an image. It can be used to changes faces's positions from foreground to background, and vice versa, and contains information like skin color, age and genders for the faces.

2.2. Server

2.2.1. Logic Controller

This subsystem controls the requests and responses between the server and the client.

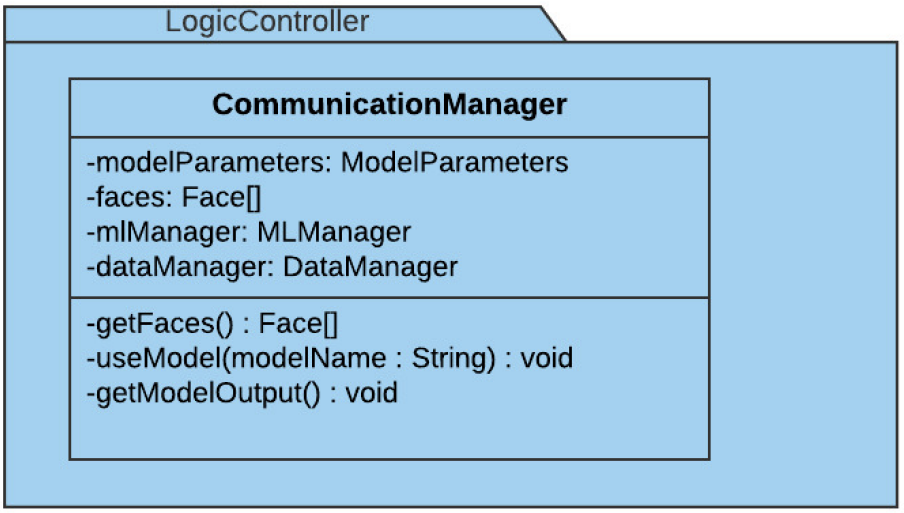

Figure 3: Subsystem Decomposition: LogicController

CommunicationManager

This class handles the requests and responses between the Client and Server subsystems. This class also formats the resulting image or other outputs coming from the components to adapt to the requirements of each side.

2.2.2. Model Controller

This package holds the different ML models to be used in the application and is responsible for their operation.

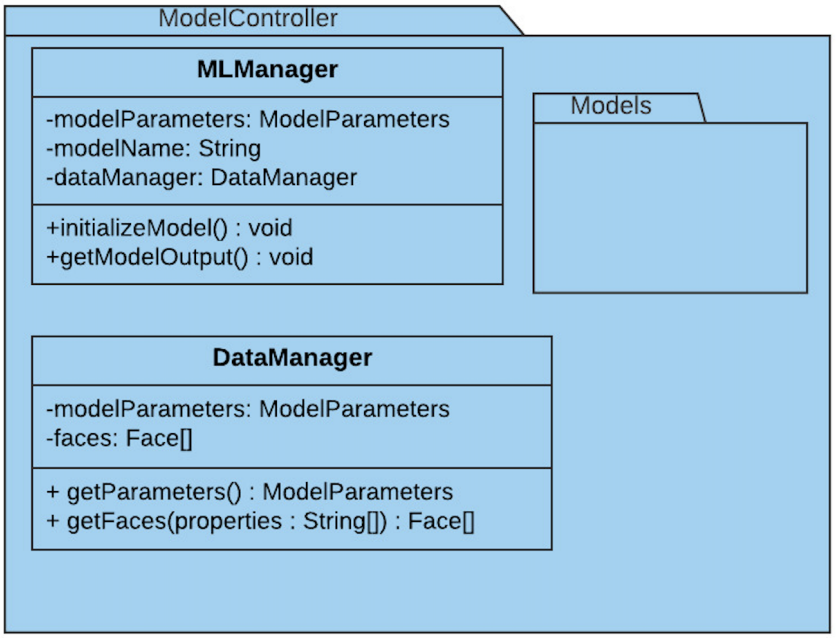

Figure 4: Subsystem Decomposition: ModelController

MLManager

This class initializes the ML models and manages the input and output of the models.

DataManager

This class handles the communication between the Data subsystem and the MLManager so that the ML model can use the pre-decided parameters and pre-generated faces can be used instead of new face generation if required information for a face has a match in the database.

2.2.3. Data

This package holds the required data (model weights) to process and generate images and is responsible for passing them onto the ML models. It also contains the pre-generated Face (from the online database) which are used to eliminate the face generation time in the applications.

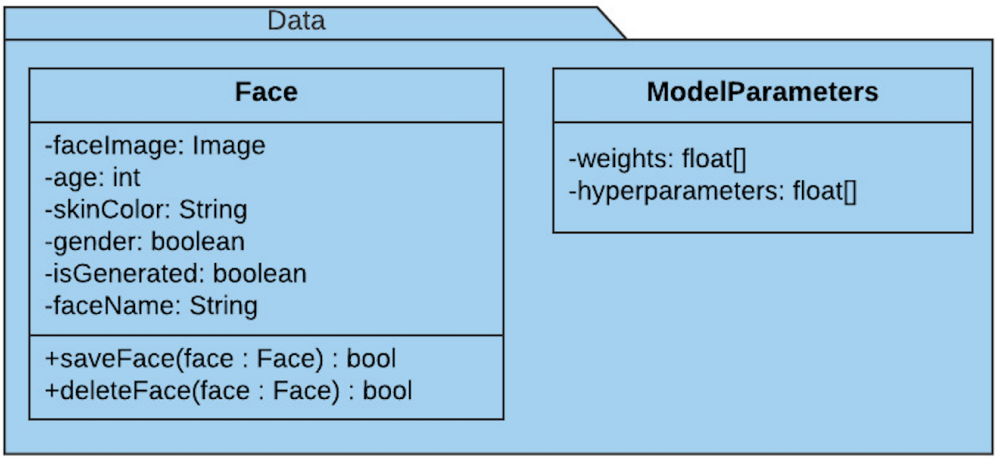

Figure 5: Subsystem Decomposition: Server Data

Face

The artificially generated faces to be used as a replacement for the people that appear in the background of an image. These faces are stored in the online database to reduce the runtime of the application.

ModelParameters

Parameters used in the ML models. These parameters can be weights of pretrained models in addition to hyperparameter values for any model that needs to be re-trained.

3. Class Interfaces

In this section, signatures, properties and methods of the classes will be provided. In addition, their specific duties will be indicated in detail.

3.1. Client Subsystem

In this section the subsystems are dedicated for the client part namely mobile and desktop applications. Due to the nature of the client subsystem, many classes are dedicated for the frontend development. Hence, the main two subsystems under the client subsystem are named as “frontend”. Names of the functions and classes that are listed may alter along the development life cycle of the project.

3.1.1. Mobile Frontend

The attributes and methods marked with an “*” (asterisk) are only accessed for the Desktop application.

HomeManager	
HomeManager loads the application on at the initial startup. This is a view class, responsible for displaying the initial home page where there is the possibility of advancing to the tutorial, having more information about the application or choosing to upload an image from the user's gallery. At this stage, the application will establish a connection with the server.	
Attributes	
private CommunicationManager communication private FileManager database* private Navigation navigation	
Methods	
public void initApp() public Face[] loadFaces()*	Boots up the application Loads the local faces from the file system

TutorialManager	
TutorialManager is responsible for displaying instructions about using Fakenstein including a demo video.	
Attributes	
private Video demo private Text instructions	
Methods	
public void runDemo() public void skipDemo()	Starts the demo. Provides users to skip the demo.

BlurManager	
BlurManager is responsible for blurring the areas selected by the user on an image.	
Attributes	
private double blurStrength	
Methods	
public void getBlurred(Image image, int factor)	returns the blurred image piece according to its factor

FaceEditor	
This view is responsible for displaying the foreground and background images, and allowing the user to manually select the faces which they wish to be changed.	
Attributes	
private Face[] foreFaces private Face[] backFaces private int ageValue private String skinColor private boolean gender	
Methods	
public void moveToFore(Frame frame) public void moveToBack(Frame frame) public void removeFace(Frame frame) private Face getFaceInFrame(Frame frame)	moves a background marked image to foreground moves a foreground marked image to background removes face from selected faces returns the allocated face from the user input of a touched area (frame)

ImageUploader
This view is responsible for displaying the user their gallery to choose an image to upload to Fakenstein. (In the mobile app, there will be size and quality limitation to image uploading whereas the desktop will accept original quality and size.)
Attributes
private Size imageSize private ImagePicker imagePicker

private Image image	
Methods	
public void openGallery()	redirects the user to their photo gallery of their device and returns a selected resized photo that is uploaded

ImageManager	
ImageManager allows the user to make changes on the image by extending respective classes for Image Exporter and Image Magnifier. In addition, the modifications on the image will be selected, requested, and received in this view.	
Attributes	
private Image image private Frame selectedFrame private Area touchedArea private ImageExporter exporter private ImageMagnifier magnifier private BlurManager blurTool private CommunicationManager communication	
Methods	
public void export() public Area blur(Area area)	calls exporter to export the final image sends the touched area to the blurring tool and returns the new blurred area
public void touchArea(Area area)	selects, deselects or increases count of select on an area according to the state
public void touchFrame(Frame frame)	selects or deselects a frame according to the state
public Area requestChange(ModelAction action)	requests a change on a face from the modals in the server and returns the resulting image area
private void merge(Area area)	merges the changed image area into the main image

ImageExporter	
This class allows the user to export the resulting image by saving the image to their gallery.	
Attributes	

private Image image	
Methods	
public void exportToGallery()	exports the image to the gallery

ImageMagnifier	
This class allows the user to magnify the image for easing selection of individual faces or blurring areas.	
Attributes	
private double zoomFactor	
Methods	
public void resize(Image image) public void rescale(Image image)	rescales the image resizes the image

3.1.2. Desktop Frontend

AdvancedFaceEditor	
This view allows the desktop users more advanced editing options on top of the mobile ones such as displaying different generated face options for replacement.	
Attributes	
private Face[] faceOptions private FaceLibraryManager faceLibrary	
Methods	
public Face[] displayOptions(Face face)	displays generated face options for a given real face

FaceLibraryManager
This class handles the face library, which allows the users to save/delete faces to/from the face library, display the faces in the library and choose a face from the library for replacement.

Attributes	
private Face selectedFace private FileManager fileManager	
Methods	
public boolean saveFace(Face face) public boolean deleteFace(Face face) public Face[] getFaces() public Face[] getFaces(String[] properties) public void selectFace(Face face)	saves the given face to the library deletes the given face from the library gets all the faces in the library gets faces from the library that fit the given properties(age, gender, skin color) select a face from the library

FileManager	
This class handles the communication between the FaceLibraryManager and the local database, LocalFaces, to add, get, modify, or delete an entry in LocalFaces.	
Attributes	
private LocalFaces localFaces	
Methods	
public Face[] getFaces() public Face[] getFaces(String[] properties) public boolean saveFace(Face face) public boolean deleteFace(Face face) public boolean modifyFaceName(Face face, String name)	gets all the faces from the local faces gets the stored faces that fit the given properties(age, gender, skin color) from the local faces saves a given face to the local faces deletes a given face from the local faces modifies the name of the given face in the local faces

3.1.3. Data

LocalFaces	
A local database which contains the generated faces saved by the user with their names.	
Attributes	
private String pathToFaces private Face[] faces	
Methods	

public Face[] loadFaces() public Face[] loadFaces(String[] properties) public boolean saveFaces(Face[] faces) public boolean saveFace(Face face) public boolean deleteFace(Face face) public boolean modifyFaceName(Face face, String name)	loads the local faces from the filesystem loads the stored faces that fit the given properties(age, gender, skin color) from the filesystem saves the given faces to the filesystem saves a given face to the filesystem deletes a given face from the filesystem modifies the name of the given face in the filesystem

3.2. Server Subsystem

3.2.1. Logic Controller

CommunicationManager	
This class handles the requests and responses between the Client and Server subsystems. This class also formats the resulting image or other outputs coming from the components to adapt to the requirements of each side.	
Attributes	
private ModelParameters modelParameters private Face[] faces private MLManager mlManager private DataManager dataManager	
Methods	
public Face[] getFaces() public void useModel(String modelName) public {} getModelOutput()	retrieves and returns the generated faces from the database directs the request to MLManager for the given model to be run retrieves the model output from MLManager and responds to the client with the output (because there are different output types for different models, the return type is given as a dictionary object)

3.2.2. Model Controller

MLManager

This class initializes the ML models and manages the input and output of the models.	
Attributes	
private ModelParameters modelParameters private String modelName private DataManager dataManager	
Methods	
public void initializeModel() public {} getModelOutput()	initializes the model specified by modelName with modelParameters returns the output of the model (because there are different output types for different models, the return type is given as a dictionary object)

DataManager	
This class handles the communication between the Data subsystem and the MLManager so that the ML model can use the pre-decided parameters and pre-generated faces can be used instead of new face generation if required information for a face has a match in the database.	
Attributes	
private ModelParameters modelParameters private Face[] faces	
Methods	
public ModelParameters getParameters() public Face[] getFaces(String[] properties)	gets the model parameters (can be hyperparameter values or weights of pretrained models) gets the generated faces that fit the given properties

3.2.3. Data

Face
Face is an object class which represents artificially generated faces and real faces in the image. The generated faces are stored in the online database to reduce the runtime of the application.

Attributes	
private Image faceImage private int age private String skinColor private boolean gender private boolean isGenerated private String faceName	
Methods	
getter/setter methods for the attributes public boolean saveFace(Face face) public boolean deleteFace(Face face)	 saves the given face to the online database delete the given face from the online database

ModelParameters	
Parameters used in the ML models. These parameters can be weights of pretrained models in addition to hyperparameter values for any model that needs to be re-trained.	
Attributes	
private float[] weights private float[] hyperparameters	
Methods	
getter/setter methods	

4. References

[1] R. Eveleth, “How Many Photographs of you are out there in the world?,” *The Atlantic*, 03-Nov-2015. [Online]. Available: https://www.theatlantic.com/technology/archive/2015/11/how-many-photographs-of-you-are-out-there-in-the-world/413389/. [Accessed: 15-Nov-2021].

[2] “Contributed street view imagery policy,” Google. [Online]. Available: https://www.google.com/intl/en_uk/streetview/policy/. [Accessed: 09-Oct-2021].

[3] “Photoshop free trial | official adobe photoshop.” [Online]. Available: https://www.adobe.com/products/photoshop/free-trial-download.html. [Accessed: 9-Oct-2021].

[4] E. Then, “Pixel 6 magic eraser removes uninvited people from photos,” *SlashGear*, 20-Oct-2021. [Online]. Available: https://www.slashgear.com/pixel-6-magic-eraser-removes-uninvited-people-from-photos-19695941/. [Accessed: 15-Nov-2021].

[5] “Remove people from photo: The easy way,” *Inpaint*. [Online]. Available: https://theinpaint.com/tutorials/pc/how-to-remove-unwanted-people-from-photo. [Accessed: 09-Oct-2021].

[6] ”This person does not exist,” *This Person Does Not Exist*. [Online]. Available: https://thispersondoesnotexist.com/. [Accessed: 09-Oct-2021].

[7] “Unified modelling language.” [Online]. Available: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/. [Accessed: 27- Feb- 2022].

[8] “IEEE Reference Guide.” [Online]. Available: https://ieeeauthorcenter.ieee.org/wp-content/uploads/IEEE-Reference-Guide.pdf. [Accessed: 27-Feb-2022].