Bilkent University

Department of Computer Engineering

SENIOR DESIGN PROJECT

Photonom

Low Level Design Report

Group Members	Omer Faruk Babademez	21601216
	Idil Hanhan	21601289
	Beyza Kalkanli	21600944
	Berfin Kucuk	21502396
	Kerem Yilmaz	21601223
Supervisor	Asst. Prof. Hamdi Dibeklioğlu	
Jury Members	Prof. Ozcan Ozturk	
	Assoc. Prof. Selim Aksoy	
Innovation Expert	Mehmet Surav	

Low Level Design Report
Feb 17, 2019
This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfilment of the requirements of the Senior Design Project course CS491/2.

Contents

1 Introduction **2**

1.1 Object design trade-offs 2

1.1.1 Understandability vs. Complexity 2

1.1.2 Portability vs. Development Time 3

1.1.3 Response Time vs. Memory 3

1.1.4 Security vs. Backup 3

1.2 Interface documentation guidelines 3

1.3 Engineering standards 4

1.4 Definitions, acronyms, and abbreviations 4

2 Packages **5**

2.1 Client 5

2.1.1 Presentation 6

2.2 Server 7

2.2.1 Modules 9

3 Class Interfaces **11**

3.1 Client 11

3.1.1 Presentation 11

3.2 Server 17

4 Glossary **22**

1 Introduction

Taking photos have turned into a daily practice, whether we are visiting a new country, celebrating a birthday or just having a normal day. In fact it has been estimated that more than 100 million photos and videos are uploaded to Instagram everyday [1]. Even though this is an impressive number, it does not even include the number of photos that people take for each upload. For some this number can be 20 and for other it might go up to 200 [2].
There are many reasons why one would have to take a lot of photos to get one shot they like. The light does not look right in one of them, someone walked behind them in another one, the building they want to be on the background did not fit the frame, they look too serious in that one and so on. Taking more photos can be a possible solution, but in most cases you might realise these problems when its too late. Going through your vacation photos to realise that picture you took in front of a historical church is missing the tip of the church is frustrating. Even if you do realise these problems when you have the chance to take more photos, some things cannot be changed. If it is raining and water droplets are in your lens, it is impossible to change the weather. And even if the problem is something you can change, why waste time taking tons of photos when it is possible for you to enhance or modify the ones you already have.
There are multiple tools that one can use to alter their photos. Advanced photoshop tools are often too complicated for occasional use. It is possible to find a tool that will remove a background item and then another one to alter your head position and/or facial expression and then find a filter in another app that reflects your style. In this case, it is not common to find all these features in one place and the users have to jump from one app to another to get what they want.
With Photonom, we are proposing a mobile application that will allow users to modify their photos interactively and in one place whilst conserving the true nature of the image itself. In the next section, Photonom will be described in more detail and constraints related to it will be explained. Then both the functional and nonfunctional requirements of Photonom will be explained. The references can be seen at the end of this report.

1.1 Object design trade-offs

1.1.1 Understandability vs. Complexity

Our application is designed with the aim of providing users an opportunity to complete their edits with only one application. In order to achieve this purpose we included multiple functionalities during the design process. However, it might be problematic for users to learn the use of all of the functionalities at the beginning. In order to find the balance between understandability and complexity, we limited the number of functionalities and added a user manual to the application.

1.1.2 Portability vs. Development Time

In order to ensure that Photonom is easily portable and available in multiple platforms, we decided to use Flutter framework which enables us to deploy the resulting product into multiple platforms. Although we all have experiences with Android development, we decided to continue with Flutter considering the trade off between the importance of portability and the longer development time.

1.1.3 Response Time vs. Memory

Photonom is an app that will be used for photo editing. Therefore at various stages of the app's usage, the images selected/edited by the user will have to be processed. Both the remote server and the user's device can be used for this purpose. Using the user's device will ensure better response time but will also require some memory of the device to be used. With the server we wouldn't have to rely on the memory of the user's device however the response time might suffer. In order to create a balance, we have decided that simple processes will be executed in the user's device to ensure fast response and the rest will be send to the server.

1.1.4 Security vs. Backup

When users upload their photos to Photonom, or take their photos using Photonom, these photos will persist in the server. Therefore it is crucial that we ensure our users personal data is secure and ensure it is only used for operations they have given consent for. Therefore, even though it would have been ideal for us to save every version of the photo by default, we have decided to make security a priority and only save such information if the user has given their permission.

1.2 Interface documentation guidelines

The table of interface documentation guideline are given in this section.

Table 1: Interface documentation guidelines

Class	Class Name
	Class description
Attributes	
Attribute	Attribute description
Methods	
method(args)	Method description

1.3 Engineering standards

We have used Unified Modelling Language (UML)[3] during the modelling of Photonom through class, package, activity and sequence diagrams. We have used Institute of Electrical and Electronics Engineers (IEEE) style [4] in citing references.

1.4 Definitions, acronyms, and abbreviations

- Object Removal: Tool of Photonom which can be used to remove an object from a photo.
- Image Stitching: Tool of Photonom which can be used to stitch multiple photos together to make one photo.
- Face Expression Modifier: Tool of Photonom which can be used to modify the face expression of a recognised face in the photo.
- Image Quality Evaluation: Tool of Photonom that evaluates selected photo(s) according to both aesthetics and technical details.
- Style Transfer: Tool of Photonom that modifies a photo to match the style of a provided style photo.
- Rain Removal: Tool of Photonom that removes the effects of rain from a photo.
- Head Position Modifier: Tool of Photonom which can be used to modify the head position of a recognised face in the photo.
- Server: The system which processes the images sent to it according to specified tool.

- OpenFace: Python and Torch implementation of face recognition with deep neural networks. It is used for face detection, face tracking and extraction of action units. [5]
- PyTorch: An open source machine learning library.
- GAN: Generative Adversarial Network. These networks create new instances based on the training data.[6]

2 Packages

Photonom is formed by the interaction of two systems: the client and the server.

2.1 Client

The client can be used through devices that has Android or iOS operating system. The client is the user interface layer of Photonom, where the user interact with his/her photos and edit them. The client contains the presentation subsystem. The presentation subsystem contains user interface elements for the user to interact with.

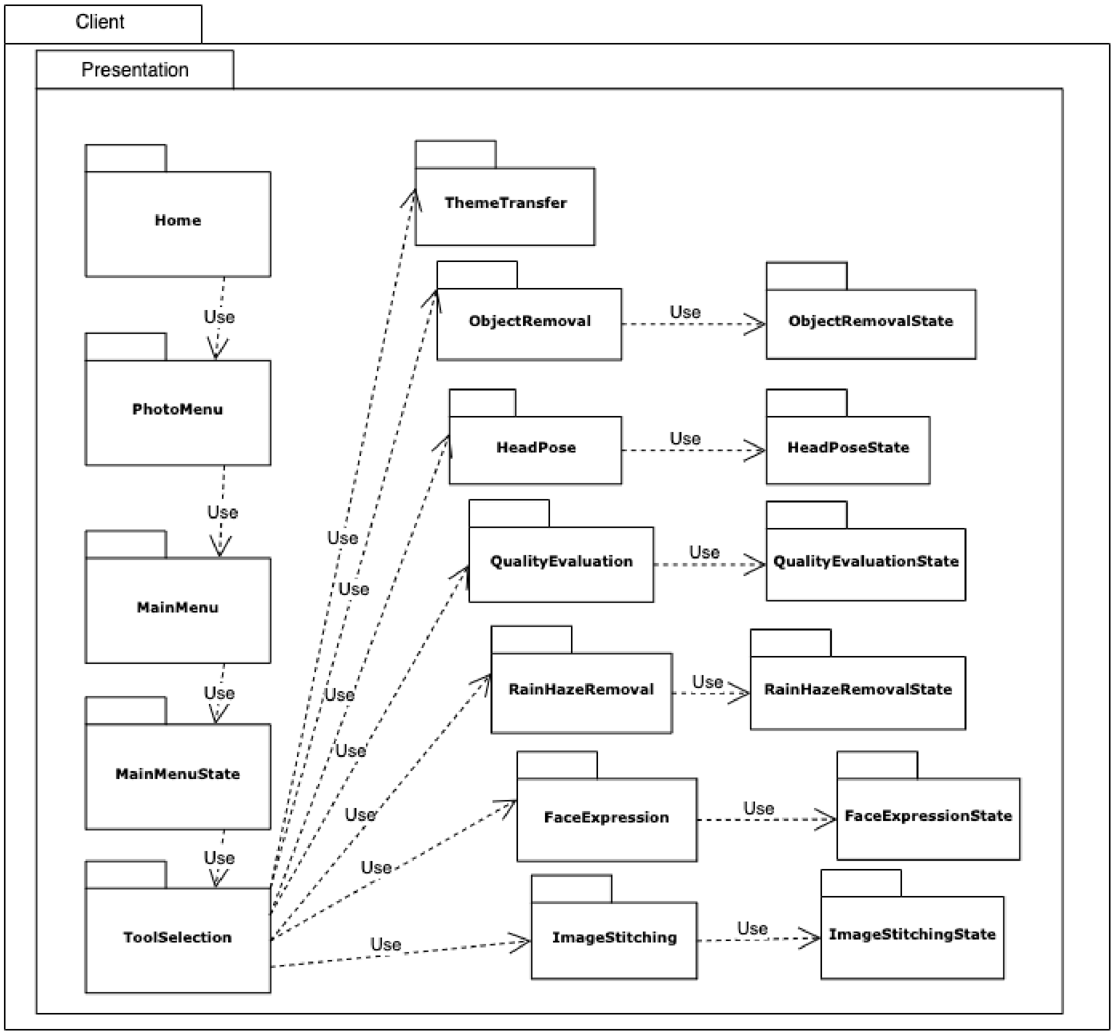

Figure 1: The hierarchy of modules inside the client system

2.1.1 Presentation

The presentation layer contains views and their event firing mechanisms.

Home: The first page that is seen when Photonom is opened. It contains options for selecting a new photo or continuing from a saved one.

MainMenu: Creates the menu page for selecting between camera and gallery.

MainMenuState: Displays the menu page.

PhotoMenu: Creates the connections for tools and their respective page.

ToolSelection: Creates the page that includes all of the tools of Photonom.

ThemeTransfer: Creates the page for theme transfer tool.

QualityEvaluation: Creates the page for quality evaluation tool.

QualityEvaluationState: Displays the image to be evaluated and it's score.

ImageStitching: Creates the page for image stitching tool.

ImageStitchingState: Displays the resulting image after the stitching operation is completed.

ObjectRemoval: Creates the page for object removal tool.

ObjectRemovalState: Displays the resulting image after object is removed.

RainHazeRemoval: Creates the page for rain and haze removal tool.

RainHazeRemovalState: Displays the resulting image after rain and/or haze is removed.

FaceExpression: Creates the page for face expression tool.

FaceExpressionState: Displays the resulting image after the face expression is modified.

HeadPose: Creates the page for head pose modification tool.

HeadPoseState: Displays the resulting image after the head pose is modified.

2.2 Server

All the processing happens in the server. An image with the required operations on that image arrive as a request to the server. Then, the server processes the image with the given modules and creates an output. The output is then sent back to the client as a response. The server consists RequestHandler, ImageOperator and the Modules layer. The Modules layer consists of 7 image operations such as image stitching and sytle transfer, etc.

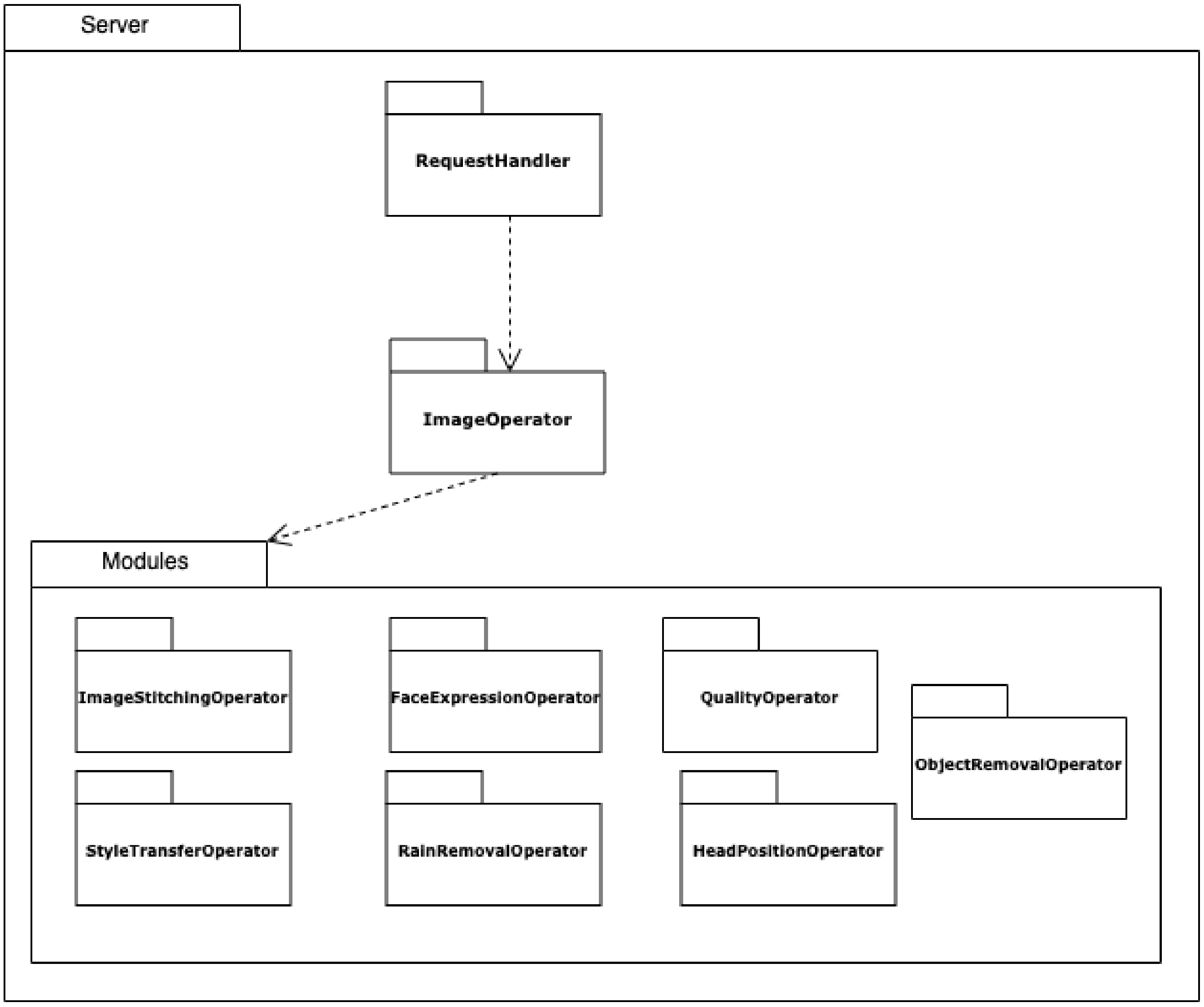

Figure 2: The hierarchy of modules inside the server system

RequestHandler: Handles client request by turning them into tasks and runs them on ImageOperator. Creates responses and sends them back to the client.

2.2.1 Modules

Module package contains the module used to edit the image(s).

ImageStitchingOperator: Stitches selected images into an image.

FaceExpressionOperator: Changes the expression of the selected face in the image.

QualityOperator: Evaluates the quality of the selected images.

StyleTransferOperator: Transfers the style of the secondly selected image to the first image and returns a new image.

RainRemovalOperator: Removes rain drops from the image and returns the output image.

HeadPositionOperator: Modifies the head position of the selected face in the image.

ObjectRemovalOperator: Removes the selected object from the image and fills the background accordingly.

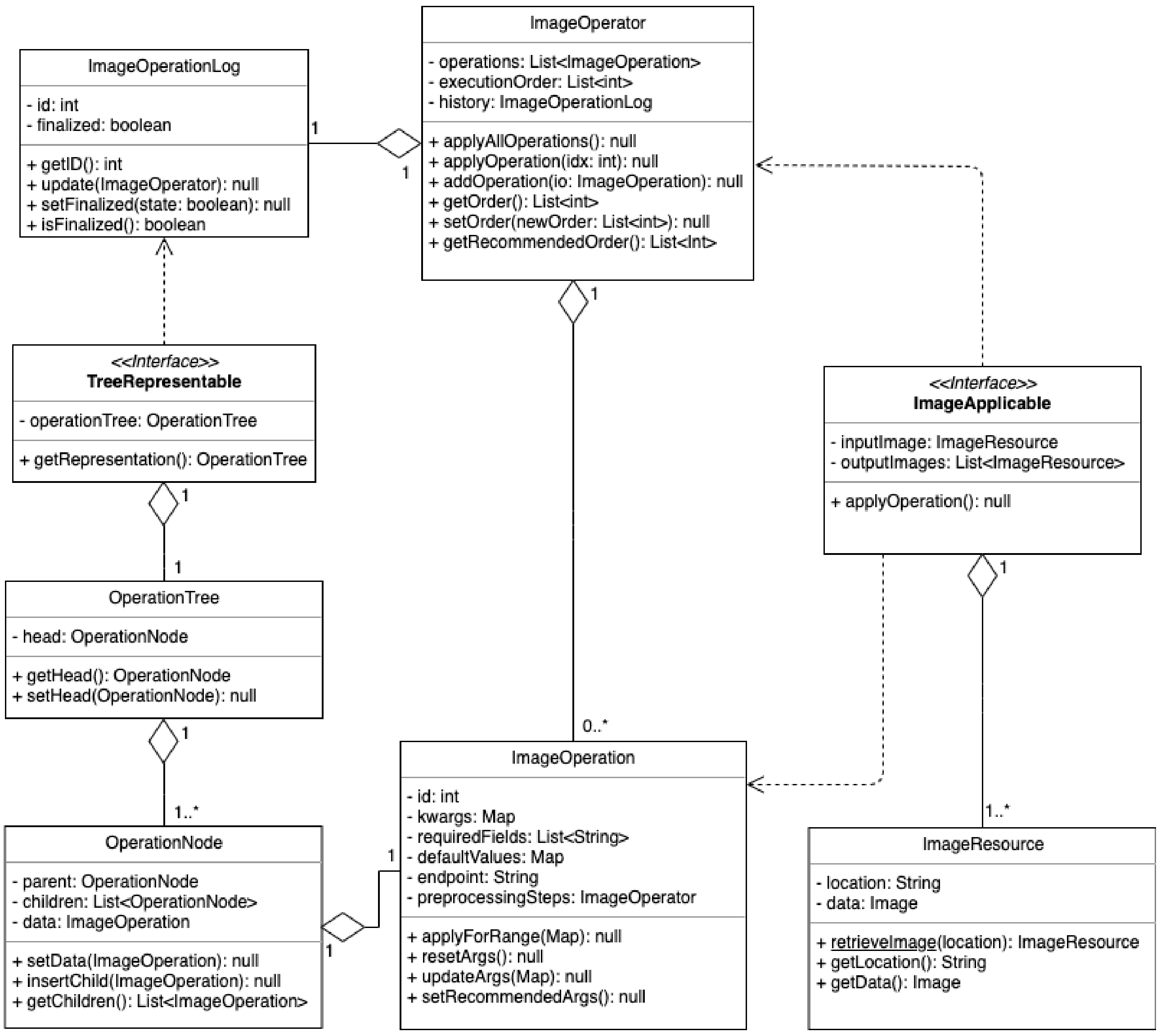
ImageOperationLog
- id: int
- finalized: boolean
+ getID(): int
+ update(ImageOperator): null
+ setFinalized(state: boolean): null
+ isFinalized(): boolean
ImageOperator
- operations: List<ImageOperation>
- executionOrder: List<int>
- history: ImageOperationLog
+ applyAllOperations(): null
+ applyOperation(idx: int): null
+ addOperation(io: ImageOperation): null
+ getOrder(): List<int>
+ setOrder(newOrder: List<int>): null
+ getRecommendedOrder(): List<Int>
<<Interface>>
TreeRepresentable
- operationTree: OperationTree
+ getRepresentation(): OperationTree
<<Interface>>
ImageApplicable
- inputImage: ImageResource
- outputImages: List<ImageResource>
+ applyOperation(): null
OperationTree
- head: OperationNode
+ getHead(): OperationNode
+ setHead(OperationNode): null
ImageOperation
- id: int
- kwargs: Map
- requiredFields: List<String>
- defaultValues: Map
- endpoint: String
- preprocessingSteps: ImageOperator
+ applyForRange(Map): null
+ resetArgs(): null
+ updateArgs(Map): null
+ setRecommendedArgs(): null
OperationNode
- parent: OperationNode
- children: List<OperationNode>
- data: ImageOperation
+ setData(ImageOperation): null
+ insertChild(ImageOperation): null
+ getChildren(): List<ImageOperation>
ImageResource
- location: String
- data: Image
+ retrieveImage(location): ImageResource
+ getLocation(): String
+ getData(): Image
1
1
1
1
1
0..*
1
1
1..*
1
1
1..*

Figure 3: ImageOperator Package UML Diagram

3 Class Interfaces

The class interfaces of client, presentation, and server are given in this section.

3.1 Client

The interface documentation of classes in Client side are given below.

3.1.1 Presentation

The interface documentation of classes in Presentation package are given below.

Table 2: Home Class Interface

Class	Home
	Builds the home page
Attributes	
None	None
Methods	
+Widget build()	Builds the page

Table 3: MainMenu Class Interface

Class	MainMenu
	Builds camera and gallery selection page
Attributes	
+String title	Title of app
Methods	
+_MainMenuState createState()	Adds the camera and gallery buttons

Table 4: MainMenuState Class Interface

Class	MainMenuState
	Displays camera and gallery selection page
Attributes	
None	None
Methods	
+Widget build()	Displays the camera and gallery buttons

Table 5: PhotoMenu Class Interface

Class	PhotoMenu
	Creates background and connections for tools
Attributes	
None	None
Methods	
+Widget build()	Provides the connections

Table 6: ToolSelection Class Interface

Class	ToolSelection
	Builds the page that includes all tool option
Attributes	
+Object selectedImage	Image the user has selected for editing
Methods	
+Widget build()	Builds the page

Table 7: ThemeTransfer Class Interface

Class	ThemeTransfer
	Builds the page for the Theme Transfer tool
Attributes	
-File themeImage	Image whose theme will be transfered
+File originalImage	Image the user has selected for editing
Methods	
+Widget build()	Builds the page

Table 8: QualityEvaluation Class Interface

Class	QualityEvaluation
	Creates the state page for quality evaluation
Attributes	
+String title	None
Methods	
+_QualityEvaluationState createState()	Creates the states

Table 9: QualityEvaluationState Class Interface

Class	QualityEvaluationState
	Shows the images and their scores
Attributes	
-String $_{e}valuationScore$	Calculated score of the image
+List<Asset> images	Images from camera or gallery
Methods	
+Widget build()	Gets the scores from the server and displays them

Table 10: ImageStitching Class Interface

Class	ImageStitching
	Builds the page for Image Stitching tool
Attributes	
+File selfieImage	Image the user has selected for editing
Methods	
+ImageStitchingState createState()	

Table 11: ImageStitchingState Class Interface

Class	ImageStitchingState
	Displays the resulting image after the stitching operation is completed
Attributes	
+File selfieImage	Image the user has selected for editing
+File panoromaImage	Image that will be stitched to the initially selected image
+File resultImage	The resulting image
Methods	
+Widget build()	Builds the page

Table 12: ObjectRemoval Class Interface

Class	ObjectRemoval
	Creates the state page for object removal
Attributes	
+String title	None
Methods	
+_ObjectRemovalState createState()	Creates the states

Table 13: ObjectRemovalState Class Interface

Class	ObjectRemovalState
	Shows the resulting images after object removal
Attributes	
+File currentImage	Image from camera or gallery
+List<File> resultingImages	Images that are returned from the server
Methods	
+Widget build()	Gets the resulting images from the server and displays them

Table 14: RainHazeRemoval Class Interface

Class	RainHazeRemoval
	Creates the state page for rain and haze removal
Attributes	
+String title	None
Methods	
+_RainHazeRemovalState createState()	Creates the states

Table 15: RainHazeRemovalState Class Interface

Class	RainHazeRemovalState
	Shows the resulting image after rain/haze removal
Attributes	
+File currentImage	Image from camera or gallery
+File resultingImage	Image that is returned from the server
Methods	
+Widget build()	Gets the resulting image from the server and displays it

Table 16: FaceExpression Class Interface

Class	FaceExpression
	Creates the state page for facial expression modification
Attributes	
+String title	None
Methods	
+_FaceExpressionState createState()	Creates the states

Table 17: FaceExpressionState Class Interface

Class	FaceExpressionState
	Shows the resulting image after facial expression modification
Attributes	
+File currentImage	Image from camera or gallery
+List<File> resultingImages	Images that are returned from the server
-int sliderCoefficient	Smiling coefficient
Methods	
+Widget build()	Displays slider, gets slider coefficient, gets the resulting image from the server and shows it

Table 18: HeadPose Class Interface

Class	HeadPose
	Creates the state page for head pose modification
Attributes	
+String title	None
Methods	
+_HeadPoseState createState()	Creates the states

Table 19: HeadPoseState Class Interface

Class	HeadPoseState
	Shows the resulting image after head pose modification
Attributes	
+File currentImage	Image from camera or gallery
+List<File> resultingImages	Images that are returned from the server
-int angle	Pose coefficient
Methods	
+Widget build()	Displays sphere, gets pose coefficient, gets the resulting image from the server and shows it

3.2 Server

The interface documentation of classes in Server side are given below.

Table 20: ImageOperator Class Interface

Class	ImageOperator
	Executor class for applying operations
Attributes	
-List<ImageOperation> operations	List of operations supported by Photonom
-List<int> executionOrder	The order in which operations will be handled
-ImageOperationLog history	An ImageOperationLog that contains all of the changes made by the user
Methods	
+void applyAllOperation()	Apply all of the operations according to the order specified in executionOrder
+void applyOperation(int idx)	Applys the operation indicated by the given id
+void addOperation(ImageOperation io)	Adds a new operation to Photonom
+List<int> getOrder()	Get the order of execution.
+void setOrder(List<int> newOrder)	Set the execution order
+List<int> getRecommendedOrder()	Get the recommended order, meaning the default order of execution

Table 21: ImageOperation Class Interface

Class	ImageOperation
	Represents an operation with its parameters
Attributes	
-int id	Id of the operation ranging from 1 to 7.
-Map kwargs	Arguments necessary for image operation
-List<String> requiredFields	The necessary fields that must be in *kwargs*
-Map defaultValues	The default values for arguments of the image operation
-String endpoint	URL to the one of the servers endpoints where the operation for the operation will be sent to.
-ImageOperator preprocessingSteps	Operations that should be applied before the actual operation
Methods	
+void applyForRange(Map)	Applies operation with given range of input values
+void resetArgs()	Reset the arguments of the operation.
+void updateArgs(Map)	Update the arguments of the operation.
+void setRecommendedArgs()	Set the arguments of the operation to default values.

Table 22: ImageApplicable Class Interface

Class	ImageApplicable
	Interface for anything that can be applied to an image
Attributes	
-ImageResource inputImage	Input image
-List<ImageResource> outputImages	Output images
Methods	
+void applyOperation()	Operation definition

Table 23: ImageResource Class Interface

Class	ImageResource
	Represents an image
Attributes	
-String inputImage	Location of the image.
-image data	The image itself
Methods	
+ImageResource retrieveImage(location)	Gets the image from the given location
+String getLocation()	Returns the location of the image
+image getData()	Returns the matrix of the image

Table 24: ImageOperationLog Class Interface

Class	ImageOperationLog
	Holds an action log for an image
Attributes	
-int id	Id of the image operation
-boolean finalized	Whether editing of the image is completed
Methods	
+int getID()	Gets the id of the image operator
+void update(ImageOperator io)	Update the log to include the latest image operation
+void setFinalised(boolean state)	Updates the finalised attribute according to given state
+boolean isFinalized()	Checks if editing of the image is finalized.

Table 25: TreeRepresentable Class Interface

Class	TreeRepresentable
	Interface for anything that can be represented in a tree structure
Attributes	
-OperationTree operationTree	A tree that stores operations and their order
Methods	
+OperationTree getRepresentation()	Returns a representation of the tree

Table 26: OperationTree Class Interface

Class	OperationTree
	Data structure for representing the life of an image
Attributes	
-OperationNode head	Head node
Methods	
+OperationNode getHead()	Returns head node
+void setHead(OperationNode)	Updates the head

Table 27: OperationNode Class Interface

Class	OperationNode
	Represents an operation in OperationTree
Attributes	
-OperationNode parent	Parent node
-List<OperationNode> children	Children nodes
-ImageOperation data	Related operation object
Methods	
+void setData(ImageOperation)	Sets data attribute
+void insertChild(ImageOperation)	Inserts an operation as a child
+List<ImageOperation> getChildren()	Returns all children nodes

4 Glossary

Photonom: A word derived from the merging of two words: photoshop and autonom.

References

[1] "Instagram by the numbers (2019): Stats, demographics fun facts," 2019. [Online]. Available: https://www.omnicoreagency.com/instagram-statistics/

[2] J. Brucculieri, "4 instagrammers show us how many photos they took before nailing 'the shot'," 2018. [Online]. Available: https://www.huffpost.com/entry/instagramphoto-camerarolls_n_5ac4ed48e4b063ce2e58131f

[3] "Unified modeling language," 2020. [Online]. Available: https://www.uml.org/

[4] "How to cite references: Ieee documentation style," 2020. [Online]. Available: https://ieee-dataport.org/sites/default/files/analysis/27/IEEE%20Citation%20Guidelines.pdf

[5] "Openface," 2019. [Online]. Available: https://cmusatyalab.github.io/openface/

[6] "Introduction | generative adversarial networks," 2019. [Online]. Available: https://developers.google.com/machine-learning/gan