Bilkent University

Department of Computer Engineering

Senior Design Project II

DeePaint: Smart Photo Editing on Your Pocket

Low Level Design Report

Project Group Members:

Yavuz Bakman - 21703309

Hande Sena Yılmaz - 21703465

Alperen Öziş - 21703804

Zübeyir Bodur - 21702382

Duygu Nur Yaldız - 21702333

Supervisor: Uğur Doğrusöz

Jury Members: Tağmaç Topal and Erhan Dolak

Low Level Design Report

Feb 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 CS492.

Table of Contents

[This part is intentionally left blank]

Low Level Design Report

DeePaint: Smart Photo Editing on Your Pocket

1. Introduction

During the last decade, social media applications have become widely used and part of our daily lives due to the easy accessibility of mobile devices and the increasing quality of their cameras. All around the world, people take photos constantly and share them on social media. For instance, 1074 photos are uploaded on Instagram every second [1]. While sharing photos, people have an urge to get rid of the imperfections in the photo such as a stranger in the background of a selfie or a car in a beautiful scene.

However, applying these changes requires knowledge and experience in photoshop. A regular person can not achieve to remove an object from a photo in a realistic way. Nevertheless, this became a need for regular people on a daily basis. Therefore, applications that make these adjustments automatically have gained more and more demand. Those applications with a user-friendly interface, fast modification speed, and low error rate in modification (i.e. more realistic results) are preferred by the users.

Considering the demand for easy and accurate photoshop applications, we came up with the DeePaint mobile application. In this report, we provide an overview of the architecture and design of the application, as well as the purpose of the system, design goals, consideration of various factors in engineering design, and teamwork details.

1.1. Object Design Trade-offs

1.1.1. Usability vs Functionality

The main purpose of DeePaint is to ease the photoshop process and thus usability is maybe the most important design goal we have. However this

doesn’t stop us from offering complex features. We are still able to manage such complex things like filling the background of the removed object but we are doing it in a way that minimum user effort is involved thanks to the automatization coming with deep learning techniques.

1.1.2. Compatibility vs Extensibility

Computer vision is an area where almost every day there is a new advancement. So there are many aspects of our project where we can improve our current features or add new ones. Therefore extensibility is something we have to achieve if we want to keep our app up-to-date and compete in the market. Compatibility is also an important factor to reach a higher number of users. Though we will be releasing only on Android for now, we can easily go live on other operating systems thanks to java being an environment friendly programming language. We just need to change OS dependent features while keeping the underlying structure the same to export our project, say iOS. Using java for the app and AWS servers for the backend will help us to easily extend new features while keeping main functionalities compatible with different operating systems.

1.1.3. Space vs Time

We will be using big models which would be a burden to store locally. It also wouldn't be fast to execute since they require high computation power which most of the mobile devices lack. Keeping our models in cloud servers which offer us strong GPUs will enable us to both avoid the redundant data storage and achieve faster computations. Though there will be a time overhead sending and receiving requests to/from servers, it is an overhead we can tolerate because the benefits will outdo the disadvantages.

1.2. Interface Documentation Guidelines

The following interface will be used in the rest of the report:

Class Name

Class name is given as subtitle and explained underneath the title.

Attributes:

- Atributes are listed below Attributes section. Attribute types such as int, float is given before the attribute name.

Methods:

- Methoda are listed below Methods section. Method return types given at the beginning, while required parameters are biven inside the paranthesis.

1.3. Engineering Standards

In this report, UML guidelines are followed to visualize our system design [2]. Also, we used IEEE referencing style for citations [3].

1.4. Definitions, acronyms, and abbreviations

API

Application Programming Interface. The connection between computers or between computer programs. It is a type of software interface, offering a service to other pieces of software.

AWS

Amazon Web Services. Reliable, scalable, and inexpensive cloud computing services provided by Amazon. Remote servers can be rented through this service.

CPU

Central Processing Unit. The main processor or just processor is the electronic circuitry that executes instructions comprising a computer program.

GUI

Graphical User Interface. An interface through which a user interacts with electronic devices such as computers, hand-held devices, and other appliances.

2. Packages

According to the system's high-level design, there are two main layers, client and server, consisting of packages that include their class interfaces. Since there were no changes made in the high-level design of the system, the report will continue with class diagrams of each package. External packages (third parties) are ignored for simplicity.

2.1. client.view

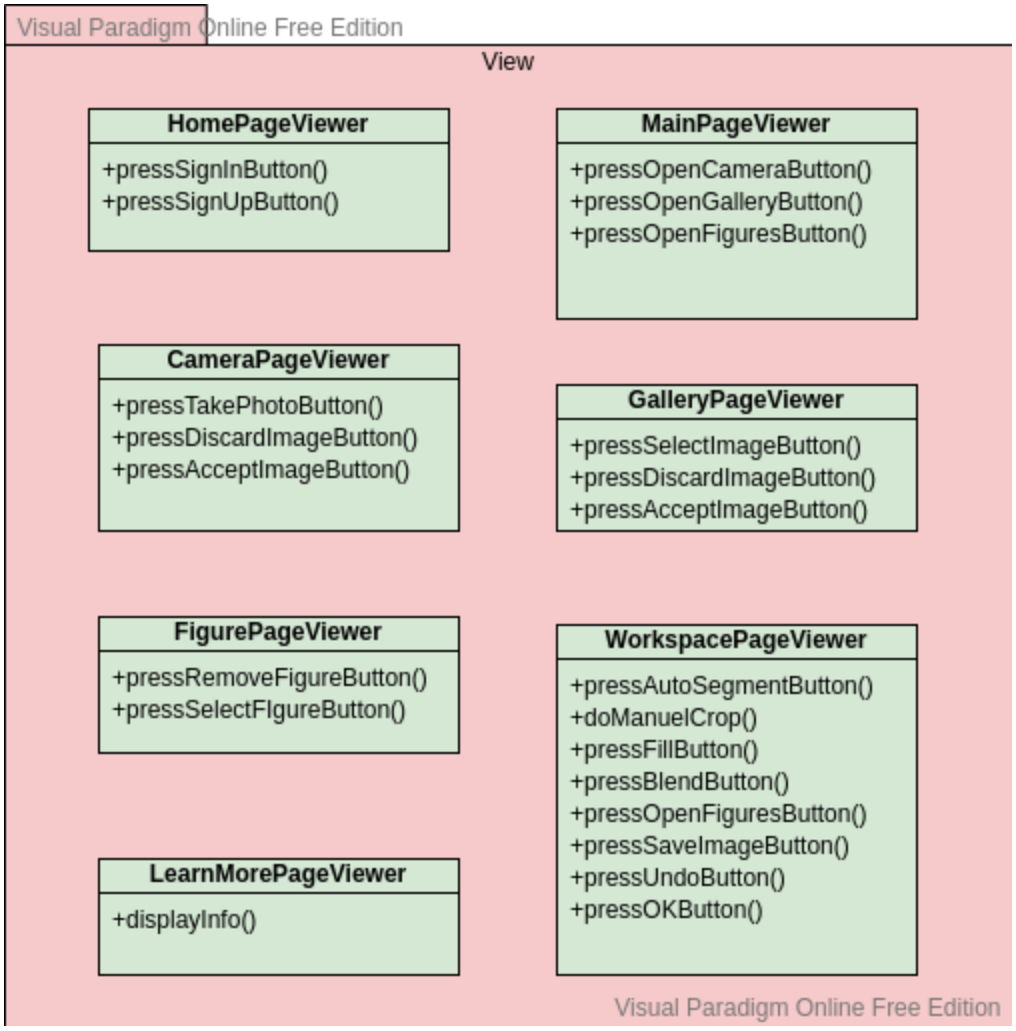

Figure 1: Class diagram for the client.view package

2.2. client.controller

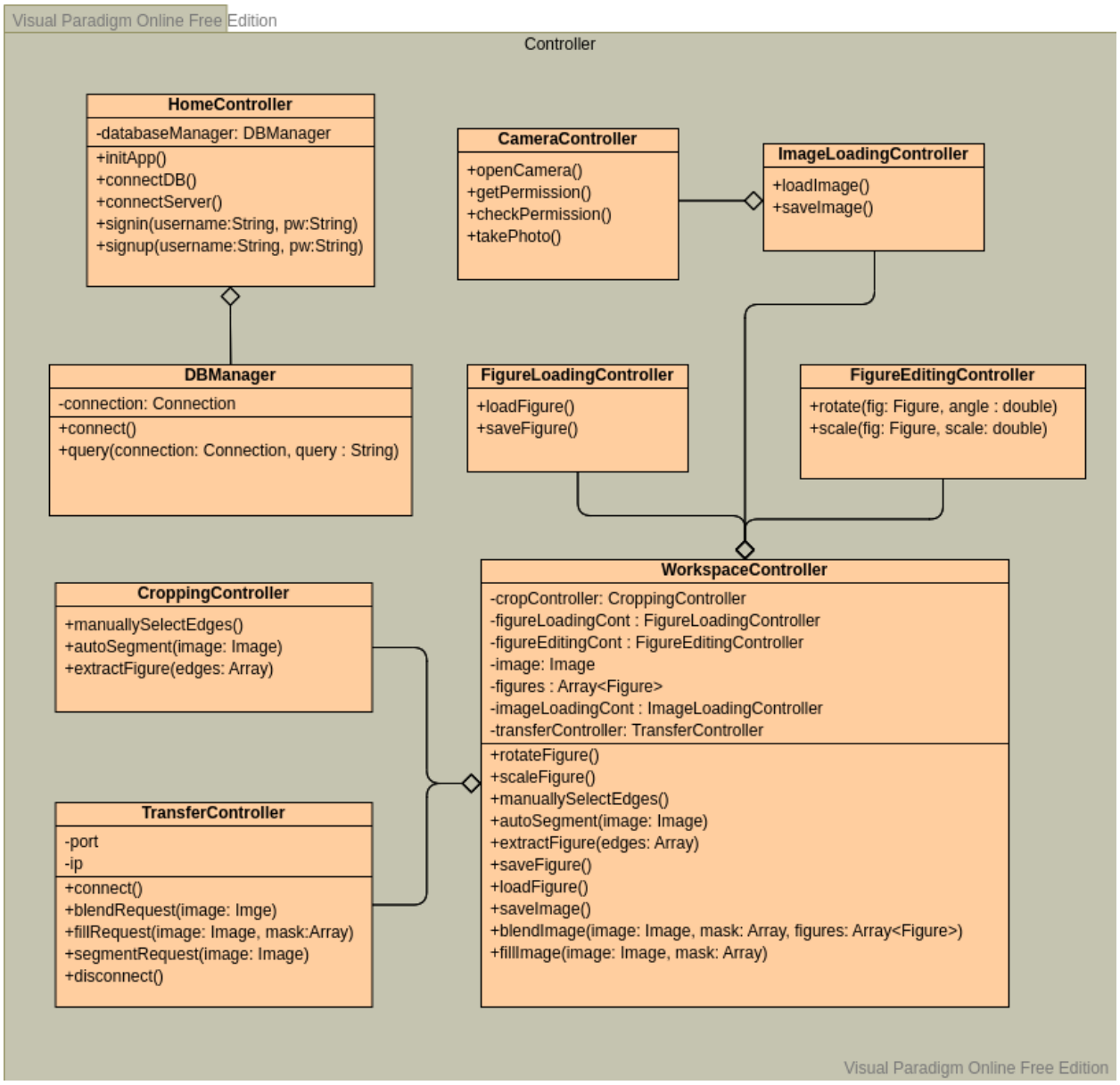

Figure 2: Class diagram for the client.controller package

2.3. client.data

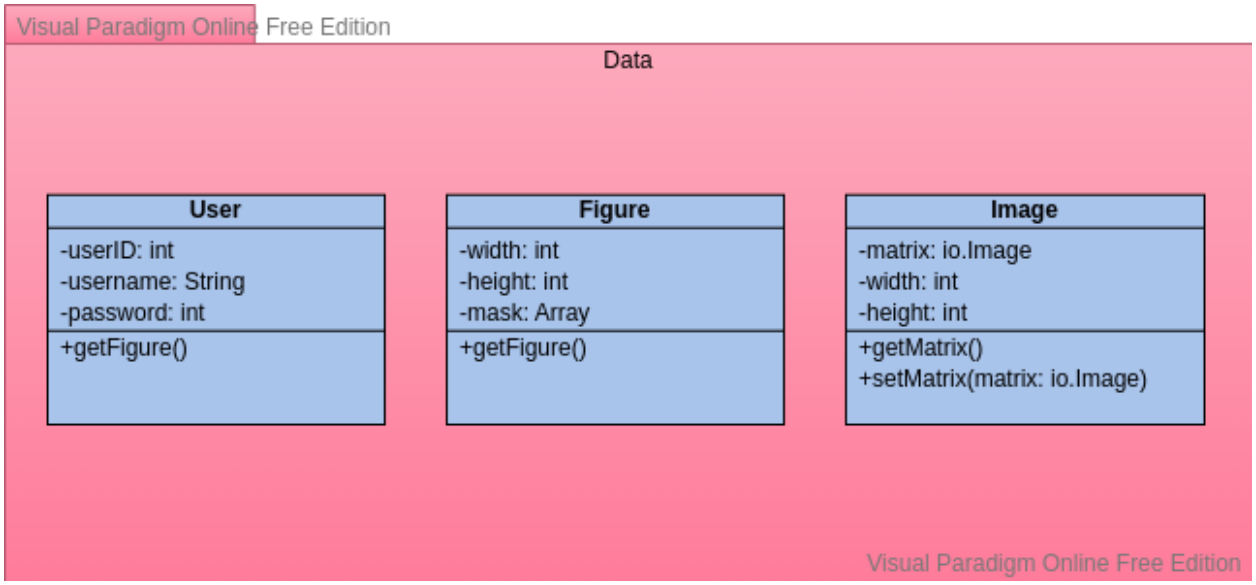

Figure 3: Class diagram for the client.data package

2.4. server.applicationLogic

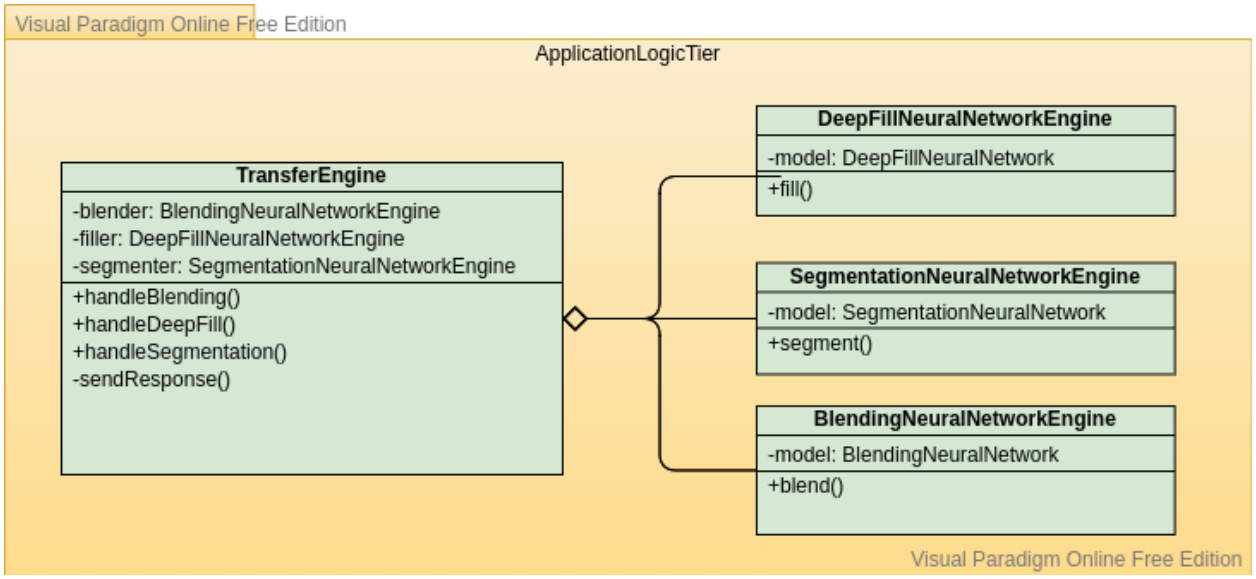

Figure 4: Class diagram for server.applicationLogic package

2.5. server.data

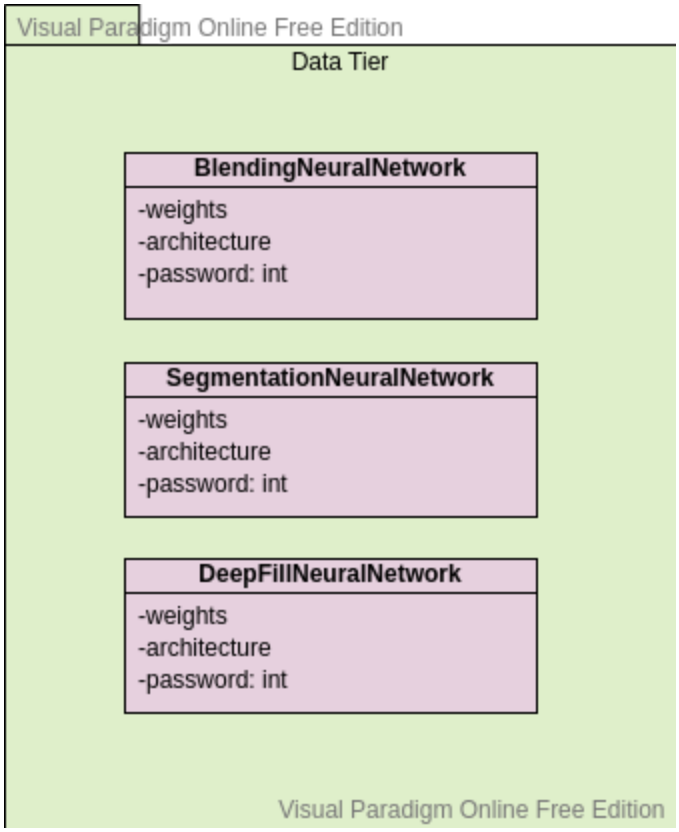

Figure 5: Class diagram for server.data package

3. Class Interfaces

Below is the list of the interfaces of the system, for each layer and tier.

3.1. Client Layer

Below is the list of the interfaces of the system, for each tier.

3.1.1. client.view

3.1.1.1. HomePageViewer

HomePageViewer is responsible for displaying the welcome page when the application is started and allowing users to sign in/signup for the app.

Methods:

- **protected void pressSignInButton():** Directs app to sign in page.
- **protected void pressSignUpButton():** Directs app to sign up page.

3.1.1.2. MainPageViewer

MainPageViewer is responsible for showing the initial actions a user can take, i.e. opening the camera, displaying saved figures, or exporting an image from the gallery.

Methods:

- **protected void pressOpenCameraButton():** Directs app to Camera Page and opens the camera for taking photos.
- **protected void pressOpenGalleryButton():** Directs app to Gallery Page and lists the pictures already exists in the users' gallery.
- **protected void pressOpenFiguresButton():** Directs app to previously saved figures page for users to check them.

3.1.1.3. CameraPageViewer

CameraPageViewer is responsible for displaying the camera and allowing the user to take photos, which they will be editing in the workspace view.

Methods:

- **protected void pressTakePhotoButton():** Takes the photo and directs it to the state where users decide to accept or discard the image and take a new one.
- **protected void pressDiscardImageButton():** Discards the taken image and directs to the photo-taking page.
- **protected void pressAcceptImageButton():** Accepts the taken image and directs to the main editing page.

3.1.1.4. GalleryPageViewer

GalleryPageViewer is responsible for showing the initial actions a user can take, i.e. opening the camera, displaying saved figures, or exporting an image from the gallery.

Methods:

- **protected void pressSelectImageButton():** Selects the already existing image from the gallery of the user and directs it to the state where users decide to accept or discard the image and choose a new one.
- **protected void pressDiscardImageButton():** Discards the selected image and directs to the gallery page to choose a new one.
- **protected void pressAcceptImageButton():** Accepts the selected image and directs to the main editing page.

3.1.1.5. FigurePageViewer

FigurePageViewer is responsible for opening the saved figure page whenever the user presses the Saved Figures button on the Main Page. It will show the list of saved figures so far by the user.

Methods:

- **protected void pressRemoveFigureButton():** Removes the selected figure from the figure page.
- **protected void pressSelectFigureButton():** Allows users to select from multiple figures to remove them.

3.1.1.6. LearnMorePageViewer

LearnMorePageViewer includes the information about the app and displays the buttons, “Contact Us”, “FAQ” and “Team”. The “Contact Us” button will redirect the user to the DeePaint website. The FAQ

button will redirect the FAQ page of the DeePaint application. Finally, the Team button will display a modal that shows the names of developers who contributed to this project.

Methods:

- **protected void displayInfo():** Displays all the information included in LearnMorePage by directing to "Contact Us", "FAQ" or "Team" based on the preference of the user.

3.1.1.7. WorkspacePageViewer

WorkspacePageViewer is the main page where the users will interact with the photo and edit them. It includes GUI items for displaying the saved figures, segmenting the image, manually editing the image, saving the edited images, and more for the manipulation of the image.

Methods:

- **protected void pressAutoSegmentButton():** Starts auto segmentation on the image and displays the segmented figures afterward.
- **protected void doManuelCrop():** Allows users to manually crop the figures by hand instead of auto segmentation, then display the segmented figure.
- **protected void pressFillButton():** Fills the removed areas smoothly after figures are removed by using the deep fill technique.
- **protected void pressBlendButton():** Operates the image blending of the figures within the specific area of the main editing image that the user desires.
- **protected void pressOpenFiguresButton():** Opens a panel where the already saved figures are resided and allows users to choose from the figures that

they want to use for manipulation on the image editing.

- **protected void pressSaveImageButton():** Saves the edited image on the main editing page to users' gallery.
- **protected void pressUndoButton():** Undos the changes that have been made on the edited image one by one in each press.
- **protected void pressOKButton():** Approves the changes made so far for the image editing and manipulation.

3.1.2. client.data

3.1.2.1. User

This class represents the User table of the application's database as entities.

Attributes:

- **private int userID:** Stores the id of the user.
- **private String username:** Stores the name of the user.
- **private int password:** Stores the password of the user.

3.1.2.2. Figure

This class represents the segmented figures as a figure object for the local storage.

Attributes:

- **private int width:** Stores the width of the figure.
- **private int height:** Stores the height of the figure.
- **private Array mask:** Stores the filtered/labeled image representation of the figures.

Methods:

- **public Figure getFigure():** Returns the specified figure from the storage.

3.1.2.3. Image

This class represents the edited images as an image object for local storage.

Attributes:

- **private int width:** Stores the width of the image.
- **private int height:** Stores the height of the image.
- **private io.Image matrix:** Stores the matrix representation of the image pixel by pixel.

Methods:

- **public io.Image getMatrix():** Returns the matrix representation of the edited image.
- **public void setMatrix(io.Image matrix):** Sets the matrix representation of the edited image.

3.1.3. client.controller

TransferController, DBManager, and WorkspaceController are singleton classes, and WorkspaceController is the façade class.

3.1.3.1. WorkspaceController

This class manages all of the operations related to the workspace the user is interacting with, such as image editing, selecting a figure, saving the image into the gallery, etc.

This class is also singleton, as façade classes usually have only one instance at any given time. This is also true for the DeePaint.controller.WorkspaceController as well, there will be only one workspace.

Attributes:

- **private CroppingController cropController:** The cropping controller the workspace will use. It will provide operations for retrieving manually selected edges and retrieving the figure.
- **private FigureLoadingController figureLoadingCont:** The figure loading controller the workspace will use. It will manage to load a selected figure and save a specific figure.
- **private FigureEditingController figureEditingCont:** The figure editing controller the workspace will use. Will provide the workspace operations for rotating and scaling a figure.
- **private Image image:** The image the workspace is currently working on.
- **private ArrayList<Figure> figures:** The list of figures in the device so far. Will be stored in the WorkspaceController as an attribute as well to easily access those figures. If there is a change in those lists of figures, the device storage will be updated as well.
- **private ImageLoadingController imageLoadingCont:** The image loading controller will manage operations related to the gallery, such as saving or loading an image locally.
- **private static TransferController = null:** The transfer controller will send the necessary requests to AWS for segmentation, blending, and filling (inpainting). It is a singleton class since connection classes are usually singletons.

Methods:

- **public void rotateFigure(Figure f, double angle)**: Encapsulated version of the `FigureEditingController.rotate()` method.
- **public void scaleFigure(Figure f, double scale)**: Encapsulated version of the `FigureEditingController.scale()` method.
- **public void manuallySelectedEdges()**: Encapsulated version of the `CroppingController.manuallySelectedEdges()` method.

- **public void autoSegment(Image image):** Sends the AWS request for segmenting the image in the workspace, using the workspace's `TransferController`. After receiving the response, processes the `ArrayList<Image> binaryEdges`, so that at the end, those edges are placed at the top of the image in the View (not in the actual image in the controller, as we want to select those segmented lines and do operations with them, such as filling, saving the figure, etc.).
- **public Figure extractFigure(byte[] edges):** Encapsulated version of `CroppingController.extractFigure()`.
- **public void saveFigure(Figure f):** Basically encapsulates `FigureLoadingController.saveFigure()`.
- **public void loadFigure(String figureUri):** Basically encapsulates `FigureLoadingController.loadFigure()`, and also communicates with the view.
- **public void saveImage(Image image):** This method also encapsulates `ImageLoadingController .saveImage()`.
- **public void blendImage(Image sourceImage, byte[] mask, ArrayList<Figure> figures) :** Sends the request to the AWS, so that the source image is blended into the image in the workspace. Basically, this method encapsulates the request transfer controller sends. The mask will be either drawn by the user, or a mask that is a result of a segmentation operation.
- **public void fillImage(Image image, byte[] mask):** Sends the request to the AWS, so that the image in the workspace is inpainted by the given mask, which will be either provided by the user or retrieved from a segmentation.

3.1.3.2. TransferController

This class will manage the connection between the AWS and the application. It will be also responsible

for sending the HTTP requests to the server and receiving the HTTP response.

Attributes:

- **private String port:** The port number of the connection.
- **private String IP:** The IP address of the connection.

Methods:

- **public void connect():** Establishes the connection between the AWS and application, using the port number and IP address of this singleton instance.
- **public void disconnect():** Removes the connection between the AWS and the application.
- **public Image blendRequest(Image dest, Image src, Image mask):** Sends an image blending request to AWS to execute.
- **public Image fillRequest(Image dest, byte[] mask):** Sends an image filling request to AWS to execute.
- **public Image segmentRequest(Image dest, byte[] mask):** Sends a segmentation request to AWS to execute. Returns an image that represents the segmented regions.

3.1.3.3. CroppingController

This class is responsible for manual cropping operations.

Methods:

- **public void manuallySelectEdges() :** Opens the palette in the View so that the user can manually select their own edges.
- **public void autoSegment(Image image) :** Automatically segments the image ?

- **public Figure extractFigure(Image src, byte[] edges):** Given an array of edges, extracts a Figure from the source Image.

3.1.3.4. FigureLoadingController

This class is responsible for file operations regarding the data.Figure class, which is an extension of the implementation of Images in Java, ImageIO.

Methods:

- **public Figure loadFigure(String figureUri):** Returns a selected figure from the gallery, where the selected figure is specified by `figureUri`.
- **public void saveFigure(Figure f):** Save a given figure f, to the constant path DeePaint app will use: "`~/DeePaint/Workspace/Figures/figure_file_name.figure_extension`".

3.1.3.5. FigureEditingController

This class is responsible for manipulating figures, namely rotation and scaling.

Methods:

- **public void rotate(Figure f, double angle):** Given a figure f and an angle alpha, rotates f alpha degrees, in a counterclockwise direction.
- **public void scale(Figure f, double scale):** Given a figure f and a scaling coefficient s, ranging from 0 to `Integer.MAX_VALUE`, scales the figure so that size is multiplied by that scale. If the resulting figure is too small or too large, an exception will be thrown.

3.1.3.6. ImageLoadingController

This class is responsible for file operations on client.data.Image class, which is a

not-so-straightforward implementation of WhatsApp Stickers in Java.

Methods:

- **public Image loadImage(String imageUri):** Returns a selected figure from the gallery, where the selected figure is specified by `figureUri`.
- **public void saveImage(Image img):** Save a given figure f, to the constant path DeePaint app will use: "`~/DeePaint/Workspace/Images/image_file_name.image_extension`".

3.1.3.7. CameraController

This class is responsible for connecting to the camera of the device of the user and letting the user take a picture.

Methods:

- **public void openCamera():** Initiates the camera of the device of the user based on the permission to let the user take a picture.
- **public void getPermission():** Gets the users' permission for asking the device to initiate the camera for taking pictures.
- **public void checkPermission():** Checks whether the permission that the user gives is stable before connecting to the camera of the device.
- **public void takePhoto():** If the camera is reached by the application, this allows users to take pictures of their preference to further use it for editing.

3.1.3.8. HomeController

This class will manage the user log-in and sign-up operations and will use the DBManager to access the database of users. The DBManager is a singleton that HomeController will use.

Attributes:

- **private static DBManager databaseManager = null:** This is the singleton instance is responsible for establishing a connection between the database and the mobile device, using the JDBC driver.

Methods:

- **public void initApp():** Communicates with the view classes so that the application is initialized.
- **public void connectDB():** Establishes the connection between database and application using a hardcoded connection string, which consists of a URL, DB user, and DB password.
- **public void connectServer():** Establishes the connection between the AWS and application (?)
- **public void signIn(String username, String pw):** Tells the DBManager to execute such a query so that the user with the given information exists, and credentials are correct. After that, the user will have their own session as the application is open.
- **public void signUp(String username, String pw):** Tells the DBManager to execute such a query so that the user with the given information does not exist, and a row in the User table is created. After that, the user will be asked to sign in, so this method will also redirect the view in such a way.

3.1.3.9. DBManager

This singleton class is responsible for establishing a connection and then storing it, using the JDBC driver for MySQL. It can also execute any SELECT query, and then retrieve the results.

Attributes:

- **private Connection connection:** The connection established will be stored in this attribute.

Methods:

- **public ResultSet query(Connection c, String query):** Executes any MySQL query using the connection provided. If there is a result, it is returned. If there is any MySQL exception, they are also handled.
- **public ArrayList<T> processResultSet(ResultSet rs):** A ArrayList<T> processResultSet() method will be implemented if necessary for SELECT queries, that is if the database gets larger.

3.2. Server Layer

Below is the list of the interfaces of the system, for each tier.

3.2.1. server.applicationLogic

3.2.1.1. BlendingNeuralNetworkEngine

This class is responsible for the Blending task.

Attributes:

- **private model BlendingNeuralNetwork:** This attribute stores the blending neural network parameters.

Methods:

- **public Image blend(Image i, Figure f, Arraylist<int> mask):** This method makes the neural network work and applies the blending task with the given parameters.

3.2.1.2. SegmentationNeuralNetworkEngine

This class is responsible for the Segmentation task.

Attributes:

- **private model SegmentationNeuralNetwork:** This attribute stores the segmentation neural network parameters.

Methods:

- **public ArrayList<int> segment(Image i):** This method makes the segmentation.

3.2.1.3. DeepFillNeuralNetworkEngine

This class is responsible for the Deep Fill task.

Attributes:

- **private model DeepFillNeuralNetwork:** This attribute stores the deep fill neural network parameters.

Methods:

- **public Image fill(Image i, ArrayList<int> mask):** This method makes the neural network works and fill the selected area

3.2.1.4. TransferEngine

This class gets the request from the client and make the appropriate models work to handle the request and send the response to the client

Attributes:

- **private blender BlendingNeuralNetworkEngine:** This attribute provides the methods for the blending task.
- **private filler DeepFillNeuralNetworkEngine:** This attribute provides the methods for the deep filling task.
- **private filler SegmentationNeuralNetworkEngine:** This attribute provides the methods for the segmentation task.

Methods:

- **public Image handleBlending(Image i, Figure f, ArrayList<int> mask):** This method handles the blending task
- **public Image handleDeepFill(Image i, ArrayList<int> mask):** This method handles the deep filling task.
- **Public ArrayList<int> mask handleSegmentation (Image i):** This method handles the segmentation task
- **public void sendResponse(Image i, Figure f, ArrayList<int> mask):** This method sends the output of neural networks into the client

3.2.2. server.data

We will be holding the neural network models in the cloud since they are big files and it would be redundant to store them locally. Moreover running these models requires high computing power which most of the mobile devices lack. Therefore holding these models in the cloud helps us to avoiding redundant data storage and better computation power.

3.2.2.1. BlendingNeuralNetwork

This is the NN model responsible for the blending task.

Attributes:

- **Private Weights:** This represents the learned parameters by the model to do the task.
- **Private Architecture:** This represents the underlying architecture of the blending NN.
- **Private Password**: This is the password needed to access the models.

3.2.2.2. SegmentationNeuralNetwork

This is the NN model responsible for the segmentation task.

Attributes:

- **Private Weights:** This represents the learned parameters by the model to do the task.
- **Private Architecture:** This represents the underlying architecture of the segmentation NN.
- **Private Password**: This is the password needed to access the models.

3.2.2.3. DeepFillNeuralNetwork

This is the NN model responsible for the deepfilling task.

Attributes:

- **Private Weights:** This represents the learned parameters by the model to do the task.
- **Private Architecture:** This represents the underlying architecture of the deepfill NN.
- **Private Password**: This is the password needed to access the models.

4. References

[1]"Instagram by the Numbers: Stats, Demographics & Fun Facts," Omnicore Agency, Jan 3, 2021 [Online]. Available: https://www.omnicoreagency.com/instagram-statistics/ [Accessed Oct 10, 2021].

[2] "Unified modelling language." https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/. [Accessed: 27- Feb- 2022].

[3] "Ieee reference guide." https://ieeeauthorcenter.ieee.org/wp-content/uploads/IEEE-Reference-Guide.pdf. [Accessed: 27- Feb- 2022].