Bilkent University

Department of Computer Engineering

Senior Design Project

EyeSight

Low-Level Design Report

Group Members: Cemil Şişman, Derviş Mehmed Barutcu, A. A. M. Jubaeid Hasan Chowdhury, Mustafa Azyoksul, Onur Mermer

Supervisor: Varol Akman

Jury Members: Fazlı Can, Hamdi Dibeklioğlu

Low-Level Design Report
Oct 5, 2020

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 Contents

1. Introduction **4**
1.1. Object Design Trade-offs 4
1.1.1. Usability vs Functionality 4
1.1.2. Performance vs Complexity 4
1.1.3. Reliability vs Functionality 4
1.1.4. Scalability vs Cost 5
1.2. Interface Documentation Guidelines 5
1.3. Engineering Standards 5
1.4. Definitions, Acronyms, and Abbreviations 6

2. Packages **7**
2.1. Presentation Layer 7
2.1.1. Views 7
2.1.1.1. Camera View 7
2.1.1.2. SignUp/Login View 7
2.1.1.3. Startup Page View 8
2.1.1.4. Settings View 8
2.1.1.5. Main Page View 8
2.1.1.6. Emergency Contact View 8
2.1.2. Controllers 8
2.1.2.1. EyeSightClient 8
2.1.2.2. VisionManger 8
2.1.2.3. VoiceManager 8
2.1.2.4. LocalDataManager 8
2.1.2.5. CommunicationManager 8
2.1.3. Models 9
2.1.3.1. User 9
2.1.3.2. Contact 9
2.1.3.3. Frame 9
2.1.3.4. EyeSightObject 9
2.2. Server Layer 9
2.2.1. EyeSightServer 9
2.2.2. VisionAnalyzer 9
2.2.3. DataManager (Server-side) 9
2.2.4. Communication 10
2.3. Data Layer 10

3. Class Interfaces **10**
3.1. Presentation Layer 10
3.1.1. Views 10
3.1.1.1. CameraView 10
3.1.1.2. SignupView 11
3.1.1.3. LoginView 11

3.1.1.4. StartupView 12
3.1.1.5. SettingsView 12
3.1.1.6. MainMenuView 12
3.1.1.7. EmergencyContactView 13
3.1.2. Controllers 13
3.1.2.1. EyeSightClient 13
3.1.2.1.1. Client 13
3.1.2.2. VisionManager 14
3.1.2.3. Voice Manager 14
3.1.2.3.1. VoiceManager 14
3.1.2.4. LocalDataManager 15
3.1.2.5. CommunicationManager 15
3.1.2.5.1. ClientTCPSocket 15
3.1.2.5.2. ClientUDPSocket 16
3.1.3. Models 17
3.1.3.1. User 17
3.1.3.2. Contact 17
3.1.3.3. Frame 18
3.1.3.4. EyeSightObject 18
3.2. Server Layer 19
3.2.1. EyeSightServer 19
3.2.1.1. Server 19
3.2.2. VisionAnalyzer 19
3.2.2.1. ObjectDetection 19
3.2.2.2 FaceDetector 20
3.2.2.2.1. Detector 20
3.2.2.2.2. Recognition 20
3.2.2.2.3. ObjectDetectionModel 21
3.2.3. DataManager (Server-side) 21
3.2.3.1. DataManagement 21
3.2.4. Communication 22
3.2.4.1. ServerTCPSocket 22
3.2.4.2. ServerTCPHandler 22
3.2.4.3. ServerUDPSocket 23
3.2.4.4. ServerUDPHandler 24
3.3. Data Layer 24
3.3.1. MySQL 24
3.3.2. Filesystem storage 24
4. References 25

1. Introduction

EyeSight is intended to hold visually impaired people's hands to make their life less challenging. Despite the rapid development of computers, visually impaired people still use a stick to navigate themselves through obstacles. EyeSight is imagined to become a new eye for those people. Using the camera of the smartphones, EyeSight will describe the user's surroundings such as objects, walls, humans etc. to them in real time. Users will also be able to register their relatives, friends and family members to EyeSight. The app will recognize human faces and will inform users specifically on who those people are.

EyeSight will have a user friendly UI for visually handicapped people. It will be accomplished by using voice overlay and taking voice inputs from the users which will be used to help them interact with the app. We are determined to ensure that visually impaired users will have no issue in using the app without any help from other people. Since the deadlines of this project are limited, we decided to implement EyeSight just for indoor usage. EyeSight will initially not be suited for outdoors because of the vital risks that might occur.

This report is an overview of the low level design of EyeSight. It starts with the trade-offs of the software, then interface documentation guidelines, engineering standards and some definitions are explained. After these, packages and class interfaces are presented.

1.1. Object Design Trade-offs

1.1.1. Usability vs Functionality

This trade off is one of the most crucial aspects of our application. Since our application is for visually impaired people, we have to keep our application simple for them. If we put too much functionality into EyeSight, our application will be unusable by the users as intended. That is why we have to offer a simple interface and purposive functions so that the users can adapt easily and use the application with ease. As a result, we can say that we prefer usability rather than functionality.

1.1.2. Performance vs Complexity

In life, every second is important to us, however if you cannot see the environment, it can be dangerous for you. Since our application will do complex processes such as real time object detection and face detection, we choose the machine learning algorithms to speed up the performance and serve with minimum delay. We limit the inputs to reduce the complexity. Thus, we choose the performance over complexity to provide best services to the user.

1.1.3. Reliability vs Functionality

To increase reliability, we need to sacrifice some of the functionality of the detection systems. We want to help visually impaired people, not to be a burden. Adding many features increases the risk of having reliability issues which could result into serious problems for visually impaired users, therefore we prefer reliability over functionality. If the detection system detects too many things, it will be too hard to handle for the application, so we limit it and choose to increase reliability of the detection systems.

1.1.4. Scalability vs Cost

We want everyone to use our application and ease their life. EyeSight can be used not only by visually impaired people, but also by children. Since the application detects objects and notify the users about these objects by voice, it can be an educational tool. We do not want to restrict how our application will be used, that is why we choose stability over cost to be used by everybody who wishes to use our application.

1.2. Interface Documentation Guidelines

In this report, all class interface documentation is unified in the following format.

class “ClassName”
Class Description: Description for the class “ClassName”
Attributes
- “attribute1Name” : “attribute1Type” - “attribute2Name” : “attribute2Type”
Functions
- “method1Name”(“parameter1Name” : “parameter1Type”, ...) - “method2Name”(“parameter2Name” : “parameter2Type”, ...)

1.3. Engineering Standards

We are using IEEE standards to write software reports such as 830-1984-IEEE software requirements specifications guidelines[1] and 1016-1987-IEEE software design description guidelines[2].

For modelling the software, we are using widely used, standard UML diagrams such as subsystem decomposition diagrams and class diagrams.

1.4. Definitions, Acronyms, and Abbreviations

TTS	Text to Speech
STT	Speech to Text
API	Application Program Interface: This is the method of communication. The server layer exposes an API for the presentation layer
MVC	Model View Controller: A standard architectural pattern used in the presentation layer.
UML	Unified Modelling Language: This is the standard diagramming and modeling technique
GCP	Google Cloud Platform
HTTP	Hyper-Text Transfer Protocol: This is the standard protocol used to communicate between presentation and server layer
IEEE	The Institute of Electrical and Electronics Engineers: The engineering and reporting standards used.
UI	User Interface: The interface that user interacts with while using our application
POCO	Plain-Old Clr Object: Refers to classes, which are object oriented design building blocks, which only have attributes and have no functions. These classes are used to model objects in our app.
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
SQL	Structured Query Language: This language is used for querying relational databases.

2. Packages

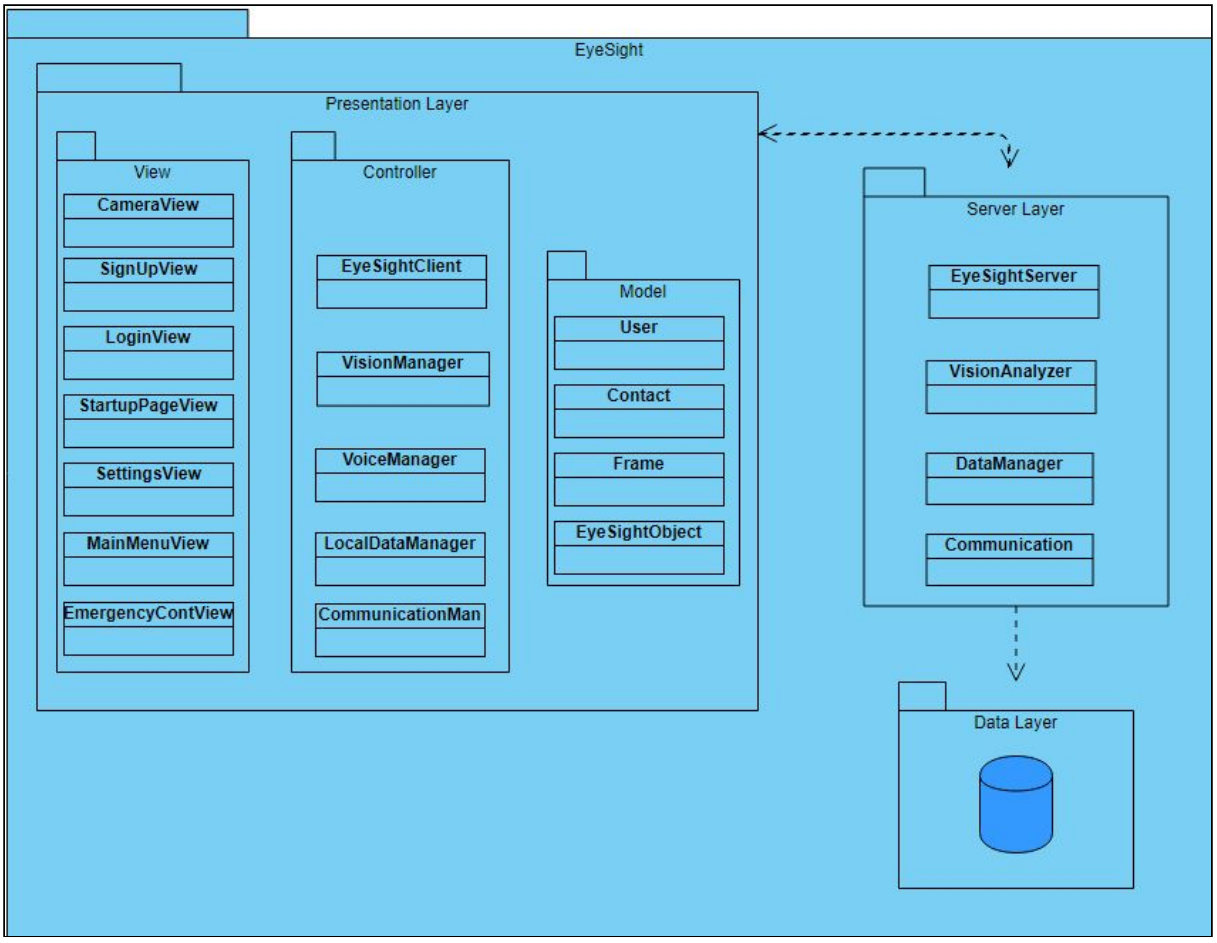

Figure 1: Package overview

2.1. Presentation Layer

This is our application's frontend layer. It is structured in the MVC pattern. This part of the application is responsible for user interaction. User gives different types of inputs such as

- Voice
- Touch

and receives different types of outputs such as

- Visuals on the screen
- Sound
- Vibrations

2.1.1. Views

2.1.1.1. Camera View

This is the main function. Opens the camera and starts vision analysis.

2.1.1.2. SignUp/Login View

Collects sign up or login information to be sent to the server for authentication.

2.1.1.3. Startup Page View

Collects information about users and family members when the app is used for the first time.

2.1.1.4. Settings View

Display and update any settings information and store them in the server.

2.1.1.5. Main Page View

Displays the main menu.

2.1.1.6. Emergency Contact View

Display and update emergency contact information.

2.1.2. Controllers

2.1.2.1. EyeSightClient

This client is responsible for making HTTP calls to the backend server therefore fetching data from it and sending data to it.

2.1.2.2. VisionManger

This manager handles most of the main camera functionality. it consists of ObjectDetector and FaceDetector packages.

2.1.2.3. VoiceManager

This manager is responsible for dealing with voice inputs.

2.1.2.4. LocalDataManager

The system stores a few small data in local data store such as

- User preferences
- Settings
- Login credentials

Local data manager is responsible for writing to and reading from the local data storage.

2.1.2.5. CommunicationManager

This package is responsible for establishing the connection between the client-side and the server-side systems. Client-side creates a single socket to achieve this.

2.1.3. Models

This package has the POCO classes to every domain model we have in our system.

2.1.3.1. User

This class consists of necessary functions and attributes for the current user such as

- User
- User profile
- User login
- User relations

2.1.3.2. Contact

This class consists of necessary functions and attributes for the corresponding contact -that the user adds.

2.1.3.3. Frame

This class consists of necessary functions and attributes for the Frame class. This class will be the model for the output that is generated by object detection processes on the server side.

2.1.3.4. EyeSightObject

This class consists of necessary functions and attributes for the EyeSightObject class. The objects will be recognized by the machine learning algorithms and will be modeled by this class.

2.2. Server Layer

This layer is the backend of our application. It handles most of the processing using custom made functions and 3rd party APIs.

2.2.1. EyeSightServer

This package contains functionality to communicate with different clients through TCP and UDP sockets. It is used for both vision analysis and data flow for users between database and client.

2.2.2. VisionAnalyzer

This package is responsible for processing and analyzing visual data sent from clients using 3rd party APIs and custom functions using outside libraries and hand-written code.

2.2.3. DataManager (Server-side)

This package is responsible for communication to the SQL database providing necessary information to other functions and writing data to the permanent data store.

2.2.4. Communication

This package is responsible for establishing the connection between the client-side and the server-side systems. Server-side creates a many sockets, one for each client, to achieve this.

2.3. Data Layer

This is the persistent data layer of our application. Other than some small amount of information that is stored in the user's local device, all of the persistent data is stored here.

There are 2 types of persistent data: text data and media data. Most of the information is saved as text and numbers in a relational database server. The rest of the data, namely the media type data such as images and video, is saved in a standard filesystem storage.

3. Class Interfaces

The main classes that are crucial for our application will be listed and explained in the following subtopics.

3.1. Presentation Layer

Class explanations for the frontend. Architectured in an MVC fashion.

3.1.1. Views

3.1.1.1. CameraView

class CameraView
This view is responsible for the main functionality of the application. It has a camera view on it that will get frame inputs from the camera and sends it to the server. It also expects analysed frames from the server and converts the text into voice for the user.
Attributes
- currentUser: User - cameraView: CameraView - EyeSightObjects(): EyeSightObject[]
Functions
- processFrame(frame: Frame): void - sendFrameToServer(frame: Frame): Frame + fetchAnalyzedFrame(frameId: int): EyeSightObject[] - tellTheObject(object: EyeSightObject): void - launchVoiceListener(): void

- **processFrame(frame: Frame): void** : Gets frame from the camera, preprocesses it and forwards some of them to the server side periodically in an asynchronous manner.
- **sendFrameToServer(frame: Frame): Frame** : Calls the network class to send a frame-to-be-analyzed to the server.
+ **fetchAnalyzedFrame(frame: Frame): void** : This method is called by the network class whenever a response from the server brings in an analyzed frame
- **tellTheObject(object: EyeSightObject): void** : This method calls the appropriate function from the voice manager package to read out an object in the analyzed frame.
- **launchVoiceListener(): void** : This method launches the voice listener.

3.1.1.2. SignupView

class SignupView
This view is responsible for signup.
Attributes
- user: User - userName: String - password: String
Functions
- passwordSignup(username: String, password: Password): boolean - googleSignup(email: String): boolean

- **passwordSignup(username: String, password: Password): boolean** : Logging in using password
- **googleSignup(email: String): boolean** : Logging in using Google identity provider

3.1.1.3. LoginView

class LoginView
This view is responsible for login.
Attributes
- currentUser: User - userName: String - password: String
Functions
- passwordLogin(username: String, password: Password): boolean - googleLogin(): boolean

- **passwordLogin(username: String, password: Password): boolean** : Signing up using password
- **googleLogin(email: String): boolean** : Signing up using Google identity provider

3.1.1.4. StartupView

class StartupView
This view is responsible for setting up user information when the user first logs in.
Attributes
- currentUser: User - contactInfo: contact
Functions
+ setUpUserInfo(user: User): void

+ **setUpUserInfo(user: User): void** : Changes the user information.

3.1.1.5. SettingsView

class SettingsView
This view is responsible for changing user preferences.
Attributes
- currentUser: User - settings: Settings
Functions
- saveSettings(settings: Settings): void

- **saveSettings(settings: Settings): void** : Saves the changes in user preferences

3.1.1.6. MainMenuView

class MainMenuView
This view is responsible for the main menu, selecting between the main functionality and settings page.
Attributes
- currentUser: User
Functions
- launchCamera(): void - launchSettings(): void

- **launchCamera(): void** : Launches the main camera function view
- **launchSettings(): void** : Launches the settings view

3.1.1.7. EmergencyContactView

class EmergencyContactView
This view is responsible for calling the contact information.
Attributes
- contact: Contact - currentUser: User
Functions
+ fetchEmergencyContact(user: User): contact: Contact - callEmergencyContact(contact: Contact): boolean

+ **fetchEmergencyContact(user: User): contact: Contact** : Gets the emergency contact information from the local data store
- **callEmergencyContact(contact: Contact): boolean** : Calls the system function to make a landline call to the emergency contact.

3.1.2. Controllers

3.1.2.1. EyeSightClient

3.1.2.1.1. Client

class Client
This class consists of main function and properties which will use additional threads to provide functionalities of the whole program. This class will use vision and voice package class instances to provide main functionality of the program in different threads while also using UDP and TCP socket class instances to communicate with the server program to transfer or receive necessary data to the mobile phone.
Attributes
- tcpSocket: ClientTCPSocket - udpSocket: ClientUDPSocket - currentUser: User - final_IP: InetAddress - final_PORT: int
Functions
+ main(String args[]): void

+ **main(String args[]):** This function is the main thread that will control view sections of the menu and run other thread instances to provide main functionality.

3.1.2.2. VisionManager

class VisionManager
Visionmanager class initializes the camera and does the related adjustments for the camera.
Attributes
-
Functions
- startCamera() : void

- **startCamera() :** This method starts the camera and sends frames to the server to be analyzed.

3.1.2.3. Voice Manager

3.1.2.3.1. VoiceManager

class VoiceManager
TTSManager class manages the conversion of speech to text and text to speech processes
Attributes
- soundPath : string - text : string
Functions
+ convertToText (soundPath : String) : string + convertToSpeech (soundPath : String) : string + checkForInvalidText (text : String) : boolean + checkForInvalidSpeech (soundPath : String) : boolean - callPerson (person : Contact) : void

- **public checkForInvalidSpeech(string soundPath):** checks for invalid words in the given speech, returns false if there are any
- **public checkForInvalidText(string text):** checks for invalid words in the given text, returns false if there are any
- **callperson(Contact person):** Auxiliary function to make a call to a recorded person if necessary command given via speech.

3.1.2.4. LocalDataManager

class LocalDataManager
This class handles the data that will be stored locally.
Attributes
- databaseName : String
Functions
+ openOrCreateDatabase(path : String, factory : SQLiteDatabase.CursorFactory) : void + getContact() : Contact + addContact(contact : Contact) : void + deleteContact (name : String) : void + setSettings(i : int, p : int) : void + addRelative(name : String, picPath : String) : void + setRelative(name : String, picPath : String) : Void + deleteRelative(name : String) : void + reset() : void

3.1.2.5. CommunicationManager

3.1.2.5.1. ClientTCPSocket

class ClientTCPSocket
This class consists of the thread function that is required to create a connection to the server to get necessary data for the user from the database or save user data into the database through the server application.
Attributes
- socket: Socket (java.net.Socket) - dinstream: DataInputStream (java.io.DataInputStream) - doutstream: DataOutputStream (java.io.DataOutputStream)
Functions
+ ClientTCPSocket(socket : Socket, dinstream : DataInputStream, doutstream : DataOutputStream) + run(): void

+ **ClientTCPSocket(socket : Socket, dinstream : DataInputStream, doutstream : DataOutputStream):** Constructor function for the socket. Takes input and output streams with socket object to provide communication functionality to the main client program.
+ **run():** This is the thread execution function that will be used for sending or receiving data as a string which will be required for certain functionalities such as saving user data records, fetching user data.

3.1.2.5.2. ClientUDPSocket

class ClientUDPSocket

This class consists of the thread function that is required to be executed to create a connection to the server to stream vision to analyze it on server and get feedback from analysis.

Attributes

- socket: DatagramSocket (java.net.DatagramSocket)
- finstream: FileInputStream (java.io.FileInputStream)
- foutstream: FileOutputStream (java.io.FileOutputStream)

Functions

+ ClientUDPSocket(socket : DatagramSocket, finstream : FileInputStream, foutstream : FileOutputStream)
+ send(o : FileOutputStream): void
+ receive(i : FileInputStream): void
+ run(): void

+ **ClientUDPSocket(socket : DatagramSocket, finstream : FileInputStream, foutstream : FileOutputStream):** Constructor function for the socket. Takes input and output file streams with socket object to provide communication functionality to the main client program.
+ **send(o : FileOutputStream):** This function streams the file to the server to analyze.
+ **receive(i : FileInputStream):** This function receives file stream from server.
+ **run():** This is the thread execution function that will be used for streaming vision to the server to get the analysis service and provide feedback to the user in the main thread.

3.1.3. Models

3.1.3.1. User

class User
This class consists of necessary functions and attributes for the current user.
Attributes
- name: String - userID: String - email: String - dateOfBirth: int - contactList: Contact[]
Functions
+ getUsername() : String + getUserID() : String + getUserEmail() : String + getUserDateOfBirth() : int + setUsername(name : String) + setUserID(id : String) + setUserEmail(email : String) + setUserDateOfBirth(dob : int) + getContactList(): Contact[] + addToContactList(newContact : Contact)

3.1.3.2. Contact

class Contact
This class consists of necessary functions and attributes for the corresponding contact -that the user adds.
Attributes
- name: String - phoneNumber: int - photoPath: String
Functions
+ getName() : String + getPhoneNumber() : int + getPhotoPath(): String + setName(String name) + setPhoneNumber(int pNo) + setPhotoPath(path: String): void

3.1.3.3. Frame

class Frame

This class consists of necessary functions and attributes for the Frame class. This class will be the model for the output that is generated by object detection processes on the server side.

Attributes

- objectList: EyeSightObject[]
- output: String

Functions

+ getObjectList() : EyeSightObject[]
+ getOutput() : String
+ setObjectList(objectList: EyeSightObject[]) : void
+ setOutput(output: String) : void
+ addToObjectList(newObject : EyeSightObject)

3.1.3.4. EyeSightObject

class EyeSightObject

This class consists of necessary functions and attributes for the EyeSightObject class. The objects will be recognized by the machine learning algorithms and will be modeled by this class.

Attributes

- name: String
- xPosition: int
- yPosition: int
- xSize: int
- ySize: int

Functions

+ getName() : String
+ getXPosition() : int
+ getYPosition() : int
+ getXSize(): int
+ getYSize(): int
+ setXSize(xSize: int)
+ setYSize(ySize: int)
+ setName(name: String) : void
+ setXPosition(xPos: int) : void
+ setYPosition(yPos: int) : void

3.2. Server Layer

3.2.1. EyeSightServer

3.2.1.1. Server

class Server
This class consists of main function and properties to communicate with different clients through TCP and UDP sockets for both vision analysis and data flow for users between database and client.
Attributes
- tcpSocket: ServerTCPSocket - udpSocket: ServerUDPSocket - final PORT: int
Functions
+ main(): void

+ **main():** This function is the main thread that will control socket interactions and database functions to communicate with different clients via multiple socket threads to provide main functionality.

3.2.2. VisionAnalyzer

3.2.2.1. ObjectDetection

Class ObjectDetection
It analyses the input from the camera and classifies the objects. It also sends the classified objects to the voiceManager.
Attributes
- options : val - objectDetector : val - classifiedObject : String + data : val + image : val
Functions
- sendImagetoVoice(item : String) : String - startDetection(frame : Frame, callback : String) : FirebaseVisionImage[3] - identifyObjects(frame : Frame) : void

- **sendImagetoVoice(item : String):** This function basically takes the identified object and sends it to voiceManager to notify the user.
- **startDetection(frame : Frame, callback : String):** This function starts to analyze frames from the camera.
- **identifyObjects(frame : Frame):** This function starts processing the images comes from the startDetection(...) method.

3.2.2.2 FaceDetector

FaceDetector package consists of the functions that analyze the frame and extract the faces and label them if there are any saved faces.

3.2.2.2.1. Detector

class Detector
Detector class initiates the instances for the face detection.
Attributes
- faceDetector : FaceDetector - detector : FaceDetector
Functions
- faceSettings(multiFace : Boolean) : void - identifyFace(frame : Frame) : void - sendImagetoVoice(item : String) : void

- **faceSettings(multiFace: Boolean):** This function sets the basic settings for FaceDetector. It takes a boolean variable to detect whether it detects multiple faces in the frame.
- **identifyFace(frame : Frame):** It takes a frame, analyzes it. If the face in the frame previously saved it calls the sendImagetoVoice(...) method.

3.2.2.2.2. Recognition

class Recognition
This class is the inner class of the interface Classifier. It holds the variables of the face detection.
Attributes
- confidence : float - extraData : Object

3.2.2.2.3. ObjectDetectionModel

class ObjectDetectionModel
This class is about training the model with inputs, and holds the dataset for saved faces.
Attributes
- output_size : int - buffer : float[][] - registered : Hashmap
Functions
- register(name : String, record : Recognition) : void - findFace(inp : float[]) : Pair - recognizeImage(bitmap : Bitmap, storeExtra : boolean) : List

- **register(name : String, record : Recognition):** it adds the input face and name to the dataset.
- **findFace(inp : float[]):** This method searches the dataset and returns the name and the confidence.
- **recognizeImage(bitmap : Bitmap, storeExtra : boolean):** Finds the face and matches it with the existing face.

3.2.3. DataManager (Server-side)

3.2.3.1. DataManagement

class DataManagement
DataManagement class interacts with the SQL database.
Attributes
- databaseName : String
Functions
+ setConnection(dbName : String) : void + getContact() : Contact + addContact(contact : Contact) : void + deleteContact (name : String) : void + setSettings(i : int, p : int) : void + addRelative(name : String, picPath : String) : void + setRelative(name : String, picPath : String) : Void + deleteRelative(name : String) : void + reset() : void

3.2.4. Communication

3.2.4.1. ServerTCPSocket

class ServerTCPSocket
This class holds the main thread function of server TCP socket which allocates handler sockets to handle multiple client requests in different threads. Main requests are done to send auxiliary data for users which to be used by clients and save records for users to be later used.
Attributes
- socket: Socket (java.net.Socket) - dinstream: DataInputStream (java.io.DataInputStream) - doutstream: DataOutputStream (java.io.DataOutputStream)
Functions
+ ServerTCPSocket(Socket socket, DataInputStream dinstream, DataOutputStream doutstream) + run(): void

+ **ServerTCPSocket(Socket socket, DataInputStream dinstream, DataOutputStream doutstream):** Constructor function for the socket. Takes input and output streams with socket object to provide communication functionality to the main server program.
+ **run():** This is the thread execution function that will be used for accepting client requests and creating new TCP sockets for sending or receiving data as a string which will be required for certain functionalities such as saving user data records, fetching user data.

3.2.4.2. ServerTCPHandler

class ServerTCPHandler
This class holds the thread function of the sockets that are created by the main server socket to communicate with multiple clients in different threads. It is responsible for communicating with one specific client socket to provide service.
Attributes
- socket: Socket (java.net.Socket) - dinstream: DataInputStream (java.io.DataInputStream) - doutstream: DataOutputStream (java.io.DataOutputStream) - uid: int
Functions
+ ServerTCPHandler(Socket socket, DataInputStream dinstream, DataOutputStream doutstream, uid int) + run(): void

+ **ServerTCPHandler(Socket socket, DataInputStream dinstream, DataOutputStream doutstream, int uid):** Constructor function for the socket. Takes input and output streams with socket object to provide communication functionality to the main server socket.
+ **run():** This is the thread execution function that will be used for sending or receiving data as a string which will be required for certain functionalities such as saving user data records, fetching user data.

3.2.4.3. ServerUDPSocket

class ServerUDPSocket
This class holds the main thread function of the server UDP socket which allocates different udp sockets in different threads to receive vision from different clients to analyze in the server.
Attributes
- socket: DatagramSocket (java.net.DatagramSocket) - finstream: FileInputStream (java.io.FileInputStream) - foutstream: FileOutputStream (java.io.FileOutputStream)
Functions
+ ServerUDPSocket(socket : DatagramSocket, finstream : FileInputStream, foutstream : FileOutputStream) + run(): void

- **ServerUDPSocket(socket : DatagramSocket, finstream : FileInputStream, foutstream : FileOutputStream):** Constructor function for the socket. Takes input and output streams with socket object to provide communication functionality to the main server program.
- **run():** This is the thread execution function that will be used for accepting client requests by creating new udp sockets for sending or receiving file stream which will be required for vision analysis.

3.2.4.4. ServerUDPHandler

class ServerUDPHandler
This class holds the thread function of the sockets which are created by the main UDP socket to communicate with multiple clients and receive vision to analyze in the server. It's responsible for communicating with one specific client socket to provide service.
Attributes
- socket: DatagramSocket (java.net.DatagramSocket) - finstream: FileInputStream (java.io.FileInputStream) - foutstream: FileOutputStream (java.io.FileOutputStream)
Functions
+ ServerUDPSocket(socket : DatagramSocket, finstream : FileInputStream, foutstream : FileOutputStream) + send(o : FileOutputStream): void + receive(i : FileInputStream): void + run(): void

- **ServerUDPHandler(socket : DatagramSocket, finstream : FileInputStream, foutstream : FileOutputStream):** Constructor function for the socket. Takes input and output streams with socket object to provide communication functionality to the main server socket.
- **send(o: FileOutputStream):** This function streams a file to the client.
- **receive(i : FileInputStream):** This function receives vision stream from client.
- **run():** This is the thread execution function that will be used to receive vision and provide necessary feedback though file streams.

3.3. Data Layer

The data layer consist of a singular SQL database and a filesystem storage

3.3.1. MySQL

EyeSight data layer consists of a MySQL relational database that stores all non-media persistent data.

3.3.2. Filesystem storage

All media data such as user profile pictures and contact faces are stored in a filesystem storage. Uri's to these images are also stored in the relational database.

4. References

[1] IEEE Guide for Software Requirements Specifications," in IEEE Std 830-1984 , vol., no., pp.1-26, 10 Feb. 1984, doi: 10.1109/IEEESTD.1984.119205.

[2] IEEE Recommended Practice for Software Design Descriptions," in IEEE Std 1016-1987 , vol., no., pp.1-16, 13 July 1987, doi: 10.1109/IEEESTD.1987.122643.

[3] “Documentation | Firebase,” *Google*. [Online]. Available: https://firebase.google.com/docs?gclid=CjwKCAjw8J32BRBCEiwApQEKgS9kFhIIKvglPdbqCoUwcRrYJzyoZvVEkwQuT4t7hZtrEedrvFfgjxoCyFoQAvD_BwE.