Bilkent University

Department of Computer Engineering

Senior Design Project

Emolyst

Low-Level Design Report

Group Members: Elif Kevser Arslan, Ali Bulut, Musab Erayman, Muammer Tan, Ömer Faruk Karakaya

Supervisor: Dr. Özcan Öztürk

Jury Members: Dr. Varol Akman and Dr. Mustafa Özdal

Innovation Expert: Barış Misman

Low-Level Design Report

18.02.2019

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

CONTENT

1. Introduction

In these days, people are the fundamental interface of all businesses. People's emotion and behavior affect their vital decisions which direct the businesses. If machines can know what a person is feeling, incredible insights can be unlocked. While detecting and analyzing one's feelings and comprehending results in a smart way, computer science comes into play. With the help of trusted algorithms, collected data is processed and rational inferences can be made. Hence, the value for any business can be created in an incisive way. Concordantly, we are planning to bring this idea into our senior design project. We are planning to create an application called Emolyst that can detect emotion, gender, and age through the integration of several algorithms. Basically, real-time data that is collected through cameras will be analyzed. Later on, statistical results will be given to our clients to make them smart and profitable choices for their businesses. Since our application will process images for only real-time data analysis under the Code of Ethics, it certainly won’t break the Law for Protection of Personal Data. When all the benefits of Emolyst are considered, it can be said that the “mainstreaming” of emotion, gender and age recognition of Emolyst will enable companies to engage it in a multitude of profitable uses, across industries.

Emolyst is a CRM (Customer Relationship Management) application for legal entities and it provides analysis of their customers based on their emotional status and age-gender categories. Emolyst uses and integrates face detection, emotion recognition, age, and gender detection algorithms. It processes these data to provide analysis to our clients about their customers. Mainly, Emolyst aims to improve customer services and quality standards of companies.

It is surely beyond doubt that customer ratings are valuable sources to understand their satisfaction and are critical for designing better customer experiences and recommendations. Conventionally most of the companies were using an inefficient way to collect data like creating surveys. Thus, we decided to create a program for helping companies to get real-time reliable data about their customers and increase their customer satisfaction.

One of the most innovative and useful functions of Emolyst is providing real-time analysis. It will analyze all customers' data in the given interval. While doing this, Emolyst will not keep or use any confidential data of people and will abide by personal data protection law. Any of the faces will not be recognized or sold to third parties. Each company can only see the analysis of their customers.

In this report we aimed to provide low-level design and architecture of our system. Firstly, object design trade-offs and engineering standards are discussed. Afterwards, the packages in our system and their functionalities are described along with detailed class diagram views. Thirdly, the class interfaces in all packages are shown. Moreover, we provide the clarifications of the descriptions of functionalities of each software component in the system.

1.1. Object design trade-offs

1.1.1. Performance vs Cost

We want our system has fast computation and to do fast real data analysis. We will use more costly server services, hardware and network technologies to achieve fast data transfer and fast computation.

1.1.2. Scalability vs Complexity

We want our system to work scalable with many data sources and streams. In order to achieve this scalability we need horizontally scalable design which is more complicated than traditional ways. Also scalable tools are needed such as scalable database and streaming servers.

1.1.3. Security vs Usage

In order not to violate personal data protection law, our system must be secure enough which may decrease the effective usage from the end-user perspective. Since our main concern is security, we may sacrifice from ease of use a bit. However, while we are trying to create a safe system, we are also considering making Emolyst to have an easier level of usage.

1.1.4. Performance vs Complexity

Since we are planning to make real time data analysis, Emolyst should have a good level of performance and it should response in a short time period. To maximize the performance, the implementation should be less complex. During implementation, although we will use complex algorithms, we will also mainly focus on maximizing our performance. In this way, we can minimize the response time and conserve the users’ satisfaction.

1.1.5. Compatibility vs Extensibility

Extensibility is significant for our system since we will need updates and bug fixes to give reliable and consistent results. Although making a system extensible might cause some compatibility problems and we are preferring extensibility over compatibility, we are also planning to create a compatible system as much as we can. In this way, Emolyst can be bug-free and compatible with the different systems.

1.2. Interface documentation guidelines

In the implementation level, we follow the conventional ClassName format for naming our classes and functionName() format for naming the methods. A well-documented comments for classes and the methods are also present above the function and class declaration lines.

1.3. Engineering standards (e.g., UML and IEEE)

We used git[1] for code contribution and source code management among contributors. We used Gitlab[2] for issue management and tracking. Also, other standart software engineering charts and diagrams are used such as flowcharts, use case diagrams, sequence diagrams and UML[3] diagrams in reports to document architecture and functionality of our system and the software.

1.4. Definitions, acronyms, and abbreviations

UI: User Interface
API: Application Programming Interface
CRM: Customer Relationship Management
JSON: Abbreviation of JavaScript Object Notation which is a lightweight data-interchange format.
SQL: Abbreviation of Structured Query Language which is a programming language for managing the database.

2. Deployment Diagram

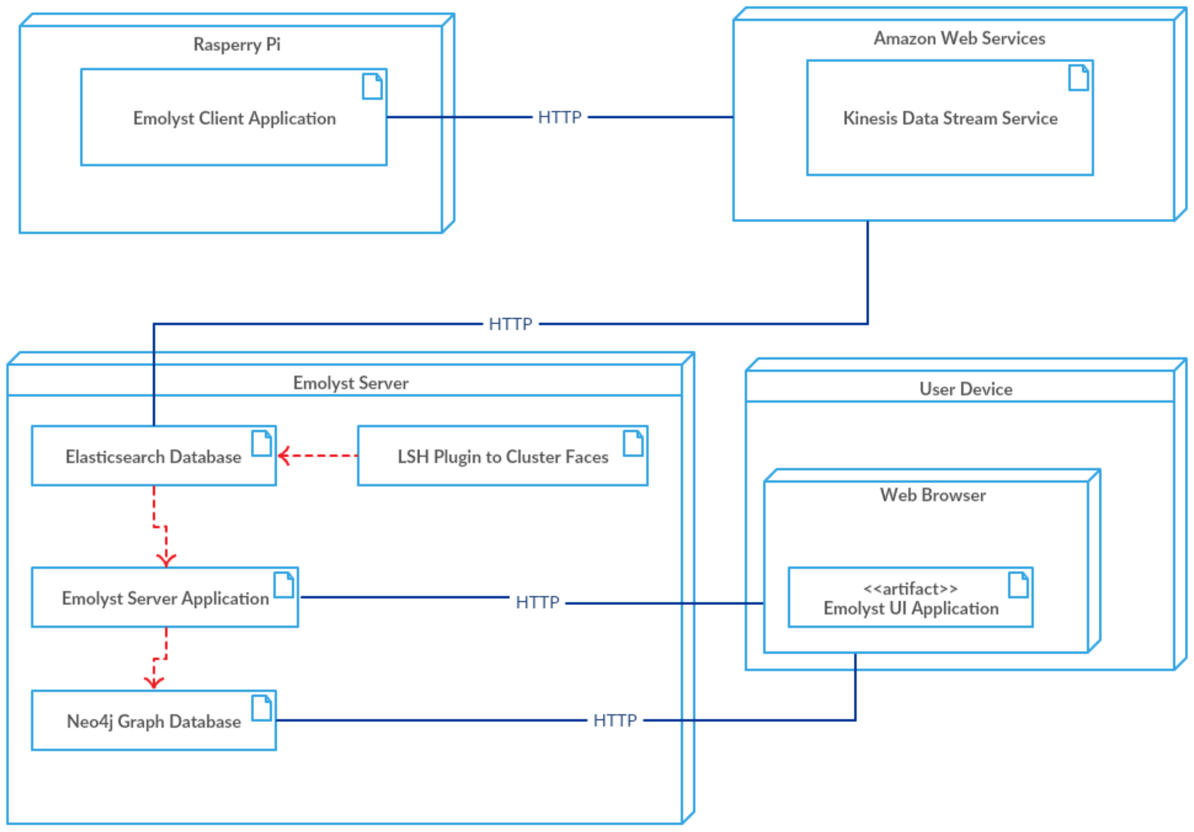


Figure 1: Deployment Diagram

2.1. Raspberry Pi

Raspberry Pi is a tiny computer that will serve us a computation power for capturing camera input and real time processing on this camera input.

2.1.1. Emolyst Client Application

Python application that contains image processing and streaming packages. Detailed class diagram can be found in Class Interfaces section. Single Emolyst Client Application consumes about 500 Mb of memory and produces 5 sample per second.

2.2. Amazon Web Services

Kinesis[4] is a real time data streaming service that provided by Amazon Web Services. Emolyst client applications will be using Kinesis for sending data to Emolyst Server.

2.3. Emolyst Server

Emolyst Server will be responsible for the management, control and analysis of the data which are gathered from video streams.

2.3.1.Elasticsearch Database

Single Emolyst Client Application will push approximately 5 many analyzed face data per second. This data is in a log structure and will not be deleted or updated later on. Therefore Elasticsearch[5] will be used for fast data indexing and performing time based analyses. Here you can find data mapping as JSON object:

```
{
  "mappings":{
    "doc":{
      "properties":{
        "face_feature":{
          "type":"keyword"
        },
        "emotion":{
          "properties":{
            "angry":{
              "type":"long"
            },
            "fear":{
              "type":"long"
            },
            "happy":{
              "type":"long"
            },
            "sad":{
              "type":"long"
            },
            "surprise":{
              "type":"long"
            },
            "neutral":{
              "type":"long"
            }
          }
        },
        "age":{
          "type":"integer"
        },
        "gender":{
          "type":"keyword"
        },
        "created":{
          "type":"date",
          "format":"strict_date_optional_time||epoch_millis"
        }
      }
    }
  }
}
```

2.3.2.LSH Plugin

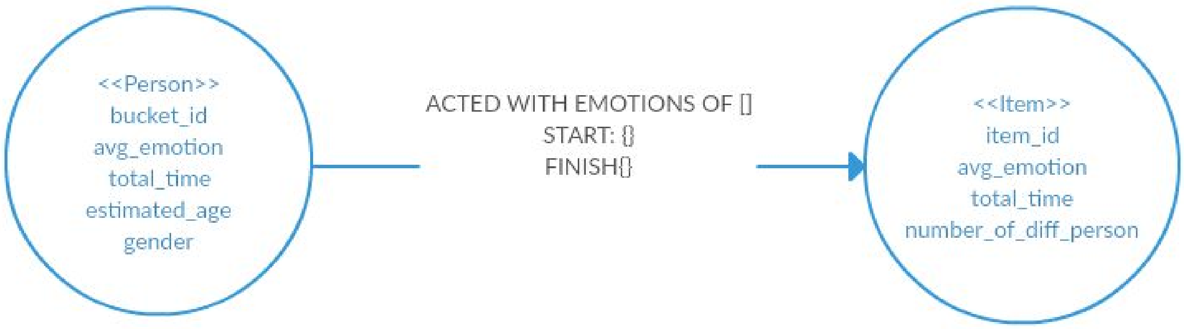
LSH plugin is an Elasticsearch plugin that allow us to find same users on different samples with applying Locality Sensitive Hashing[6] on face vector field.

2.3.3.Emolyst Server Application

Emolyst Server Application will use detection algorithms to detect and cluster faces from frames and process the inputs accordingly. After processing the inputs, server transfer the output to the Neo4j[7] Graph Database

2.3.4.Neo4j Graph Database

Here is the data our data mapping for Neo4j graph database:



3. Packages

3.1. Image Processing

ImageProcessing package will take frames from streamer as input. Using the algorithms in the AgeDetector, GenderDetector, FacialExpressionDetector and FaceDetector classes, this package will hold the core functionality of *Emolyst*. FaceVerifier class clusters the face and looks in the database to learn whether the recognized face is in the database or not. After those processes, Predata class modifies the data and makes it more appropriate for database system.

AgeDetector: This class is responsible for age detection. This class has functionality of estimating age range of any given face.

GenderDetector: This class is responsible for gender detection. This class has functionality of estimating gender of any given face with a confidence.

FaceDetector: This class is responsible for face detection. This class has functionality of detecting all faces in any given image and extracting feature vectors for all faces.

FaceVerifier: This class is responsible for face verification. This class has functionality of verifying a given face and deciding whether this face is already encountered. To do this, we compare feature vectors of faces by using LSH(Locality Sensitive Hashing) algorithm to scale.

PreData: This class is a container class for data obtained from faces. This class includes face feature vector, gender, age range, and facial expression.

FacialExpressionDetector: This class is responsible for detecting facial expressions. This class has functionality of estimating facial expression of given face.

Gender: This enumerates gender into male and female.

FacialExpression: This enumerates facial expressions into happy, sad, angry, surprise, fear, and neutral.

3.1.1 Models

Models subpackage includes trained neural network models for age detection, gender detection, age detection, facial expression detection.

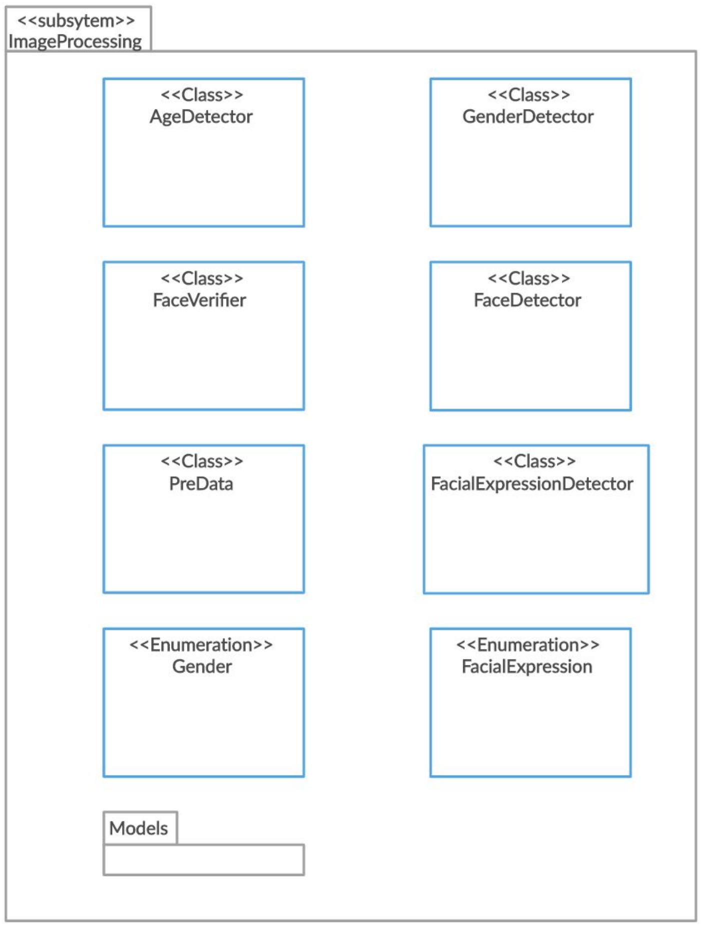


Figure 2: ImageProcessing package

3.2. Streaming

Streaming package will continuously send the video data to Emolyst servers. It will also be responsible for the connection between servers and video capturing.

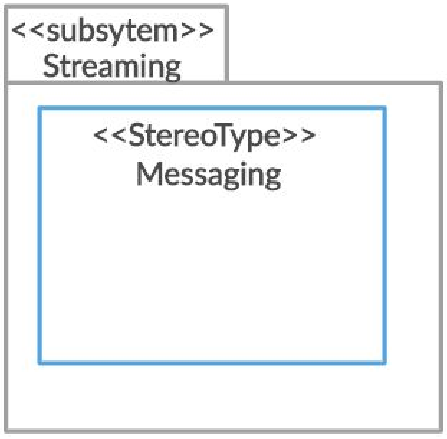


Figure 3: Streaming package

3.3. Data Manager

Data Manager packet controls data flow between Kinesis Data Stream Service and Emolyst Server system. DataReceiver class constantly listens and gets data from Kinesis Service as a JSON object and differentiate it’s content. After that it sends feature vector of faces to face clustering in order to track same person’s reaction. After identifying received face’s group, this system redirects processed data to related Analyzer Class in the Emolyst Server.

DataModel: This class is responsible for decomposing received JSON data. It grub ups feature vector of face, age, gender, and facial expression from this JSON data.

DataReceiver: This class is responsible for listening data from Kinesi Data Stream Service and handling these data. It sends request to ElasticSearch DB to find group of feature vector. After detecting its group this class redirect decomposed data to Data Analysis Package.

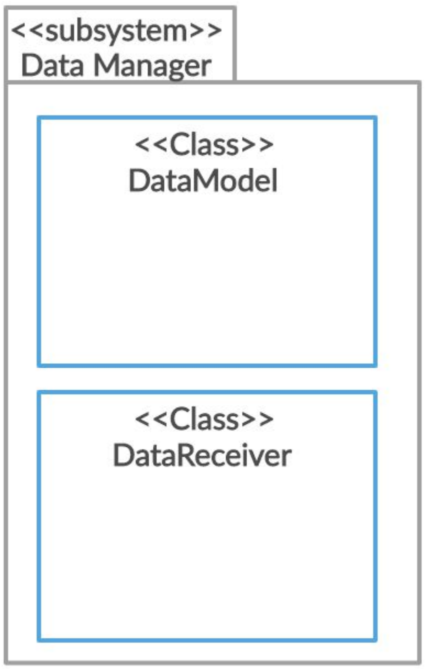


Figure 4: DataManager package

3.4. Data Analysis

This package will keep connection with databases and also analyze customer data.

GraphVisualizer: This class will process the data taken from Neo4j and modify it to easily visualize it. After some modifications, it will be able to visualize the data.
Logger: Logger class will insert the faces which are taken from video captures to Neo4j DB.

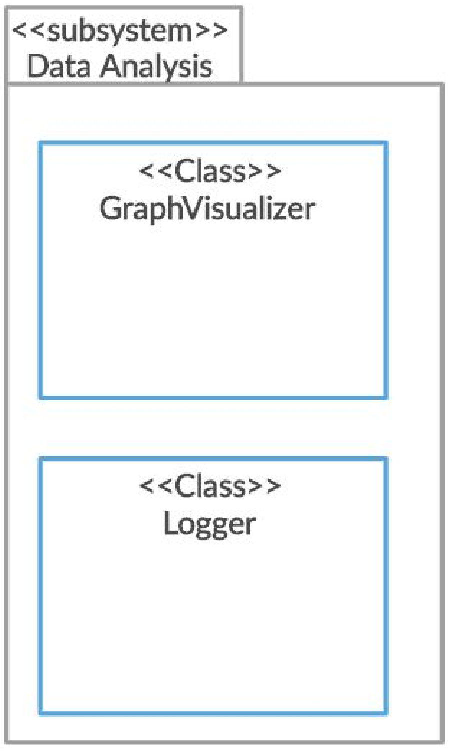


Figure 5: DataAnalysis package

3.5. UI Manager

UI manager package is the design and demonstration part of the *Emolyst.*
GraphUI: This class will be responsible to optimize the visualized graph taken from GraphVisualizer class and also there will be responsive webpage design constraints in this class.
LoggingUI: Clients will log in to the system with this class.

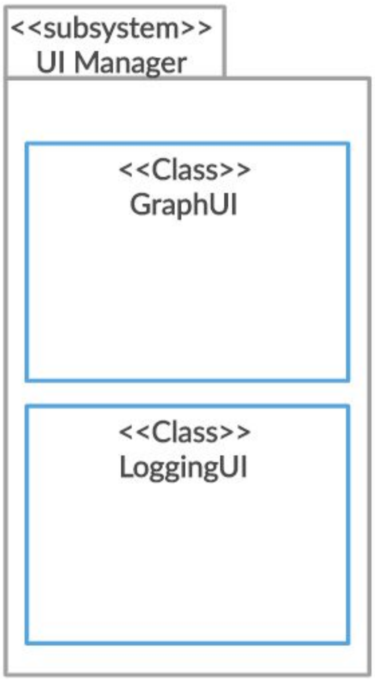
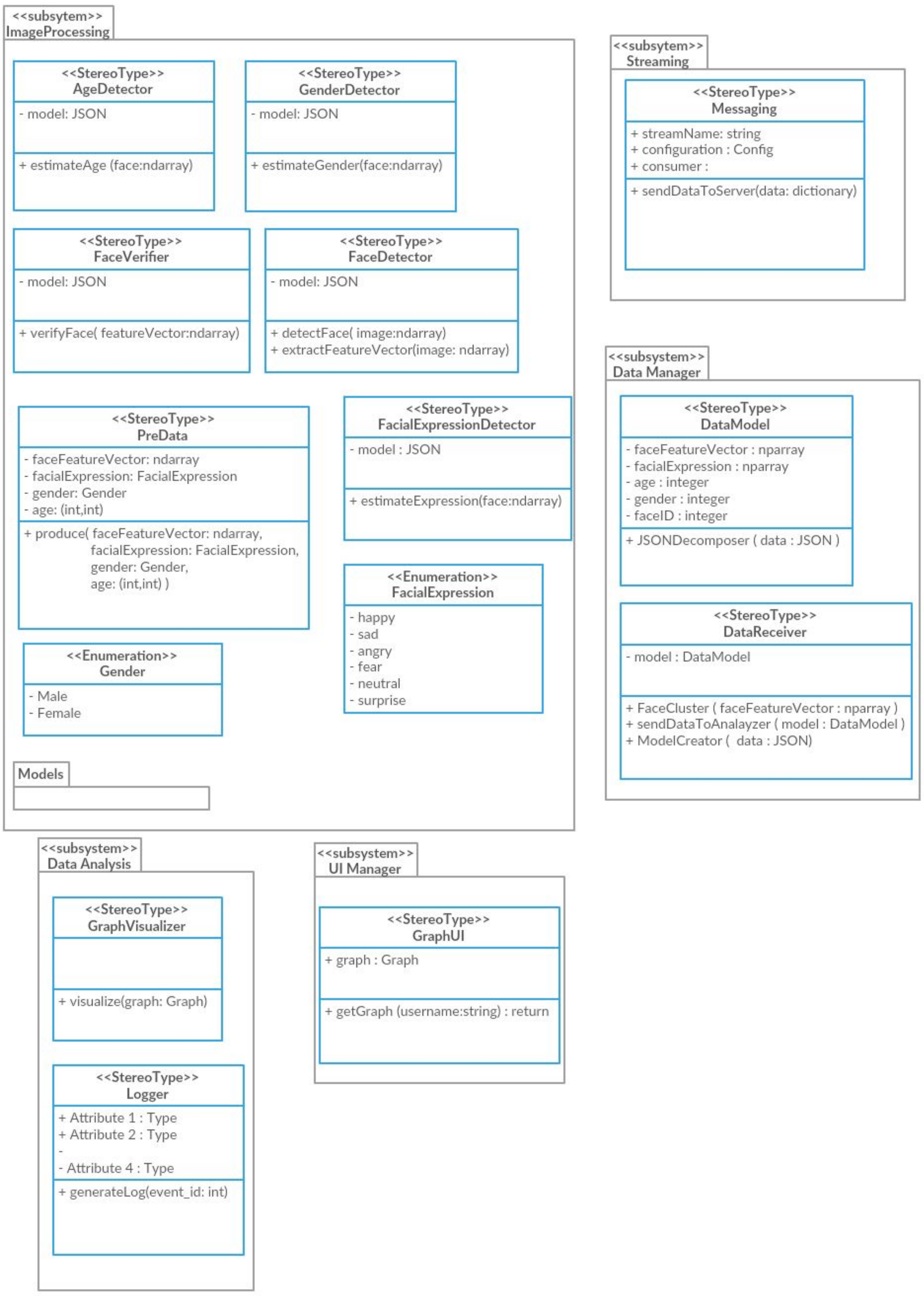


Figure 6: UIManager package

4. Class Interface



<<subsytem>> ImageProcessing
<<StereoType>> AgeDetector
- model: JSON
+ estimateAge (face:ndarray)
<<StereoType>> GenderDetector
- model: JSON
+ estimateGender(face:ndarray)
<<StereoType>> FaceVerifier
- model: JSON
+ verifyFace(featureVector:ndarray)
<<StereoType>> FaceDetector
- model: JSON
+ detectFace(image:ndarray)
+ extractFeatureVector(image: ndarray)
<<StereoType>> PreData
- faceFeatureVector: ndarray
- facialExpression: FacialExpression
- gender: Gender
- age: (int,int)
+ produce(faceFeatureVector: ndarray, facialExpression: FacialExpression, gender: Gender, age: (int,int))
<<StereoType>> FacialExpressionDetector
- model : JSON
+ estimateExpression(face:ndarray)
<<Enumeration>> FacialExpression
- happy
- sad
- angry
- fear
- neutral
- surprise
<<Enumeration>> Gender
- Male
- Female
Models
<<subsytem>> Streaming
<<StereoType>> Messaging
+ streamName: string
+ configuration : Config
+ consumer :
+ sendDataToServer(data: dictionary)
<<subsystem>> Data Manager
<<StereoType>> DataModel
- faceFeatureVector : nparray
- facialExpression : nparray
- age : integer
- gender : integer
- faceID : integer
+ JSONDecomposer (data : JSON)
<<StereoType>> DataReceiver
- model : DataModel
+ FaceCluster (faceFeatureVector : nparray)
+ sendDataToAnalayzer (model : DataModel)
+ ModelCreator (data : JSON)
<<subsystem>> Data Analysis
<<StereoType>> GraphVisualizer
+ visualize(graph: Graph)
<<StereoType>> Logger
+ Attribute 1 : Type
+ Attribute 2 : Type
-
- Attribute 4 : Type
+ generateLog(event_id: int)
<<subsystem>> UI Manager
<<StereoType>> GraphUI
+ graph : Graph
+ getGraph (username:string) : return

Figure 7: Class Interfacess

5. References

[1] *Git*. [Online]. Available: https://git-scm.com/. [Accessed: 18-Feb-2019].

[2] “The first single application for the entire DevOps lifecycle,” *GitLab*. [Online]. Available: https://gitlab.com/. [Accessed: 18-Feb-2019].

[3] "Welcome To UML Web Site!", *Uml.org*, 2019. [Online]. Available: http://www.uml.org/. [Accessed: 18- Feb- 2019].

[4] "Amazon Kinesis", *Amazon Web Services, Inc.*, 2019. [Online]. Available: https://aws.amazon.com/tr/kinesis/. [Accessed: 18- Feb- 2019].

[5] “Elasticsearch,” *Elastic Blog*. [Online]. Available: https://www.elastic.co/products/elasticsearch. [Accessed: 18-Feb-2019].

[6] "Locality Sensitive Hashing (LSH) Home Page", *Mit.edu*, 2019. [Online]. Available: https://www.mit.edu/~andoni/LSH/. [Accessed: 18- Feb- 2019].

[7] “Neo4j Graph Platform – The Leader in Graph Databases,” *Neo4j Graph Database Platform*. [Online]. Available: https://neo4j.com/. [Accessed: 18-Feb-2019].