Mugla Journal of Science and Technology

HIGH PERFORMANCE FACIAL RECOGNITION MATCHER

Gulsum AKKUZU KAYA*, Computer Engineering /Faculty of Engineering and Architecture, Kirsheir Ahi Evran University/ Turkey, gulsum.akkuzukaya@ahievran.edu.tr
(https://orcid.org/0000-0003-1806-7759)

Received: 14.12.2022, Accepted: 23.04.2023 Research Article
*Corresponding author DOI: 10.22531/muglajsci.1218915

Abstract

The utilization of biometric products is an expanding landscape; from general consumers employing it for authenticating into their devices to governments deploying it at the forefront of crime and border control. One sizeable organization required an expansion in their offering within the industryThis study aims to develop a facial matching solution that offers high performance and meets the requirements of the organization's biometric Subject Matter Experts in order to meet the current gap in the offering. A facial recognition approach known as FaceNet was utilized along with the GO language and MongoDB to produce an application capable of performing enrolments and matches against a persistent set of candidates. This solution was validated against the labeled Faces in the Wild dataset, a challenging set of facial biometric data in function, performance, and accuracy testing. For a subset of 6000 images from the dataset, a 100 % accuracy was recorded from multiple test runs demonstrating no false matches. The application's performance against this subset was averaged over multiple executions using two concurrent connections, which concluded an average enroll response time of 70ms and 236ms for match requests giving transactions per second values of 29 and 8 respectively.

Keywords: Face recognition, Machine learning, Robust, Reliable, Functionality

YÜKSEK PERFORMANSLI YÜZ TANIMA EŞLEŞTİRİCİSİ

Özet

Biyometrik ürünlerin kullanımı sürekli genişleyen bir çerçevede olup günden güne artmaktadır; cihazlarında kimlik doğrulaması yapmak için kullanan genel tüketicilerden, onu suç ve sınır kontrolünün ön saflarında konuşlandıran hükümetlere kadar hemen her alanda biyometrik urunler kullanilmaktadir. Bu çalışma, buyumekte olan bir siber güvenlik sirketinin mevcut program boşluğunu doldurmak için yüksek performans ve kuruluşun biyometrik gereksinimlerini karşılayan bir yüz eşleştirme çözümü sunan programi geliştirmeyi amaçlamaktadır. Sirket bünyesinde sürekli bir calişan grubuna ait kayıtları ve eşleşmeleri hızlı bir şekilde gerçekleştirebilen bir uygulama üretmek için; FaceNet olarak bilinen bir yüz tanıma yaklaşımı, GO programlama dili ve MongoDB kullanıldı. Bu çözüm, işlev, performans ve doğruluk testi biçiminde iyi bilinen ve zorlu bir yüz biyometrik veri seti olan Labelled Faces Wild veri setine karşı doğrulandı. Veri kümesindeki 6000 görüntüden oluşan bir alt küme için, yanlış eşleşme olmadığını gösteren çoklu test çalıştırmalarından %100 doğruluk kaydedildi. Uygulamanın bu alt kümeye karşı performansının, iki eşzamanlı bağlantı kullanılarak birden fazla yürütme üzerinden ortalaması alındı; bu, sırasıyla 29 ve 8 saniye başına işlem değerleri veren eşleştirme istekleri için ortalama kayıt yanıt süresi 70 ms ve 236 ms olarak sonuçlandı. Bu araştırmanın yazılım ürünü, türetilmiş gereksinimleri karşılamada başarılıydı, bu da calismamizi bir kurumsal çözüm için ideal bir temel haline getirdi.

Anahtar Kelimeler: Yüz tanıma, Makine öğrenimi, Sağlam, Güvenilir, İşlevsellik

Cite
Akkuzu Kaya, G. (2023). "High Performance Facial Recognition Matcher", Mugla Journal of Science and Technology, 9(1), 42-52.

1. Introduction

Biometrics are a fundamental concept utilized within systems across various industries, from security to crime prevention. They are used knowingly by everyone daily to unlock their phone or unknowingly through technology such as public closed-circuit television (CCTV). Although there is a plethora of facial recognition solutions across the community, there are minimal applications that offer the full functionality of a facial matcher without significant cost. This is partially due to the immense complexity of such solutions that must utilize cutting-edge technology in fields such as machine learning in order to stay competitive. Not only do biometric applications need to be performant but also ethical, secure, and reliable. From the biometric point, the image processing methods and Machine Learning (ML) techniques are needed to be used; image processing

is needed for face recognition. ML techniques then is needed to classify the processed images.
Although the biometric industry contains a vast landscape of corporations, the company, which is the case study of this research, is one of the up-and-coming giants in IT that are branching out into this product space. Currently, they have a minimal offering with products such as a palm vein-based authentication method. In order to breach the facial recognition market, they need to develop a new solution that provides considerable benefits over the competition, such as high performance, scalability, and utilization of fully open-source technologies. The aim of this research is to design, develop and evaluate a solution that provides a high-performant, scalable, and accurate facial matcher that addresses a current gap within the biometric market. This aim was derived from an internal organization's problem in improving its rapidly growing bio-metric with a focus on crime prevention and border security.
The research objectives of this study are as follows:

• To design and develop an application that meets the requirements of SMEs in the industry and the overall research aim.

• To conduct performance and accuracy tests to benchmark where the solution fits within the market and if the application has met its requirements.

• To perform functional testing to confirm the application is robust and reliable.

The structure of this paper is as follows; Section 2 includes the design of this research and its methodological steps explanation. In Section 3, detailed information of the developed applicationis given. In the following section, we explained the implementation phases and details of testing. In Section 5, the research is concluded. The evaluation of this work is also given in this section.

2. Research Design and Methodology

Software development projects, in general, are cumbersome and may involve large numbers of people to achieve end the goal. In order to be actioned efficiently, there are frameworks that can be followed which provide an outline of the entire project cycle from initial planning to execution [5–7]. However not all projects are alike, they vary in size, complexity, and time frames, meaning one framework can not suit all. One of the most mature software methodologies that are still widely used is Waterfall [16]. This framework is very rigid and sequential, following a process of requirements gathering, design, development, validation, and support. The waterfall is used extensively for its simple structure that scales well and has resulted in many successfully delivered projects. However, as technology advances, projects are changing and, in many situations, requiring a different style of framework. One of the biggest transitions in modern development is the need for fluidity in requirements and the ability to quickly adapt to changing environments [1,8]. This is something that Waterfall does not inherently support, which has led to numerous variations of the framework that make it more iterative to combat its inelasticity[4]. Although there is a variety of Waterfall implementations that aim to provide longevity in a world of ever-adapting technology and projects, new frameworks built for these scenarios are becoming more commonly used; with one of the major players being Agile. This framework at its core aims to be iterative and designed for a highly adaptive environment. The Agile methodology outlines some key principles that a subset of frameworks abide by; one of the most popular being Scrum [2,11]. However, the flexibility of Agile poses some challenges and confusion around what makes a project comply with the principles and how to fully define if one is implementing the framework correctly. This is explored by Koubaa et al. [3,9,10] who concluded that less than 15% of projects in their study of 556 data points completely comply with the full Agile principles. However, they did find a trend in the adoption of agility and how this has a positive correlation with project success. Agile is not perfect for all scenarios but, when implemented correctly, has been observed to enhance the likelihood of achieving the goals of a software development project that is operating with modern technologies and requirements. We used Agile because it is better suited for small and fast-paced projects.

2.1. Data Collection

There are three main sections of data collection to achieve the research aims that require a range of data analysis methods:

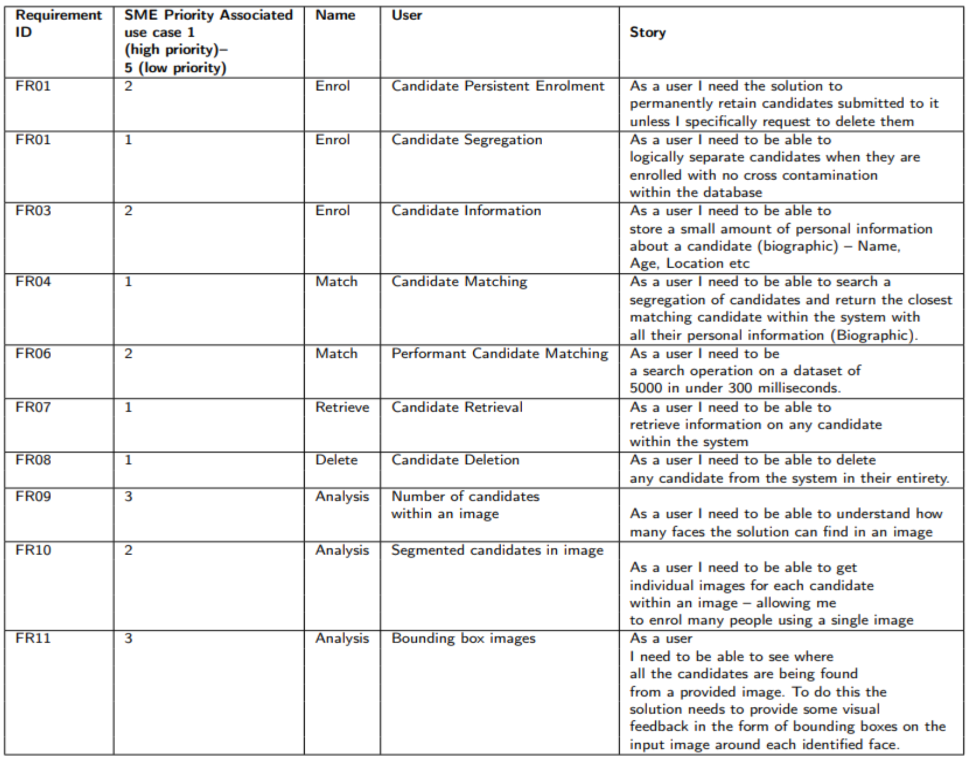
• Performance metrics for the solution are collected to evaluate the efficiency of the system under various scenarios such as how the application throughput changes with the size of candidates in the database. This data has a correlation analysis and it is an interval type.

• Accuracy metrics for the solution are collected to evaluate the precision of the system. Multiple runs of accuracy tests are used on a large set of data to get a range of metrics which are then analyzed to extract the average false match percentage.

• Requirements gathering utilizes surveys to construct and validate the requirements of the software.

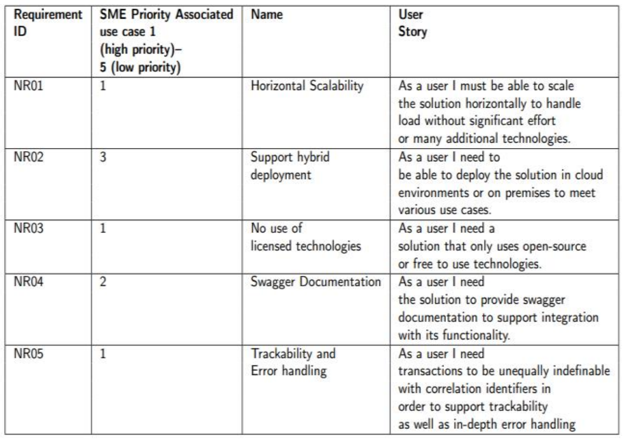
2.2. Requirements Gathering

Through the research conducted within the literature re-view, a base set of requirements has been derived to address current gaps within the biometric market. However, further requirements gathering is conducted to gain an understand- ing of priority and additional requirements. This action aims to validate and prioritize the requirements to align them with the needs of a real-world company. Therefore, to achieve this, a survey is issued to a small subset of employees within the company's bio-metric division. The employees in question are targeted as subject matter experts (SMEs) and are able to provide an in-depth understanding of what is required from a facial matcher. Candidates for the survey are identified through the Chief Technology Officer of the company's bio-metric offering. This approach is taken as it provides an efficient way of sending a survey to the required individuals. Figure 1 illustrates the functional requirements and detailed explanations of each requirement. The non-functional requirements of this research are presented in Figure 2.



Requirement ID	SME Priority Associated use case 1 (high priority)– 5 (low priority)	Name	User	Story
FR01	2	Enrol	Candidate Persistent Enrolment	As a user I need the solution to permanently retain candidates submitted to it unless I specifically request to delete them
FR01	1	Enrol	Candidate Segregation	As a user I need to be able to logically separate candidates when they are enrolled with no cross contamination within the database
FR03	2	Enrol	Candidate Information	As a user I need to be able to store a small amount of personal information about a candidate (biographic) – Name, Age, Location etc
FR04	1	Match	Candidate Matching	As a user I need to be able to search a segregation of candidates and return the closest matching candidate within the system with all their personal information (Biographic).
FR06	2	Match	Performant Candidate Matching	As a user I need to be a search operation on a dataset of 5000 in under 300 milliseconds.
FR07	1	Retrieve	Candidate Retrieval	As a user I need to be able to retrieve information on any candidate within the system
FR08	1	Delete	Candidate Deletion	As a user I need to be able to delete any candidate from the system in their entirety.
FR09	3	Analysis	Number of candidates within an image	As a user I need to be able to understand how many faces the solution can find in an image
FR10	2	Analysis	Segmented candidates in image	As a user I need to be able to get individual images for each candidate within an image – allowing me to enrol many people using a single image
FR11	3	Analysis	Bounding box images	As a user I need to be able to see where all the candidates are being found from a provided image. To do this the solution needs to provide some visual feedback in the form of bounding boxes on the input image around each identified face.

Figure 1. An Example of Functional Requirements



Requirement ID	SME Priority Associated use case 1 (high priority)– 5 (low priority)	Name	User Story
NR01	1	Horizontal Scalability	As a user I must be able to scale the solution horizontally to handle load without significant effort or many additional technologies.
NR02	3	Support hybrid deployment	As a user I need to be able to deploy the solution in cloud environments or on premises to meet various use cases.
NR03	1	No use of licensed technologies	As a user I need a solution that only uses open-source or free to use technologies.
NR04	2	Swagger Documentation	As a user I need the solution to provide swagger documentation to support integration with its functionality.
NR05	1	Trackability and Error handling	As a user I need transactions to be unequally indefinable with correlation identifiers in order to support trackability as well as in-depth error handling

Figure 2. An Example of Non-Functional Requirements

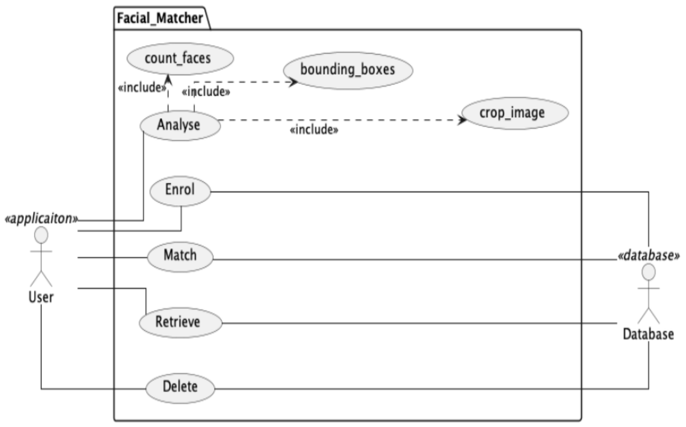


Figure 3. Use Case Diagram for facial matcher design

3. Application Design

As per the requirements, there is a variety of functionality required in order to produce a useful matcher product. Figure 1 illustrates the various functions of the system.

• Enrol: The enroll functionality provides the ability to add a new candidate into the system. It takes bio- metric information (an image of a face) and biographic data (candidate information such as name, age etc. . .) and uses this data to store the candidate in the solution. The storing of both biometric and biographic data aims to satisfy requirement FR03. One piece of biographic data is the collection a candidate is being stored in and is used by the solution to segregate candidates in order to comply with requirement FR02. The biometric data stored should be the feature vector of the candidates' faces as this is the minimum requirement for matching.

Once a candidate is enrolled, their data is persisted indefinitely until a user specifically requests for them to be deleted through their unique identifier (This satisfies requirement FR01). Therefore, they can be searched against using the match functionality while they exist in the solution. Finally, this functionality is expected to occur performantly as per requirement FR04. This use case utilizes an external database.

• Match: The matching functionality provides the ability for a user to take a face image and use it to search against the solution and its enrolled candidates to look for the closest possible match. If a close enough match is found the matcher returns all the information it has on the candidate. This functionality complies with requirement FR05. The matching process is completed performantly to comply with requirement FR06. This use case utilizes an external database that has been populated using the enroll functionality disclosed above.

• Retrieve: The retrieve functionality provides a user the ability to get all the information of a previously enrolled candidate, including both biometric and biographic data, using their unique identifier. This use case satisfies requirement FR07 and has an external dependency on the solutions database where candidates are stored.

• Analyze:The analyze functionality provides a user the ability to understand their biographic data and how it gets interpreted by the solution. It should allow the ability to count the number of faces it finds in images, an image with boxes around all identified faces, and segmented images of each face from an image. All this functionality aims to satisfy requirements: FR09, FR10 FR11. This use case is the only one without any other external dependencies as it does not interact with the database and is a purely stateless flow.

3.1. High Level Architecture

In order to meet the requirements, the proposed architecture is depicted in Figure 4a and Figure 4b. This illustration comprises three key components: a load balance, instances of the matcher service, and finally a database. A proposed cloud variation of this can be seen in Figure 5. The load balancer component in Figure 4 is used to allow the matcher services to scale horizontally. It does this by routing incoming traf- fic to a pool of registered servers. The figure shows that HTTPS traffic will be transmitted into the load balancer but forwarded on as HTTP, this is important as SSL is crucial for securing the solutions' communication with the outside world. However, once a connection has come into the private network (either on-premises or in a VPC) continuing with SSL would add additional CPU overhead. Keeping the encryption on a dedicated server such as an F5 load balancer reduces this burden and therefore, increases the performance of the system. This is crucial to meet requirements. To do so, the key design decision is the matcher needs to accept HTTP traffic and should not be expected to deal with encryption. A load balancer is utilized to do this.

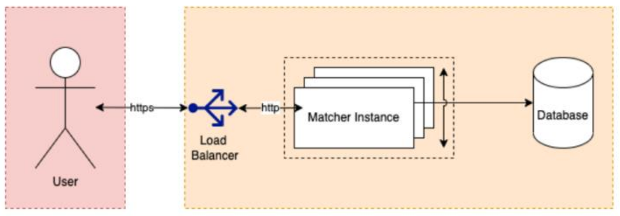


Figure 4a. Use Case Diagram for facial matcher design

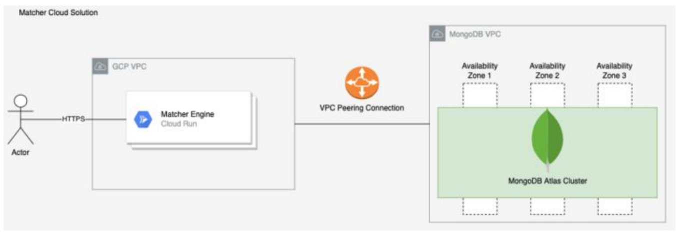


Figure 4b. Use Case Diagram for facial matcher design

3.2. Request and Response Models

Each transaction shouldn't have an expected request and response body, due to the underlying interface being REST, these objects are in JSON format as this is the standard. The JSON formats are explored more in the swagger definition of the application, however, to provide context to the sequences of transactions in the interface section Figure 5 illustrates the class diagram for the internal application models for all accepted and return objects. This shows what the application expects for each type of transaction and what it theoretically responds with.

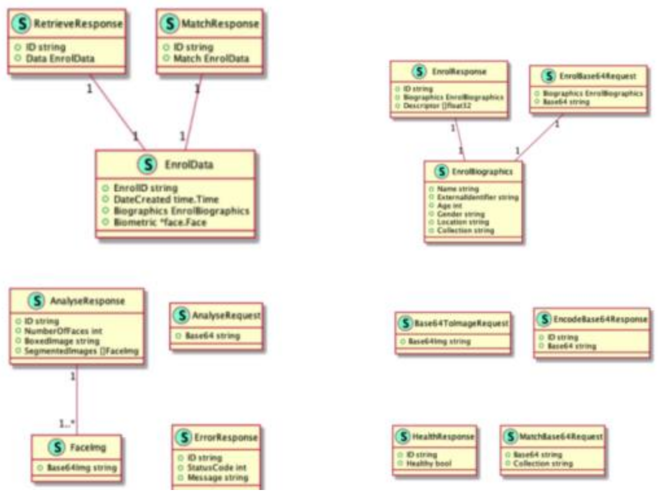


Figure 5. Class Diagram of Request and Response Objects

3.3. Enroll Sequence Flow, Match Sequence Flow, Delete Sequence Flow, Retrieve Sequence Flow, and Analyse Sequence Flow

The enroll flow is given in Figure 6 and is initialized with a post request to the /enroll resource. The first step is to parse the request and validate it to be valid. If a request is invalid, it immediately closes the connection with a status of 404. Once a request is considered valid, the application takes the parsed image and runs a facial detection algorithm on it. In that case, the recommended trained model from the go-face library is used($mmodhuman\ facedetector.dat$). Once facial detection has been completed, the number of faces is known, if this value does not equal exactly one face in an image the connection is closed with a 404 status code as only one face can be accepted per request due to biographic data being required for a candidate. Now it is concluded there is a valid request in the system with a single candidate's biometrics, the 128-vector embedding is calculated using the $dlib\ face\ recognition\ resnet\ modelv1$ trained model. Due to the detection step, there should never be an error during the templating process (although the application should gracefully handle any unexpected error with a status 500). The generated template is then stored in the database, within a specific collection defined in the request, as well as all other biographic information of the candidate. If the database interaction fails, a 500 status will be returned. If the write was successful, the application completes a JSON response and returns it with status 200.

The match flow can be seen in Figure 7 and illustrates the process of doing a match transaction. Similarly, to the enroll flow, the first steps in handling the request are to parse it and verify it – if this fails a 404 response is returned to the caller. Once the request is parsed, it then goes through the same process as the enroll – any faces are detected with a 404 error being returned if there are too many of few faces followed by the generation of the biometric embedding. Now an embedding is calculated for the input candidate, the logic flow deviates from enrolling with the next step being to retrieve all candidate biometrics from a specific collection (stored in the database). If an error occurs during the retrieve process a 500 error is returned. Once the embedding of the input candidate is known as well as all the previously enrolled candidates for a given collection being searched, a classification machine learning model is run. This model uses clustering to find the closest match to the input candidate. The enrolled candidate with the highest similarity is returned only if it is within a given threshold of similarity (this should be configurable in the application settings). If the classification model fails, a 500 is returned to the caller with a message – this message should give context as to why a match could not be returned such as "no candidates within threshold" or "internal error, unable to classify". If successful, a response is created using the matched candidates' stored biometric and biographic data which are returned to the caller.

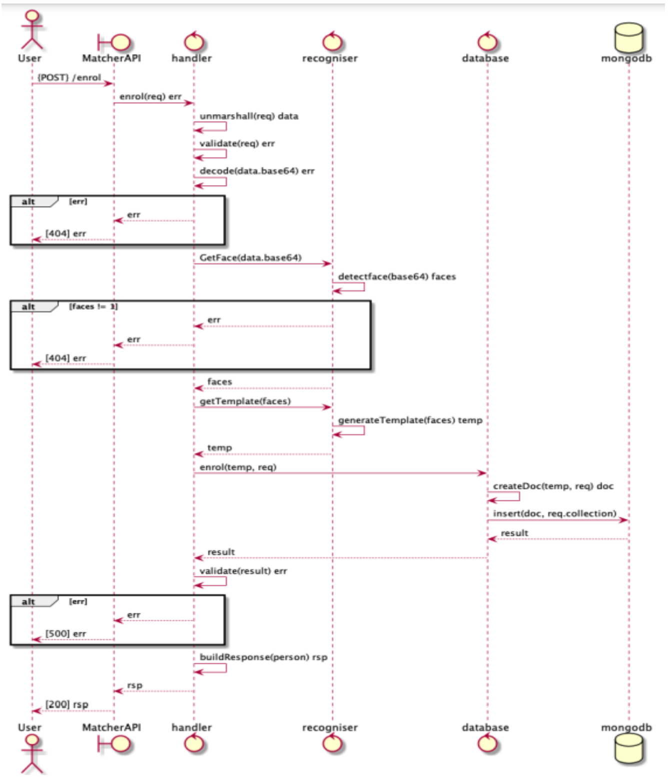


Figure 6. Enrol Sequence Diagram

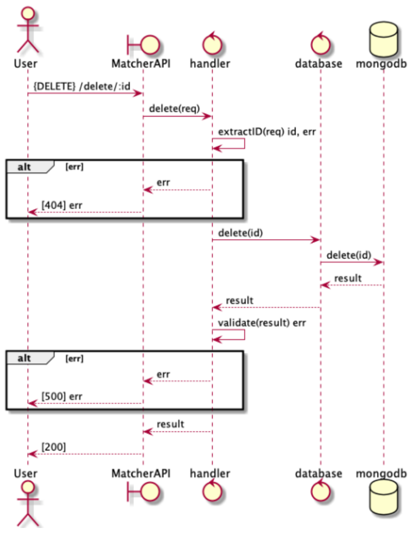


Figure 7. Match Sequence Diagram

The delete sequence diagram is shown in Figure 8 which illustrates the logic flow for this transaction. Delete is a straightforward request using the REST DELETE type and Figure 7: Match Sequence Diagram it has no interaction with underlying machine learning models. Once a request is received, the unique identifier of an enrolled candidate is passed in as a URL parameter. The matcher first extracts this ID – if it is not present or in the right format a 404 error is returned. Once the ID is known, a delete request is sent to the database for that candidate. If the delete fails, for example, if the candidate does not exist or if the database connection is lost – a 500 error is returned with a message. If a successful delete is returned from the database a status of 200 is returned to the user with no response body.

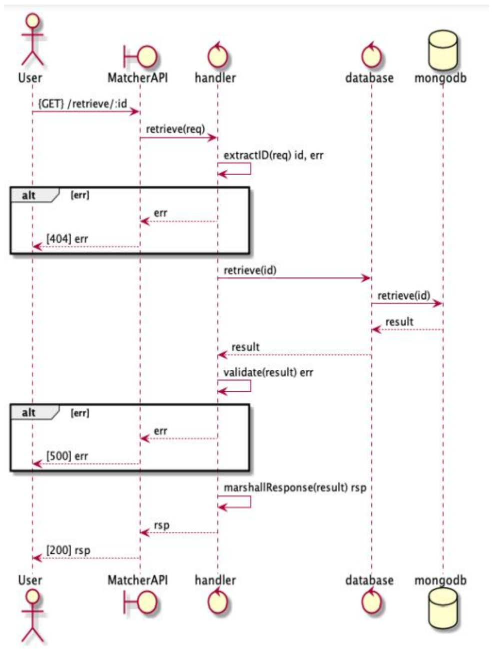


Figure 8. Delete Sequence Diagram

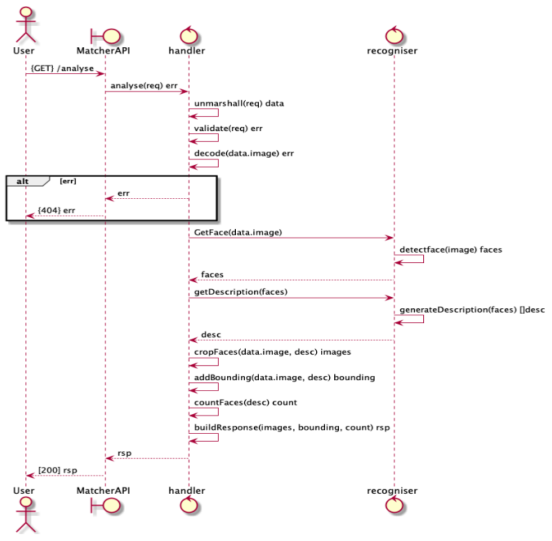


Figure 9. Retrieve Sequence Diagram

The retrieve sequence diagram is seen in Figure and is very similar to that of a delete. However, for this transaction rather than using the URL passed ID to delete from the database, it gets the candidate and returns all data stored for them. Here a 200 is returned with a response body when compared to a deleted stand-alone status code. The data returned is explored in the model's section. The analyze sequence diagram is illustrated in Figure 10 and is one of the most complex flows due to the many operations it performs. However, this type of transaction does not have any interaction with the database and is purely within the matcher. The first stage of the flow is the same as enroll and match – the request is received, parsed, and validated with 404 being returned if any of this fails. The next step is like the aforementioned transactions in terms of the detection process; however, it does not care how many faces are found (only unexpected errors are handled here with a 500- status code). For every face found, a template will be generated for each as well as other information such as location in the image. Using the location in the image which represents the coordinates of a rectangle, each face can be individually cropped out as well as have boxes drawn on the original image. This results in three new pieces of information; the number of faces in the input image, individually cropped images per face, and bounding boxes drawn on the original image for each face. This new data is bundled up into a response object and returned to the user with a 200-status code.

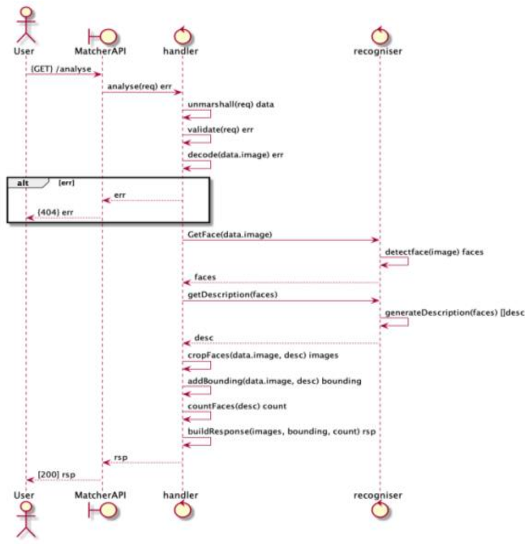
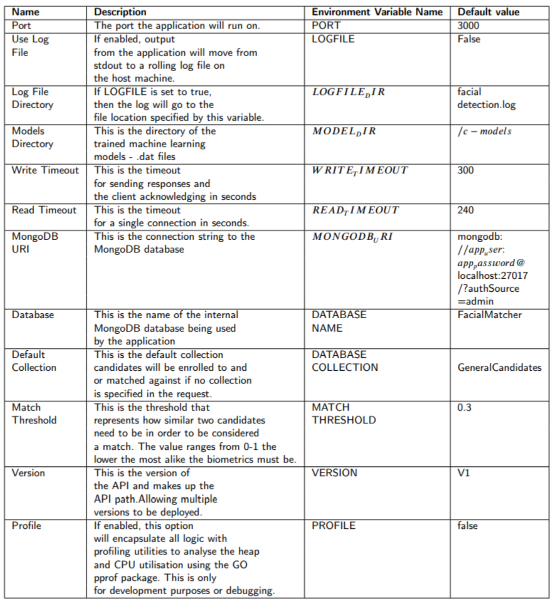


Figure 10: Analyze Sequence Diagram

4. Implementation and Testing

4.1. Implementation

Although not a core package of the matcher, many other internal modules have a dependency on the configuration module. This module encapsulates all the configuration options and populates them on initialization with environment variables – if these variables are not present then default values are automatically populated. All available options are given in Figure 11 and can be configured by any administrator deploying the facial matcher solution. A depiction of how these variables sit within the internal module is presented with a class diagram in Figure 12. The Recogniser package encapsulates all the machine learning logic to perform facial detection, embedding creation, and matching. Figure 13 illustrates the class diagram of this component. As shown, it is initialized with a dependency on a populated con-fig structure and exposes five functions.



Name	Description	Environment Variable Name	Default value
Port	The port the application will run on.	PORT	3000
Use Log File	If enabled, output from the application will move from stdout to a rolling log file on the host machine.	LOGFILE	False
Log File Directory	If LOGFILE is set to true, then the log will go to the file location specified by this variable.	$LOGFILE_DIR$	facial detection.log
Models Directory	This is the directory of the trained machine learning models - .dat files	$MODEL_DIR$	$/c - models$
Write Timeout	This is the timeout for sending responses and the client acknowledging in seconds	$WRITE_TIMEOUT$	300
Read Timeout	This is the timeout for a single connection in seconds.	$READ_TIMEOUT$	240
MongoDB URI	This is the connection string to the MongoDB database	$MONGODB_URI$	mongodb: $//app_user$: $app_password@$ localhost:27017 /?authSource =admin
Database	This is the name of the internal MongoDB database being used by the application	DATABASE NAME	FacialMatcher
Default Collection	This is the default collection candidates will be enrolled to and or matched against if no collection is specified in the request.	DATABASE COLLECTION	GeneralCandidates
Match Threshold	This is the threshold that represents how similar two candidates need to be in order to be considered a match. The value ranges from 0-1 the lower the most alike the biometrics must be.	MATCH THRESHOLD	0.3
Version	This is the version of the API and makes up the API path.Allowing multiple versions to be deployed.	VERSION	V1
Profile	If enabled, this option will encapsulate all logic with profiling utilities to analyse the heap and CPU utilisation using the GO pprof package. This is only for development purposes or debugging.	PROFILE	false

Figure 11: All available configuration options

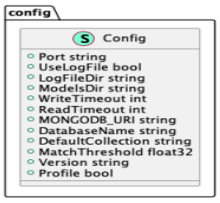


Figure 12. Retrieve Sequence Diagram

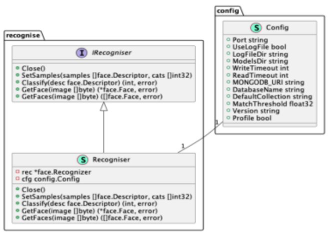

recognise
IRecogniser
Close()
SetSamples(samples []face.Descriptor, cats []int32)
Classify(desc face.Descriptor) (int, error)
GetFace(image []byte) (*face.Face, error)
GetFaces(image []byte) ([]face.Face, error)
Recogniser
rec *face.Recognizer
cfg config.Config
Close()
SetSamples(samples []face.Descriptor, cats []int32)
Classify(desc face.Descriptor) (int, error)
GetFace(image []byte) (*face.Face, error)
GetFaces(image []byte) ([]face.Face, error)
config
Config
Port string
UseLogFile bool
LogFileDir string
ModelsDir string
WriteTimeout int
ReadTimeout int
MONGODB_URI string
DatabaseName string
DefaultCollection string
MatchThreshold float32
Version string
Profile bool
1
1

Figure 13. Recogniser Package Class Diagram

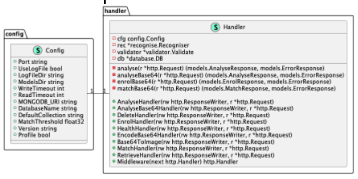

config
Config
Port string
UseLogFile bool
LogFileDir string
ModelsDir string
WriteTimeout int
ReadTimeout int
MONGODB_URI string
DatabaseName string
DefaultCollection string
MatchThreshold float32
Version string
Profile bool
1
1
handler
Handler
cfg config.Config
rec *recognise.Recogniser
validator *validator.Validate
db *database.DB
analyse(r *http.Request) (models.AnalyseResponse, models.ErrorResponse)
analyseBase64(r *http.Request) (models.AnalyseResponse, models.ErrorResponse)
enrolBase64(r *http.Request) (models.EnrolResponse, models.ErrorResponse)
matchBase64(r *http.Request) (models.MatchResponse, models.ErrorResponse)
AnalyseHandler(rw http.ResponseWriter, r *http.Request)
AnalyseBase64Handler(rw http.ResponseWriter, r *http.Request)
DeleteHandler(rw http.ResponseWriter, r *http.Request)
EnrolHandler(rw http.ResponseWriter, r *http.Request)
HealthHandler(rw http.ResponseWriter, r *http.Request)
EncodeBase64Handler(rw http.ResponseWriter, r *http.Request)
Base64ToImage(rw http.ResponseWriter, r *http.Request)
MatchHandler(rw http.ResponseWriter, r *http.Request)
RetrieveHandler(rw http.ResponseWriter, r *http.Request)
Middleware(next http.Handler) http.Handler

Figure 14. Handler Class Diagram

The handler package contains all the HTTP logic. The package can be illustrated in Figure 14 with a class diagram. As well as many of the internal packages of the matcher it has a dependency on the con-fig, however, it also needs to be initialized with the recognizer and database structures. Private functions hold the business logic of the operations whereas public functions are those that are bound to the server and handle all HTTP logic. The Database package contains all the logic regarding the database. The package has a dependency on the application configuration. The class diagram for this package can be seen in Figure 15. As shown it has an interface, this is to allow for dependency injection and standalone unit tests. However, it also allows for another database package to be created and, if it complies with the interface, all other areas of the application can use it without change. For example, a MySQL package could be created as well as MongoDB – the application could then easily support both types of databases.

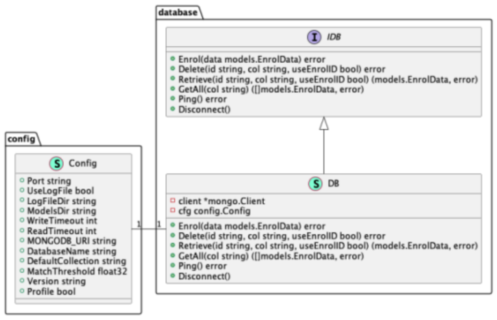
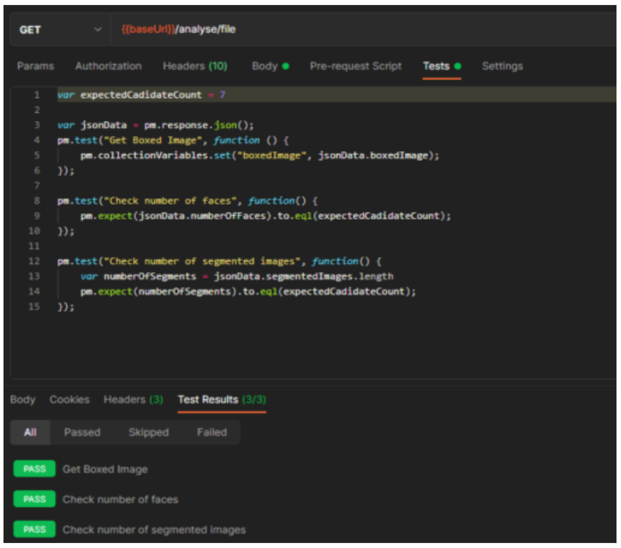

database
IDB
Enrol(data models.EnrolData) error
Delete(id string, col string, useEnrollD bool) error
Retrieve(id string, col string, useEnrollD bool) (models.EnrolData, error)
GetAll(col string) ([]models.EnrolData, error)
Ping() error
Disconnect()
DB
client *mongo.Client
cfg config.Config
Enrol(data models.EnrolData) error
Delete(id string, col string, useEnrollD bool) error
Retrieve(id string, col string, useEnrollD bool) (models.EnrolData, error)
GetAll(col string) ([]models.EnrolData, error)
Ping() error
Disconnect()
config
Config
Port string
UseLogFile bool
LogFileDir string
ModelsDir string
WriteTimeout int
ReadTimeout int
MONGODB_URI string
DatabaseName string
DefaultCollection string
MatchThreshold float32
Version string
Profile bool
1
1

Figure 15. Database Package Class Diagram

4.2. Testing

The application is evaluated with functional testing, accuracy and performance. The validation of image objects is a difficult problem, however, Postman does atomically check: the number of faces identified is seven, the number of segmented images returned is also seven and the boxed base64 object isn't empty. The application is evaluated with functional testing, accuracy and performance. The validation of image objects is a difficult problem, however, Postman does atomically check: the number of faces identified is seven, the number of segmented images returned is also seven and the boxed base64 object isn't empty. The test illustrated in Figure 16 shows how a test is written in postman and that all of them passed – allowing simple automation of testing.



```
var expectedCadidateCount = 7

var jsonData = pm.response.json();
pm.test("Get Boxed Image", function () {
    pm.collectionVariables.set("boxedImage", jsonData.boxedImage);
});

pm.test("Check number of faces", function() {
    pm.expect(jsonData.numberOfFaces).to.eql(expectedCadidateCount);
});

pm.test("Check number of segmented images", function() {
    var numberOfSegments = jsonData.segmentedImages.length
    pm.expect(numberOfSegments).to.eql(expectedCadidateCount);
});
```

```
PASS Get Boxed Image
PASS Check number of faces
PASS Check number of segmented images
```

Figure 16: Screenshot of Analyse test in Postman

4.3. Performance and Accuracy Testing

Performance and accuracy testing of a biometric solution is a challenging and complex problem. This is made increasingly more challenging when these kinds of tests need to be run consistently through the application's life cycle to test how changes affect both its performance and accuracy. In order to do this effectively, it needs to be simple to perform a consistent test with very little configuration and developer effort. To overcome this challenge a specific tool has been designed to accompany the matcher solution and allow developers to test changes while also giving the potential for users to check how it performs with their hardware and data requirements. This tool has been labeled as FBAP (Facial Bio-metric Accuracy and Performance testing). Figure 17.a shows the basic flow of logic for FBAP and how it internally works. FBAP is a CLI application that was developed using the swagger definition of the facial matcher- using this a GO client can be generated. Using the client, once images are read from disk, they can be converted to base64 using one of the matcher helper endpoints. These images get piped into an in-memory queue, along with some file data. Multiple threads then pull from the queue to perform simultaneous enrolls, using the file name as the candidate ID in the request. The time taken for the enroll to complete is recorded and published to another internal queue for later processing along with the enrolled candidate's information. Once all the enrolls have been performed, the matching process is triggered – pulling of the same candidate queue as enroll a match is done and the returned candidate and response time is recorded and sent to another queue. Finally, the result stage pulls in all the enroll and match results and generates a PDF output with a row being a candidate read from disk with an associated enroll and match response time. As well as response time (measuring performance) the match result is compared to the input candidate; if they are not equal then the match has failed, and this is also shown in the CSV output. Figure 17.b presents an example of the

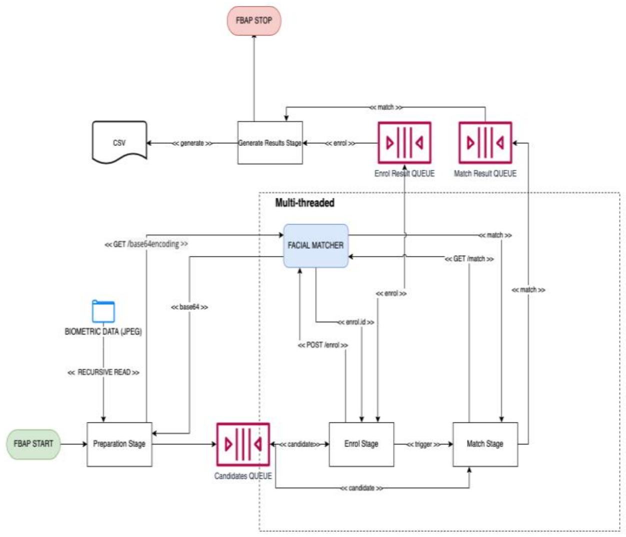
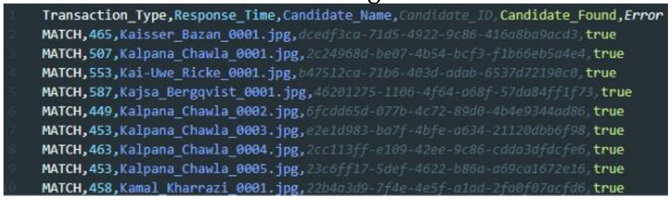


Figure 17.a : A high-level overview of FBAP and its internal logic



```
Transaction_Type,Response_Time,Candidate_Name,Candidate_ID,Candidate_Found,Error
MATCH,465,Kaisser_Bazan_0001.jpg,dcedf3ca-71d5-4922-9c86-416a8ba9acd3,true
MATCH,507,Kalpana_Chawla_0001.jpg,2c24968d-be07-4b54-bcf3-f1b66eb5a4e4,true
MATCH,553,Kai-Uwe_Ricke_0001.jpg,b47512ca-71b6-403d-adab-6537d72190c0,true
MATCH,587,Kajsa_Bergqvist_0001.jpg,46201275-1106-4f64-a68f-57da84ff1f73,true
MATCH,449,Kalpana_Chawla_0002.jpg,6fcdd65d-077b-4c72-89d0-4b4e9344ad86,true
MATCH,453,Kalpana_Chawla_0003.jpg,e2e1d983-ba7f-4bfe-a634-21120dbb6f98,true
MATCH,463,Kalpana_Chawla_0004.jpg,2cc113ff-e109-42ee-9c86-cdda3dfdcfe6,true
MATCH,453,Kalpana_Chawla_0005.jpg,23c6ff17-5def-4622-b86a-a69ca1672e16,true
MATCH,458,Kamal_Kharrazi_0001.jpg,22b4a3d9-7f4e-4e5f-a1ad-2fa0f07acfd6,true
```

Figure 17.b : An example of FBAP results CSV output

When performing testing using FBAP for this research, a subset of the LFW dataset will be used; around 6000 candidates in order to comply with requirement FR06 with 1000 extra candidates to assess a worst-case scenario. The features of the machine used for this study are; Intel(R) Core (TM) i7- 10750H CPU @ 2.60GHz,2592 MHz, 6 Core(s), 12 Logical Processor(s), RAM 16.0 GB, OS Microsoft Windows 10 Home, DELL XPS 15 9500, X64. When analyzing the enroll results the following findings are derived from FBAP output:

- Average Response Time: 69.83ms (rounded to two decimal places)
- Transactions Per Second: 28.64 (rounded to two decimal places)
- Error Rate: 0
- Slowest Transaction: 84ms
- Quickest Transaction: 55ms

As illustrated in Figure 18 the spread of response times for enrolling transactions was consistent with minimal variation. There were also no apparent trends seen – this is reassuring as the response time did not seem to increase in conjunction with the number of candidates inserted into the system. Therefore, an assumption can be made that the response times for enrolls should be consistent no matter the size of the solution.

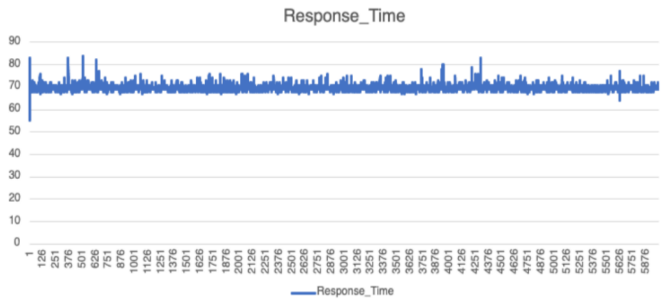


Figure 18: The spread of transaction times for enroll requests from the Test

When analyzing the Match results the following findings can be derived from FBAP output:

- Average Response Time: 236.45ms (rounded to two decimal places)
- Transactions Per Second: 8.46 (rounded to two decimal places)
- Error Rate: 0% Accuracy: 100% (no false matches were made)
- Slowest Transaction: 371ms
- Quickest Transaction: 212ms

Figure 19 demonstrates the spread of response times for the match transaction type. Although the data is mostly consistent there are more spikes in transaction times compared to enrolments with slightly more spread. When investigating the CPU profiles, it was maxed out during the long running transactions; indicating a CPU bottleneck that correlated to high usage on MongoDB. Therefore, the database appears to be the cause of the transaction spikes. This investigation also raises concerns about the scalability of the solutions as the database usage correlates to the number of candidates stored. Therefore, the bottleneck grows with the size of the solution, and match response times will increase accordingly. This was also seen when a smaller test of 1000 candidates was performed giving an average enroll time of 70ms (consistent with the larger dataset) but match times were significantly less with an average of 73.17ms. The results of this reduced match test are shown in Figure 20. After running multiple performance tests, the database had 11,616 candidates in the database. This amounted to a total collection size of 17.7MB of data.

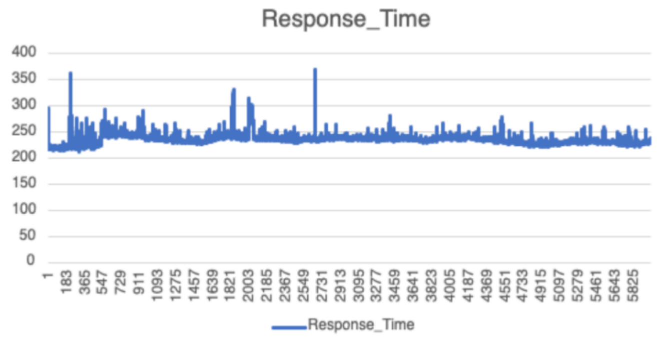


Figure 19: The spread of transaction times for match requests from Test

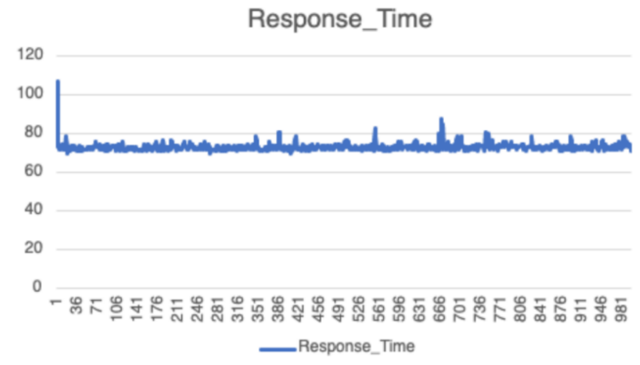


Figure 20: The spread of transaction times for a reduced match Test

4.4. Test Summary

The application held up to all the requirements it was developed to accomplish. With a range of functional tests to cover the basis of these requirements, a 100% success rate was observed with no defects coming to light. The test pack within postman provides a robust implementation that can be automated into a CI/CD pipeline going forward, however, the general coverage of testing could be improved with multiple tests validating each requirement with more focus on edge cases. For example, testing every biographic field individually to ensure it doesn't produce a bug. Thanks to the development of FBAP, performance and accuracy testing is now an easy task if the data is available. Using the tool, performance was easily measured and showed great promise–meeting all requirements derived by SMEs while maintaining very high accuracy. This testing showed a 100% accuracy for direct matches – enrolling and matching against identical biometric information. However, the tool needs to be updated to provide paired biometrics to calculate the accuracy of matches using a variety of biometrics for a single candidate. This initial test phase of a prototype application shows incredible promise and a working solution against the requirements.

5. Conclusion

We aimed to design, develop and evaluate a solution that provides a high-performant, scalable, and accurate facial matcher that addresses a current gap within the biometric market. The solution needed to comply with requirements derived from SMEs within the industry and against organization's internal business needs. Requirements gathering was conducted through an anonymous survey in order to not just understand the requirements of SMEs but also their priority. Through an evaluation of the current literature for facial matching, the concepts of FaceNet were explored where a convolutional neural network developed by Google in 2015 generates a 128 vector face embedding used for matching and is at the cutting edge of facial recognition technologies [3]. Combining the facial recognition techniques of FaceNet with the modern database of MongoDB and the performance of a compiled language like GO has resulted in highly performant and scalable solution. In order to evaluate the performance of the software, a custom testing application was developed to run various load scenarios with multiple concurrent transactions simulating multiple external users. Before running the performance testing, however, a fully automated functional testing suite was used to validate if the application met all the defined requirements – the results of which were all successful with all functional requirements receiving a good level of test coverage. There was a minimum of one test per functional requirement. Although no apparent bugs were seen, the overall test coverage could have been improved with more tests around edge and error cases. The results of the performance testing were also very promising, with all the data complying with the requirements and demonstrating very high throughput. In summary, the enrolled transactions consistently averaged response times of 70ms and match transactions getting 236ms. However, when varying the number of background candidates in the solution, enroll transactions remained consistent at 70ms whereas match response times correlated with the number of candidates within the system. This is to be expected as the more candidates there are the more comparisons need to be made as well as the higher volumes of data being transported, but this is one area that can be optimized going forward. Accuracy on the other hand was 100% with no false matches being returned for direct biometric matches; this is particularly impressive when considering the dataset being used was LFW which is known for being one of the most challenging for facial recognition [12].

As seen in the Performance and Accuracy Testing performant, show an section, the rate of match transactions, although increase in processing time in correlation to the number of candidates in the database. One way of improving the degradation of performance is to implement a cache into the architecture [13] to reduce the strain on the database and access larger amounts of data quicker. This could be through an external application such as Redis which is a very popular caching system that many solutions utilise to increase performance [14]. A more convoluted approach could be to design a new algorithm for sorting 128 categorising them –- vectors and reducing the number of similar embeddings to match against for a given candidate. One section of functionality SME's showed considerable interest was in the analysis of images using various machine learning techniques to perform functions such as quality assessment, tampering analysi s and soft biometric detection like age. These kind of analysis are possible as outlined by Galbally et al. [15], however, they do add considerable complexity.

In conclusion, the aims and objectives devised for this research have been met with the software solution. It shows incredible promise for being the foundation of an enterprise solution in the biometric industry and could be part of the organization's offering going forwards. It is a prototype and there are many areas that can be improved as well as further levels of functional and performance testing required to consider the solution enterprise-ready.

6. References

[1] Vijayasarathy, L.R., Butler, C.W., "Choice of software development methodologies". *Colorado State University.* 2016.

[2] Hoda, R., Salleh, N., Grundy, J.,. "The rise and evolution of agile software development". *IEEE software* 35, 58–6, 2018.

[3] Almeida, F., Simões, J., . "Moving from waterfall to agile: Perspectives from it portuguese companies". *International Journal of Service Science Management, Engineering, and Technology* (IJSSMET) 10, 30–43, 2019.

[4] Hill, K. "The secretive company that might end privacy as we know it*". In Ethics of Data and Analytics* (pp. 170-177), 2020.

[5] Kortli, Y., Jridi, M., Al Falou, A., and Atri, M.. "Face recognition systems: A survey". *Sensors*, 20(2), 342, 2020.

[6] Lim, K.Y.H., Zheng, P., Chen, C.H., . "A state-of-the-art survey of digital twin: techniques, engineering product lifecycle management and business innovation perspectives". *Journal of Intelligent Manufacturing* 31, 1313–1337, 2020.

[7] Wang, Z., Wang, G., Huang, B., Xiong, Z., Hong, Q., Wu, H., ... and Liang, J. "Masked face recognition dataset and application". *arXiv preprint* arXiv:2003.09093, 2020.

[8] Koubaa, A., Ammar, A., Kanhouch, A., AlHabashi, Y., "Cloud versus edge deployment strategies of real-time face recognition inference." *IEEE Transactions on Network Science and Engineering* 9, 143–160, 2021.

[9] Wang, M., and Deng, W.. "Deep face recognition: A survey." *Neurocomputing*, 429, 215-244, 2021.

[10] Smith, M., and Miller, S.. "The ethical application of biometric facial recognition technology". *Ai and Society*, 37(1), 167-175, 2022.

[11] Almeida, D., Shmarko, K., and Lomas, E. . "The ethics of facial recognition technologies, surveillance, and accountability in an age of artificial intelligence: a comparative analysis of US, EU, and UK regulatory frameworks". *AI and Ethics, 2(*3), 377-387, 2022.

[12] Zhang, N., and Deng, W. . Fine-grained LFW database. In 2016 International Conference on Biometrics (ICB) (pp. 1-6). IEEE, June, 2016.

[13] Rao, A. S., and Ganguly, P. Implementation of Efficient Cache Architecture for Performance Improvement in Communication based Systems. In 2017 International Conference on Current Trends in Computer, Electrical, Electronics and Communication (CTCEEC) (pp. 1192-1195). IEEE, September, 2017.

[14] Li, D., Dong, M., Yuan, Y., Chen, J., Ota, K., and Tang, Y.. SEER-MCache: A prefetchable memory object caching system for IoT real-time data processing. IEEE Internet of Things Journal, 5(5), 3648-3660, 2018.

[15] GALLALLY, J., Marcel, S., and Fiérrez, J. Image quality assessment for fake biometric detection: application to iris, fingerprint and face recognition [J]. IEEE Transactions on Image Processing, 23(2), 710-724, 2014.

[16] Budiarti, R. P. N., FATHIN, A. N., and Sulistiyani, E. Website-Based Student Achievement Book Using the Waterfall Method. IJRSM: International Journal of Scientific Research and Management, 10(3), 797-808, 2022.