National College of Ireland

Configuration Manual

MSc Research Project
Cloud Computing

VIJAYAKUMAR KANNIAH
Student ID: x21188955

School of Computing
National College of Ireland

Supervisor: Sean Heeney

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	VIJAYAKUMAR KANNIAH
Student ID:	x21188955
Programme:	Cloud Computing
Year:	2022
Module:	MSc Research Project
Supervisor:	Sean Heeney
Submission Due Date:	18/09/2023
Project Title:	Configuration Manual
Word Count:	1316
Page Count:	8

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	VIJAYAKUMAR KANNIAH
Date:	18th September 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

VIJAYAKUMAR KANNIAH
x21188955

1 Introduction

This document's main objective is to give reader a thorough manual for effectively set up and run the project. The system architecture, installation procedures, configuration options, execution flow, and troubleshooting advice are all covered. Before deploying the project, please carefully read this document.

2 Prerequisites

Users are expected to have basic knowledge of Ubuntu, python programming language, node js and machine learning algorithms.

3 Environment Setup

I am using AWS cloud for my implementation. I have used configuration for EC2 instance as specified in Figure 1. It is recommended to use the latest version of ubuntu and a minimum of RAM: 16 GB. We are using backed in FAST API and frontend in React JS

I have developed my project using python programming language and python version of 3.7 or later is required. You can install python latest version from `https://www.python.org/downloads/`

* Need to download environment setup file 2 :

FastAPI - A modern, fast (high-performance), web framework for building APIs with Python 3.7+ based on standard Python type hints. You can find information and documentation for FastAPI on its official website: https://fastapi.tiangolo.com/
Uvicorn - A lightning-fast ASGI server that serves as the interface between your FastAPI application and the internet. You can find more information and documentation for Uvicorn on its GitHub repository: https://github.com/encode/uvicorn
CMake - A widely used open-source, cross-platform family of tools designed to build, test, and package software. You can find information and documentation for CMake on its official website: https://cmake.org/documentation/
Dlib - A C++ library containing machine learning algorithms and tools, including the Histogram of Oriented Gradients (HOG) feature descriptor. You can find information and documentation for Dlib on its official website: http://dlib.net/
OpenCV (cv2) - An open-source computer vision and machine learning software library.

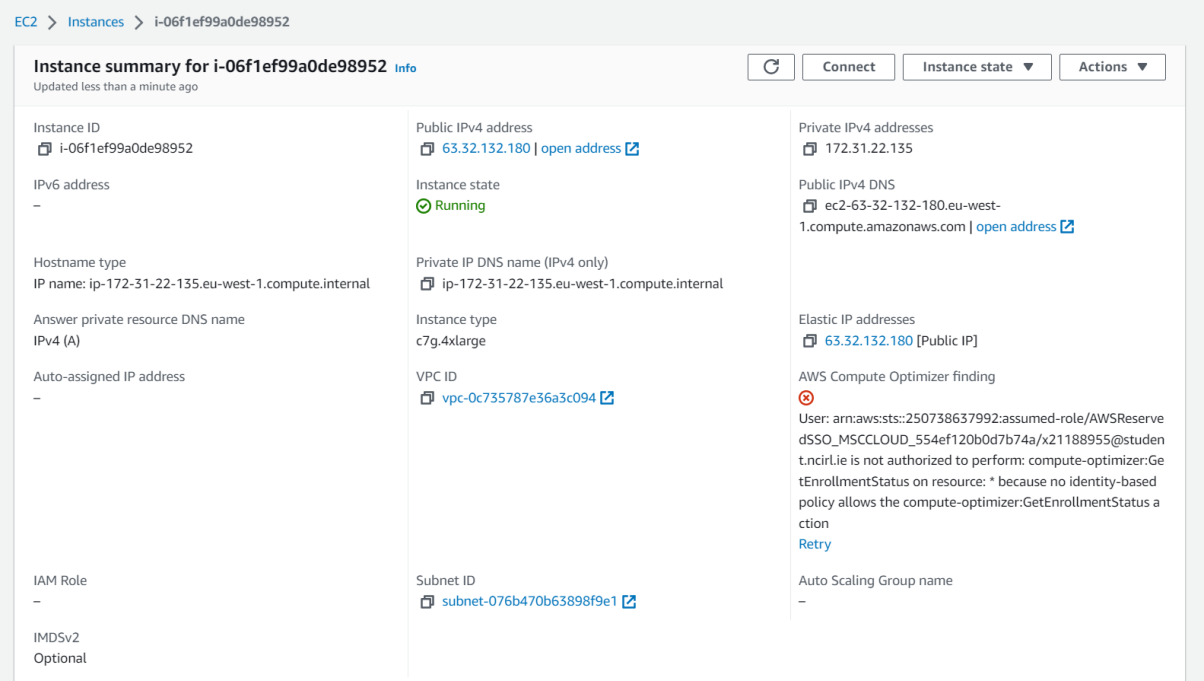


Figure 1: AWS EC2 instance

You can find information and documentation for OpenCV on its official website: https://docs.opencv.org/
Node.js and npm - Node.js is a JavaScript runtime, and npm is the package manager for Node.js. You can find information and documentation for Node.js and npm on their official websites: https://nodejs.org/ and https://www.npmjs.com/
Serve - A simple static file server for serving web pages. You can find information and documentation for Serve on its GitHub repository: https://github.com/vercel/serve

* Need to download the dependency files from the requirement.txt file using the following commands - pip install -r requirements.txt 3

4 Implementation steps

4.1 Methodology

File structure of the project for front (React JS) and backend (Fast APi) 4

Config Files to handel the face menthod, face folder and E.t.c- 5

GenerateEncodings : The model trained with HOG and the output is generated in the GenerateEncodings folders. Code snippet of it 6 7 8

* Commands to run the Frontend - pm2 "npm start" start –name frontend
* Commands to run the backend - pm2 start "python3 main.py" –name "hawkeye"
* Commands to check for the log - sudo pm2 logs
* Commands to check the status of the application - pm2 status 9

```
pip install fastapi
sudo apt install python3-pip
pip install uvicorn
pip install cmake
sudo apt-get update && sudo apt-get install libgl1
pip3 install dlib --force-reinstall --no-cache-dir --global-option=build_ext
sudo apt-get install ca-certificates curl gnupg
pip3 install cv2
sudo apt-get install python3-distutils
sudo apt-get install python3-apt
sudo apt-get install python3.9-dev
sudo apt install nodejs
sudo apt install npm
sudo npm install -g serve
```

Figure 2: Dependency Files to Install dependency

```
ubuntu@ip-172-31-22-135: ~/
dlib==19.22.1
face-recognition==1.3.0
face-recognition-models==0.3.0
fastapi==0.70.1
filetype==1.0.9
imutils==0.5.4
opencv-python==4.5.5.62
pandas==1.1.5
Pillow==8.4.0
pydantic==1.8.2
python-multipart==0.0.5
uvicorn==0.16.0
requests==2.26.0
SQLAlchemy
~
~
~
```

Figure 3: Requirement Files to Install dependency

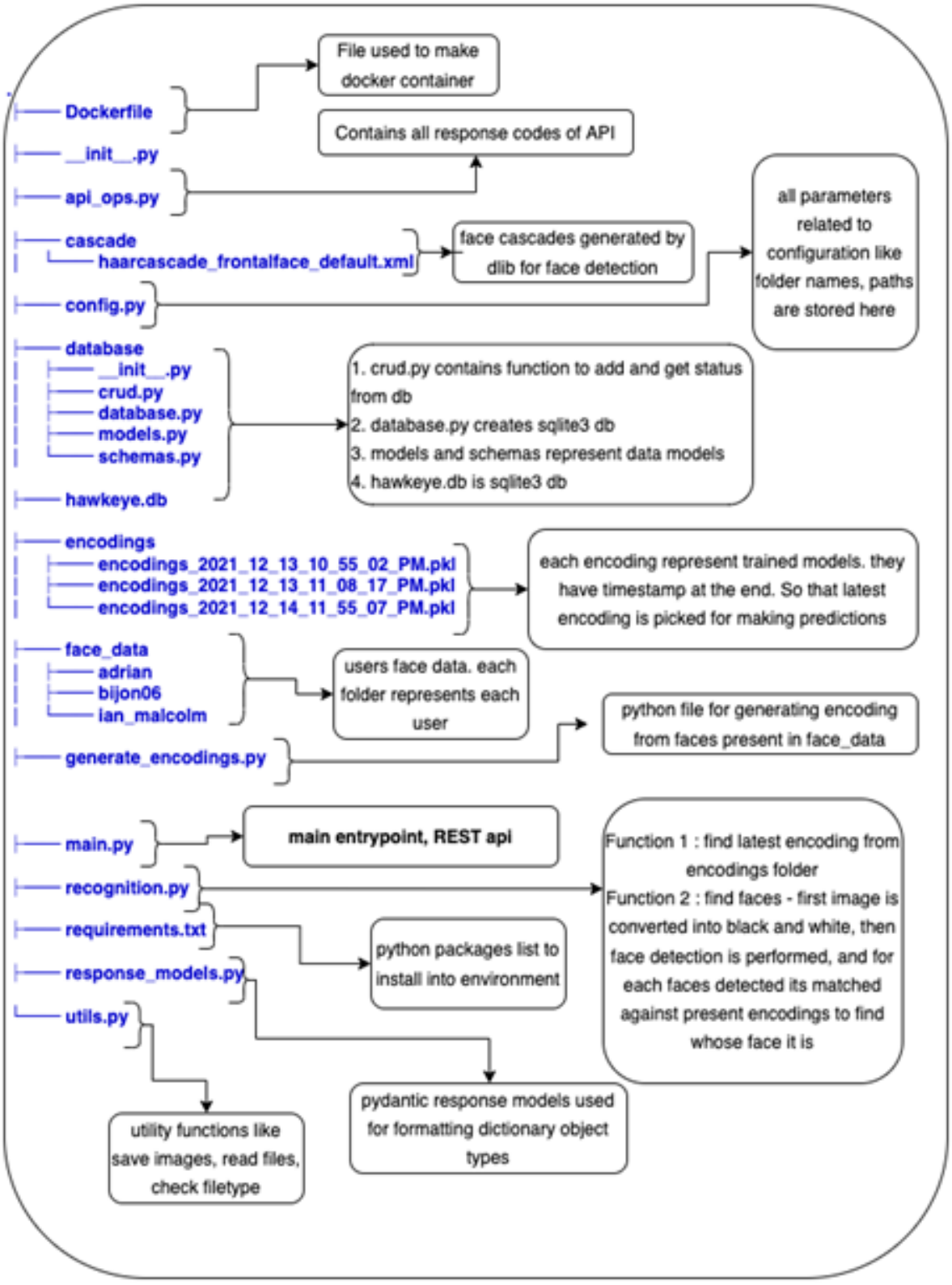


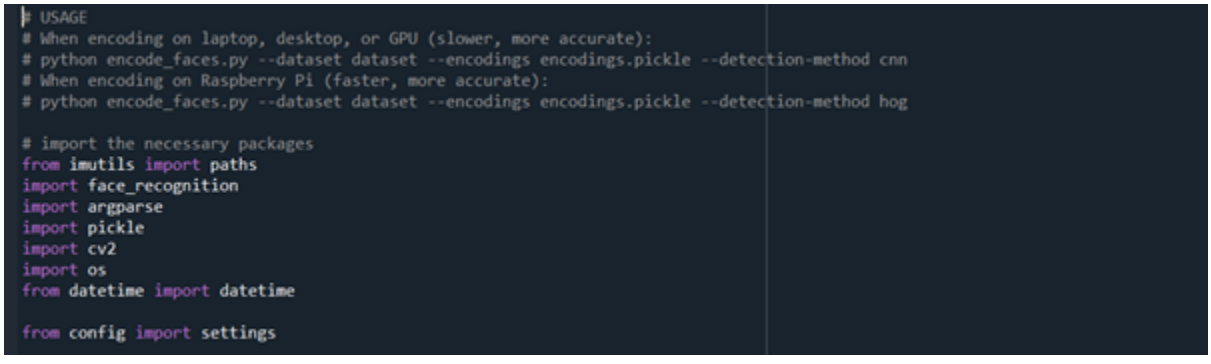
Figure 4: File Structure of the project (Backend)

```python
import os
from pydantic import BaseSettings

class Settings(BaseSettings):
    version: str = "V-1"
    app_name: str = "HawkEye"
    base_path: str = os.getcwd()
    api_files_folder: str = "received_files"
    logs_folder: str = "logs_folder"
    pytest_report: str = "pytest_report"
    face_folder: str = "face_data"
    encoding_folder: str = "encodings"
    cascade_folder: str = "cascade"
    temp_files_path: str = os.path.join(os.getcwd(), "temp_files")
    temp_images_path: str = os.path.join(os.getcwd(), "temp_images")
    facedet_method: str = 'hog'

settings = Settings()
```

Figure 5: Config File of the Backend



```python
# USAGE
# When encoding on laptop, desktop, or GPU (slower, more accurate):
# python encode_faces.py --dataset dataset --encodings encodings.pickle --detection-method cnn
# When encoding on Raspberry Pi (faster, more accurate):
# python encode_faces.py --dataset dataset --encodings encodings.pickle --detection-method hog

# import the necessary packages
from imutils import paths
import face_recognition
import argparse
import pickle
import cv2
import os
from datetime import datetime

from config import settings
```

Figure 6: GenerateEncodings1 (Model Training)

```python
def encode_faces():
    # grab the paths to the input images in our dataset
    print("[INFO] quantifying faces...")
    imagePaths = list(paths.list_images(settings.face_folder))

    # initialize the list of known encodings and known names
    knownEncodings = []
    knownNames = []

    # loop over the image paths
    for (i, imagePath) in enumerate(imagePaths):
        # extract the person name from the image path
        print("[INFO] processing image {}/{}".format(i + 1,
            len(imagePaths)))
        name = imagePath.split(os.path.sep)[-2]

        # load the input image and convert it from RGB (OpenCV ordering)
        # to dlib ordering (RGB)
        image = cv2.imread(imagePath)
        rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        # detect the (x, y)-coordinates of the bounding boxes
        # corresponding to each face in the input image
        boxes = face_recognition.face_locations(rgb,
            model=settings.facedet_method)

        # compute the facial embedding for the face
        encodings = face_recognition.face_encodings(rgb, boxes)

        # loop over the encodings
        for encoding in encodings:
            # add each encoding + name to our set of known names and
            # encodings
            knownEncodings.append(encoding)
            knownNames.append(name)
```

Figure 7: GenerateEncodings2 (Backend)

```python
    # dump the facial encodings + names to disk
    print("[INFO] serializing encodings...")
    data = {"encodings": knownEncodings, "names": knownNames}
    date = datetime.now().strftime("%Y_%m_%d_%I_%M_%S_%p")
    enc_name = os.path.join(settings.encoding_folder, settings.encoding_folder + '_' + date + '.pkl')
    f = open(enc_name, "wb")
    f.write(pickle.dumps(data))
    f.close()

    return enc_name

if __name__ == '__main__':
    enc_name = encode_faces()
    print(enc_name)
```

Figure 8: GenerateEncodings3 (Backend)

id	name	namespace	version	mode	pid	uptime	↺	status	cpu	mem	user	watching
2	frontend	default	N/A	fork	0	0	15	errored	0%	0b	ubuntu	disabled
0	hawkeye	default	N/A	fork	3287914	6h	2	online	0%	352.4mb	ubuntu	disabled
1	ui	default	N/A	fork	0	0	15	stopped	0%	0b	ubuntu	disabled
3	ui-new	default	N/A	fork	3763519	0s	256…	online	0%	4.5mb	ubuntu	disabled

```
[PM2][WARN] Current process list is not synchronized with saved list. App main static-page-server-8082 differs. Type 'pm2 save' to synchronize.
ubuntu@ip-172-31-22-135:~/Hawkeye-Thesis/HawkEye_v2/backend$
```

Figure 9: Status of the application (Backend)

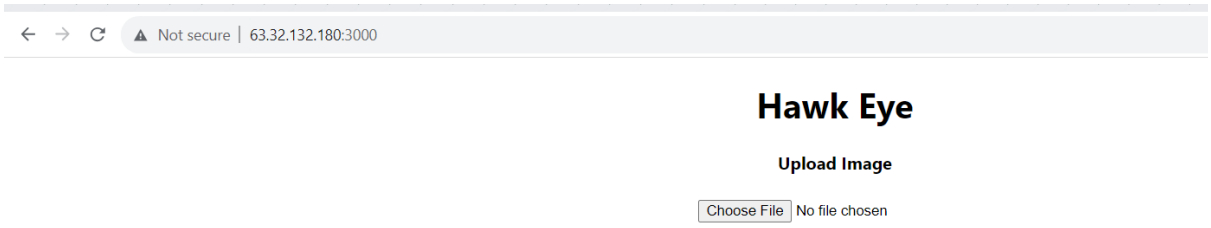


Figure 10: UploadImage for identification

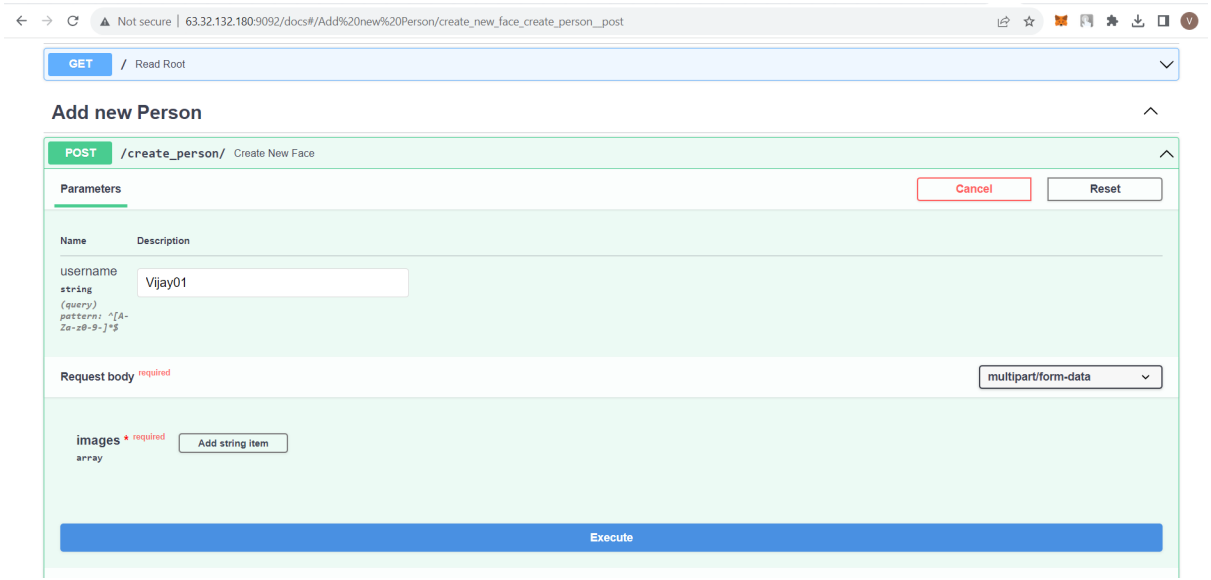


Figure 11: Addperson known person to the application

5 Applications

* UploadImage - Upload the image to identify the face 10
* AddPerson - Add the known person to the application 11
* Train Model - Train the model for the know persons 12

6 Execution and Results

In this study, two small image databases were constructed, sourced from Unsplash and Google, aimed at evaluating model performance across varied image types and sizes. The assessment of these datasets revealed that dlib consistently outperformed alternative models. Randomly selected facial images from Google were employed to validate the trained models, resulting in the classification of individuals as "UNKNOWN." Subsequent testing of non-facial images demonstrated the models' ability to correctly refrain from predicting faces and instead presented a reset option. Through training a DLIB-based face detection model on images of seven renowned personalities from the Labelled Face Database, we achieved accurate results. The training process and subsequent accuracies were obtained using an AWS EC2 cloud environment, facilitated by a dedicated backend

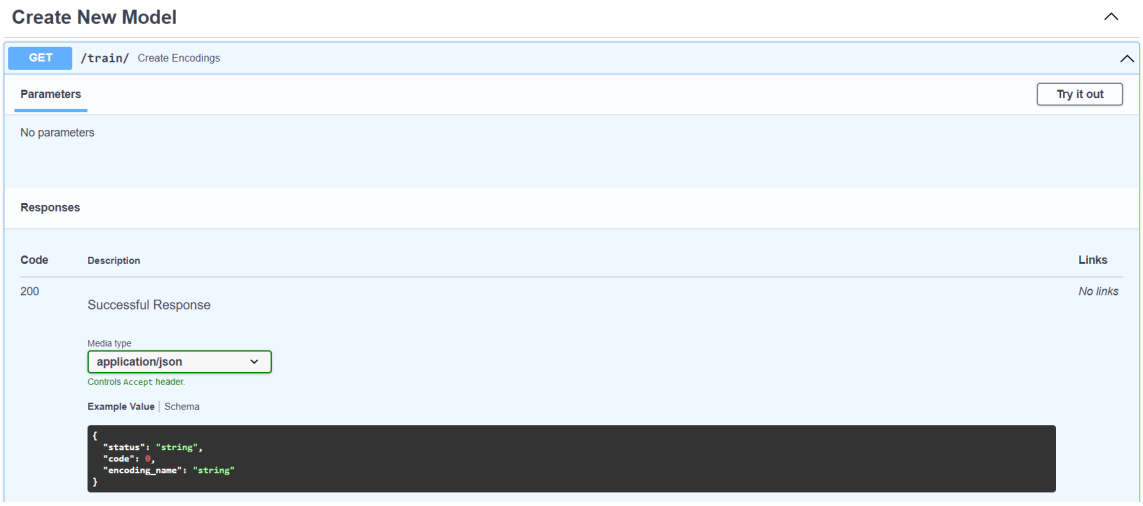


Figure 12: Model Training

API system developed for this project.