**CS673 Software Engineering (AIDAN)**

**Team 5 - Fafi**

**Software Design Document**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| Zengrui Luo | Security leader | *Zengrui Luo* | 9/26/2022 |
| Matthew Kluska | Team leader | [Matthew Kluska](mailto:kluska@bu.edu) | 9/26/2022 |
| Brendan Truong | QA leader | *Brendan Truong* | 9/26/2022 |
| Aidan Chang | Design and Implementation leader | *Aidan Chang* | 9/26/2022 |
| Derric Syme | Configuration leader | *Derric Syme* | 9/26/2022 |
| Patounezambo  Ouedraogo | Requirement leader | *Patounezambo* | 9/26/2022 |
|  |  |  |  |
|  |  |  |  |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
| 1 | Aidan Chang | 9/26/2022 | Iteration 1 submission |
| 2 | Zengrui Luo, Derric, Aidan | 10/11/2022 | Database design and Security Design, sequence diagram, project descriptions, and class diagram |
| 3 | Aidan | 10/18/2022 | Final update on some of the terminologies and final features. |

[Introduction](#_87t9hln2vjz0)

[Software Architecture](#_3ipvmjgn6clp)

[Class Diagram](#_ky60nv8suxxm)

[UI Design (if applicable)](#_7ucksmkf6rzx)

[Database Design (if applicable)](#_tcmuor4nl1kz)

[Security Design](#_x18fj36s1121)

[Business Logic and/or Key Algorithms](#_mtfbusfb0eq3)

[Design Patterns](#_9zvwkmc4luo5)

[Any Additional Topics you would like to include.](#_15tmymhipvdv)

[References](#_50ojo9i46ytq)

[Glossary](#_8n34lvocupub)

# Introduction

The FaFi application implements the YOLOv5 pre-trained model to detect person objects in a live streaming video through the computer's camera. Inside the app, a user can sign up for an email notification which stores a snapshot of the video along with other information whenever people are detected inside the video.

The design goal of FaFi are the following:

1. Efficiency- The ML algorithm used in this application must be efficient.
2. Reliability- The person detection and email notification must be reliable.
3. Security- The security of the end user’s account and the privacy of the history must be secured.
4. Robustness- The application must be able to run in all environments, handle errors, be fault tolerant and pass thorough testing.

# Software Architecture

Frontend - Flask, Flask-Login, Flask-Mail

Backend server - SQlite3, Flask Framework

ML framework - Pytorch, Yolov5

# Class Diagram

Entities:

* Users
* Video Streaming

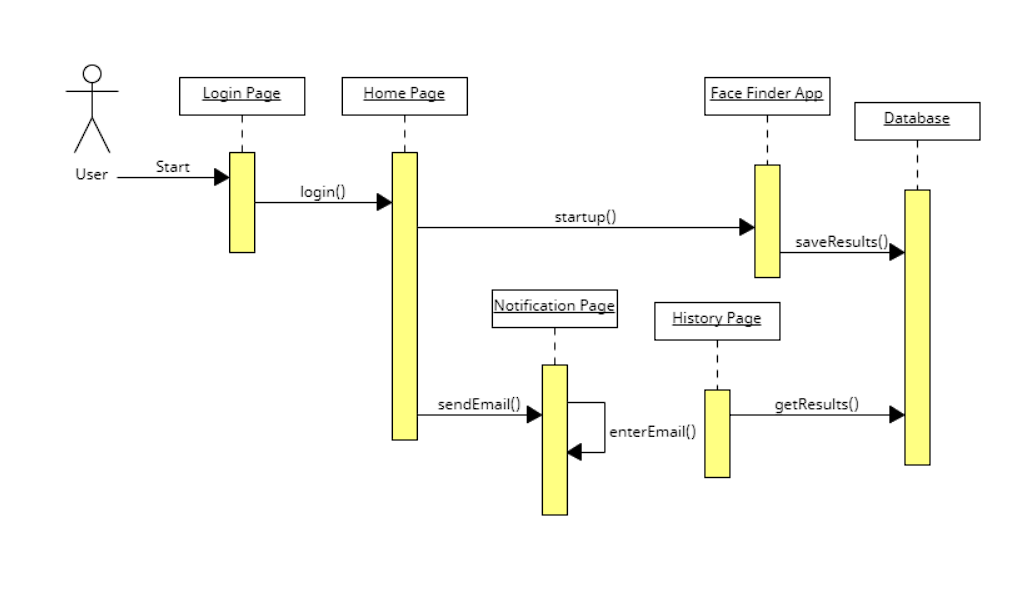
Controls:

* Login
* Signup
* Set Notification
* View History

Boundaries:

* Main page
* Login Page
* Notification page
* History Page

# Sequence Diagram



A user will login, and then go to the Home page. The Stream page starts the Face Finder application, and sends the results to our Database. If the user signed up for email notification, Fafi sends a notification email to the address provided on the Setting page. In the History page, we can view the notification results from our Database.

# UI Design

* + Login page - Users can create and login to their account
    - Fields: Username and Password
    - Buttons: Sign Up and Sign In
      * Sign In redirects to Home Page
      * Sign Up redirects to the Login interface. Accounts with passed username and password are created.
  + Home page:
    - Navigation Bar: Links to Stream, History, and Setting pages
  + Stream page
    - Functionalities:
      * Displays the live stream video with people detection feature

* + History page- Displays a history of events where people are detected.
    - Buttons: Click on an event to view the screenshot enlarged.
    - History Feature: Each event shows a screenshot on the left with time it took place, and number of people detected on the right.
  + Setting page- A page that an end user can sign up for email notification to receive potential detection notifications from the application.
    - Fields: Email Address to receive notifications
    - Buttons:

1. An end user can sign up for an email notification which will send a detection notification to the user's email address, and a screen shot of a video will be saved and stored in the History page.

# Database Design

The Database Design will consist of two tables. These tables were created natively in python using the SQLite3 module.

The first table houses metadata about the uploaded image including attributes like person detected or not, unique identifying code, and timestamp. This SQL Database is used by the history tab, notification tab and future features.

We had initial thoughts to store the uploaded photo directly inside the sql database as a base64 encoded text. We took a safer approach of storing an image in a folder outside of the sql database and instead on the server, referencing the file directly instead of encoding/decoding its contents. This is to ensure images are not corrupted and retain their resolution.

The second table in the same database is used to save the user information. Including user email, username and password. For preventing sql injection, all the user information will be encoded to base64 before writing into the database.

Future enhancements could be to replace our SQLite3 module with a dedicated database system and encode/decode the image.

# Security Design

We implemented an authentication mechanism to prevent unauthenticated use of our application by creating a username and password. These values are encrypted then stored within our database. We make use of the flask-login plugin to provide user session management. This makes use of a login manager to handle tasks of logging the user in and out, along with managing a user’s session over a period of time. The web application is enforced to only display and authorize history access to that corresponding to a given user.

Authorization:

We have an index page, which only allows users to sign up and login. When new users sign up, it will compare input email with all existing user email addresses in the database. Only when this email is not registered before, it can be signed up as a new user. In login page, both the email and password will be compared with the information in database. After user successfully login, it will jump to the Home page. Home page allows user to navigate to other pages such as submit files, view history and use Facial Detection function.

Preventing sql injection:

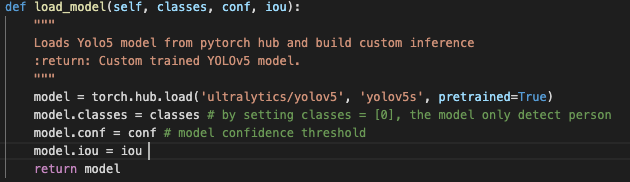
When we request user information on the sign up page. We encrypt the string to base64 before we commit them into the database. We also encrypt the login information before we compare it with the data in database. So the sql command won’t work in user information request pages.

# Business Logic and/or Key Algorithms

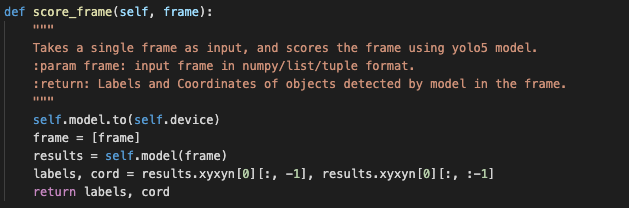
In order to detect people in a live stream video, we first use cv2.VideoCapture() method from OpenCV to capture the video, then use the .read() method to break the video into frames (images).



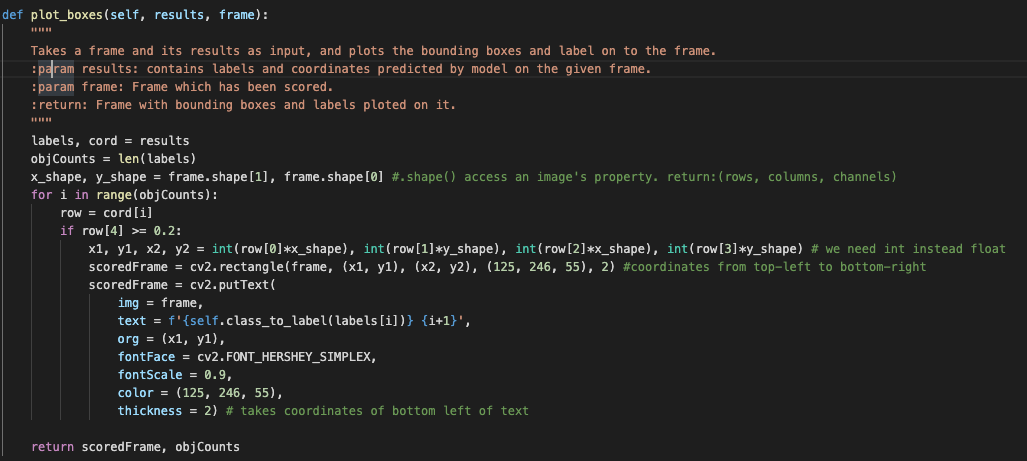
Then, we apply a pre-trained YOLOv5 model in the load\_model() method with custom inference, where our application requires higher confidence rate and person detection only.



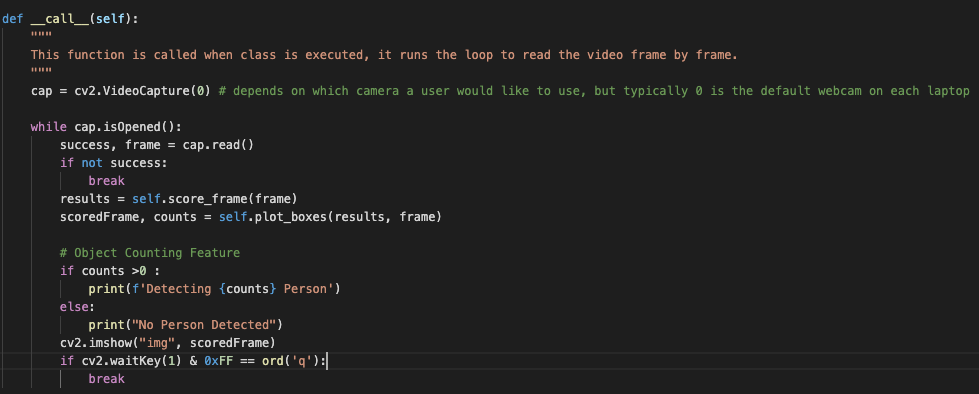
After we obtain the model and the frames (images), the model is ready to score the frames (images). The implementation is very easy thanks to Pytorch, with the first step being to transform the frames into numpy/list/tuple format, then declare whether our Pytorch is a CPU or GPU version. Lastly, pass in the frames into the model as shown below. We can further breakdown the results into Classification and Coordinates of each object.



The Coordinates represent the top-left and bottom-right corners of a detected object. By using cv2.rectangle() method, we are able to plot the bounding boxes of each object for each model it detected.



Lastly, by using cv2.imshow() method, we can display scoredFrames. Also, a while loop is added to continuous display scoredFrames as a video output.



# Design Patterns

We are using Flask, a micro web framework written in Python, to build the Views and buttons, and Pytorch as the ML framework for the Model.

We used the Observer Pattern to design the History page and the Email Notification feature since each detection events happens frequently and each event needs to be stored, sent and displayed.

# References

* SQLite 3: <https://docs.python.org/3/library/sqlite3.html>
* Flask Login: <https://flask-login.readthedocs.io/en/latest/>
* Yolov5 Github: ​​<https://github.com/ultralytics/yolov5>

# Glossary

* ML: Machine Learning
* FaFI: Facial Finder