

Skopos Security

Project Summation

**Prepared by Mia Ward
Sinan Fathulla
Merrell Reed
Todd Robinson
Zachary Edwards Downs**

1. Preface	3
1.1. Purpose	3
1.2. Summary	3
2. Software Design	4
2.1. Class Diagrams	4
2.1.1. Website Diagram	4
2.1.2. Database Diagram	5
2.1.3. Camera System Diagram	5
2.2. Use Case Diagram	6
2.3. Sequence Diagrams	7
2.3.1. Account Management Diagram	7
2.3.2. Motion Alert Diagram	8
3. Project Implementation	9
3.1. Implementation Overview	9
3.2. Server Implementation	9
3.2.1. Server Overview	9
3.2.2. Apache Server	9
3.2.3. MYSQL Database	9
3.2.4. File Transfer	10
3.2.5. Mail Server	10
3.3. Website Implementation	11
3.3.1. Homepage	11
3.3.2. Login	11
3.3.3. Registration	12
3.3.4. User Home	12
3.3.5. User Schedules	13
3.3.6. User Settings	13
3.4. Vision System Implementation	14
3.4.1. System Overview	14
3.4.2. Name Generation	14
3.4.3. Footage Recording	14
3.4.4. Motion Detection	15
3.4.5. Human Recognition	16
3.4.6. Alert System	17
3.4.7. Clip Recording	17
3.4.8. Image Capture	17
3.4.9. File Uploading	17
3.5. Unimplemented Functionalities	18
3.6. Implementation Issues	18
4. Member Contributions	19

1. Preface

1.1 Purpose

The purpose of the Skopos Security Project was to produce an affordable security measure to protect against home invasion and package theft. Our aim was to do this by creating an object recognition and motion detection camera system which detects humans and packages. This system was meant to have the advantage of only detecting serious threats compared to standard motion detecting cameras that detect motion of any kind. Through our camera system we hope to provide the user with peace of mind, and provide them protection against porch pirates.

1.2 Summary

We believe we were able to implement the core functionalities of our camera system and achieved what we originally set out to do in the creation of this prototype. The motion detection and human recognition software we implemented were both functional and ran well on lower end hardware such as a raspberry pi. The ability for this software to execute seamlessly on lower end hardware bodes well for running on smaller, cheaper camera hardware. Further implementation of this product by improving the motion detection software and streamlining the camera's functionality would allow the camera to do well in the market of front porch security cameras.

2. Software Design

When decomposing our project we identified three critical systems that function together to make one. These systems are our:

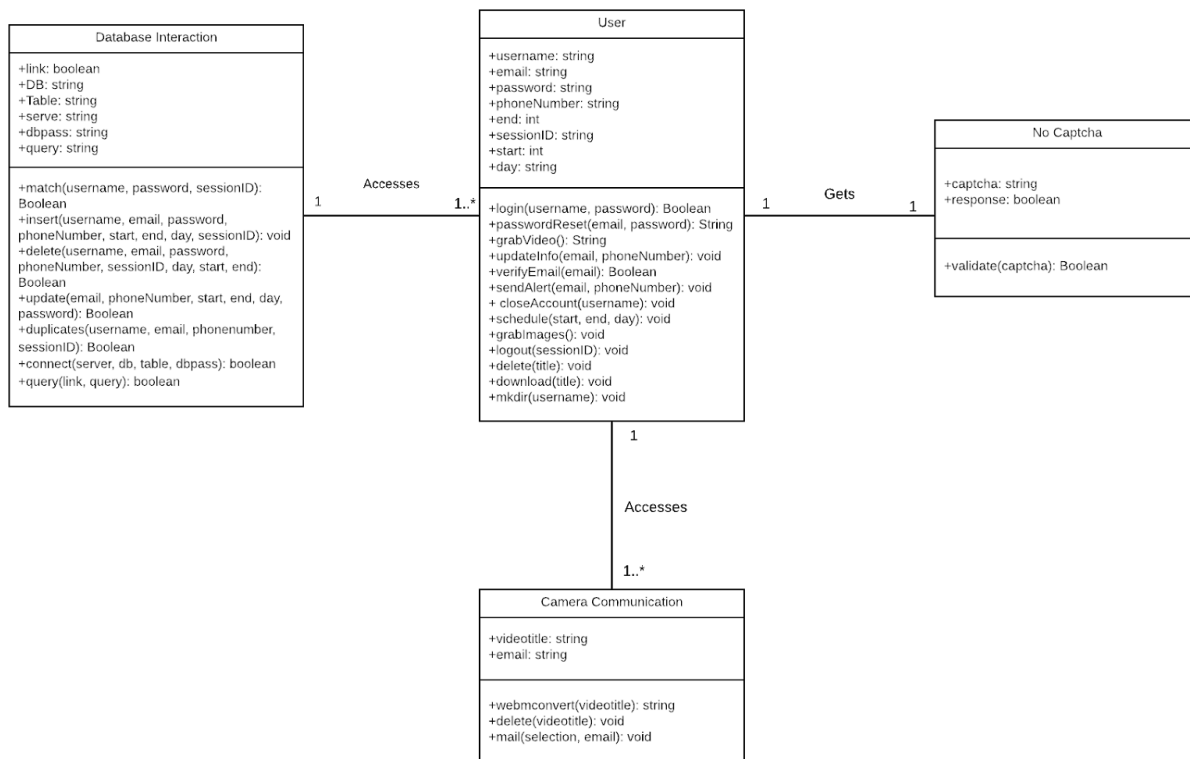
- Website
- Database
- Camera

To properly visualize this breakdown we have created a number of diagrams.

2.1 Class Diagrams

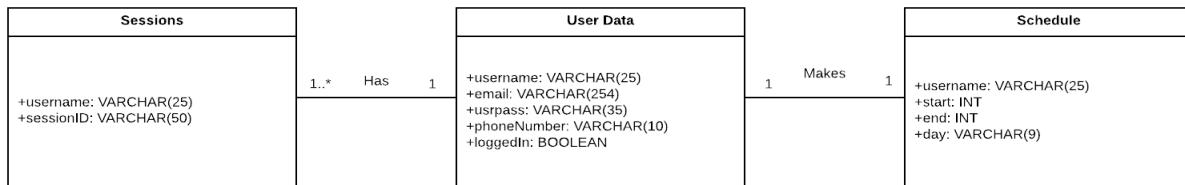
2.1.1 Website Diagram

We have chosen to implement four modules that communicate with each other to successfully run our website. The first module, database interaction, is how our database interacts with our website and is accomplished through PHP. Through the use of session IDs specific to a given user, the module is able to write to, update, delete, or check for duplicate/matched information from our database. The second and most important module, our user module, holds all information that is linked to a single specific user account. The user module entails scheduling, video and image data, alerts, and general account preferences. Third is our no captcha module that serves the purpose of keeping bots from creating fake/spam accounts on our website. Fourth, and finally, is the camera communication module that helps communicate data from our camera such as when to email the user a picture of the intrusion, convert video files to the appropriate type, and delete videos once they are uploaded.



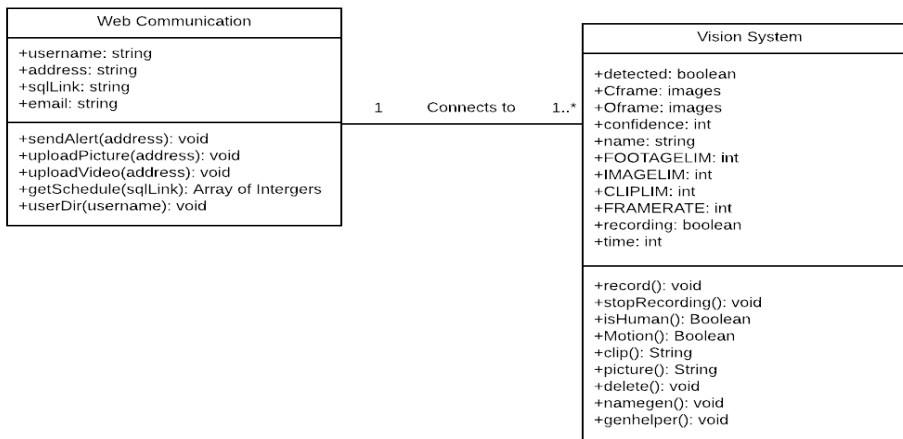
2.1.2 Database Diagram

Our database features three tables. The first table, user data, is used to store user information such as username, email, user password, phone number, and if the user is currently logged in. The second table, schedule, is where the information for a user's schedule is stored. It holds what day, and times the user wants the camera to be active. Lastly, the sessions table which stores the username and session ID for every session generated by user login.



2.1.3 Camera System Diagram

The camera system for our project has only two modules that require it to operate. The first module, web communication, administers all web communications between our website and the FTP server by uploading images and videos, getting schedule information, and sending alerts to the website to alert the user. The vision system, our second module, is the whole of how we get images and videos from our camera. It has constants for video and image limits on a user's account, and the frame rate at which to record. Its functions include generating unique file names, detecting motions and humans, recording footage, clips, and taking images.



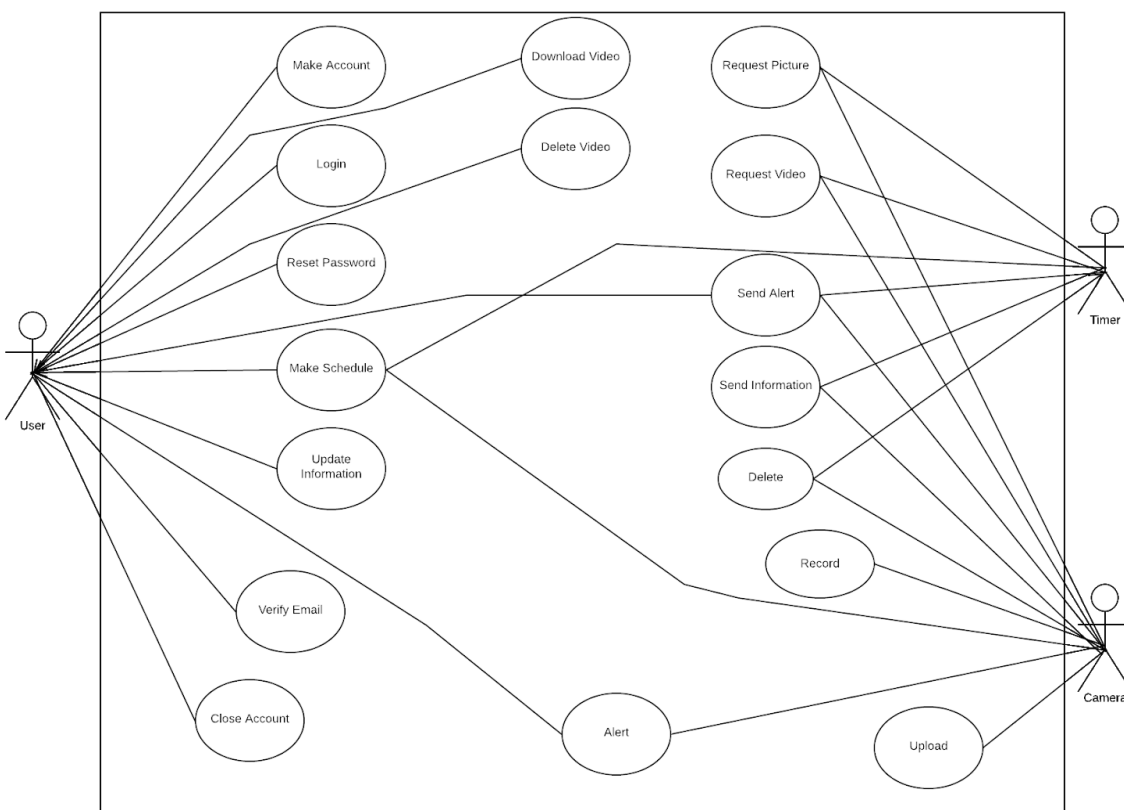
2.2 Use Case Diagram

For an overall look at how our system functions and interacts with our user, we offer a use case to better support our vision for the project. We have three “actors” that interact with our numerous use cases, a user, timer, and camera. The following paragraphs will explain how each of these actors relate to the given use cases.

The user is the most important actor as the system was built for them. They are able to do many actions on our website including more important tasks like making an account, giving a precise schedule for when the system should be active, downloading videos that are recorded, and deleting videos they no longer wish to have on their account page.

The timer is an essential actor in our system. It allows for the functionality of many use cases that help signal to the camera when to do tasks. Such tasks like requesting a picture and video from the camera, sending alerts to the user, and deleting videos once they are uploaded to the website are pivotal to the overall service of the system.

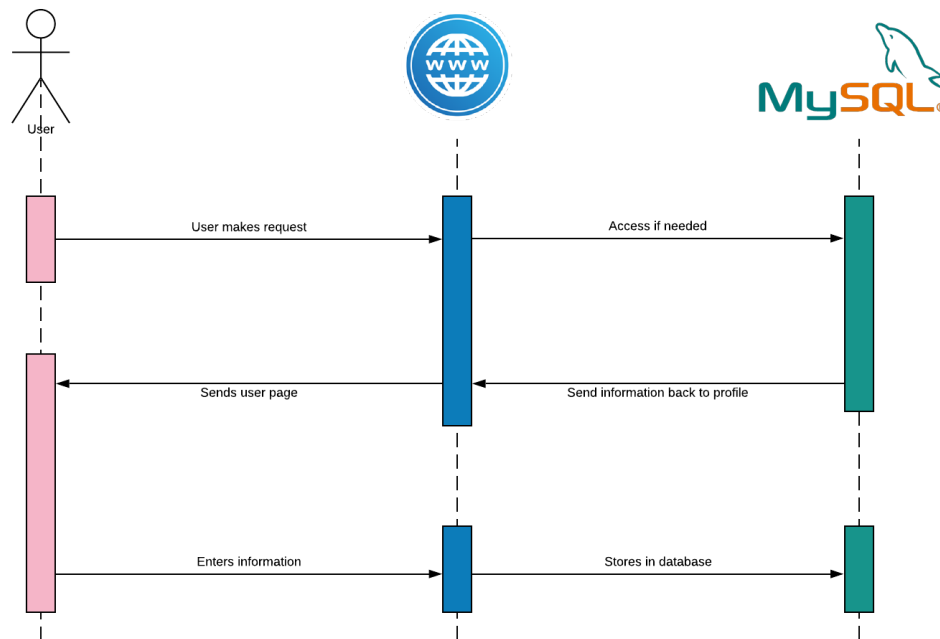
Similar to the timer actor, the camera actor is crucial and without it our system would not be useable. Relying on the timer's signal, the camera has many important functionalities such as recording videos, uploading videos, and alerting the user of such activities.



2.3 Sequence Diagrams

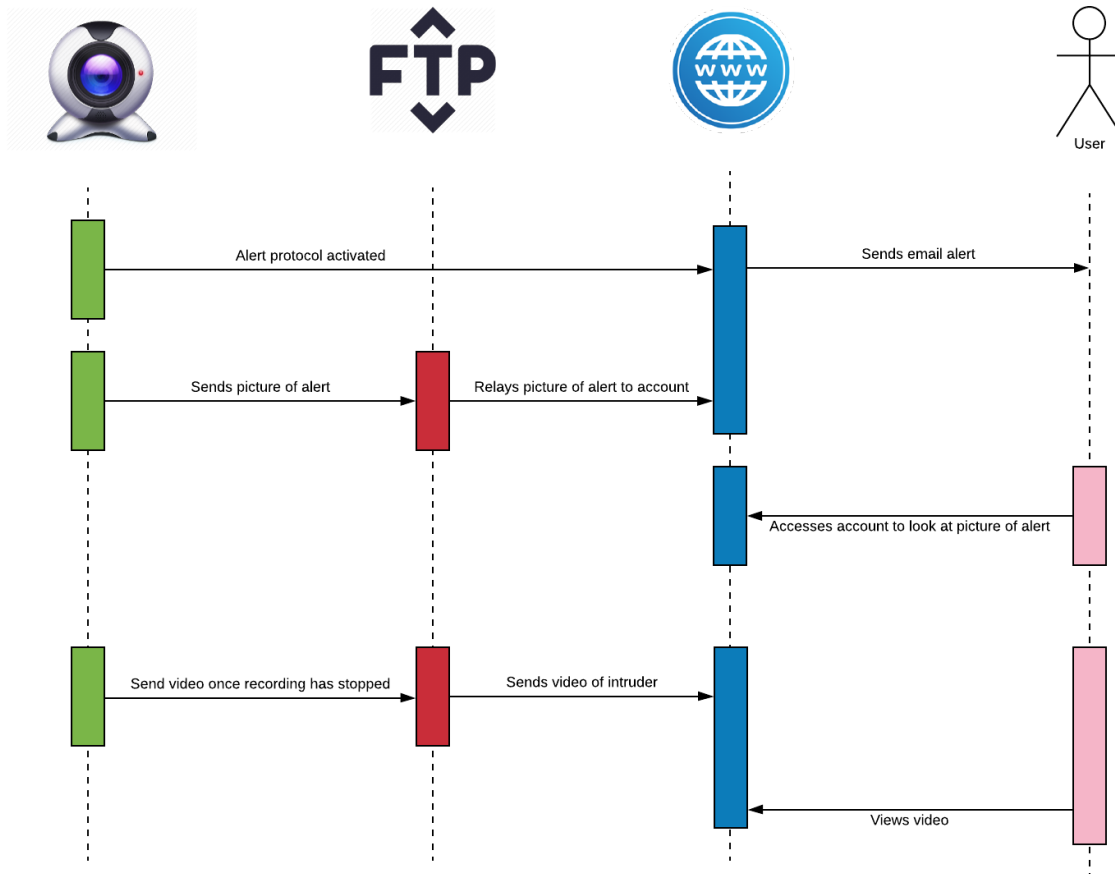
2.3.1 Account Management Diagram

This diagram illustrates how the general interaction between user, website, and database works. Some of the major functionalities include logging into an account, registering an account, adding schedules, and changing settings. First, the user makes a request to the website, for example, requesting to change their password. The website then connects to the database and sends the appropriate information back to the user profile. This information is then sent back to the user and displayed. The user, in this example, inputs a new password on the website. The website connects to the database again and stores the password in the database.



2.3.2 Motion Alert Diagram

As shown in our use case diagram, this diagram shows how the system reacts when motion from a human is detected by the system. When the camera detects a human in frame, it sends an alert to the website which in turn sends an email alert to the user. The camera will then send a picture of the alert to the FTP server which then relays that picture to the website. This picture is then displayed on the user's homepage. The user is then able to view the picture of the alert. After the camera has stopped detecting motion, it will send the video to the FTP server, similar to how it sent the picture, and the FTP servers sends the video to the website. Finally, the user is able to view the video on their homepage.



3. Project Implementation

3.1 Implementation Overview

In implementing this project we utilized a number of components ranging from our camera hardware to the pre-trained models used in recognizing when a human is in frame. In this section we will go through each system and break down what we used to implement each system and how it was done. The three main components of our system are:

- The Server
- The Website
- The Vision System

3.2 Server Implementation

3.2.1 Server Overview

For this project we set up the server ourselves. Using a member's home desktop computer running linux, specifically the manjaro distribution, we set up an apache and mysql server. For our member's router we had to open specific ports and direct them to the server to allow incoming requests from the web. These ports include:

- Port 80: To handle http request of the website.
- Port 443: To handle more secure https requests of the website.
- Port 22: To allow ssh and sftp to and from the website.
- Port 587: To allow sending email to user's from the server.

3.2.2 Apache Server

The apache server handles connecting our website's html and php files to the router's public ip address. It specifically redirects any http request of the website to the more secure https. However, since ssl certificates for https are very expensive we used self-signed ssl signatures when implementing https for our website. This will prompt a warning message in browsers that the website is possibly insecure. This issue would be fixed by purchasing a legitimate ssl certificate. The remainder of what our apache server does is use the rewrite module to 'rewrite' urls to remove the .html and .php extensions.

- Allows website to be accessible on the internet.
- Handles interaction between the user and the website's files.
- Redirects http to the more secure https.
- Removes the .html and .php extensions from website urls.

3.2.3 MYSQL Database

The mysql database on our server consists of a single database used to store the information explained in section [2.1.2](#). It is queried and modified by our website using PHP and it's mysqli functions meant for working with relational databases. It's job on our server is to store user information and verify user identity.

- Manipulated via the php class mysqli.
- Stores user data and verifies user identity.

3.2.4 File Transfer

Setting up a dedicated port for file transfer gave rise to problems. To combat this and allow file transfer to the server from our camera system we set up ssh by opening port 22. This allows our system to transfer files by first opening an ssh connection to our server then opening an sftp connection through the ssh. Then files can be made and transferred using python's paramiko library and PHP's ssh2. The sftp connection is used for creating a user's directory when they register and uploading clips and images from the camera system to a user's directory. Finally we have a server side script for converting mkv clips received from the camera system to webms on arrival using ffmpeg. The mkv is then deleted once conversion is complete. This is done because only webms can be displayed by our website and opencv doesn't record webm.

- SSH through port 22.
- SFTP through SSH.
- Python's paramiko.
- PHP's ssh2.
- ffmpeg.

3.2.5 Mail Server

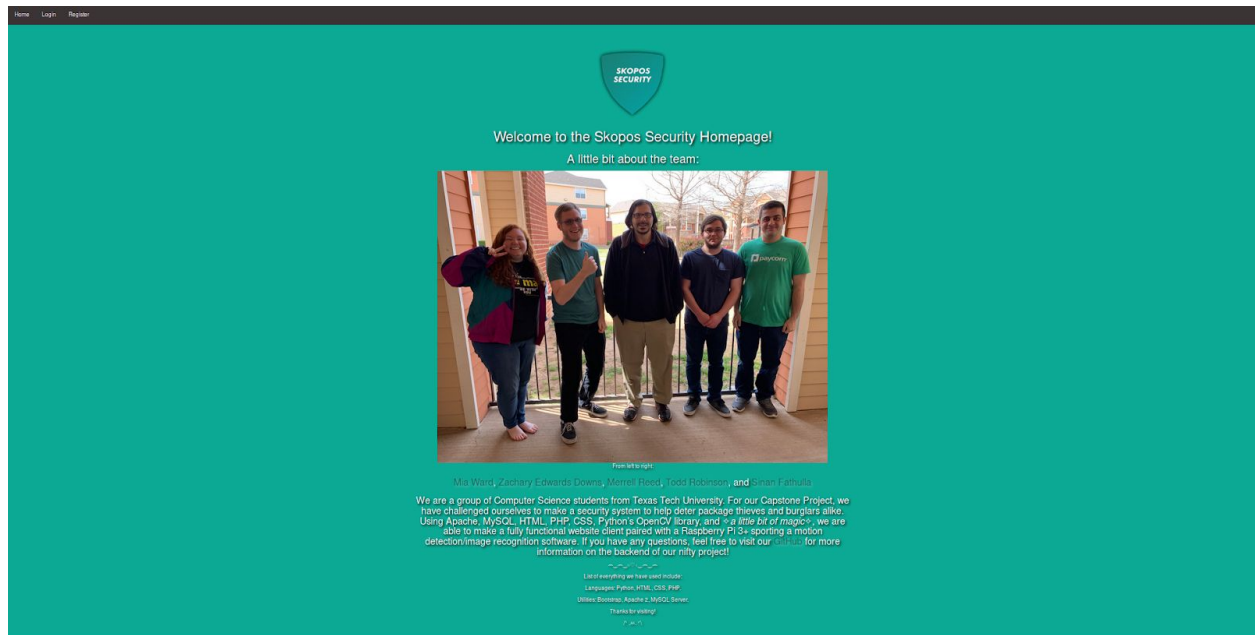
The mail server on our website did not go as originally planned. The final implementation of it uses a google account we created called 'skopos.mailer@gmail.com' and Google's SMTP. We send mail using a singular python script specifying the type of email to send and the recipient as program parameters. The script sends mail by logging in to the aforementioned account with an app password then invoking google smtp over port 587 using python's smtp library.

- Google's SMTP Server.
- The python smtp library.
- Dedicated gmail account.

3.3 Website Implementation

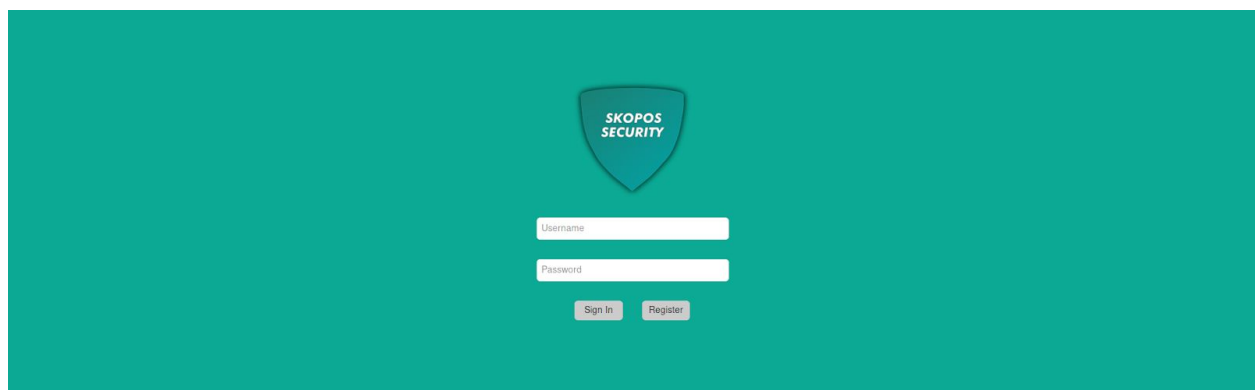
3.3.1 Homepage

Our homepage serves as a starting off point for our users. It offers a navigation bar at the top made using the tag in HTML. The bar makes it possible for the user to navigate to the sites login and registration pages. The homepage also provides a picture of the development team and offers links to their GitHub profiles. Lastly there is a synopsis detailing our purpose, what technologies we used, and a link to the project's GitHub for more information and the source code.



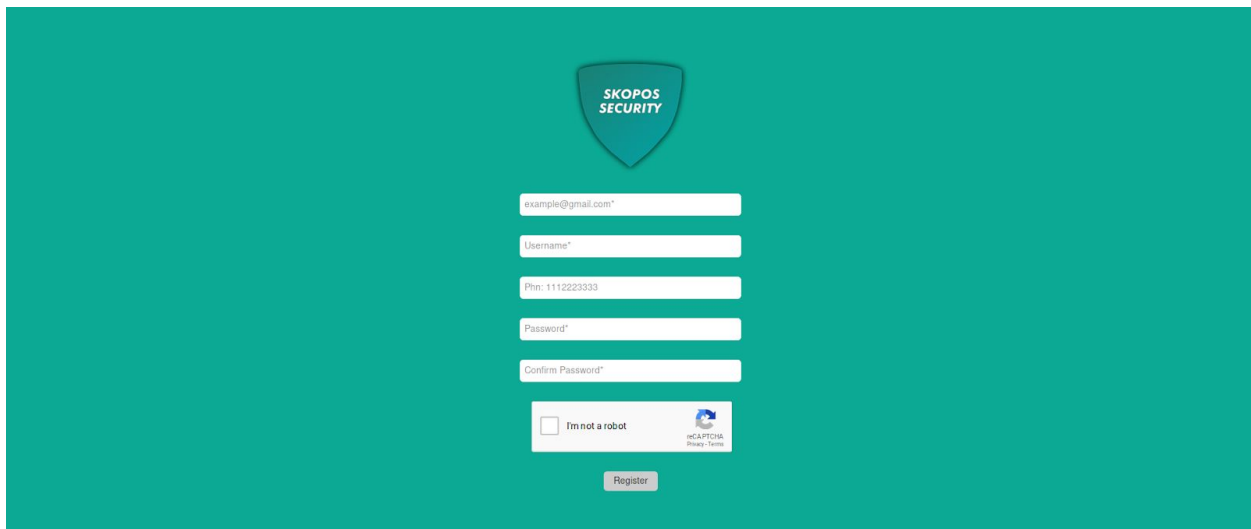
3.3.2 Login

The UI for the login page is straightforward. The Skopos Security logo takes you back to the homepage when clicked on. Under the logo are two text boxes where the user is able to input their username and password. The database is queried using PHP's mysqli(). If the user does not enter a username and password matching an account in the database they are not able to login. The login page also has a register button at the bottom that allows anyone to register for an account. Login functionality was implemented using sessions() in PHP and the hashids library in python.



3.3.3 Registration

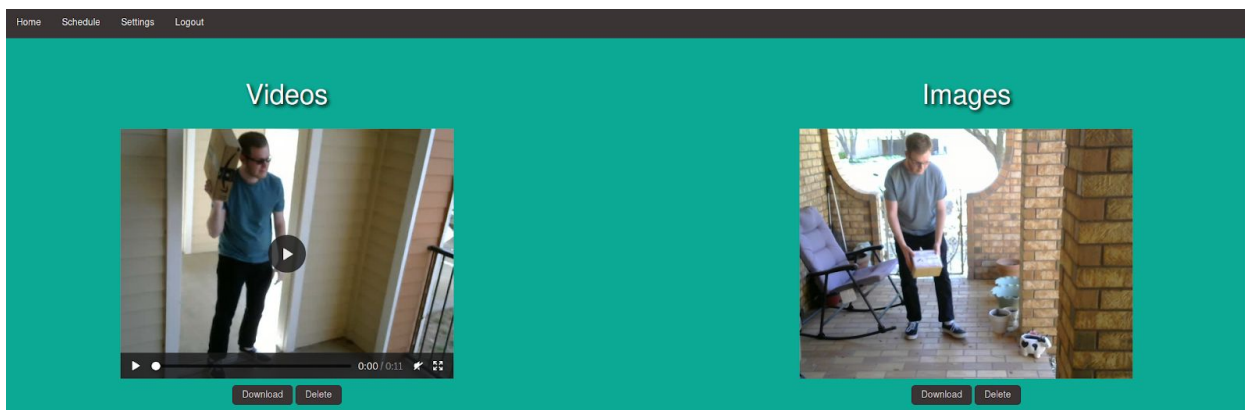
Our registration page makes it easy for users to create an account. It asks for a user's email, username, phone number, password, and to confirm their password. All fields are required except their phone number. HTML's email type ensures the user has entered an email address including the "@" symbol. A valid phone number has 10 digits, and a valid password includes a number, a lowercase letter, an uppercase letter, and a minimum of 8 characters. The confirm password field is required to match their input for the password field. Once the user has entered correct information, they must complete a captcha. This captcha prevents bots from making fake accounts. When submitted, PHP checks the database for matching usernames and emails. If a match is found, the user is returned to the registration page to enter a different username or email. If these fields are unique, then the account is created and a PHP script is called to make a personal directory for the user on the server.



The registration form is displayed on a teal background. At the top center is a shield-shaped logo with the text "SKOPOS SECURITY". Below the logo are several input fields: "Email*" (containing "example@gmail.com"), "Username*", "Pfm: 1112223333", "Password*", and "Confirm Password*". Below these fields is a checkbox labeled "I'm not a robot" next to a reCAPTCHA logo. At the bottom center is a "Register" button.

3.3.4 User Home

The user homepage uses a navigation bar with home, schedule, settings, and logout buttons that navigate the user to their respective pages. Logout destroys the user's PHP session() and removes their sessionID from the database. The homepage has two columns, where videos and images are displayed. When our camera system is activated, it dynamically uploads videos and images onto the server. Once these videos and images have been uploaded the user is able to view and watch them. Users are also able to download and delete these videos and images using HTML's download feature, and PHP's unlink().



3.3.5 User Schedules

The user schedule page takes input from the user on what days and time they want their camera system to be activated. This gives the user control over when their camera is recording. It uses PHP to display the user's schedules and any currently active ones based on their database entries.

Home Schedule Settings Logout

SKOPOS SECURITY

Fake's Schedule

Current Schedule

Day	Hour	Minute
Monday	12	45 pm
Tuesday	6	30 pm

Add to Schedule

Day	Hour	Minute
<input type="text" value="Day of the week"/>	<input type="text" value="Hour part of time"/>	<input type="text" value="Minute part of time"/>

Add Schedule

3.3.6 User Settings

User settings is a page that allows the user to edit and update their account information. The user can update their email or phone number, and change their password. If their current account information is used as input it return them to the settings page without doing anything. The user is also able to delete their account. A dialogue prompt occurs asking for confirmation of account deletion. If confirmation is given they are logged out, all account information is purged from the database, and all of their images and videos are delete along with their user folder.

Home Schedule Settings Logout

SKOPOS SECURITY

Account Settings

Password Reset:

<input type="text" value="New Password"/>
<input type="text" value="Confirm New Password"/>

Reset

Update Email or Phone Number:

<input type="text" value="example@gmail.com"/>
<input type="text" value="Phn: 1112223333"/>

Update

Delete Account

3.4 Vision System Implementation

3.4.1 System Overview

The vision system consists of a webcam connected to a raspberry pi. The software was written in python with the main module for the vision system being mirVision.py. It consists of several functions that enable the operation of the vision system. The module utilizes the opencv library for vision and object detection. Paramiko is used for ssh and sftp communication with the server, and numpy to process camera frames. The vision system's operations focus computation around a single frame. The frame will be converted to several different images for object detection and then be appended to a stream of frames to create video.

The vision system begins by first connecting to the server using SSH. Once it establishes a connection to the server it then sets up the codec to correctly format the video. The vision system calculates a unique name for all generated files to prevent errors with existing files. The system then checks that the camera is scheduled to run using a user's defined schedule.

Every frame the vision system sends the current frame and the previous frame to the motion detection function. The function processes the frames and determines if something has changed between frames. If something changed that means that motion has occurred. The vision system then checks if the motion is coming from a human by running the frame through a convolutional neural network. If the motion is caused by a human the system creates a clip that will continue to record until the motion has stopped or 15 minutes have passed. This clip will be uploaded to the server upon completion. The system also takes an image upon motion and sends it to the server. An email is then sent to the user's email account to notify them. If however there is no motion or the motion is not produced by a human the frame will be added to a the existing video and a new frame will be produced.

3.4.2 Name Generation

There are three file categories that the vision system produces:

- Footage - Always recording while the camera is active.
- Clips - Video that is recorded when human motion is detected.
- Images - A snapshot of the person that caused the motion.

After determining which type of file needs to be produced the NameGen function checks that there is available storage. If the category is footage and there is not available storage then the oldest footage is removed. Otherwise the user's directory on the server is searched using the UserDir() function to check if there is space, if not then no image/clip is produced. It then generates a unique name using the category, a number, and the extension.

3.4.3 Footage Recording

Footage recording is accomplished with a time limit of five minutes before the current recording is ended and a new recording begun. This footage generation loop continues until motion by a human is detected or the camera is deactivated. Recording the video was accomplished using the opencv functions VideoWriter(), read(), and write(). The function VideoWriter() defines the video file using a name, codec, framerate, and resolution. The name is generated by our name generation function, and the framerate is globally defined as 10 frames per second. The codec is defined as MJPG because of issues using any other codec effectively. The video format encoded to is mkv as that is what we got working best. Finally the resolution was chosen to be 640 by 480 to easily transfer over to an appropriate size for the website. The read() function reads from the camera and returns the state of the camera and a video frame. Finally after testing for motion and humans if both were not detected the read frame is written to the video file using the write() function. Replacing footage older than 10 minutes is accomplished using the mv command in linux to copy over the older footage.

3.4.4 Motion Detection

The motion detection function takes the previous frame and the current frame and compares them to see if anything has changed by taking the difference between the two frames. Before it can detect the difference the system must process the images to make detecting the changes between frames easier. To do this we first make the images grayscale so all pixel colors are between 0 and 255. Then we use a gaussian blur to remove excess noise from the image to make changes more easily detectable.

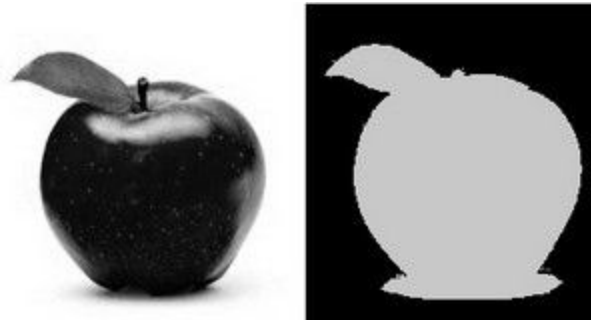
```
f1_gray = cv2.cvtColor(Oframe,cv2.COLOR_BGR2GRAY) #convert to gray for motion detection
f2_gray = cv2.cvtColor(Cframe,cv2.COLOR_BGR2GRAY) #convert to gray for motion detection

Oframe_blur = cv2.GaussianBlur(f1_gray,(21,21),0) #blur it to reduce noise
Cframe_blur = cv2.GaussianBlur(f2_gray,(21,21),0) #blur it to reduce noise
```

Next the difference between the 2 frames is produced:

```
diff = cv2.absdiff(Oframe_blur,Cframe_blur)
```

At this point there will still be too much noise in the image. To further reduce the noise we next contract the range of values in the image. Pixels that are too light the will be clamped down to a smaller value. Here is an example of what threshold does:



The absdiff function has produced an image where white pixels represent differences between the frames. To capture just those white pixels and discard the rest we transform the image into a numpy matrix and divide each pixel by 255 to produce only the white pixels:

```
white_pixels = np.sum(thresh) / 255 #find all white pixels in the image
rows, cols = thresh.shape # get the matrix of the image
total = rows * cols # # of rows * # of columns
```

Finally if more than 1% of the pixels are white we will consider that movement:

```
if white_pixels > 0.01 * total: #if the image contains 1% white pixels than something has moved
    return True
```

3.4.5 Human Recognition

To detect if motion is being produced by a human, a convolutional neural net is used to process the frame. When the vision system is first executed it will load in a pre-trained neural network. The pre-trained neural network has a model and classification definitions for recognizing humans. Those files are loaded into opencv's deep neural network (dnn):

```
#model paths
model_name = 'neural_net_models/MobileNetSSD_deploy.caffemodel'
model_proto = 'neural_net_models/MobileNetSSD_deploy.prototxt.txt'

#load trained model into opencv
net = cv2.dnn.readNetFromCaffe(model_proto, model_name)
```

The frame is first resized to 300 by 300. This is to lower the amount of resources needed to process the frame. Next the image needs to be transformed into a data structure that the neural network can process. In opencv that data structure is called a blob. The mean value is subtracted from each pixel to scale it. The neural net is next told to use this blob as its input and net.forward is called to produce the output from the neural net.

```
blob = cv2.dnn.blobFromImage(resized, 0.007843, (300, 300), 127.5)
net.setInput(blob)
#run detection
detections = net.forward()[0][0]
```

The neural net itself will work by splitting the blob into a set of images that each represent a different feature of the image. For example it will produce a separate image for all the horizontal lines, vertical lines and contours and several other features of the image.

For every classification that the neural network defines, in this case there are 15, a confidence number is produced which represents the probability that the image represents that type of object. We only care about humans, classification 15, and if the network has a confidence greater than 60% then the function will return true.

```
for i in range(len(detections)):
    #get how sure the net is of this detection
    confidence = round(detections[i][2] * 100,2)
    #get the index of the category the net thinks this object is in
    category_index = int(detections[i][1])
    if confidence > 60 and category_index == 15:
        return True

return False
```


3.4.6 Alert System

We were only able to implement email alerts for our vision system with the time allotted. The script for sending these emails is server side. The vision system runs this server side script using paramiko's `exec_command()` function. The email provided to this script has been coded to be taken as a program parameter for the vision system as we were unable to hook up the camera to a user account in the end. The script is called during the footage function after motion has been detected by a human and an image has already been uploaded.

3.4.7 Clip Recording

Clips are recorded the exact same way as footage, so that section [\[3.4.3\]](#) should be visited to understand how video is recorded. What differs is that clips record until motion has ended for at least five seconds. There is no human detection implemented in clip recording, thus it will only stop recording until any type of motion has ceased. After the recording has ended it is uploaded to the server in the user's directory. Paramiko is called at this point to convert the mkv clip to webm using a server side converter script.

3.4.8 Image Capture

Images are taken at the time of human motion detection in the footage function. It is given the frame on which motion was detected as input and uploads this image using a unique file name from the name generation function. It is uploaded to the user's directory on the server using our upload function and paramiko.

3.4.9 File Uploading

Files are uploaded with the `Upload()` function. It takes a filename as an input then opens an sftp connection over paramiko using `open_sftp()`. Then the file is uploaded to the user's directory using the `put()` function and then the sftp connection is closed. If the file is a video clip then the server side converter is called using `exec_command()` to execute it with python. Once converted the mkv is deleted and webm can be viewed on the website.

```
def Upload(file):

    sftpclient = sshclient.open_sftp() # Opens an sftp connection.
    sftpclient.put(file, "/srv/http/ftpsrv/" + username + "/" + file) # Writes file to the user's folder.
    sftpclient.close() # Closes the sftp connection.

    # If uploading a video file then call a server-side script to convert it.
    if 'mkv' in file:
        sshclient.exec_command("python3 /srv/http/ftpsrv/converter.py")
```

3.5 Unimplemented Functionalities

Data Encryption:

With the time we were allotted encryption of our data was deemed low priority and thus was not completed in time. It was given this priority status because for this project our database was not storing any sensitive information. It was originally meant to be implemented using the python library passlib.

System Communication

We were not able to implement communication between our vision system and server in the end. It likely would have been implemented using http requests and scripts on the server to handle these requests.

Camera States

It was originally planned to use system states for the camera to better organize its function. This was not needed for the system to operate however so it was of low priority and did not get implemented.

High Alert

High alert was meant to handle scenarios where a clip was recorded and hit the 15 minute limit. It was meant to send an extra alert and make preparations to simultaneously upload and record using a background process. The feature wasn't very important and thus unimplemented.

Package Detection

Package detection was going to be a staple feature of the system and use a pre-trained neural network like our human recognition, but there was not enough time to get the feature implemented.

3.6 Implementation Issues

Video Playback

Video, including both footage and clips, often plays back at a slowed down or sped up speed. We are unsure as to the cause of this and were unable to rectify it before the project's end.

Camera Link

The team was unfamiliar with using http requests to link our server and camera system and the link between a user's account and a camera never became a feature.

Mail Server

We setup our own postfix server only to find out that home ip addresses are blocked from sending mail to prevent mass spam. We were able to work around this however using google's smtp as explain in the mail server section [[3.2.5](#)].

4. Member Contributions

Mia Ward

- Helped design presentation slides for all projects.
- Designed class, use case, and sequence UML diagrams.
- Responsible for front end implementation of the homepage and user home.
- Created the UI front end of the user scheduling.
- Wrote Software Design Portion (UML diagram explanations) and helped write Website Implementation on final report.

Sinan Fathulla

- Worked with others to implement the website's front end UI.
- Specifically assisted with creating UI for user settings.

Merrell Reed

- Implemented the motion detection algorithm using openCV.
- Added human detection functionality using a pre-trained neural network.
- Worked on adding package recognition for assessing if a package was removed from frame.

Todd Robinson

- Assisted with back end password verification.
- Assisted with registration page front end UI.
- Helped with front end UI for user settings page.

Zachary Edwards Downs

- Hosted the website with Apache and the database with MySQL Server.
- Created the login and registration pages.
- Implemented login and logout functionality.
- Set up file transfer between the website and camera system.
- Implemented the footage recording function with openCV.
- Implemented post-motion clip recording and uploading,
- Implemented picture taking and uploading to user account.
- Implemented clip viewing/deleting/downloading on user account.
- Created email notification upon motion detection.