**Information Systems 1D Project: Group 1E**

**Problem Statement**

How can we create an affordable home surveillance system?

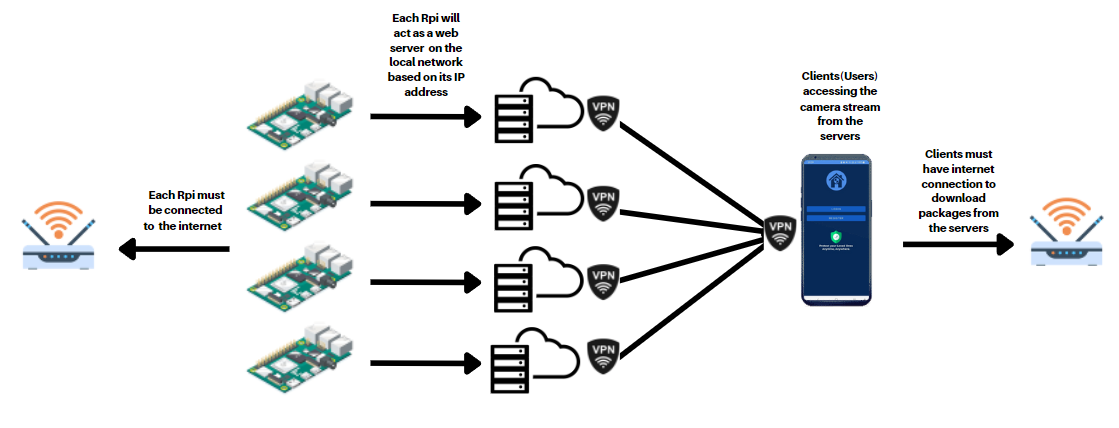
**System Design and Implementation**

Most users only require a lightweight camera surveillance system to keep track of simple, single-use monitoring of activities, such as checking in on their pet at home. However, modern surveillance systems available on the market offer services with additional functionalities which may not be needed, and are also highly-priced for the average user.

Taking this into consideration, we wanted to create a solution for this group of users; users who require a simple and affordable surveillance system. Hence, we designed a system that is highly customizable. Users can decide the scale of the system depending on their individual needs, as our solution allows for the installation of as many cameras as they wish. As the cameras only require a small power source, such as a power bank, to power themselves, the cameras are portable, and the user is free to decide where to position the cameras as needed and move them to different places at any time. This is unlike stationary, installed surveillance cameras which are typically available on the market.

Our system consists of a combination of Raspberry Pi (Rpi) devices, cameras, and an Android application. We are hosting a HTTP server on the Rpi using its own IP address with the port set as 8082 by default, and streaming the live camera feed to that server. In the Android application, the user registers their camera under their account by logging into their account and keying in the IP address of the Rpi for the camera and adding this camera to their list of cameras. The database (Firebase) then connects the camera to the app. However, due to security reasons, the user device must be connected under the same network as the Rpi to be able to access the stream. Our solution to remove this limitation is to implement a Virtual Private Network (VPN). The user’s Android device and the Rpi need to be connected to the same VPN in order to achieve seamless real-time monitoring without any restrictions.

**System Architecture**

****

**Main Application Features**

**Security**

Privacy is our utmost priority. From hardware to software, our system offers three layers of protection to the users. The first protection layer is the Rpi itself. Since each Rpi itself is hosting a HTTP server for the stream, it can block out connection requests from devices that are not connected to the same network. The second protection layer is the VPN. In our system, we used Tailscale VPN which requires users to set up an account in order to use the VPN service. Both the Rpi and the user’s mobile device must be connected to the VPN under the same network, else the user will not be able to access their cameras outside of their house/on mobile data. This is possible because, under Tailscale, every connected device gets a stable IP and an auto-assigned domain that stays consistent, no matter what WiFi or mobile data the device is on. The IP address of the Rpi is only available on the VPN’s dashboard, hence it is very difficult to know the IP address of the cameras without having access to the VPN account. The third layer of protection is on the app itself; users need to register an account on the app in order to use the app. We implemented FirebaseAuthentication for our login system, protecting the user's credentials even from the developer team.

**Live Camera Feed**

As the stream will be uploaded to the HTTP server hosted on the Rpi, the users can access their cameras by visiting [http://ip\_address\_of\_the\_Rpi:8082/](http://ip_address_of_the_rpi:8082/). However, this is very inconvenient and time-consuming, especially if the user has multiple cameras. Hence, our app ensures that users only need to key in the IP address once, which will be converted into the URL automatically and is ready to be streamed on the app.

**Camera Management**

First way of camera management is the addition of cameras. Users can add a camera with a button, by filling in the IP address of the Rpi of that camera, and giving it a name that can serve as a label for the user to know what the camera is for.

Due to there being only one unique IP address per camera, there cannot be a duplicate of IP addresses within the list of cameras a user adds. The app is designed such that if a user attempts to add a camera with a duplicating IP address, it will prompt the user that this camera has already been added and will not add the camera. Similarly, for camera names, the app also does not allow a user to name cameras with different IP addresses with the same name to prevent confusion.

However, different users can use the same camera. For example, User A can only have one camera named “Living Room” with the IP “100.101.1”. But, it is possible for User B to have a camera with the exact same name and IP in their list. This design allows for multiple users with different accounts to control and view their communally shared cameras, without having to share any passwords or private information.

Moreover, users can delete a camera by tapping on the camera, and then tapping on the ‘Remove Camera’ button. This allows users to individually remove the cameras that the user may not be using anymore, such as idle cameras or malfunctioning cameras.

Lastly, the camera list can be reset. If the user seeks to clear all the cameras from his list, simply tap the “Reset” button from the options menu. This clears all their cameras in the database.

**Call for Help**

While viewing footage from a camera, if the user notices something amiss, the “Call for Help” button brings up the device’s default phone app with 999 dialled, so the user can call for help as soon as they need it.

**Activities of the app**



**Helper Classes for the app**

|  |  |
| --- | --- |
| CurrentUser.java | Takes in the user email and splits it to obtain the unique userID for every user. This userID will be used to identify the user in the Realtime Database. |
| Devices.java | Takes in the camera name and IP address and provides it to the ListView Adapter, which will be shown in the MainActivity (Home page) of the app |
| IpHashMap.java | When there is a valid camera name and IP address input by the user, IpHashMap creates a hash map, where the camera name and IP address are hashed to each and will be updated to the Realtime Database. |
| URLGetter.java | Takes in the IP address of each camera and puts it into an URL string. The WebView widget in WebActivity will be obtaining the camera stream from this URL. |

**Concepts used**

|  |  |
| --- | --- |
| Singleton Design Pattern | - urlGetter.java  - The public static urlGetter method getInstance checks if an instance with a particular camera name and IP already exists.  - If that instance does not exist, the private constructor is called and a new instance is created. Finally, the instance is returned. |
| ListView | -ListView and ArrayAdapter |
| Unit Test | -CurrentUser, Devices, IpHashMap, and urlGetter are debugged at a modular level to ensure it outputs the intended result |
| Firebase | -Used Firebase Authentication for user accounts and Firebase Realtime Database to store user data |
| Refactoring | -Refactored ExampleUnitTest to UnitTest  -Refactored the activities into another name  -Renamed the whole app into its final name “HomeGuard” |
| Explicit Intent & Implicit Intent | -Pass data between activities.  -Start a new sub-activity  -Start another application (Phone calling app) |
| Adapter Design Pattern | -listview.setAdapter(adapter) |

**Possible Future Works**

1. Implement audio functionalities.

Design the system such that whenever the camera is accessed from the app, the Rpi will make a chime or an audio announcement “The camera is accessed”. This is to prevent exploitation of the lightweight and flexible feature of the system for unethical behaviour, such as using it as a spycam.

1. Suitable for different databases apart from only Firebase

Design the system such that it can be easily adjusted to store and access data in different databases, as our current prototype is focused solely on using Firebase. This would thus improve the adaptability of our system.

1. Additional camera management functionalities

Renaming existing cameras as they may have been repurposed over the course of their use. Add pop-ups to prevent the accidental deletion of cameras or the accidental resetting of cameras.

1. Storage of multiple camera feeds

Design the system such that it can store feeds of various cameras independently so that the users can access the feeds later (eg; in the event of a robbery)