GosHawk - Smart App Documatation

Itay Parnafes & Tomer Shalmon

June 26, 2016

GOSHAWK

SECURITY

Home Security had never been so smart

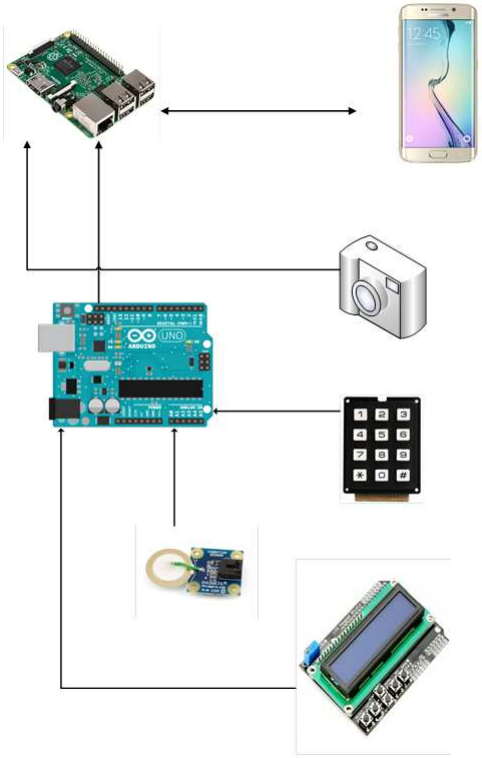
Contents

1 Introduction

GosHawk is a smart compact home security system based on open-source development platforms.
GosHawk was built and designed to be as simple as possible so that the ordinary user can set it up and use it without any previous knowledge.
The GosHawk mobile app strives to be as user-friendly as possible in order to provide an easy convenient user experience.

The current version of GosHawk is an alpha version, designated to provide a proof-of-concept for a free open-source modular smart home security system, initially started as a graduation project for computer-science bachelor degree.

2 System Architecture



2.1 Backend

2.1.1 Arduino

Arduino is an open-source hardware, software and microcontroller based development board. The specific type used in GosHawk is *Arduino Uno*. The Arduino is fed from different sensors scattered around the house, analyses its inputs and synchronizes with the main server. The arduino uses an ethernet shield[1] in order to communicate with the main server using *UDP*[2] over port 9898[3].

The Arduino code is written in *C++* and is well commented in order to provide highly detailed code for other coders to integrate their own code.

Current supported sensors are described in details in section 2.1.2.

[1] Another board assembled on top of the main board in order to provide ethernet based communication.

[2] User Datagram Protocol - Transport layer contectionless protocol used to deliver messages over IP network.

[3] By default. Can be configured differently.

2.1.2 Sensors

The system sensors were particulary chosen to give the maximum information on the house state in minimum cost.
The sensors has to be connected to the designated ports in order to work.

Sensor	Analog/Digital	I/O	Description
Keypad	Digital	Input	Keypad used to input local pincode
Vibration sensor	Analog	Input	Detects vibrations on the door
LCD Screen	Digital	Output	Displays key strokes on the keypad

2.1.3 GoServer

The server is the "heart" of the system and is in charge of communicating and synchronizing between all system components. The server runs on a *Raspberry Pi 2 Model B*[4] hardware running *Ubuntu 16.04 Xential Server* operating-system.
The server runs several component, overwatched by a *watchdog*[5].

Synchronizer

A simple server which listens on UDP port 9898 and receives event messages from the *Arduino* and updates the shared DB[6].
The synchronizer was written in Python 2.7.

Web Server

The web server is the main component of the GoServer and is designed to synchronize with the frontend components.
The web server is designed to work solely with *Android* clients[7], and provides fully integrated web app.
The web server main jobs are:

- Manage users registration and permissions
- Query shared DB as a backend component for the app
- Keep track of clients queries
- Arm and Disarm the system
- Check home presence of registered users

[4] A credit card-sized single-board computer.
[5] A process which runs using crontab and keeps all specified processes up and running.
[6] Database
[7] Currently iOS/web clients are not supported

The web server was written in *Django*[8] 1.8.7 based on Python 2.7.

Hawk-Eye

The Hawk-Eye is a python script used to take photos using the installed camera in the event of a breach.
The Hawk-Eye was written in Python 2.7.

2.2 Frontend

2.2.1 Android App

blah blah blah....

[8] Free and open-source web framework, written in Python, which follows the modelview-controller (MVC) architectural pattern.

3 Requirements

- Arduino (Uno or other)
- Ethernet shield
- Keypad
- Small LCD screen
- Vibrations sensor
- Raspberry Pi
- Foscam IP Camera[9]

3.1 Raspberry Pi

- Ubuntu 16.04 Xenial Server
- Python 2.7
- Django 1.8.7
- Nmap[10]
- GoServer package
- Active internet connection

4 System Components

4.1 Web Server

As mentioned before - the web server is the "heart" of the system and is intermediating between the clients and the data layer such as events, images, etc. Under the *GoServer* main directory, the application files are located under *app* directory.

- The database tables definitions and declarations are located in *models.py*
- The url paths are located in *urls.py*
- The methods actual implementation are located in *views.py*
- All utility methods are located in *utils.py*

[9] Any model
[10] Network Mapper

The server listens by default to port TCP/8888 from all available interfaces.
The path from which the server is searching for captured pictures can be changed and is located in *utils.py* under *IMAGE_PATH*.
All availble methods can be found in section 4.1.1.

The web-server also provides a convenient admin control panel which provides advanced users a way to visually inspect and maintain server administrators and databse.
The admin contorl panel can be accessed via: http://goserver.address:8888/admin.

4.1.1 Server Available Methods

URI	Type	Parameters	Description	Response	Remarks

/app/login	POST	name
		password
			A registered user login page	{
				'Access Granted': 'True',
				'uid': user_id,
				'user_type': permission
				}
					Cookies must be enabled

/app/register_mac	POST	mac	Register given mac to a registered user	{'Success': 1}	MAC address should be colon separated

/app/sync	GET	None	Request all unread events (by a specific user)	{
				events: [
				{
				"id": "862",
				timestamp: {
				"hour": 7,
				"month": 3,
				"second": 57,
				"year": 2016,
				"day": 5,
				"minute": 48
				},
				"type": "Breach",
				"description": "house has been breached",
				"severity": "Critical"
				}
				]
				}	

/app/create_new_user	POST	name password permission	Add new user to the local system	{'Success': 1}	Only 'Admin' can add new users

/app/get_recent_events	GET	None	Reuqest 10 most recent system events	{
				events: [
				{
				"id": "862",
				timestamp: {
				"hour": 7,
				"month": 3,
				"second": 57,
				"year": 2016,
				"day": 5,
				"minute": 48
				},
				"type": "Breach",
				"description": "house has been breached",
				"severity": "Critical"
				}
				]
				}	

/app/snapshots/¡image_id¿	GET	None	Get image taken upon specific event	Binary image data	
/app/arm	POST	None	Arm system	{'Success': 1}	
/app/disarm	POST	None	Disarm system	{'Success': 1}	
/app/status	GET	None	Get system status [Armed/Unarmed]	{'Status': Armed/Unarmed}	

/app/whos_home	GET	None	Get registered users whom their smarphone is connected to the local network	{
				Users: [
				{
				"Name": goshawk,
				"UID": 99
				}
				]
				}	

4.2 Synchronizer

The *Synchronizer* is a key component in the server functionality. It's used as an IPC between local server components and the Arduino. The Synchronizer listens by default on port UDP/9898 only from internal NAT addresses in order to prevent spoofed external intervention in designated traffic.
Both the Arduino and the web-server communicate with the Synchronizer using a simple protocol which maintains the minimal requirements needed.
The ingress packets should be 2 Bytes long and formed as follows:

8 bits	**8 bits**
alert type	UID

Notice: In the same directory from which Synchronizer.py runs from, a *sync.conf* file should be found as well.

4.2.1 Synchronizer configuration file

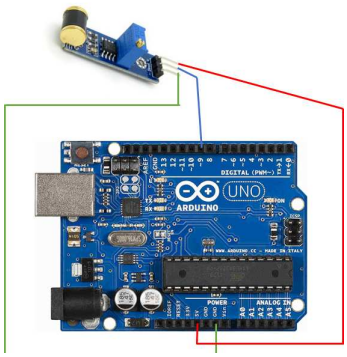
The configuration file must be found in the same directory as *Synchronizer.py*. The configuration is consist of ¡param¿=¡value¿ tuples in that very format.

- PORT - the port which the Synchronizer listens to. (9898 by default)
- DB_PATH - The shared database path. (by default should be located in the parent directory under the name db.sqlite3)
- IMAGES_PATH - The full path to save camera captures to.

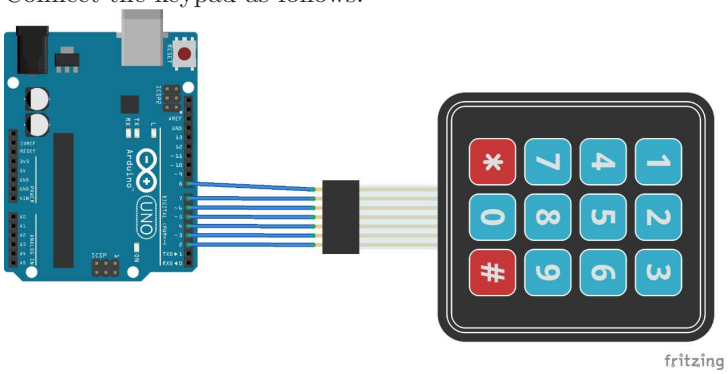
5 Installation

5.1 Arduino

1. Connect the vibration sensor as follows:



2. Connect the keypad as follows:



3. Connect the Ethernet shield on top of the board
4. Plug a USB cable to the USB port (or the AC socket to a 5v power source).
5. Plug the netwrok RJ-45 cable to the Ethernet shield

5.2 Raspberry Pi