Design Document

version 0.0.1

Chaojie Wang

Bumsik Kim

1. **Purpose**

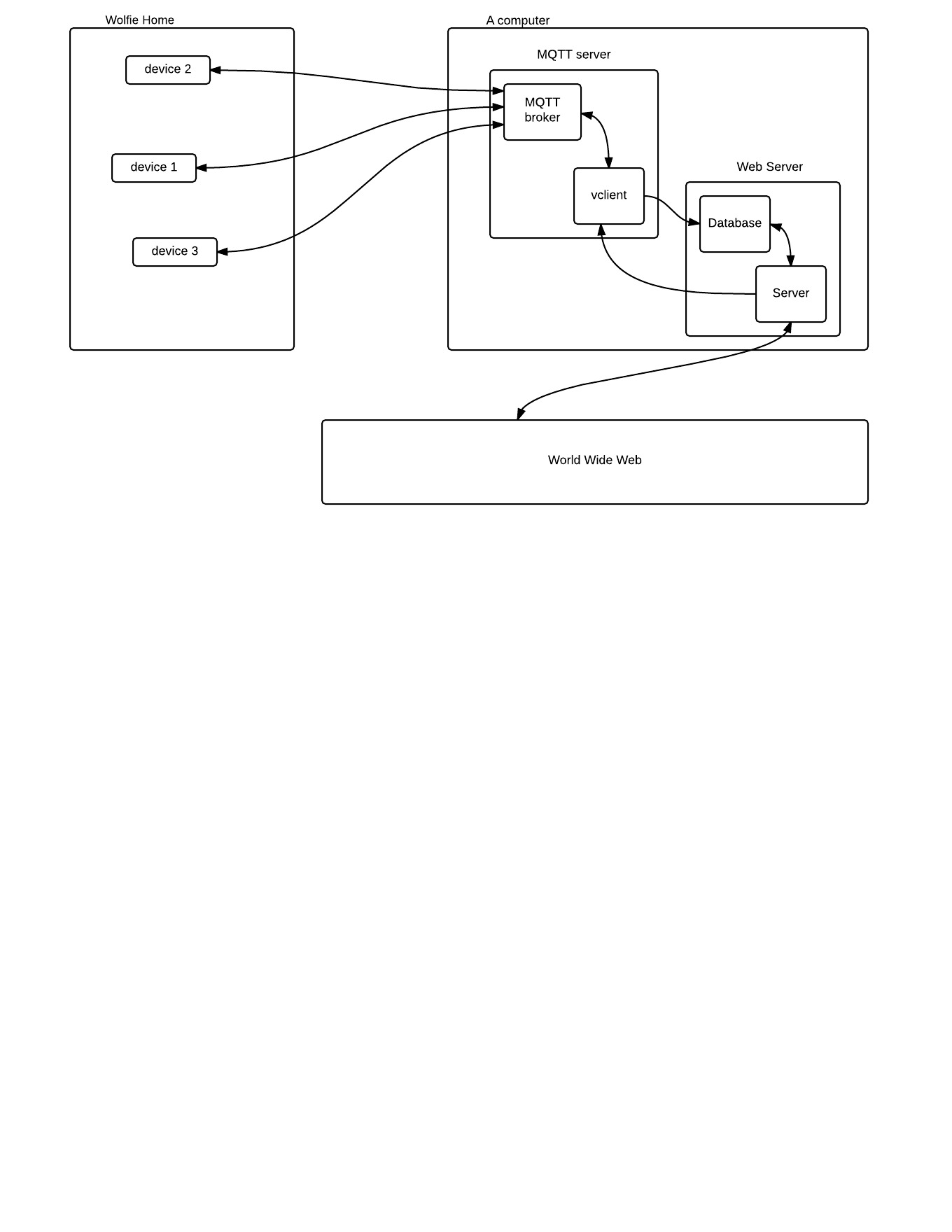
This document is a technical design document. This document specifies components developers need to implement and how pieces work together. This should be used by developers as a reference for implementing Wolfie Home.

1. **Overview**

There are three components in this project: microcontrollers that are able to connect to the Internet, a MQTT server, and a webserver with a database.

Microcontrollers communicate among each other with the MQTT server. Inside MQTT server, there are two components: one is MQTT broker, and the other one is called vclient. The MQTT broker handles the inter-communications among microcontrollers and vclient. The vclient is the virtual client that communicates with database and the webserver. The webserver communicates with users and the MQTT server.

The following topology illustrates the idea:



1. **Microcontrollers**

To be filled

1. **MQTT Server**

Both the web server and the MQTTserver are running on a Linux computer.

**4.1 MQTT broker**

The MQTT broker is implemented according to MQTT protocol. There is an open source MQTT broker called Moquitto, which is used in this project.

**4.1.1 Communication between devices and the MQTT broker**

there are two types: control message and status message.

• Status Message

Status message is a message published from devices to the broker. Following topic is used: status/(room)/(type)/(dev\_id).

• room is unique identifier of a room

• type is type of the information, such as temperature, LED status

• dev\_id, unique identifier of the device

The payload is csv formatted: (dev\_name),(value),(time)

• dev\_name: user friendly name of the device

• value: value of the status

• time: time stamp of when the value was recorded.

For example, a LED is in room1, and is off. It can publish with topic

“status/room1/led/led1”. The payload could be: “LED,off,2016-02-29 02:45:31”.

• Control Message

To control a device, format of topic is used:

control/(dev\_id)

The payload format is simply:

(value)

For example, in order to turn off a LED with led1 as the dev\_id, topic is control/led1, and the payload will be “on”

**4.2 vclient**

Moquitto also provides library for writing MQTT client. It is used for writing vclient. vclient is short for virtual client. vclient is responsible for two tasks. The first task is obtaining all status data from devices and store them into the database. The second task is creating a channel for accepting commands from the web server. The channel is a FIFO pipe. vclient passively listens commands and act accordingly.

**4.2.1 Data format of vclient FIFO pipe**

It is a FIFO pipe. vclient reads and web server writes.

Data format:

(dev\_id),(control value)

• dev\_id: unique identifier of a room

• control value: value being overwritten in the device.

1. **Web Server**

**5.1 MySQL**

The database vendor is MySQL. Use account *chaojie*, and password *dummypass* to login. The MySQL has a database called *wolfie\_home*. *wolfie\_home* has two types of table: users type and home type.

There is only one users type table and it is called *users* (sorry for lack of a better name). It stores information about *users*. For now, main purpose of *users* is to authenticate users.

|  |  |  |
| --- | --- | --- |
| COLUMN NAME | SQL TYPE | DESCRIPTION |
| uid | INT | unique identifier of the user |
| username | VARCHAR(128) | username |
| password | VARCHAR(128) | password |
| email | VARCHAR(128) | email |
| wolfie\_home | VARCHAR(128) | wolfie\_home table name associated with the user |
| devices | JSON | Array of object a user has. |

devices is an array object. Object format is {‘dev\_id’: [dev\_id], ‘controllable’: [ct]}. dev\_id is the same in the *wolfie\_home* table. ct has three possible values of type string: ‘unknown’, ‘controllable’ and ‘uncontrollable’.

Each user has a *wolfie\_home* table. The naming convention is wolfie\_home\_[username]. The table stores status of all devices in user’s house. Data stores in a *wolfie\_home* table are generated by vclient.

|  |  |  |
| --- | --- | --- |
| COLUMN NAME | SQL TYPE | DESCRIPTION |
| dev\_name | VARCHAR(128) | Device name |
| dev\_id | VARCHAR(128) | Unique identifier of the device |
| topic | VARCHAR(128) | Topic of the value |
| sensor\_value | VARCHAR(256) | Sensor value, for example temperature of a room |
| Time | TIMESTAMP | Time the sensor value was recorded |

**5.2 Server**

Overview: when a user log in, the web server queries data from *wolfie\_home* table data, and display to the user. It also accepts commands from user to control devices.

**5.2.1 Backends**

The job of backend is to communicating with vclient and the MYSQL database on behalf of users. It has following urls: api/login, api/logout, api/devices, api/control

* login: verify login information of a user. If it is verified, a cookie is sent to identify user. The cookie is the username.
* logout: destroy the cookie.
* devices: query information about devices
* control: controlling

API specifications:

* login: AJAX with POST, with fields *username* and *password*. Each field is type of string. On success, status code is 301 Moved Permanently. On failure, status code is 400 Bad Request. For example, a user can send AJAX with POST method: username=jack, password:123456.
* logout.php: AJAX with POST, with no fields. On success, status code is 200 OK. On failure, status code is 400 Bad Request.
* devices: AJAX with POST. It has a field *command* of type string. There are three possible commands:

|  |  |
| --- | --- |
| Commands | quick description |
| show\_devices | Show all devices that a user has |
| get\_device\_recent,(dev\_id)[,dev\_id[, ...] | Get the most recent data about one or more devices |
| get\_device\_all,(dev\_id)[,dev\_id[, ...] | Get all data about one or more devices |

On success, show\_devices command gets back code 200 OK, and an array of objects: {

‘dev\_id’: dev\_id,

‘dev\_name’: dev\_name,

‘controllable’: c

}\[, …\]

length of the array depends on the number of devices the user has.

get\_device\_recent,(dev\_id)[,dev\_id[, ...] commands queries data of given devices. Only most recent data is returned based on timestamp. On success, it gets back code 200 and data:

{

{

‘dev\_id’: dev\_id,

‘dev\_name’: dev\_name,

‘topic’: topic,

‘sensor\_value’: sensor\_value,

‘time\_stamp’: time\_stamp,

}\[, …\]

}

get\_device\_all,(dev\_id)[,dev\_id[, ...] commands queries all data stored related to the list of dev\_id. On success, it gets back code 200 OK and data:

{

{

‘dev\_id’: dev\_id,

‘dev\_name’: dev\_name,

‘topic’: topic,

[ {

‘sensor\_value’: sensor\_value,

‘time\_stamp’: time\_stamp,

}\[, …\] ]

}\[, …\]

}

* control: AJAX with POST. It has two fields: *dev\_id* and *control\_value*. Use *dev\_id* to uniquely identify the controlled device. Use *control\_value* to set the new value of the device. For example, if a device is LED and is off. “on” can be sent to that LED to turn it on.

Two possible error code of devices.php: 401 Unauthorized, and 400 Bad Request.

1. **Security**

Security is very important in this project. To be filled.