**COVER PAGE**

Globlock: 2 Factor Document Access Control, Repository and File Tokenization

**Document Control**

Revision History

Date Version Prepared Reviewed Approved

**Distribution List <<TABLE>>**

Name Title

Paul Hayes Project Coordinator

Jonathan McCarthy Project Lecturer

**Executive Summary**

**Introduction**

This document is a deliverable for a final year project as part of an Honours Degree in computing with the National College of Ireland.

This part is the tech report, then the Monthy

**Background**

**Initial Idea**

The idea for Globlock came about during the ‘dot conf’ I.T. conference held at the National College of Ireland. I attended the event in 2012 and had been given a small stress ball from Grafton Media as part of the welcome pack. I was researching the ‘Internet of Things’, home automation systems and RFID for personal projects at the time and something struck a chord with me. **Repositories**

Having worked with repositories in the past I have always found them to require technical

knowledge to make full use of them and found them to be out of reach of unskilled or untrained I.T.

professionals. Particularly around reducing duplication, the potential benefits are quite obvious

**NFC/RFID**

The use of NFC has become increasingly popular among phone manufacturers, payment systems and widespread in supply logistics, products production and manufacturing. The increased tech world’s focus on the ‘Internet of things’, has removed people fears over this type of technology to a certain extent and clearly identified a potential market for IOT, pervasive and ubiquitous products and services.

**Intended Audience**

The purpose of 'Globlock' is to provide a means for unskilled or professionals untrained in I.T., as well as those trained, but without the necessary infrastructure, to gain access to and utilise the benefits of a repository management system.

In order to reduce potential costs to customers, open source, freely available software and architecture was used throughout, both at a software and hardware level.

It is important to note that 'Globlock' is not a replacement for existing repository systems and

services. It is intended for use in environments where knowledge and training in concept and use of

repository systems is unavailable or unsuitable. In particular it is directed at small to medium sized offices where concurrency control and revision reposition is needed or would add value to the business processes. **Physical Tokens**

'Globlock' provides a mechanism to tokenise files and projects, and allow a physical abstraction of

these digital objects. This physical abstraction will allow strict concurrency control over the

associated files/projects.

The software and hardware interfaces required for this abstraction and control are all encapsulated

in 'Globlock'. Projects and their associated documentation can be linked to a physical object, and the system will allow users to review and maintain a repository of versions, review changes, re-assign files or projects to different physical objects, and report on progress and updates

**Aims**

**Concurrency Control**

Physical possession of globe objects is required, as well as a username and password. This 2 factor authentication aims to deliver a system with robust concurrency control on a single file or

group of files, with a revision history and reporting capability. The system would be implemented to

support a team of users who would have a business need to prevent the possibility of multiple

versions of a file or files in circulation.

**Tokenisation**

File or files would be tokenised. Each project would have a physical form; A Globe Object. The soft copy or digital representation of the file on the system (whether they be stored locally or remotely) would be represented by a physical token in the real world, in the form of a small durable ball, however any device capable of rfid communication at 125Mhz would be able to provide this abstraction, so cards, tags, buttons and fobs would allow this physical tokenisation.

By tokenising the file or files, it is then possible to prevent multiple user change access and multiple

revisions at any given time. Only the user in physical possession of the token, having granted them

access through the system prior to creating the token, would be able to make changes to the file or

files.

The physical handoff of the token in itself is also something meaningful at a psychological level such

as the passing a baton from one person to another.

**Repository Management**

The robust central API delivers a secure and tightly controlled document management system and repository. Each token and their associated files would be supported by administration of user profiles and access rights/permissions, and would allow high level users (super users or admins) to assign files to the physical Globe Objects, review change comments and current status.

**Definitions, Acrobyms and Abbreviations**

See appendix

**Technologies Used**

**Hardware**

**Arduino**

“Arduino is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. It's intended for artists, designers, hobbyists and anyone interested in creating interactive objects or environments.” - (arduino.cc, 2014)

Arduino was used in place of any other programmable hardware elements as it would allow a level of interaction that would not be possible with other systems. It would also ensure complete control over source code, and allow a high level of extensibility.

Arduino is also open source hardware and the components required for a bare bones hardware implementation could be sourced at a very low cost, and should mass production be required, the hardware elements could be very economical.

Arduino allowed interaction such as handshaking with the device and also allowed a pre formatted response from the device which improved the robustness of the system and prevented potential issues such as slow buffering and serial communication loss. It also allowed for scope for additional components such as LCD displays etc..

**RFID**

“Radio frequency identification, or RFID, is a generic term for technologies that use radio waves to automatically identify people or objects. There are several methods of identification, but the most common is to store a serial number that identifies a person or object, and perhaps other information, on a microchip that is attached to an antenna (the chip and the antenna together are called an RFID transponder or an RFID tag). The antenna enables the chip to transmit the identification information to a reader. The reader converts the radio waves reflected back from the RFID tag into digital information that can then be passed on to computers that can make use of it.” – (rfidjournal.com, 2012)

RFID is core to the entire concept and allows the necessary tokenisation of files by providing a unique identification for each Object which can then be associated with a particular project. MIFARE was used due to the small form factor of the tags and this was necessary for embedding the rfid capability in the globe objects (small, soft durable sponge balls).

**ID-12LA**

ID-12LA rfid reader chip was used in conjunction with the Arduino.

Designed and developed by ID-Innovations (http://id-innovations.com), this product is widely available and reads tags at a frequency of 125Mhz which was ideal for the Globlock system and MIFARE style RFID tags.

**ID12LA Breakout Board**

The ID20 Breakout board is a small PCB available from Sparkfun Electronics (sparkfun.com), which allows the ID-12LA (or ID20) to be mounted to the PCB and in turn mounted to a prototyping breadboard for testing and development.

**Client PC and Server Host**

There was no specific requirement for a PC but any PC/Laptop device capable of running Windows 7 and connecting to the internet or a local network was required for the client, and any system capable of hosting an apache web service, MySQL and PHP4.5 or higher was required for the back end of the system. This architecture will be described in more detail later in the document.

**Software**

**Github**

Throughout the development process Github was used as a code repository for the Globlock system, supporting references, libraries and documentation.

Github allowed code to be pushed and pulled from different PC’s and devices while still maintaining a master copy.

**Trello**

Trello was used to manage the early deliverables, tasks and milestones of the project. Once coding began,

**Development Environments**

**Arduino**

The Arduino IDE, which is an open source software IDE based on processing, allowed the use of RFID and software serial libraries. Individual code segments, called sketches are compiled into native C and uploaded to the Arduino directly from the IDE.

**Visual Studio**

Visual Studio was used for the development of the client application which was developed to demonstrate the interaction between a unique RFID tag read from the Arduino and sent to the client application, which in turn communicates with the server. Visual Studio was chosen for its extensive libraries and support for serial communcations.

**Notepad++**

For PHP scripts, as well as HTML, CSS and initialisation files, notepad++ was the only development environment used. This prevented added ‘baggage’ found with other IDE’s designed for web based technologies such as PHP, HTML5 and CSS3.

**Supporting Frameworks and Technologies**

**XAMPP**

XAMPP, again an open source product, was used to port a Lampp stack on a windows based machine. A portable version of XAMPP loaded to a usb flash drive was used to allow the deployment and publishing of php, html5 and css3 files from the different machines I developed the system on. In a production environment the same could be used.

**Apache HTTP**

The Apache Software Foundations Apache HTTP Server was used to publish and manage server files over the web. Apache was chosen as an open source alternative, due to its highly configurable environment and widely available knowledge bases and free support network on the internet

**PHP**

Although the client application was developed in C# in Visual Studio .NET, PHP an open source language and framework, was chosen as it allows greater portability for the system. PHP has greater flexibility around OS choices and this would allow the entire Globlock system to appeal to a greater audience.

**Database Technologies**

**MySQL**

MySQL, an open source database technology, was chosen due to its scalability, flexibility and cost. MySQL also comes packages with the LAMPP or XAMPP stack and this grouping of technologies has proved very successful in the past.

**SQLite**

A small portable lightweight DB was required for the client to manage transactions and users of the client application. SQLite was chosen for this and due to the fact it is again open source, and the language similarity to MySQL would reduce development time.

**Programming Languages**

**Client Application**

**C#**

C sharp was chosen for the client application in order to make use of the SerialPort libraries as well as provide a smooth and tasteful look and feel for the client application. Having worked with Java before I was confident the transition to C# would cause little difficulty.

**JSON**

“JSON is short for JavaScript Object Notation, and is a way to store information in an organized, easy-to-access manner.” – (Lengstorf, 2014)

Java Script Object Notation was used as a return string message from the Server Side API interaction as it is becoming an increasingly common message broker. It was chosen in place of XML. A broker object (discussed later) is encoded in JSON format on the server prior to being received by the client application.

**Client Device (Arduino)**

**C++**

The client C++

**Server Side API**

The Server Side API is coded almost entirely in PHP (decribed earlier) and messages encoded in JSON are also collected from the client application.

A single language for the server API allowed for a more robust and focused approach to its development.

**Management Website**

**HTML5 & CSS3**

HTML5 is used for the development of the Management Website. CSS3 is also used and the management website is a fully responsive desingn, which allows segmentation on smaller devices which improves the user experience when interacting with the system.

**PHP**

PHP was also used for the management of the website to provide the necessary table content for the different components of the system such as users, groups and documents.

**Javascript**

Javascript is used to add some sugar to the user interface and improve the look and feel of the website.

**Structure**

Brief Overview of Each of the technical chapters – TO DO

**System**

**Requirements**

**Features Overview**

Below is a brief overview of the system features. See Appendix of SRS document for full Use Case Diagram.

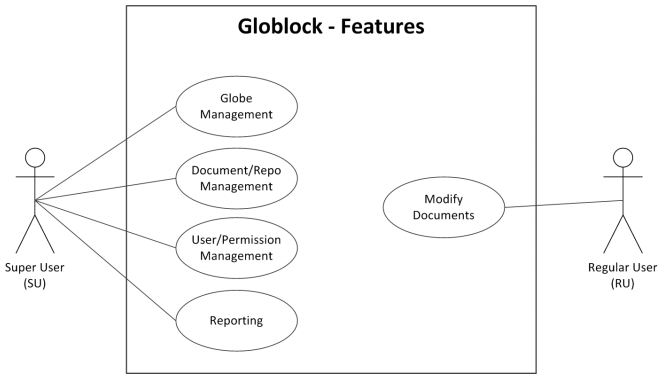


Figure 1

**User Permissions Management**

A very important feature of the initial specification ‘Globlock’ system was the ability to manage user accounts. This is still the case and has been implemented. A high level user essentially controls other user accounts and permissions over globes created and modified within the system.

Due to time constraints, the ability to assign individual permissions on specific projects is not possible however the system has been developed to allow this implementation at a later stage.

A basic user will exist which will interact with the globes and make changes to the associated files. At a later stage, these basic users will be restricted from accessing globes that they have not been given privilege to by the high level user. Privileges of read or write may be assigned to each user on each globe, but at present all users can interact with Globe Projects. **Repository and Document Management**

As each globe is manipulated with by basic users and changes made to its associated files and

documents, each iteration or revision of these will be stored and recorded. A central repository located on the server, the transparency of which will be hidden from the user, will store each

revision or version. This in essence will provide the archiving functionality and management of

documents. The location of the storage is configurable through the initialisation files on the server.

Future scope will allow a high level user to have a facility to take a revision or version of a single document, or entire group of documents, and extract from the central archived repository and associate with a new or existing Globe. This will essentially allow multi-threaded versions (branches) of documents.

**Globe Management**

Each Globe will be managed by high level users of the system. A function of the software will allow

the high level users to assign individual as well as groups of users access rights to the Globes created.

These permissions will be basic read/write but will allow users to be assigned to Globes for

information only. Users without write permissions will not be able to update or make changes to

revisions of the Globe or its associated files or documents. As stated all users currently have read/write access to all files.**Reporting**

Another feature of the system in future revisions will be basic reporting functionality which will allow high level users to track changes to Globes, files and documents and also track what users have interacted with the Globes. Each main section of the Management website currently lists all documents, globes, groups and users maintained within the system. If an addition is made, the particular sections form data is posted to a server side php script, which re-directs to the original location on successful completion.

Functional requirements

**System Features**

**User Management**

A function of the system is the facility to allow a SU to create user accounts so that they may interact

with the system. This is completed through the Management Website, hosted on the server.

**Document Management**

As the goal of the system is to provide a repository and archive for document management, the

ability to upload and download documents is a functional requirement.

**Document Upload**

When uploaded to the system, the documents are stored in a predefined location based on a combination of configurations set out in the initialisation files and the unique identity of the document in the database.

**Document Download**

Upon successful interaction with a globe object and user has been validated by the system, the document is downloaded locally to the client PC. Again the location of the storage is based on a combination of configurations set out in the initialisation files and the unique identity of the associated globe object.

**Globe Management**

Globe management is an encapsulation of all the functional requirements that will be needed for

documents to be tokenised and accessible from a RU or group of users. In the current state the system does not allow documents to be assigned individually to globe projects.

**Create Globe**

A globe can be created and added to the database. This is completed through the Management Website, hosted on the server.

**Archive Glove**

When a user uploads a new version of a document, the previous version is copied to an archive folder and replaced in a current directory with the newly uploaded document.

**Globe Access**

After Globes have been created, documents uploaded and associated and users given access to the necessary tools, there then exists a requirement for users to open the newly created Globes and modify their contents. This is made possible by the client applications interaction with the server side API.

**Open Globe**

When a user places the globe device at the reader, the client application receives its unique id, requests the associated documents from the server, and downloads them locally.

**Reporting**

A very basic functionality from the system is the ability to report on current and archived Globes that

have been created, and the changes that have occurred to them. Currently, each section of the management website lists the specific tables of entries, whether it be users, groups, globes, documents and the date they were added to the system.

**Non-Functional Requirements**

**Performance/Response time requirement**

**Client Application**

Windows service must not use more than 5% of system resources during idle periods and no more than 75% during non-idle for more than 15 seconds. This is to prevent the Globlock system being viewed as a burden or hindrance to the user.

In the current state, the application uses approximately 11.25 megabytes of system memory while the application is in an idle ‘listening’ state.



Figure 2

**Server Response**

Response from the server must not exceed 60 seconds for requests for primitive data queries. Download times should not exceed 5 minutes per document.

In the current state, response from the server after client application requests takes less than a second for any and all request types.

**Reader Response**

It should not take longer 10 seconds to read Embedded RFID in Globes from the Block device.

A certain amount of buffering is required in order to allow serial communications, however in the current state this is still taking less than a one or two seconds.

**Server Availability**

Server must have minimum of 99% uptime. This will allow 6.72 hours over a four week period to allow for scheduled changes or unscheduled down time.

If server is unreachable, documents must be stored locally until such time that the server becomes available. In the current state there has been no uptime failure, and documents are stored on the client pc. The location of the storage is based on a combination of configurations set out in the initialisation files and the unique identity of the associated globe object. If server response fails, the documents are still maintained locally.

**Recoverability & Reliability Requirement Globe Loss**

As a Globe is a physical object it may become lost. A facility to archive globes already exists and will allow previous documents associated with a lost Globe to be assigned to a new Globe.

The API has been designed in such a way as to allow globe objects to be assigned to projects by FORCE, in that the previous asset assigned to it is overwritten.

**Software Error** Response

All errors should be captured and in the event the system cannot commit changes the user will be clearly and effectively informed.

All errors are managed and stored to a logfile on the server in the case of API errors and all transactions are written locally on the client side sqlite database.

**Hardware Failure**

In the event of hardware failure the management application should continue to allow the management of users and documents, until such time that the Globe can be docked, loaded and undocked.

In the current state, all management of the system is carried out through the management website, entirely separate to the client. This allows complete separation in the event of a hardware failure.

**User Access**

Users without previously assigned access should not be able to access the files. Users with only read privileges should not be able to write or make changes. User accounts, both super (SU) and regular (RU) will require passwords. Passwords complexity will be defined the ISO27001 standard for Information Security.

In the current state all users of the client system require password identification. The password is encrypted using SHA1

**Communication Transport**

A method of encryption (TBD) will be used to encrypt data packets and files while being transported to prevent breaches.

In the current state, the system does not encrypt entire packet information, however session tokens are generated from a random string and encrypted with the addition of a salt value, using SHA1. User passwords are also encrypted using SHA1. It would be recommended that a SSL certificate be used on the server if system was implemented in a production environment.

**Maintainability & Re-usability Requirement**

Updates to platform of systems should be easy to manage and maintain with no additional training by the users.

The client application is a standalone executable, with an extremely simple user interface and as such would not require a great deal of training in order to interact with it.

**Re-Usability**

The Globes should be interchangeable and re-usable in that a super user may decide to assign new or archived information to a Globe, to prevent making the Globe objects obsolete.

The API has been designed to allow a FORCE server call which will overwrite the existing object reference with that of a new one.

**Code Re-use**

Code shall be written in such a way as too allow as much flexibility as possible around it’s re-use. Throughout the design and development of the system code re-use was high on the agenda and object orientation, segmentation and encapsulation facilitate this.

**Design and Architecture**

**Overview**

The diagram below gives a very high level overview of the separate components of Globlock.

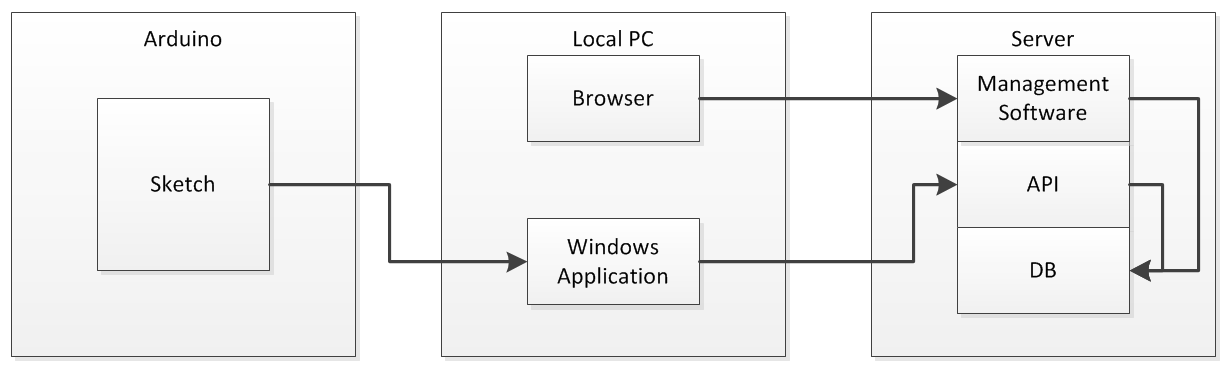


Figure 3

The Arduino device communicates tags to the client windows application. This application processes server requests, and returns the results to the client or carries out the desired client side functions depending on the result.

The API acts as middleware between the client and the database. The client may also interact with the management software which is delivered in the form of a website. Again, the website acts as middleware between the user and the database.

**Block Device**

The original block device design and schematic allocated some digital pins on the Arduino to allow the device to interact with an LCD display. There was also room for the addition of a buzzer to alert the user. This worked fine in the early stages of testing but added a level of complexity later on which slowed development.

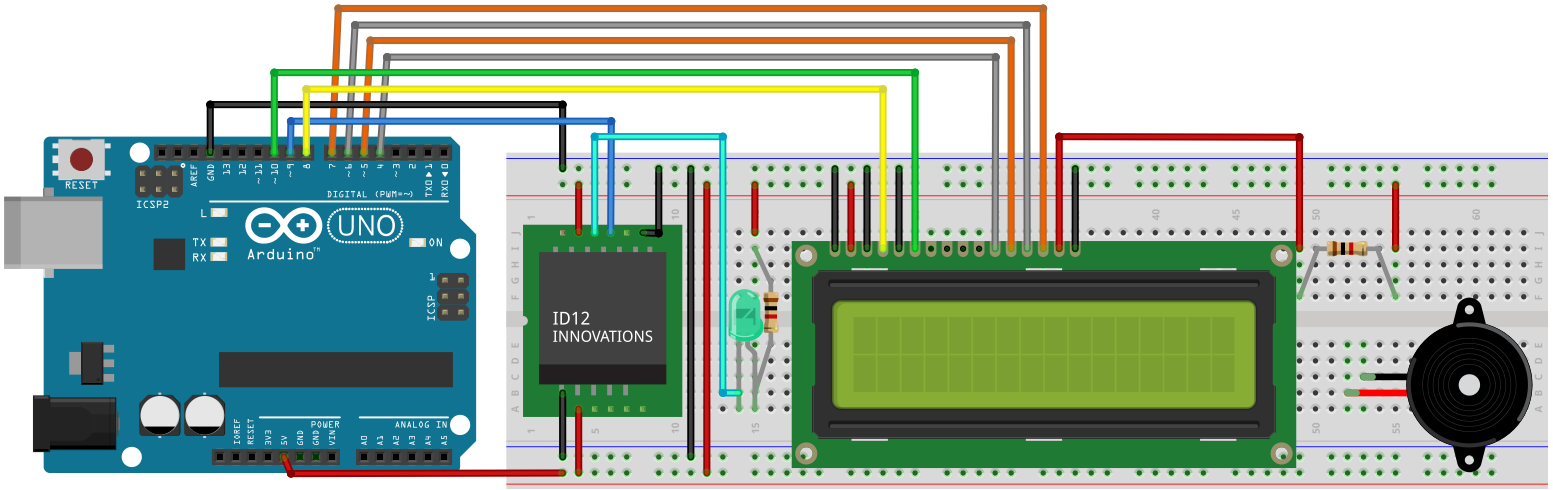


Figure 4

This additional LCD component would have provided some additional user experience, and relay information visually and aurally to the user such as system states (‘Connecting’, ‘Handshaking’,’ Listening’), however due to time constraints this additional functionality was dropped in favour of a simpler design schematic.

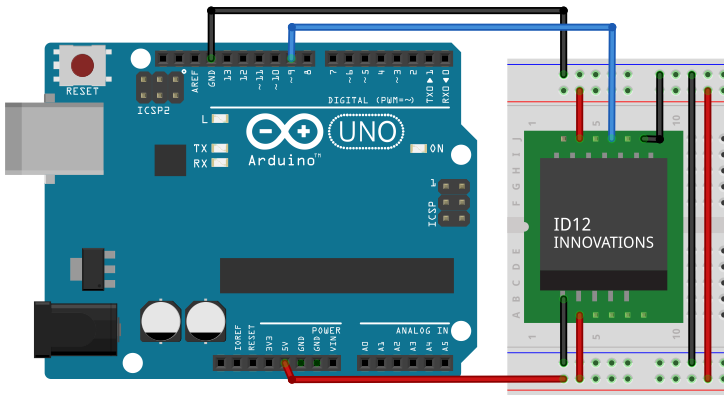


Figure 5

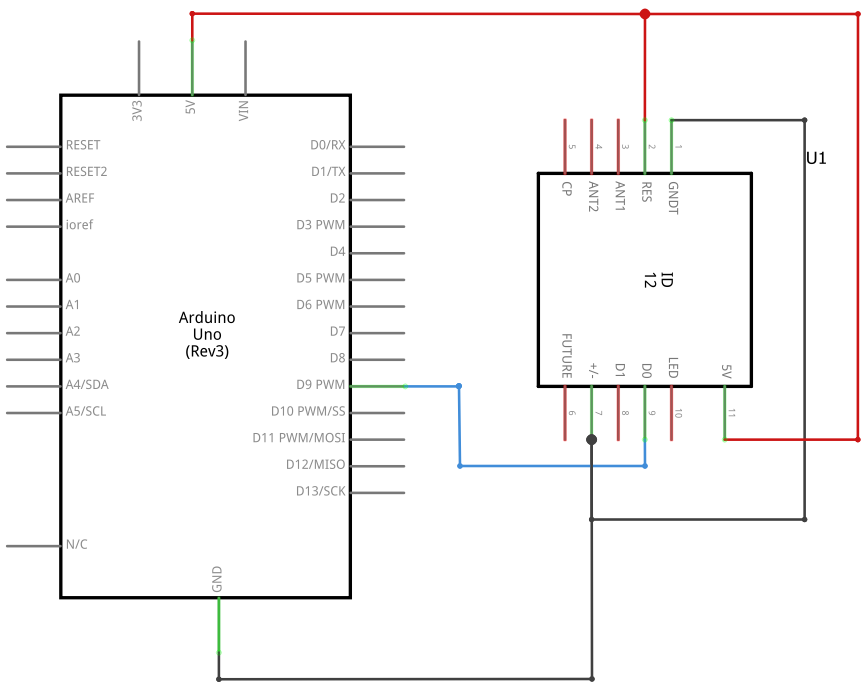


Figure 6

The diagram and schematic above were created using Fritzing (fritzing.org). The table below represents some important information contained in the figures:

|  |  |  |
| --- | --- | --- |
| Object | Colour | Description |
| Wire | Red | Live (+5v). Carries 5 volts from Arduino device to breadboard and connected components. |
| Wire | Black | Ground (-GND). Carries ground back to Arduino device |
| Wire | Blue | Carries tag id to the Arduino device using a virtual serial port (software serial). |
| Component | Black/Green | ID-12LA. Device to read rfid tags.  Carries tag ID using ASCII over I2C.  Draws approx 65mA of current  Range Approx 120mm. |
| Pin 9 |  | Used for capturing data from ID-12LA |
| Pin 13 |  | Used to flash for data received |

**Client Application**

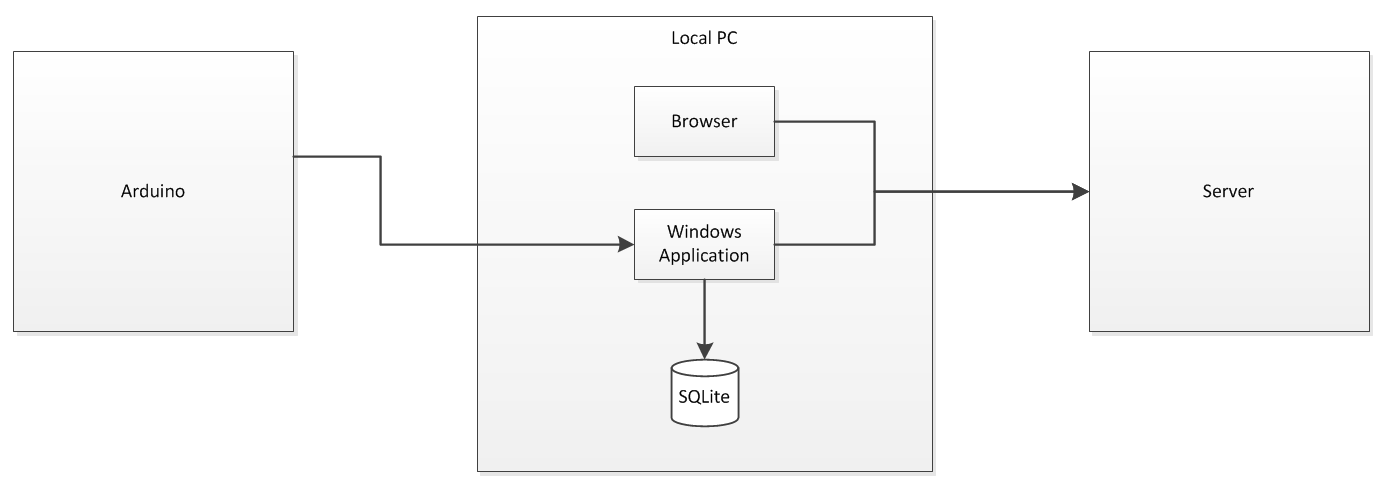
The client application uses a SerialPort library to send and receive commands from the block device through the devices USB port. 

Figure 7

Depending on the state and data received, it communicates with the server. To do this it makes use of the Web Client Library. Transaction information and login information is stored in a local SQLite database.

Through the browser, the user may make changes and create and upload new Documents, Globes, Groups and Users. This management web application is hosted on the server.

**Hosted Server**

Whether a local host or a remote host, the server provides the backbone of the entire system, and hosts a number of PHP files which are the core of the Globlock API.

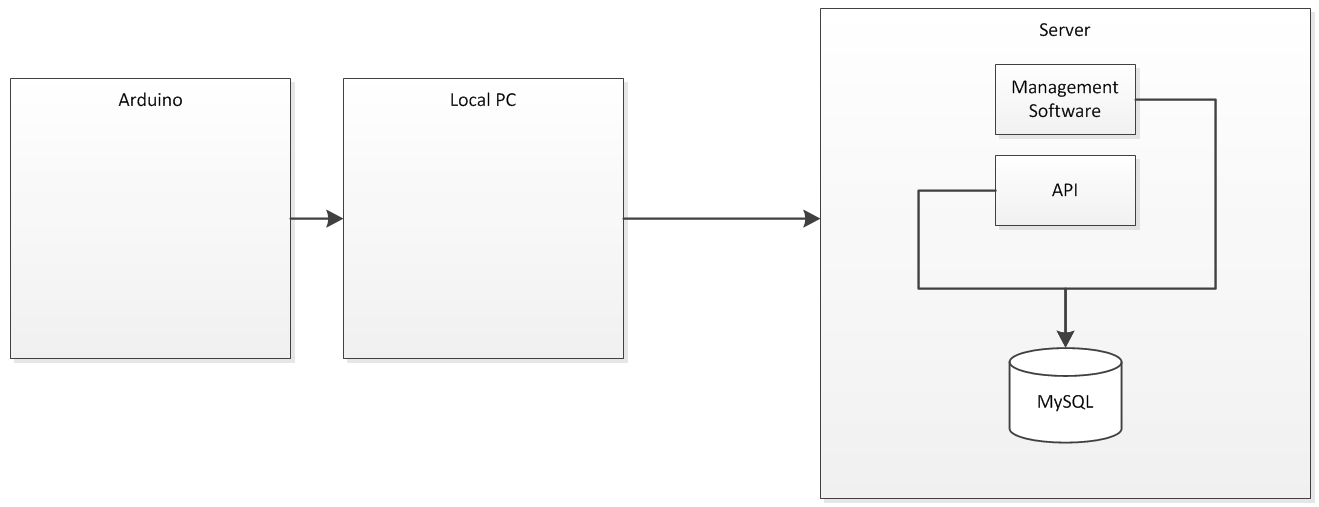


Figure 8

**Management Web Application**

From the management web applications view, the framework of the server the diagram below.

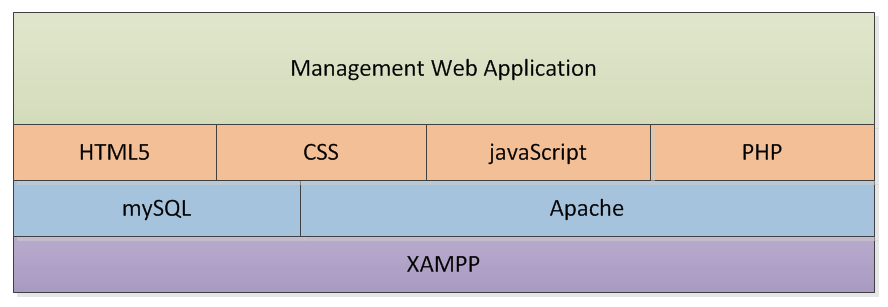


Figure 9

On top of the XAMPP stack sits the Apache HTTP server and a MySQL database. Both of which are open source and allow for great configurability.

On top of this is the web deployment such as HTML5 for the server pages, CSS3 to provide visually appealing interface components and html elements, Javascript to further add to this user experience element and finally some PHP scripts to allow server functionality.

**Globlock API**

From a purely API standpoint the framework is very similar.

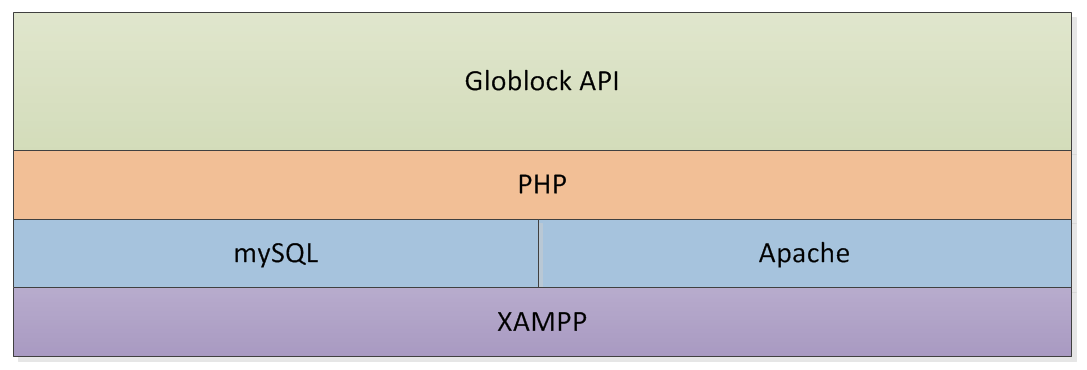


Figure 10

**Implementation**

**Database Schema**

**Server Side**

The current system state from a server perspective is outlined below.

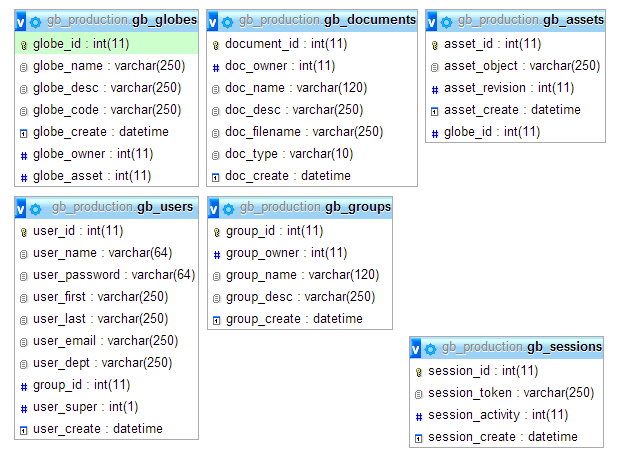


Figure 11

There are 6 tables in total in the ‘gb\_production’ database, so named after ‘Globeblock’ production system.

Each table has a unique primary key, and also maintains a default datetime stamp which allows the system to accurately identify when the particular record was inserted into the database.

**Sessions Table**

The session table (gb\_sessions) is the most frequently accessed and updated table and for this reason the schema for the table was kept to a minimum. The activity defines whether the session information is at the correct activity with the system.

The first entry point to the API is the ‘HANDSHAKE’ request, but after this if a client wishes to interact with the system they must first complete a ‘SESSION’ request. This will be discussed in greater detail later but essentially a session token is generated and inserted to the database.

The activity is then set to 1. Each time the session token is used, the activity is increased. A session token may only be used twice. The first of which must be to ‘VALIDATE’ a globe object .

All subsequent actions require the activity to be at 2 in the database, otherwise the requests will be aborted.

After an action is requested on the server at activity level 2, the session activity is set to -1, which represents an inactive session token. The session token can then no longer be used.

**Assets Table**

The assets table (gb\_assets) contains the references to individual unique globe objects and their association to globe projects (FK: globe\_id). Revision information for each asset is also maintained in this table.

**User Table**

The user table (gb\_users) maintains user information, as well as group association (FK: group\_id) and whether the user is a high level user (user\_super: 1/0) or not.

**Globe Table**

The Globe table (gb\_globes) maintains globe information, such as name, description and short code. A globe owner (globe\_owner) was put in place for future scope to allow permission around updates and changes to individual globes.

**Documents Table**

When a user uploads a document on the system, the document information is created here (gb\_documents). Again, an owner column is present for future scope to allow permission around updates and changes to individual documents.

**Groups Table**

The groups table (gb\_groups) maintains information related to the different groups with the system. As with Globes and Documents, an owner attributes allows for future scope to around permissions and updates to individual groups.

**DB Schema in SQL format**

-- Database: `gb\_production`

--

CREATE DATABASE IF NOT EXISTS `gb\_production`

USE `gb\_production`;

-- --------------------------------------------------------

-- Table structure for table `gb\_assets`

CREATE TABLE IF NOT EXISTS `gb\_assets` (

`asset\_id` int(11) NOT NULL AUTO\_INCREMENT,

`asset\_object` varchar(250) NOT NULL,

`asset\_revision` int(11) NOT NULL,

`asset\_create` datetime NOT NULL DEFAULT CURRENT\_TIMESTAMP,

`globe\_id` int(11) NOT NULL,

PRIMARY KEY (`asset\_id`)

);

-- --------------------------------------------------------

-- Table structure for table `gb\_documents`

CREATE TABLE IF NOT EXISTS `gb\_documents` (

`document\_id` int(11) NOT NULL AUTO\_INCREMENT,

`doc\_owner` int(11) NOT NULL DEFAULT '0',

`doc\_name` varchar(120) NOT NULL,

`doc\_desc` varchar(250) NOT NULL,

`doc\_filename` varchar(250) NOT NULL,

`doc\_type` varchar(10) NOT NULL,

`doc\_create` datetime NOT NULL DEFAULT CURRENT\_TIMESTAMP,

PRIMARY KEY (`document\_id`)

);

-- --------------------------------------------------------

-- Table structure for table `gb\_globes`

CREATE TABLE IF NOT EXISTS `gb\_globes` (

`globe\_id` int(11) NOT NULL AUTO\_INCREMENT,

`globe\_name` varchar(250) NOT NULL,

`globe\_desc` varchar(250) NOT NULL,

`globe\_code` varchar(250) NOT NULL,

`globe\_create` datetime NOT NULL DEFAULT CURRENT\_TIMESTAMP,

`globe\_owner` int(11) NOT NULL,

`globe\_asset` int(11) DEFAULT NULL,

PRIMARY KEY (`globe\_id`)

);

-- --------------------------------------------------------

-- Table structure for table `gb\_groups`

CREATE TABLE IF NOT EXISTS `gb\_groups` (

`group\_id` int(11) NOT NULL AUTO\_INCREMENT,

`group\_owner` int(11) NOT NULL DEFAULT '0',

`group\_name` varchar(120) NOT NULL,

`group\_desc` varchar(250) NOT NULL,

`group\_create` datetime NOT NULL DEFAULT CURRENT\_TIMESTAMP,

PRIMARY KEY (`group\_id`)

);

-- --------------------------------------------------------

-- Table structure for table `gb\_sessions`

CREATE TABLE IF NOT EXISTS `gb\_sessions` (

`session\_id` int(11) NOT NULL AUTO\_INCREMENT,

`session\_token` varchar(250) NOT NULL,

`session\_activity` int(11) NOT NULL,

`session\_create` datetime DEFAULT CURRENT\_TIMESTAMP,

PRIMARY KEY (`session\_id`)

);

-- --------------------------------------------------------

-- Table structure for table `gb\_users`

CREATE TABLE IF NOT EXISTS `gb\_users` (

`user\_id` int(11) NOT NULL AUTO\_INCREMENT,

`user\_name` varchar(64) NOT NULL DEFAULT '1',

`user\_password` varchar(64) NOT NULL DEFAULT '0',

`user\_first` varchar(250) DEFAULT 'undefined',

`user\_last` varchar(250) DEFAULT 'undefined',

`user\_email` varchar(250) DEFAULT 'undefined',

`user\_dept` varchar(250) DEFAULT 'undefined',

`group\_id` int(11) DEFAULT '0',

`user\_super` int(1) NOT NULL DEFAULT '0',

`user\_create` datetime NOT NULL DEFAULT CURRENT\_TIMESTAMP,

PRIMARY KEY (`user\_id`)

);

-- --------------------------------------------------------

**Client Side**

The current system state from a client database perspective is outlined below.

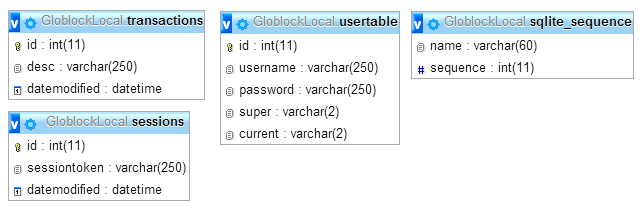


Figure 12

**Transactions**

The transaction table allows information about the continuous running success of the client application to be written to the local sqlite database.

**UserTable**

The user table maintains username and password (encrypted using SHA1) information for users who wish to remain logged into the application after the application is shut down or restarted.

**Sessions**

The sessions table maintains session token information for use in the application for server requests, and removes the need to store token information in memory.

**SQLITE\_SEQUENCE**

This table is a boilerplate sqlite table that maintains the auto increment sequence information for other tables in the database.

**DB Schema in SQL format**

-- Database: `GloblockLocal.db`

--

--------------------------------------------------------

-- Table structure for table `Sessions`

CREATE TABLE IF NOT EXISTS Sessions(

id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,

[sessiontoken] VARCHAR(250),

[datemodified] datetime

);

--------------------------------------------------------

-- Table structure for table `Transactions`

CREATE TABLE IF NOT EXISTS Transactions(

id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,

[desc] VARCHAR(250),

[datemodified] datetime

);

--------------------------------------------------------

-- Table structure for table `UserTable`

CREATE TABLE IF NOT EXISTS UserTable(

id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,

[username] VARCHAR(250),

[password] VARCHAR(250),

[super] VARCHAR(2),

[current] VARCHAR(2)

);

--------------------------------------------------------

-- Table structure for table `aqlite\_sequence`

CREATE TABLE IF NOT EXISTS sqlite\_sequence(

name,

seq

);

INSERT INTO sqlite\_sequence VALUES('Transactions',0);

INSERT INTO sqlite\_sequence VALUES('UserTable',0);

<<TO DO>>

Class diagram overview

API Point of Entry

Config broker

Request broker

Database Broker

Session Handler

Globe Handler

File Handler

Testing

Graphical User Interface (GUI) Layout

Customer testing

Evaluation

Conclusions

Further development or research

References

Appendix

Project Proposal

Project Plan

Requirement Specification

Monthly Journal

Other Material Used

References

http://www.arduino.cc/

http://www.rfidjournal.com/faq/show?49

<http://www.copterlabs.com/blog/json-what-it-is-how-it-works-how-to-use-it/>

<http://jsonviewer.stack.hu/>

<http://fritzing.org/home/>

<http://tronixstuff.com/2013/02/26/arduino-tutorial-15a-rfid-with-innovations-id-20/>

<https://www.sparkfun.com/datasheets/Sensors/ID-12-Datasheet.pdf>

<http://arduino.cc/en/uploads/Main/Arduino_Uno_Rev3-schematic.pdf>

http://www.byteparadigm.com/applications/introduction-to-i2c-and-spi-protocols/

function start(&$broker){

if (!($\_SERVER["REQUEST\_METHOD"] == "POST")) {

$broker->handleErrors("NON [POST] TYPE SERVER REQUEST ",121);

echo $broker->returnJSON();

return false;

}

return true;

}