Web Based USB Storage Manager

A Mobile Storage Application

Adrian Lowney, 20047116

02/01/2014

This report documents my progress to date for my final year project. The initial design and development stages are the central topics which are discussed in detail to outline my plan for this project.

Declaration of Authenticity

Except where explicitly stated, this report represents work that I have done myself. I have not submitted the work represented in this report in any other course of study leading to an academic award.

Table of Contents

Declaration of Authenticity 1

Project Goals 5

The Platform 6

Project Scope 7

Methodology 9

Waterfall Methodology 9

Rapid Application Development (RAD) Methodology 9

Fastest First 9

Visual Prototypes 9

Destructured Testing 10

Agile Software Development Methodology 10

Advantages: 10

Disadvantages: 11

Chosen Methodology 11

Functional Requirements 12

User Registration 12

User Login 12

Create/Delete Storage Array 12

Download/Upload a File(s) 12

Add/Remove Device 12

Format Device 12

Assign USB Disk Alias 13

File Modification/Permissions 13

IP Address Emailing 13

Internet Access 13

Non-Functional Requirements 14

Required Skills 14

Throughput 14

Storage 14

Security 14
Choosing a Primary Application Language 15
Ruby on Rails Framework 15
PHP: Hypertext Preprocessor 16
Choice: PHP 16
The User of the Application 17
Use Case Descriptions 18
Full Class Diagram 24
User Class 24
UserHandler Class 25
LoginView Class 26
RegisterView 27
StorageDevice Class 28
DeviceHandler Class 28
File Class 29
FileHandler Class 29
LoggedInView 30
DeviceView Class 30
FileView Class 31
Sequence Diagrams 32
Login Sequence 32
Registration Sequence 33
Authentication Sequence 34
View Devices Sequence 35
View Device's Files Sequence 36
Interface Design 37
Registration Page 37
Login Page 38
View Devices Page – My Devices 39
View Devices Page – Connected Devices 40

View Files Page 41
Technical Feasibility Evaluation 42
The Hardware 42
Device Software 43
Development Software 43
Partial Trial Implementation 44
Login Class 44
Login Page 45
Using PHP to View USB Disks 46
Development 48
Development Schedule 48
Development to Date 49
RegisterView Class Implementation 50
LoginView Class Implementation 51
References 53

Project Goals

The primary goal of this project is to create a web based application that allows a user to manage their USB storage devices locally and remotely via their home network. The aim is to give the user the ability to easily increase their storage capacity by adding USB storage disks to their storage array in their home. I intend that the application will enable the user to manage their data from any location as long as they have an internet connection. I aim to design the application to have an interface that is compatible with both mobile and desktop web browsers.

My goals in terms of the applications hardware are to have the BeagleBone Black and a USB hub working in tandem to allow the user to access their USB storage disks. The key piece of hardware enabling the user to perform this is the BeagleBone Black, an embedded Linux device no larger than a credit card. The second piece of hardware is a multi-port USB hub that will be attached to the BeagleBone Black and will allow the user to attach their storage devices depending on how many ports are present on the USB hub.

I also aim to develop the application to allow for multiple users. This will permit the application to provide the functionality to multiple users on a single Beaglebone Black. I wish to implement this functionality to the project as I envisage that most people using the application will appreciate the ability to allow friends or family members to use the application also.

Another area I feel is a necessity in an application such as this one is security. Given the nature of this application, insofar as that it deals with the users files, it is important that the application be designed with security implemented as required.

The Platform

I have chosen to develop my application using the Ubuntu Linux ARM distribution of the Linux operating system. I have chosen this as my operating system of choice as I am most familiar with the Ubuntu distribution of Linux. I used a pre-built image of this operating system that was preconfigured with a custom Kernel that was compatible with the BeagleBone Black. The application is PHP driven and will run on an Apache server on this device.

This setup will essentially allow anyone who wishes to implement this application on a Linux machine running an Apache server. In this sense the application is not restricted to be run on a BeagleBone Black but my intention is to focus solely on this as a platform so I can develop a high quality and focused application.

The BeagleBone Black is a relatively low cost and low power consumption, embedded Linux device that can perform all the functions of a typical desktop Linux machine but employs a miniscule footprint by comparison. The low cost of this hardware and the freely available Ubuntu Linux distribution offers an unrestricted development platform for developing this application. For this reason I have chosen this as my platform.

Project Scope

What I wish to deliver at the end of this project is an application that allows the user to manage their array of USB storage disks remotely and securely, this is all done whilst having full control over the physical hardware performing these data centric tasks. A user naturally has to invest a large amount of trust in the commercially and freely available mobile storage applications that are currently on the market. With this, the user is typically is located in a geographically separate area, or in most cases even in a separate continent to where their data is actually stored. An important goal of the application is to re-associate the user with the raw storage that is holding their data that may or may not be private in nature. Given that the hardware is present in the user's home this allows them to have full control over their devices, permitting them to use the system in a plug and play nature, if required the user can remove a disk at any time as long as the disk is unmounted. The networking scope of the device is a decision entirely to the user. The application is independent of whether the device is port forwarded through a NAT-enabled router or simply operating within the bounds of the user's local area network.

When developing the application I will provide the user(s) with the ability to:

- Register with their details, specifically their username and password.
- Authenticate with the server over HTTPS with their username and password.
- Mount or unmount a USB disk.
- Format a disk.
- Assign themselves to a USB disk or USB disks.
- Create a personal storage array of the USB disks they have assigned themselves to.
- Upload and download one file or a queue of files to a USB disk.
- Modify the name and move files on a USB disk or to another USB disk.
- Create and modify the directories of a USB disk.
- Permit read or full access to other users by username.

The system is intended to be developed to perform these tasks autonomously:

- Automatically run the Apache server when powered on.
- Check for any new devices that have been attached to the USB hub.

- Send an email notifying the user of the public IP addresses of the network the server is attached to. This is done as most home users have dynamic public IP addresses that change regularly.
- Present the user with their storage array.
- Notify the user if a USB disk is approaching capacity or has been removed.
- Notify the user if the current file operation they are performing is colliding with another user.

Methodology

In the initial design phase of my project I made the choice of using the agile software development methodology for my project. I chose this methodology as I felt it best suits my needs and I used this methodology whilst on my work placement. I also considered the waterfall method and the Rapid Application Development methodology for my project.

Waterfall Methodology

I chose to discard this method early in the design phase as it did not suit my approach to developing this project. The non-iterative fashion of the waterfall method would have hindered my projects overall refinement as I intend to develop the project in stages and it will demand design additions and alterations over the course of the applications development.

Rapid Application Development (RAD) Methodology

The primary characteristic of the RAD methodology is that the application is designed with fast delivery of work as a primary aim. Here are some of the essential characteristics of the RAD methodology:

- Fastest First

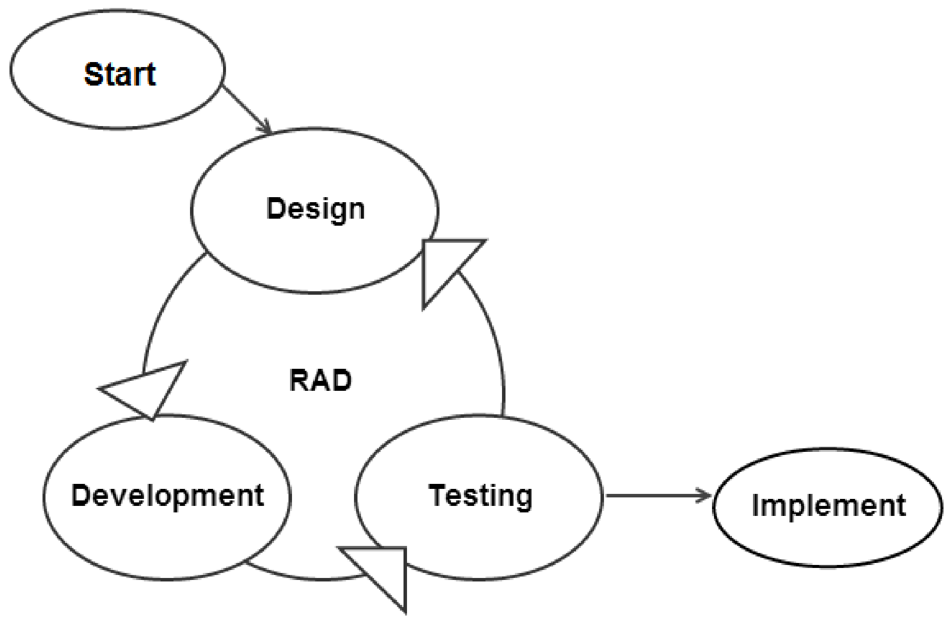
  If the solution that first comes to the developer's mind will achieve the intended goal then this is most likely the best solution. As hinted in this methodology's name, whatever gets the job done fastest, use it. The most important goal is to get the job done, if the method employed to achieve this goal can be refined afterward, then so be it. The ideology here is to give only brief thought to the design and favour getting something to work over perfected design.

- Visual Prototypes

  RAD projects require the ability to generate visual prototypes effectively. It can take many trials before the user is happy with the user interface, ultimately it is the Human user who is the greatest challenge in software development.

- Destructured Testing

  Using the RAD methodology, software is typically created in granular chunks. In this case, it makes it easier to test such chunks as they come instead of testing large, monolithic systems. RAD destructures testing as many small code fragments are more manageable and easily corrected compared to one large program. (MacFarlane, 2004)



Agile Software Development Methodology

The agile development model is a style of the incremental design and development model. In this model software is routinely created and developed in stages. The product of this design is the creation of smaller, incremental releases which build on the previous iteration of the software by adding new functional elements. Once a new piece of functionality is added, this is then rigorously tested. The agile model lends itself closely to time critical projects such as this.

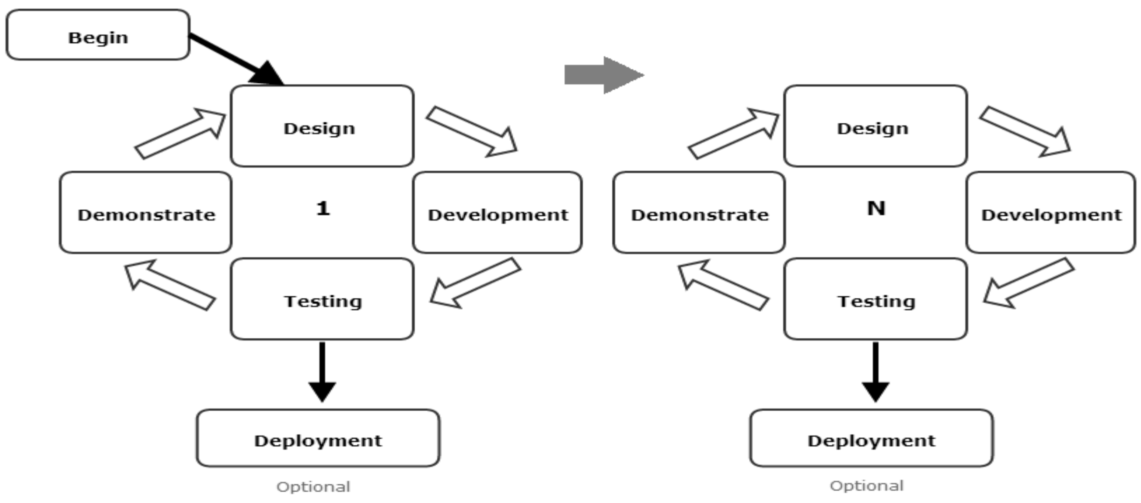
Advantages:

- Continuous attention to perfecting the applications code and design.
- Changes in design can be implemented at any stage, even late changes are adaptable.
- Software with working functionality is delivered at regular intervals compared to the weeks or months the waterfall method tends to offer.

- Rapid and continuous software releases to the customer helps to maintain their satisfaction.
- Agile emphasises the interactions between people rather than the processes and tools. Customers, testers and the developers are constantly interacting with each other to deliver the best possible application.

Disadvantages:

- The customer representative can cause the project to lose focus if they do not correctly present the application's desired outcomes.
- In larger projects, there can be issues with assessing the amount of required effort when assessing deliverables early in the design phase.
- Agile can have a lack of emphasis on necessary designing and documentation. (ISQTB, 2012)



Chosen Methodology

I have chosen to utilise the agile methodology for my project. The reason I have chosen it is due to how it is able to adapt to changes in design and addition of features. As I will be doing this project in staged increments I believe agile will be the most suitable. The constant reviewing of the applications code and design that agile requires will aid me in refining each of the applications features as they are added.

The fastest solution first ideology of the Rapid Application Development (RAD) methodology is not the style of methodology I want to implement. I believe that this model could create a project with disjointed and short-sighted code, leading to a lack of cohesion within the application.

Functional Requirements

• User Registration

All users will be required to register with the application before they can interact with it further. All new users will be presented with a registration form that will require some basic credentials but more importantly their desired username and password. This username and password will be the method of authenticating with the application for all future use.

• User Login

Existing users will be presented with a login page that will require them to enter their username and password before progressing further. Once this is done the server will authenticate the user and bring them to their homepage.

• Create/Delete Storage Array

Each user can assign a USB disk exclusively to them in order to create an array of personal storage. The storage array is simply a mapping of a user to one or more disks. In the initial iterations of this project each USB disk will be assigned exclusively to one user. I hope to make it so that multiple users can have access to a single USB disk by partitioning the disk per user.

• Download/Upload a File(s)

The application allows the user to upload and download files to the USB disks contained within the user's storage array. The user can choose to download one file or create a queue of files to download in succession. The user can also do this with the uploading of files as one upload or a queue of many uploads.

• Add/Remove Device

A user has full control over their storage array. They can add or remove a USB disk at any time from a management panel for their array. Once a device is added to the system, it will be automatically mounted to the BeagleBone Black's file system. To remove a device it must not be currently assigned to any other users' storage array. The removal process simply unmounts the device from the file system.

• Format Device

An attached USB disk can be formatted by a user as long as the device is not currently assigned to any user's storage array. This requirement is included as a disk may be corrupted or the user may desire to change the file system type of the USB disk.

- Assign USB Disk Alias

A user has the ability to assign a more readable name to a USB disk in their storage array. This will be implemented as Linux will generally assign a less user readable name to each attached USB drive. The alias functionality will be implemented to make the system more intuitive for the users.

- File Modification/Permissions

The user is able to modify the location and name of every file within their storage array. If the user desires they can move a file from one directory to another, even if the resulting directory is on another USB disk. The user will only be permitted to perform these actions on files they own. If the user permits, they can make it possible for other named users to have read or full access to a file.

- IP Address Emailing

Given that the application is intended to be ran in the home of the user it is required that the application notify the user of the public IP address of the web server. This is necessary as most household internet connections use dynamically allocated public IP addresses. The application will regularly check if the public IP address of the network has changed, and if so email the user's with the new IP address.

- Internet Access

The application will run as normal behind a network address translation (NAT) enabled router. If the user does want to use the device beyond their home LAN then they will need to port forward port 443(HTTPS) to make the application externally accessible.

Non-Functional Requirements

- Required Skills

The major skills and expertise required to complete this project are as follows:

  - Object Oriented PHP scripting.
  - Bash scripting.
  - HTML, CSS and JavaScript knowledge.
  - AJAX and jQuery skills.
  - Apache and MySQL experience.
  - Ubuntu Linux experience.
  - Embedded Device experience.
  - Networking knowledge.

- Throughput

I have some concerns regarding the speed of the BeagleBone Black device if multiple users may decide to upload files concurrently. The device may become sluggish if multiple uploads are running concurrently. I am less concerned over the concurrent download of files as most USB disks have markedly faster read rates compared to their write rates. Essentially, the device is capable of coping with concurrent uploads and downloads but performance will obviously be negatively affected when this occurs. In most cases I do not envisage this application being used concurrently by many users.

- Storage

The hardware of the Beaglebone Black contains a single USB 2.0 port. I recommend and assume that the user connects a powered USB hub to the Beaglebone Black when using this application. There is an inherent requirement with this application that requires the user to control the amount of USB disks attached to the USB hub. If the user requires more ports than are available on the hub then this is something they will have to upgrade or use larger USB disks instead. Given that 32 GB USB flash drives are relatively cheap this is not a major issue.

- Security

The Apache server will be configured to run using HTTPS. Any user who contacts the application using HTTP will be automatically redirected to the secure site. This is implemented to protect the user's data.

Choosing a Primary Application Language

For this project I have considered developing the application with PHP providing the main functionality behind the website. I have also considered using Ruby to do this using the Rails Framework. I have researched both these options in order to make the best choice of language before taking the project further.

Ruby on Rails Framework

Ruby is known for its uncluttered syntax which doesn't require much punctuation. When compared with Java, Ruby appears to be a much more streamlined language that requires less coding for the creation of structures like data fields. The main advantage of Ruby is the package manager RubyGems. RubyGems is open for anyone to upload a gem to which makes them immediately available for installation. For the most complex of websites the use of gems is crucial.

Some downsides are associated with the Ruby language also. Ruby does not lend itself well to processing performance when compared to Java and C++. Also for a website that may require a lot of simultaneous traffic, Ruby is not equipped to deal with this type of situation efficiently as it lacks parallelism.

Rails itself has become popular with Ruby as its conventions are attractive to many developers. The main benefit of the Rails Framework is that it can produce web applications that any other Rails developer can understand faster than other languages such as PHP. Developing applications the "Rails way" is designed to eliminate ambiguity when it comes to making coding decisions. Because of this programming ethos that Rails demands the framework can help produce highly reusable, well designed code. (Kehoe, 2013)

PHP: Hypertext Preprocessor

PHP is one of the easiest scripting languages to learn for developers. Having studied Java as part of my course, it is extremely simple to transfer my object oriented skills to PHP due to syntactical and other similarities. Compared to programming with Ruby on Rails it is much easier to fix problems with in most cases. This is because with each request it receives; PHP cleans up and starts fresh. Thus an issue caused by a single request may not always disrupt others. PHP lends itself well to the object oriented paradigm, allowing the developer to create custom classes from which other class can borrow from as well. PHP does not require large amounts of system resources to run; it operates much faster compared to many alternative scripting languages. PHP's maturity as a language creates a reasonably stable language that has had many of its issues defeated over time. (Taei, 2013)

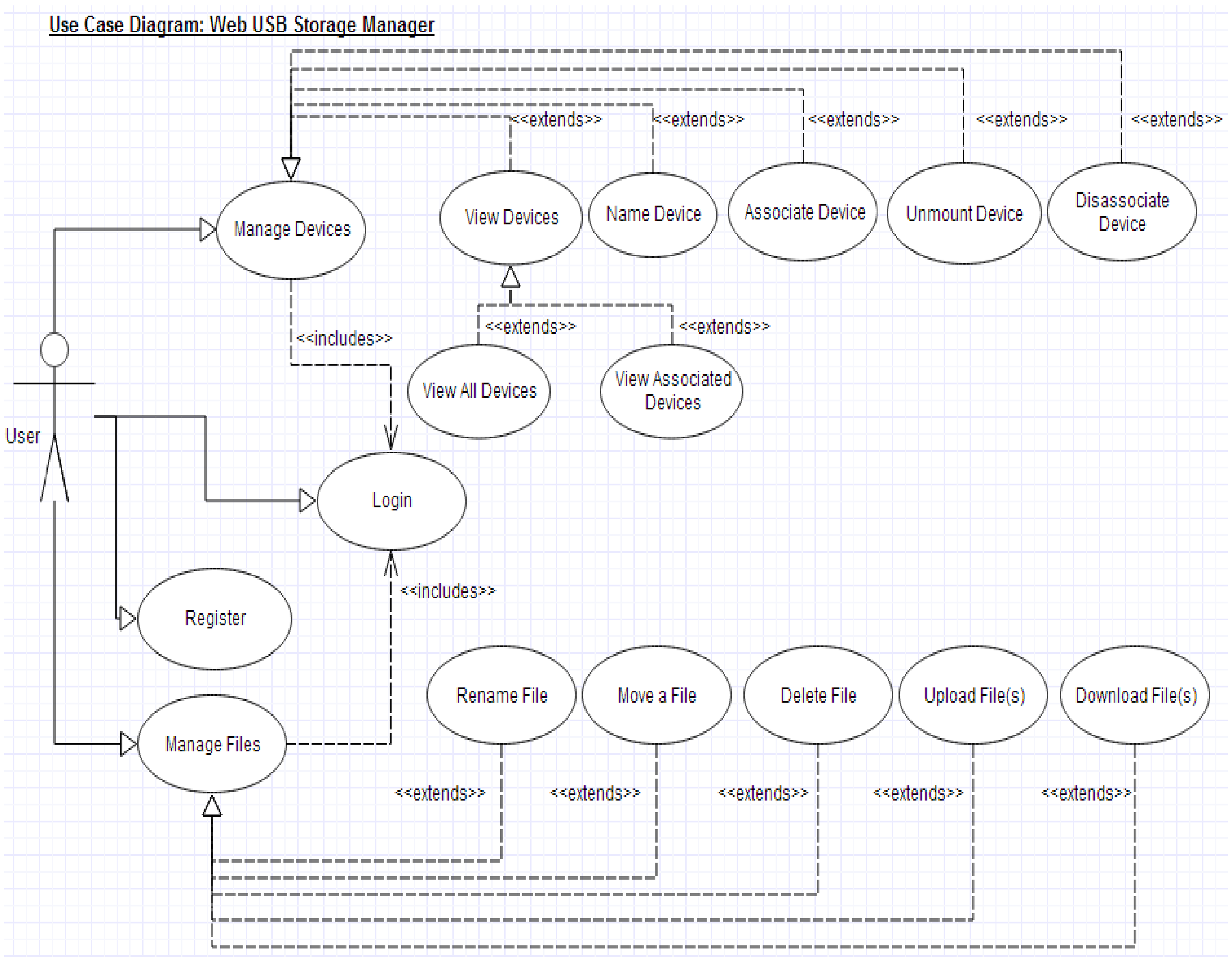
Choice: PHP

Personally, having developed web applications in both Ruby on Rails and PHP5 I can say that PHP has been my favourite of the two. I feel I will be able to develop my application to a better standard when using a language I am more comfortable with. This is not to say I would not like to learn Ruby on Rails further, but I simply do not have the time to ascend the steep learning curve inherited by such language.

I have also used PHP to develop an application I designed and developed on my work placement as part of my course. During this time I became familiar with PHP and have already conquered the initial learning curve that comes with it. For these reasons I intend to develop the application using this scripting language for the best chance of producing a quality application whilst learning the most.

The User of the Application

Below is my use case diagram of the user's interaction with the system. Each of the use cases seen in the Use Case diagram has been included in the functional requirements of the application. The use case diagram remains the same whether the user is accessing the application over the Internet or on their local network.



Use Case Descriptions

Name: Register

Description: New user registers with application.

Actors: User

Preconditions: User has not previously registered.

Flow:

1. User loads the initial home page
2. Application presents register page.
3. User enters credentials.
4. System validates user.
5. User added to database.
6. User redirected to homepage.

Name: Login

Description: Registered user returns to the site.

Actors: User

Preconditions: User is not already logged in. User is registered.

Flow:

1. User loads the main page.
2. Application presents the login page to the user.
3. User authenticates with their username and password.
4. Application presents user presented with their homepage, end of use case.

Name: View Storage Array

Description: User wishes to view their storage array.

Actors: Authenticated user

Preconditions: User has a storage array.

Flow:

1. Authenticated user navigates to their homepage.
2. Application presents the user’s storage array on homepage.

Name: Add USB Storage

Description: User wishes to add to their storage array.

Actors: Authenticated user

Precondition: USB disk is attached.

Flow:

1. Authenticated user navigates to homepage.
2. User selects add storage button.
3. User presented with unassigned USB disks.
4. User selects a disk.
5. Disk is assigned to user and returned to homepage.

Name: Remove USB storage

Description: User wishes to remove a disk from their storage array.

Actors: Authenticated user

Preconditions: The USB disk is present in the user's storage array.

Flow:

1. Authenticated user navigates to their homepage.
2. User selects remove storage button.
3. System shows user their assigned USB disks.
4. User selects a USB disk.
5. System disassociates USB disk with current user.

Name: Download File(s)

Description: Users wishes to download a file or files from a USB disk

Actors: Authenticated user

Preconditions: The file is present on a USB disk within the user's storage array.

Flow:

1. Authenticated user navigates to their homepage.
2. User selects a USB disk from their storage array.
3. User marks file or files to download.
4. User selects download.
5. Application starts to consecutively push the files to the user.

Name: Upload File(s)

Description: User wishes to upload a file or files to a USB disk within their array.

Actors: Authenticated user

Preconditions: The USB disk has enough space remaining.

Flow:

1. Authenticated user navigates to their homepage.
2. User selects the upload button.
3. User selects their file or files to upload.
4. User chooses disk from their array.
5. User stars upload.
6. Application starts to consecutively upload file(s).

Name: Modify File

Description: User wishes to rename or allow access to a file within their array.

Actors: Authenticated user

Preconditions: The file is within the user's array.

Flow:

1. Authenticated user navigates to their homepage.
2. User selects a USB disk from their array.
3. User selects a file on USB disk.
4. User presses the modify button.
5. User changes filename or gives other user access by entering username.

Name: Move File

Description: User wishes to move a file or files within their storage array.

Actors: Authenticated user

Preconditions: If the target is a directory on another disk then this disk must have sufficient space.

Flow:

1. Authenticated user navigates to their homepage.
2. User marks file or files from their array.
3. User selects move button.
4. User prompted for target disk and directory.
5. User choses directory and disk.
6. Application performs file relocation.
7. User redirected to homepage.

Name: Delete File

Description: User wishes to delete a file within their storage array.

Actors: Authenticated user

Preconditions: User has access to disk. User is allowed to modify the file.

Flow:

1. Authenticated user navigates to their homepage.
2. User navigates to file within a USB disk in their array.
3. User marks the file and selects delete.
4. Application deletes the file.

Name: Unmount USB Storage

Description: User wishes eject a USB disk from the BeagleBone Black.

Actors: Authenticated user

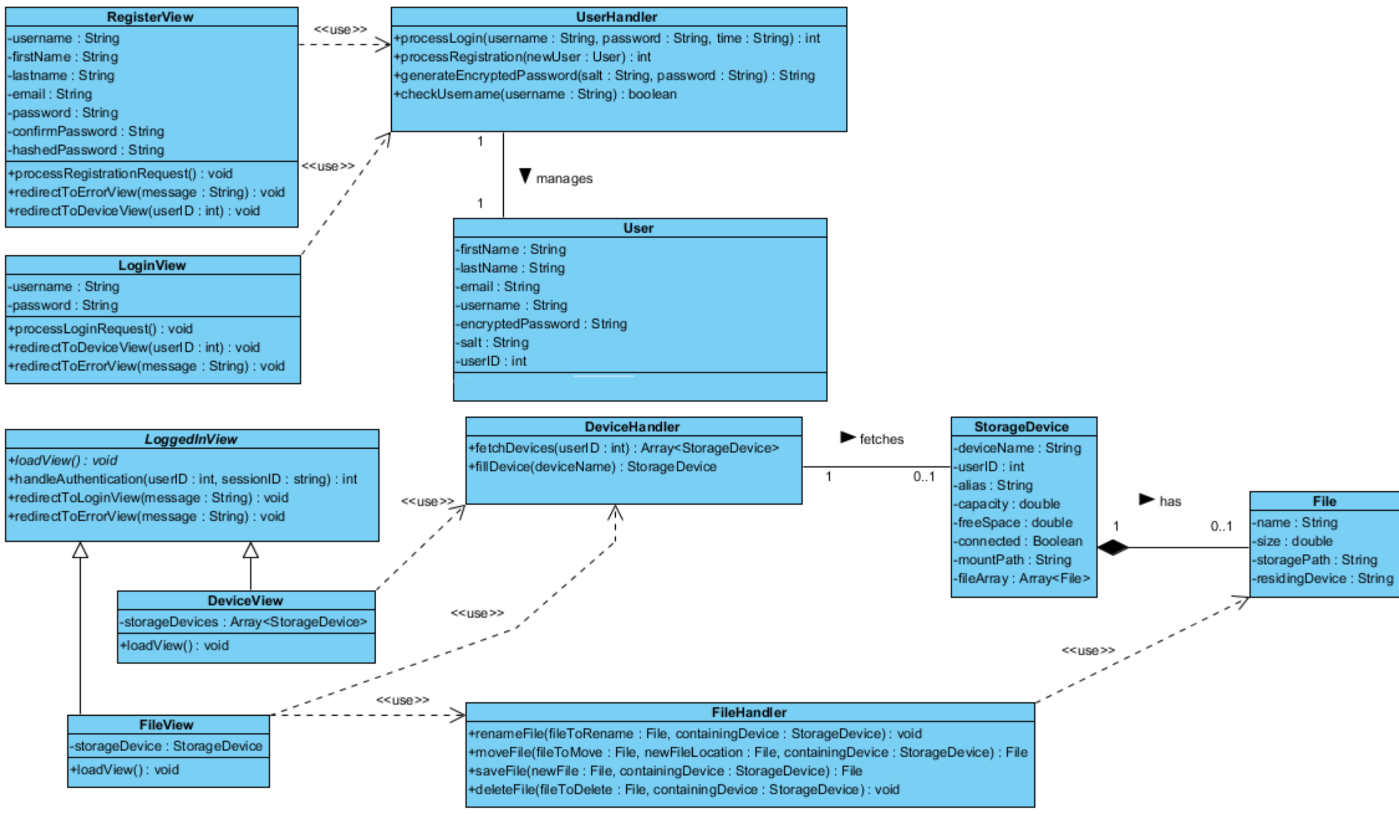
Preconditions: USB disk is mounted.

Flow:

1. Authenticated user navigates to their homepage.
2. User choses a disk in their storage array.
3. User selects unmount button.
4. Application unmounts the disk from the file system.

Full Class Diagram

The class diagram below models the classes I have envisaged for the first iteration of my project. This diagram will most likely change over time as new features are included and current ones improved. The diagram as it stands caters for the current Use Case diagram. Below this diagram are my class descriptions.


RegisterView
-username : String
-firstName : String
-lastname : String
-email : String
-password : String
-confirmPassword : String
-hashedPassword : String
+processRegistrationRequest() : void
+redirectToErrorView(message : String) : void
+redirectToDeviceView(userID : int) : void
<<use>>
UserHandler
+processLogin(username : String, password : String, time : String) : int
+processRegistration(newUser : User) : int
+generateEncryptedPassword(salt : String, password : String) : String
+checkUsername(username : String) : boolean
<<use>>
1
manages
1
LoginView
-username : String
-password : String
+processLoginRequest() : void
+redirectToDeviceView(userID : int) : void
+redirectToErrorView(message : String) : void
User
-firstName : String
-lastName : String
-email : String
-username : String
-encryptedPassword : String
-salt : String
-userID : int
LoggedInView
+loadView() : void
+handleAuthentication(userID : int, sessionID : string) : int
+redirectToLoginView(message : String) : void
+redirectToErrorView(message : String) : void
<<use>>
DeviceHandler
+fetchDevices(userID : int) : Array<StorageDevice>
+fillDevice(deviceName) : StorageDevice
fetches
1
0..1
StorageDevice
-deviceName : String
-userID : int
-alias : String
-capacity : double
-freeSpace : double
-connected : Boolean
-mountPath : String
-fileArray : Array<File>
has
1
0..1
File
-name : String
-size : double
-storagePath : String
-residingDevice : String
DeviceView
-storageDevices : Array<StorageDevice>
+loadView() : void
<<use>>
<<use>>
<<use>>
FileView
-storageDevice : StorageDevice
+loadView() : void
FileHandler
+renameFile(fileToRename : File, containingDevice : StorageDevice) : void
+moveFile(fileToMove : File, newFileLocation : File, containingDevice : StorageDevice) : File
+saveFile(newFile : File, containingDevice : StorageDevice) : File
+deleteFile(fileToDelete : File, containingDevice : StorageDevice) : void

User Class

The User class is essentially a model. The User class holds the information regarding the person who uses the application. When a user signs in this object is created by the UserHandler class. The User class will hold these fields:

- firstName : String, Holds the first name of the user.
- lastName : String, Holds the last name of the user.
- email : String, Holds the email address of the user.
- encryptedPassword : String, Holds the user's hashed and salted password.
- salt : String, The salt of the user's password.
- userID : int, Holds the user's unique ID number generated at registration.

UserHandler Class

The UserHandler class is used in a similar fashion to a controller. It contains only methods that are used to manipulate or create User objects. It also handles the logging and registration of a new user.

- +processLogin(username : String, password : String, time : Time) : int, This function is called when the LoginView wants to login in an incoming user. It takes the parameters listed to query the database and check if the user is genuine. The time is taken for logging purposes. The return type of integer is used as this method will return the userID back to a LoginView object.

- +processRegistration(newUser : User) : int, This function is called from a RegisterView object. This function takes a User object as a parameter. This newUser object is created by the RegisterView inside of its processRegistrationRequest() function. This method returns an int variable which is the userID of this newly created user. This new user is also added to the database from here.

- +fetchUser(userID : int) : User, This function is called by the abstract function *loadView()* inside the AssociatedDeviceView, AllDeviceView and FileView classes. This function is used to get a User object by searching the database by the userID, which is the only parameter of this method. This method exists so that only the userID is juggled between views instead of the actual User object which is returned by this object.

- +generateEncryptedPassword(salt : String, password : String) : String, This function is used by the RegisterView class. It is called during the processRegistrationRequest() function of the RegisterView class. It returns a String value which is the salted and hashed password for the newUser object that will be used by the RegisterView class when it calls the processRegistration(newUser : User) function above.

- +checkUsername(username : String) : Boolean, This function is used by the RegisterView class also. It is used when the incoming user tries to register. The function checks the database to see if the entered username is already in existence. If it is, it will return TRUE, thus that username is taken and then RegisterView class will call the redirectToErrorView function displaying the an error illustrating this username conflict.

LoginView Class

This class is used when an unauthenticated user navigates to the application. The class is used for creating calls to the UserHandler class, passing parameters that are required to login the incoming user using the processLogin() function of the UserHandler class. This class checks if a POST message was sent to it, if it was it will call the processLoginRequest() function. This POST is made by the form this page contains for getting the incoming user's credentials.

- username : String, This just holds the entered username from the pages main login form.
- password : String, This holds the entered password of the incoming user from the login form displayed by this view.

Functions:

- -processLoginRequest() : void, This function is called when a POST message is sent to the LoginView. All this method will then do is call the processLogin(username : String, password : String, time : String) function of the UserHandler class to check if the username and password entered are correct.

- -redirectToAllDeviceView(userID : int) : void, This function is called when the processLogin(username : String, password : String, time : String) function of the UserHandler class returns the integer value, which is used as the UserID parameter of this method. The userID parameter is used by this method when it redirects to the AllDeviceView page. Inside this redirect will be a POST message containing the userID parameter. This is then used by the AllDeviceView class's *loadView()* abstract function which requires the userID to recreate the User object. Understanding this is aided by the included sequence diagrams further on.

- -redirectToErrorView(message : String) : void, This function is called when the user enters an incorrect username or password. It is called if the call on processLogin() in the UserHandler class is not successful. It contains a message that is populated with a reason for the redirection.

RegisterView

This class holds the fields that are necessary to create a new User object. This page is used for incoming new users.

- username : String, This just holds the entered username from the pages main login form.
- firstName : String, Holds the first name of the user.
- lastName : String, Holds the last name of the user.
- email : String, Holds the email address of the user.
- password : String, This holds the entered password of the incoming user from the login form displayed by this view.
- confirmPassword : String, This holds the entered password of the incoming user from the login form displayed by this view.
- encryptedPassword : String, Holds the user's hashed and salted password.

Functions:

- -processRegistrationRequest() : void, This function is called when a POST message is sent to the RegisterView class. This post is generated when the user clicks "Register" on the registration form. If there is a POST message present it will generate a newUser object. This object is then sent as a parameter to the processRegistration(newUser : User) function of the UserHandler.

- +redirectToErrorView(message : String) : void, This function is called if the processRegistrationRequest() function encounters and issue registering the new user, i.e. the username is already taken. It accepts an error message string as a parameter.

- +redirectToAllDeviceView(userID : int) : void, This function is called when the processRegistrationRequest() is returned a userID int by the processRegistration() function of the UserHandler.

StorageDevice Class

This class is used to create storageDevice objects. These object's fields are populated by the fetchDevices(userID : Long) function of the DeviceHandler class.

- deviceName : String, This holds the Linux system name of the device.
- userID : int, This is the userID of the associated user who can use this device.
- alias : String, This is a user readable name for the device that is set by the associated user.
- capacity : double, This is the device's storage capacity.
- freeSpace : double, This is the device's remaining free space.
- connected : Boolean, This field is toggled when the device is connected or otherwise.
- mountPath : String, This is the device's mount path within the Linux filesystem.
- fileArray : Array<File>, This is an array of File objects. This array is populated by the fillDevice(deviceName) function of the DeviceHandler class.

DeviceHandler Class

This class performs operations that are used to create storageDevice and file objects.

Functions:

- +fetchDevices(userID : int) : Array<StorageDevice>, This function is called by the AssociatedDeviceView to get all the devices the userID passed is associated with from the database.

- +fillDevice(deviceName) : StorageDevice, This function is called by the FileView class. It takes the deviceName as a parameter and will then fill the fileArray of a storageDevice object. This method only operates on storageDevice objects in the database which ARE connected to the BeagleBone Black.

File Class

The File class is an object created by the fillDevice() function of the deviceHandler. It is only called by the fillDevice() function when the user accesses the FileView. This object is only created when the calling user is associated with the storageDevice it resides on.

- name : String, The name of the file.
- storagePath : String, The path of the file.
- Size : double, The size of the file.
- residingDevice : String, The device that the File belongs to.

FileHandler Class

This class is used for performing operations on File objects.

Functions:

- +renameFile(fileToRename : File, containingDevice : StorageDevice) : File, This function is used for renaming a file.

- +moveFile(fileToMove : File, newFileLocation : File, containingDevice : StorageDevice) : File, This function is used for moving a file. It takes the containingDevice object so it knows where the file is going, the file may be moved to another device this way.

- +saveFile(newFile : File, containingDevice : StorageDevice) : File, This function is used when the FileView calls it when a user starts a file upload. It takes the newFile object and the containingDevice parameter so it can place the new file into the right location and make a new file object for it.

- +deleteFile(fileToDelete : File, containingDevice : StorageDevice) : void, This function is used for deleting a file from a storage device. It will delete their corresponding objects also.

LoggedInView

This is an abstract class. It is implemented in order to remove repeating code from the other views that require a user to be logged in before proceeding.

Functions:

- loadView() : void, This is the main method that is called by all the classes that extend this class. It will assign the userID from the POST request that was sent along with the request for this view.
- handleAuthentication() : void, This function compares checks that the userID is present in the database. If it is, and if there is a corresponding user that is marked as logged in, then a user object is created using the fetchUser() function. If that user is not logged in then the user is redirected to the error view page.
- +redirectToLoginView(message : String) : void, This method is called if fetchUser() finds that the user is not acutally logged in.
- +redirectToErrorView(message : String) : void, This method is called if fetchUser() finds that the userID is not present at all in the database.

DeviceView Class

This view class is used to produce a function that will show all of the associated device's a user has. . It does not show connected, unallocated devices. This class extends the abstract class *LoggedInView.*

- storageDevices : Array<StorageDevice>, This array holds StorageDevice objects. This array is populated by the call to *loadView* , the abstract method of the abstract class.
- loadView() : This function is explained in LoggedInView below.

FileView Class

This view class is used to produce a list of all files on a storage device. This class extends the abstract class *LoggedInView.*

- storageDevice : StorageDevice, This is a StorageDevice object. This is needed to call the fillDevice(deviceName) function of the DeviceHandler class.

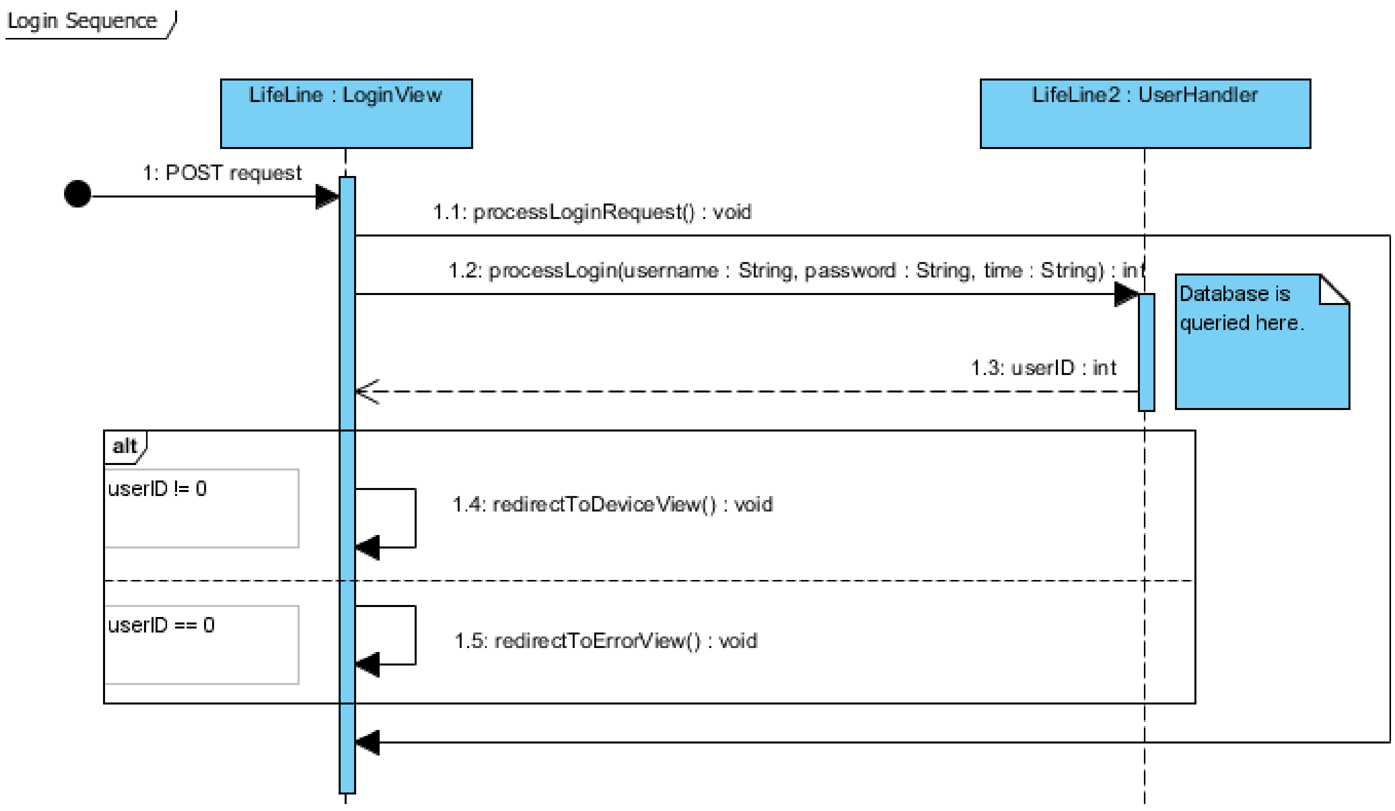
Functions:

- loadView() : This function is explained in LoggedInView below.

Sequence Diagrams

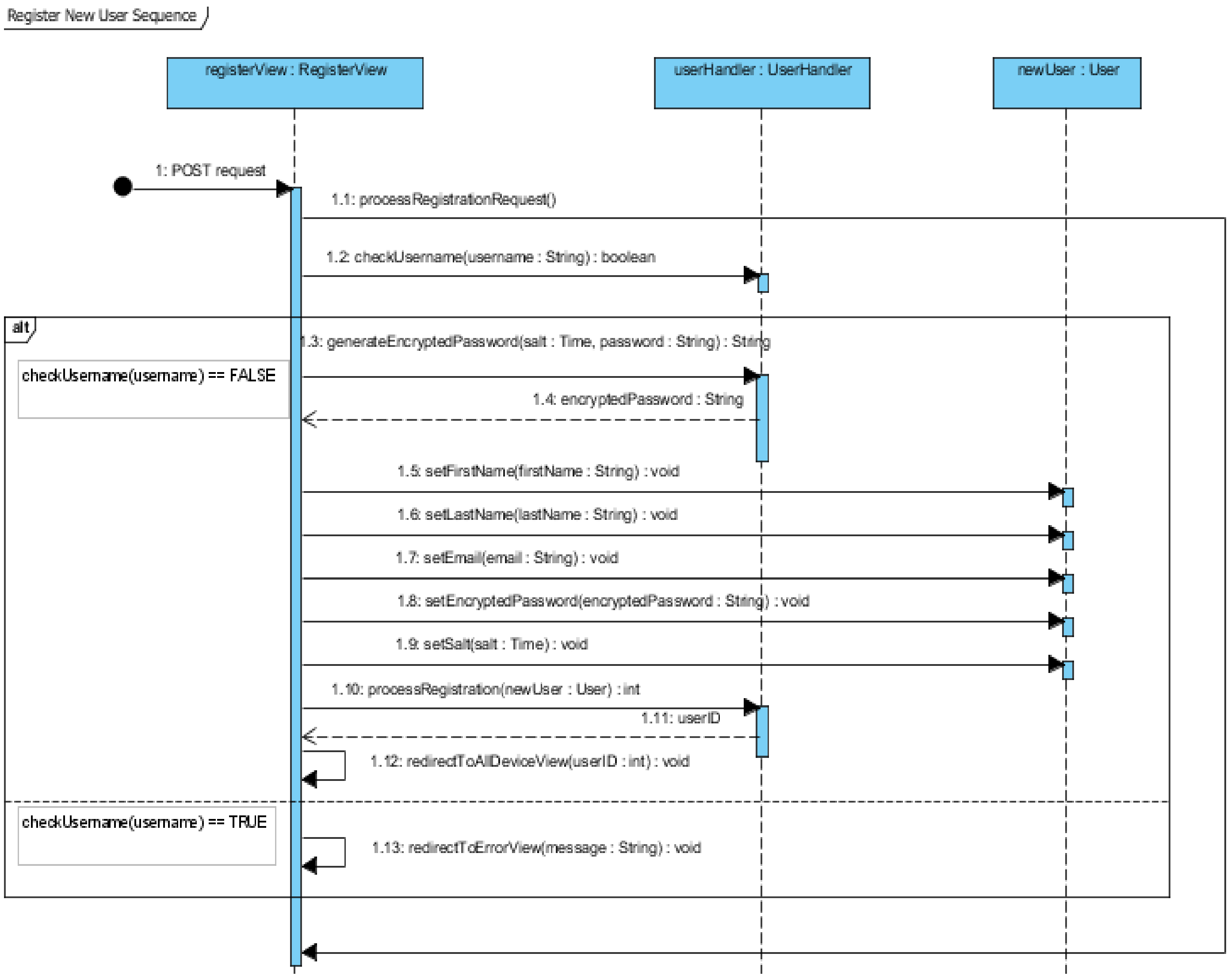
Login Sequence

This diagram depicts the sequence of functions that execute when an unauthenticated user attempts to login to the application.


Login Sequence
LifeLine : LoginView
LifeLine2 : UserHandler
1: POST request
1.1: processLoginRequest() : void
1.2: processLogin(username : String, password : String, time : String) : int
Database is queried here.
1.3: userID : int
alt
userID != 0
1.4: redirectToDeviceView() : void
userID == 0
1.5: redirectToErrorView() : void

Registration Sequence

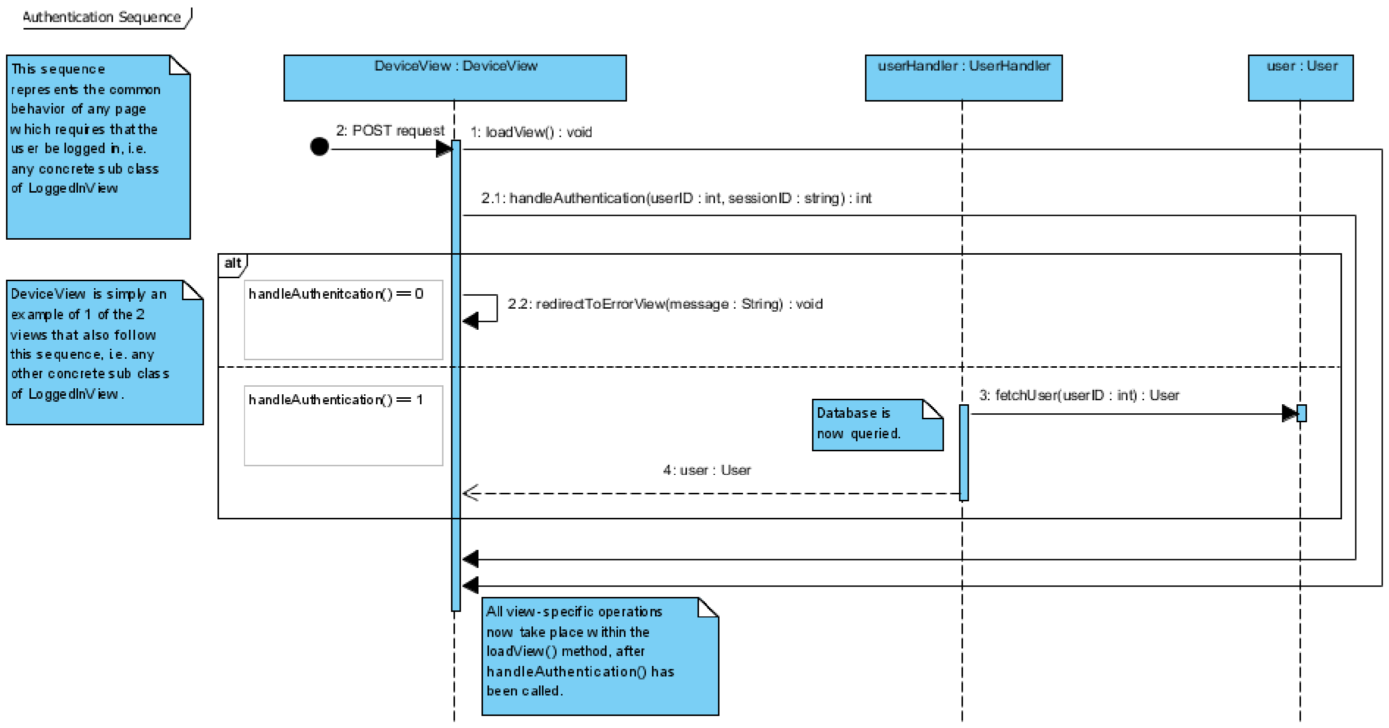
This diagram shows the sequence of functions that execute to register a new user.



Register New User Sequence
registerView : RegisterView
userHandler : UserHandler
newUser : User
1: POST request
1.1: processRegistrationRequest()
1.2: checkUsername(username : String) : boolean
alt
checkUsername(username) == FALSE
1.3: generateEncryptedPassword(salt : Time, password : String) : String
1.4: encryptedPassword : String
1.5: setFirstName(firstName : String) : void
1.6: setLastName(lastName : String) : void
1.7: setEmail(email : String) : void
1.8: setEncryptedPassword(encryptedPassword : String) : void
1.9: setSalt(salt : Time) : void
1.10: processRegistration(newUser : User) : int
1.11: userID
1.12: redirectToAllDeviceView(userID : int) : void
checkUsername(username) == TRUE
1.13: redirectToErrorView(message : String) : void

Authentication Sequence

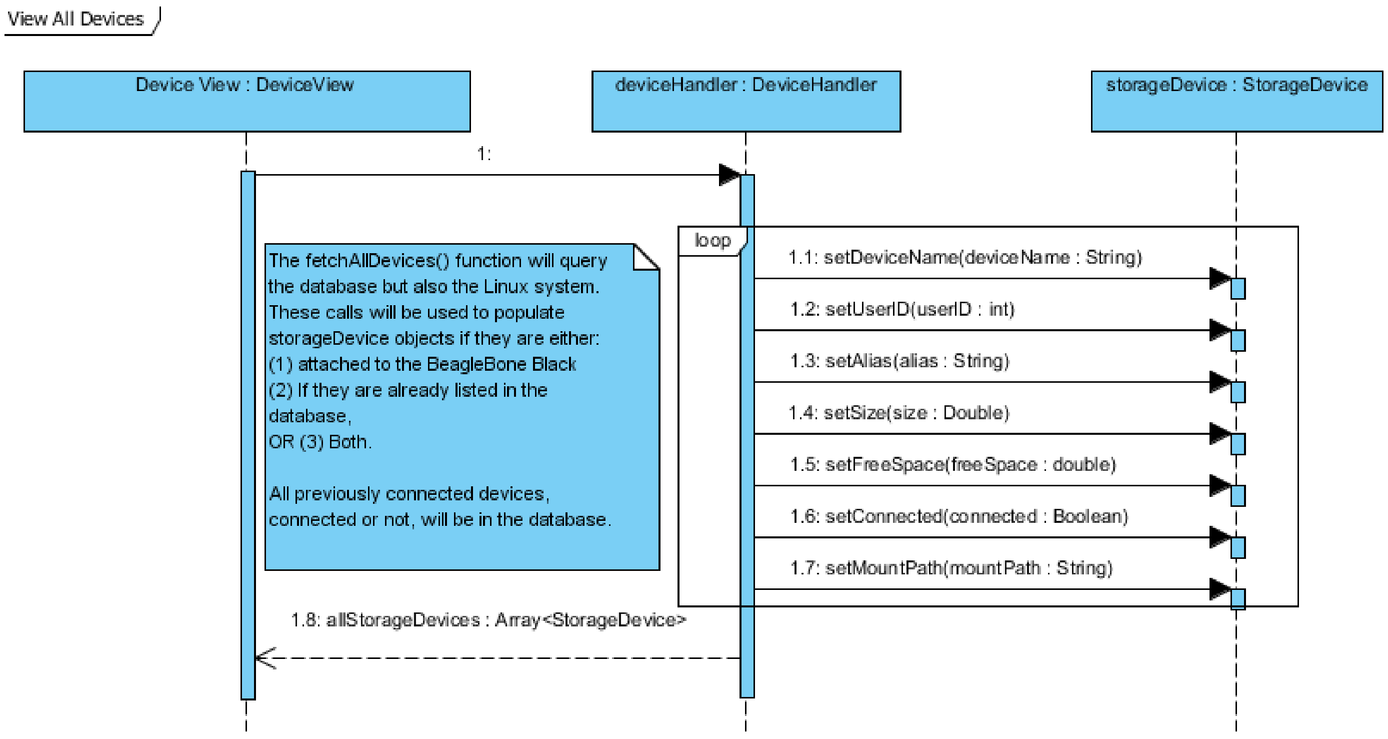
This sequence diagram depicts the authentication sequence performed by the abstract class LoggedInView which is used by the DeviceView and FileView class to determine the authenticity of the user before performing the rest of the class's tasks.



Authentication Sequence
This sequence represents the common behavior of any page which requires that the user be logged in, i.e. any concrete sub class of LoggedInView
DeviceView is simply an example of 1 of the 2 views that also follow this sequence, i.e. any other concrete sub class of LoggedInView.
DeviceView : DeviceView
userHandler : UserHandler
user : User
2: POST request
1: loadView() : void
2.1: handleAuthentication(userID : int, sessionID : string) : int
alt
handleAuthenitcation() == 0
2.2: redirectToErrorView(message : String) : void
handleAuthentication() == 1
3: fetchUser(userID : int) : User
Database is now queried.
4: user : User
All view-specific operations now take place within the loadView() method, after handleAuthentication() has been called.

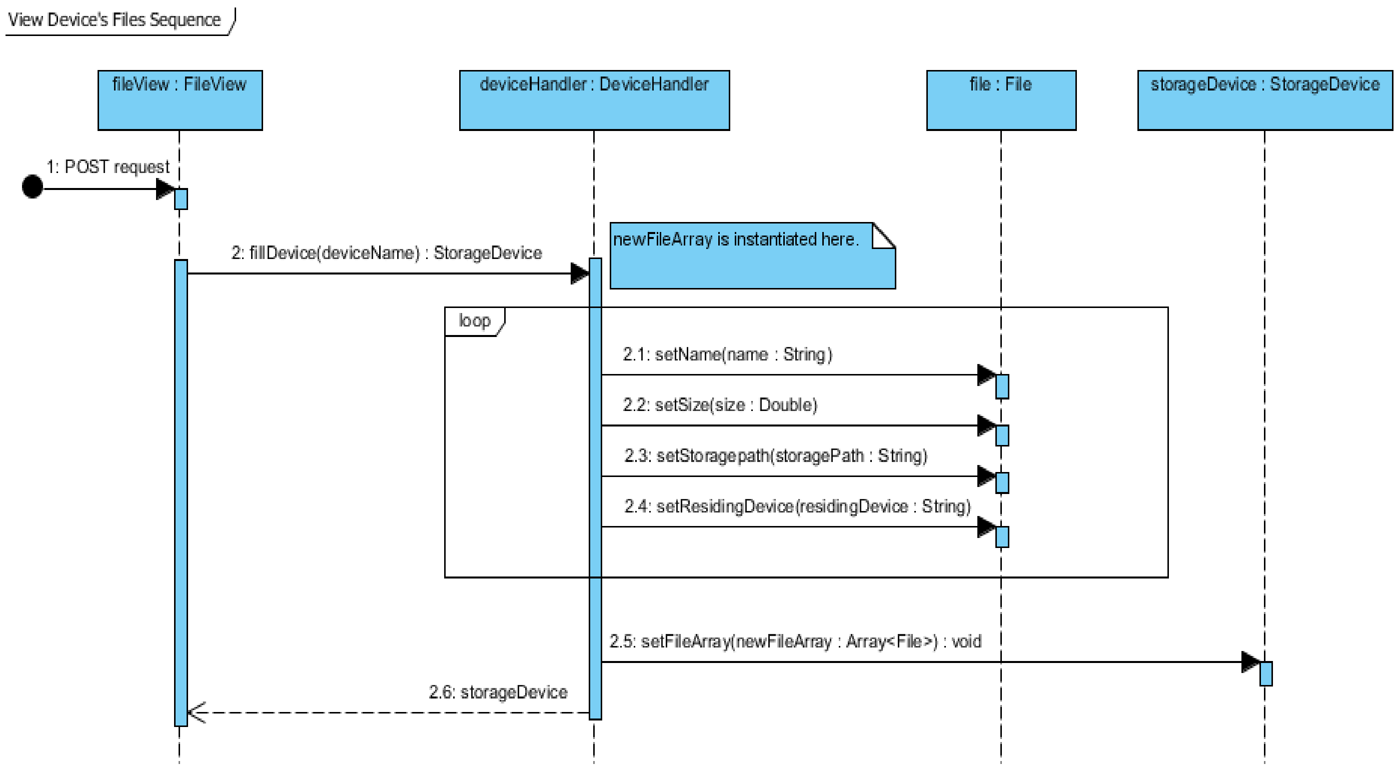
View Devices Sequence

This sequence shows the sequence of functions performed by the DeviceView class.


View All Devices
Device View : DeviceView
deviceHandler : DeviceHandler
storageDevice : StorageDevice
1:
loop
The fetchAllDevices() function will query the database but also the Linux system. These calls will be used to populate storageDevice objects if they are either: (1) attached to the BeagleBone Black (2) If they are already listed in the database, OR (3) Both. All previously connected devices, connected or not, will be in the database.
1.1: setDeviceName(deviceName : String)
1.2: setUserID(userID : int)
1.3: setAlias(alias : String)
1.4: setSize(size : Double)
1.5: setFreeSpace(freeSpace : double)
1.6: setConnected(connected : Boolean)
1.7: setMountPath(mountPath : String)
1.8: allStorageDevices : Array<StorageDevice>

View Device’s Files Sequence

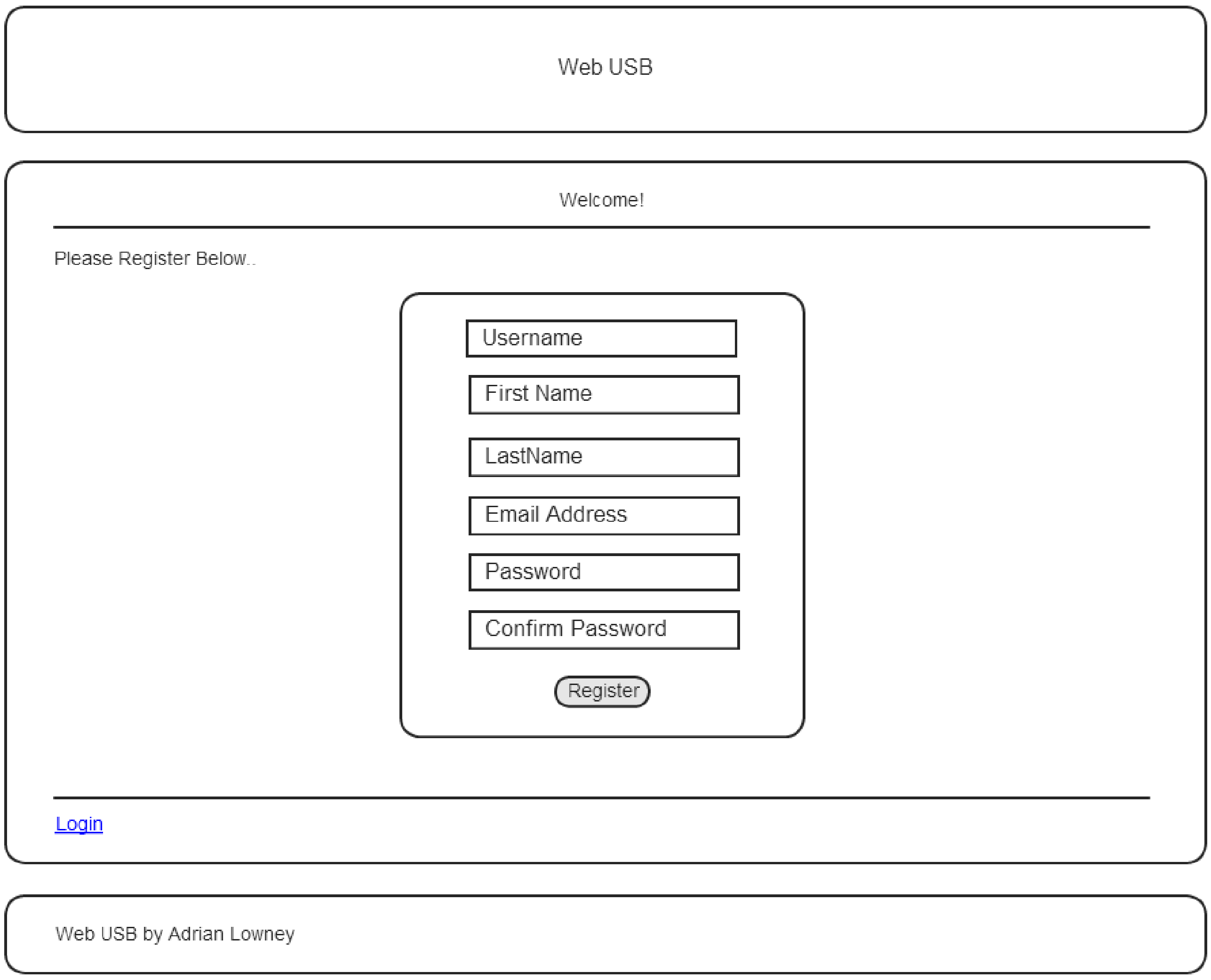
The sequence depicts the order of events of the FileView class.


View Device's Files Sequence
fileView : FileView
deviceHandler : DeviceHandler
file : File
storageDevice : StorageDevice
1: POST request
2: fillDevice(deviceName) : StorageDevice
newFileArray is instantiated here.
loop
2.1: setName(name : String)
2.2: setSize(size : Double)
2.3: setStoragepath(storagePath : String)
2.4: setResidingDevice(residingDevice : String)
2.5: setFileArray(newFileArray : Array<File>) : void
2.6: storageDevice

Interface Design

Below is a collection of wireframes that I am using to depict the design of my application's interface that the user will interact with. I intend to make the user interface as easy to use as possible and to make in intuitive to use. The wireframes below are an overall guideline that I will use when implementing each of the application's views but these are subject to change as development goes on.

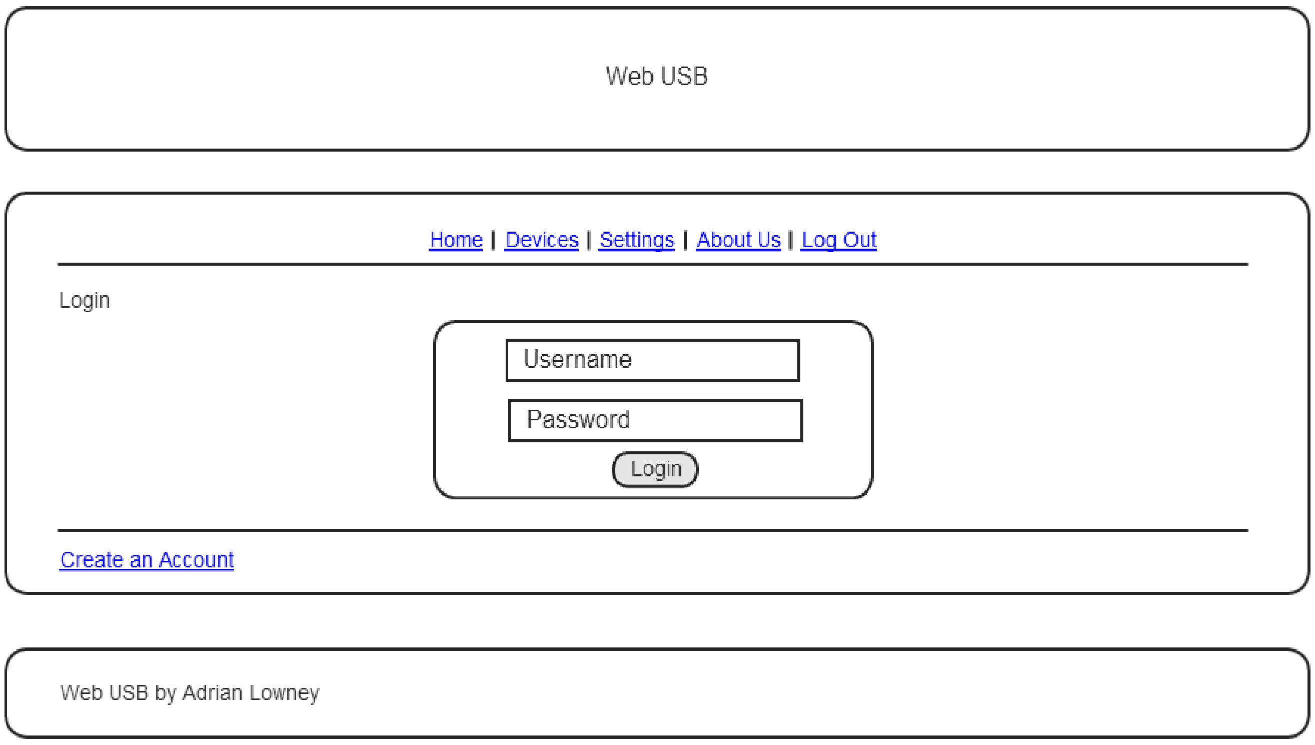
Registration Page



<u>Description</u>

This page allows a new user to create an account on the application. If all fields are filled in satisfactorily the application will redirect to login page from which the user will be prompted to enter their login details.

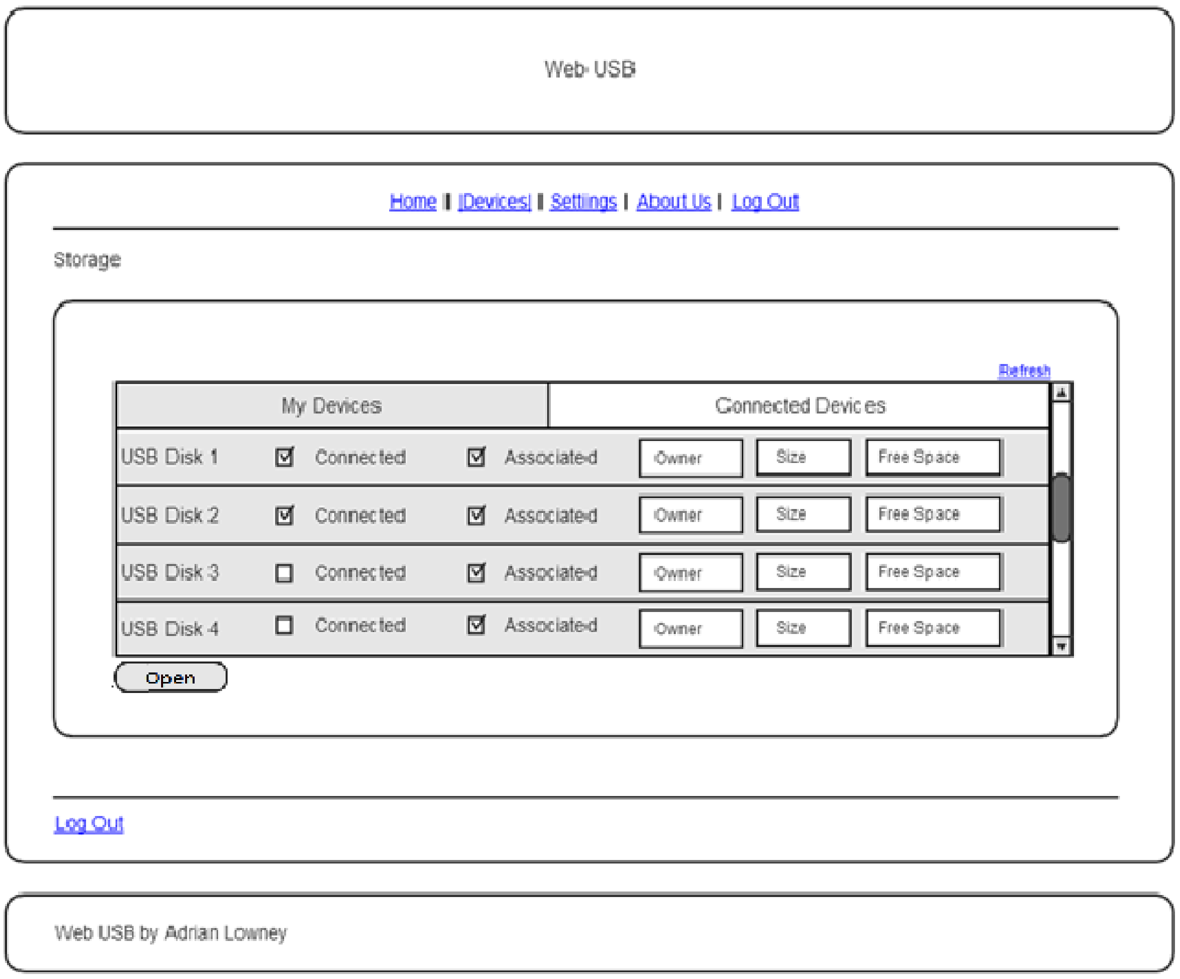
Login Page



Description

This page allows the user to login. The username and password fields are text fields that the user must fill in respectively. If the user does not input a correct username and password, this page will reload once login is selected. It will then display an error detailing that the wrong username and password combination was entered or redirect to the newly authenticated user's homepage.

View Devices Page – My Devices



Web USB

Home | [Devices] | Settings | About Us | Log Out

Storage

Refresh

My Devices					Connected Devices	
USB Disk 1	☑ Connected	☑ Associated	Owner	Size	Free Space	
USB Disk 2	☑ Connected	☑ Associated	Owner	Size	Free Space	
USB Disk 3	☐ Connected	☑ Associated	Owner	Size	Free Space	
USB Disk 4	☐ Connected	☑ Associated	Owner	Size	Free Space	

Open

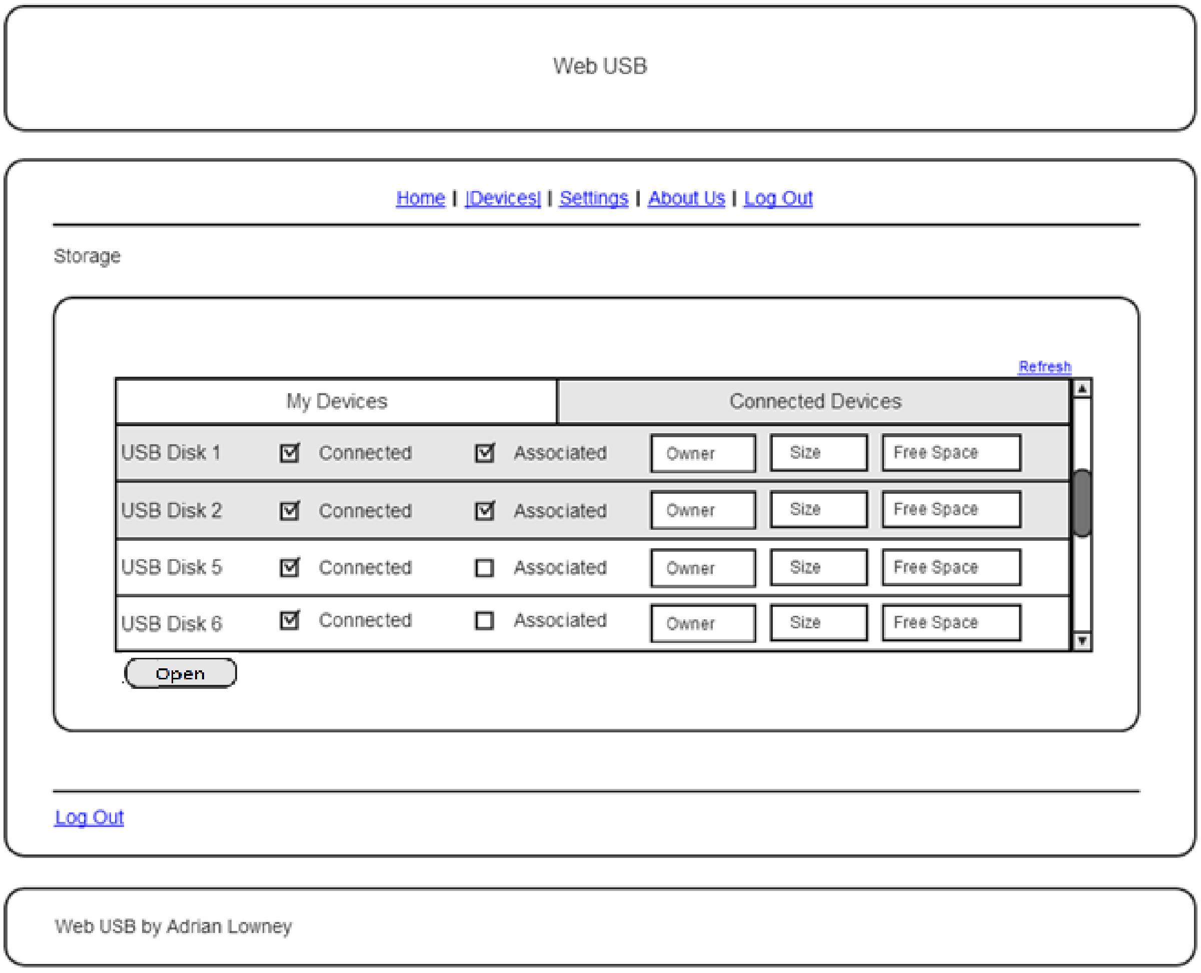
Log Out

Web USB by Adrian Lowney

Description

This page allows an authenticated user to manage their devices. The table in the centre of the page displays the list of devices the user is associated with. This can be currently attached and associated devices or historical non-attached devices which are associated with the user. The name of the device is configurable from this page and the 'Connected' and 'Associated' tick boxes can be toggled. Toggling the 'Connected' box will unmount a connected device, by default a disk is automatically toggled as 'Connected' by the application when attached to the BeagleBone Black.

View Devices Page – Connected Devices



Web USB

Home | |Devices| | Settings | About Us | Log Out

Storage

Refresh

My Devices						
USB Disk 1	☑ Connected	☑ Associated	Owner	Size	Free Space	
USB Disk 2	☑ Connected	☑ Associated	Owner	Size	Free Space	
USB Disk 5	☑ Connected	☐ Associated	Owner	Size	Free Space	
USB Disk 6	☑ Connected	☐ Associated	Owner	Size	Free Space	

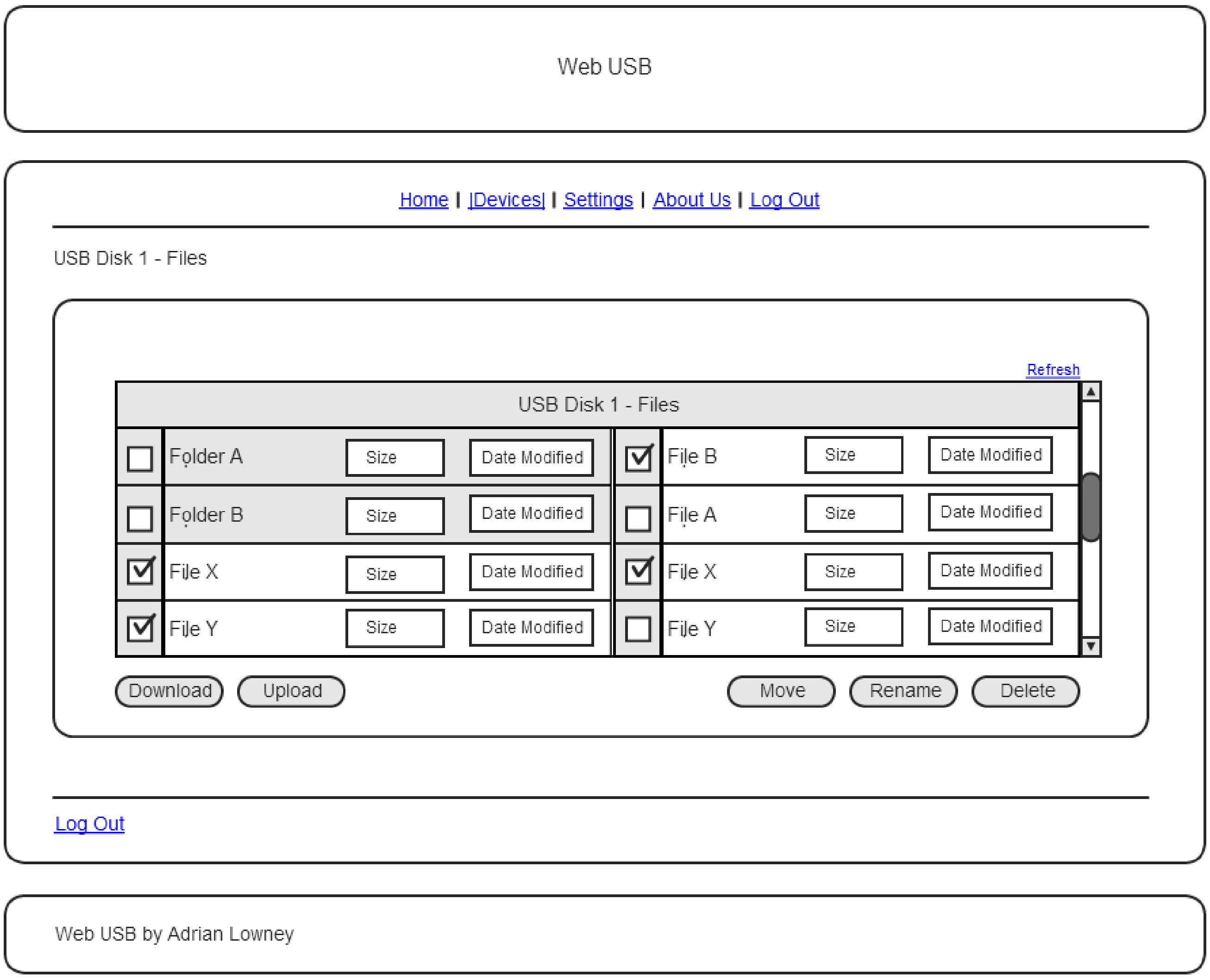
Log Out

Web USB by Adrian Lowney

Description

This is the same page as above however it is displaying the connected devices rather than the just the current user's associated devices. The connected devices list is displayed so the user can view all attached devices and associate with one that is not already associated to another user (owner). A device not associated with the current user will have the username of the user that is associated with it in the 'Owner' field.

View Files Page



Description

This page is displayed to the user when they have selected to open a disk. Folders are displayed along with the files at the current directory. Folders can be entered by left clicking their tile. Checkboxes are provided beside file and folder objects for performing actions against them. The actions available are download, upload, move, rename, delete. All of these buttons create further dialogue boxes on the page. For example, pressing ‘Rename’ when a file is selected would bring up a dialogue box prompting the user for a new file name with ‘OK’ and ‘Cancel’ buttons for executing or cancelling the rename action.

Technical Feasibility Evaluation

In order to ensure the technical feasibility of my project I have prepared a partial implementation of my class diagram. I have prepared a draft implementation of the login class which I will discuss further in this section.

The Hardware

The BeagleBone Black is the latest product from the BeagleBoard family. The device has a low-cost Cortex A8 ARM processor provided by Texas Instruments. The device is also equipped with 512MB of DDR3L RAM, 2GB of on board non-volatile memory and a single USB 2.0 port. For networking the device also has a single 100Mbit Ethernet port. The image below is that of the BeagleBone Black, for a size reference note that the footprint is no larger than a credit card.

(ELinux.org, 2013)

The device is also equipped with a USB client port that I can connect to my laptop. This allows me to communicate with the device over SSH and allows me to synchronise my application code over SFTP.

Device Software

- Ubuntu 12.04 Precise Pangolin

  The device is running a console only version of Ubuntu 12.04. This console version was picked as it offers no GUI functionality but comes preinstalled with the major packages of the normal desktop version. The lack of GUI will lower the overhead on the BeagleBone Black's CPU and will free up resources in general as they are not needed.
- Apache2

  The web server I have installed on the BeagleBone Black is the Apache2 web server. I am using this server as I am most familiar with it and have worked with it previously. Apache has been configured with SSL to allow HTTPS connections on port 443 which I intend to be used as the sole type of connection allowed to the server by clients.

Development Software

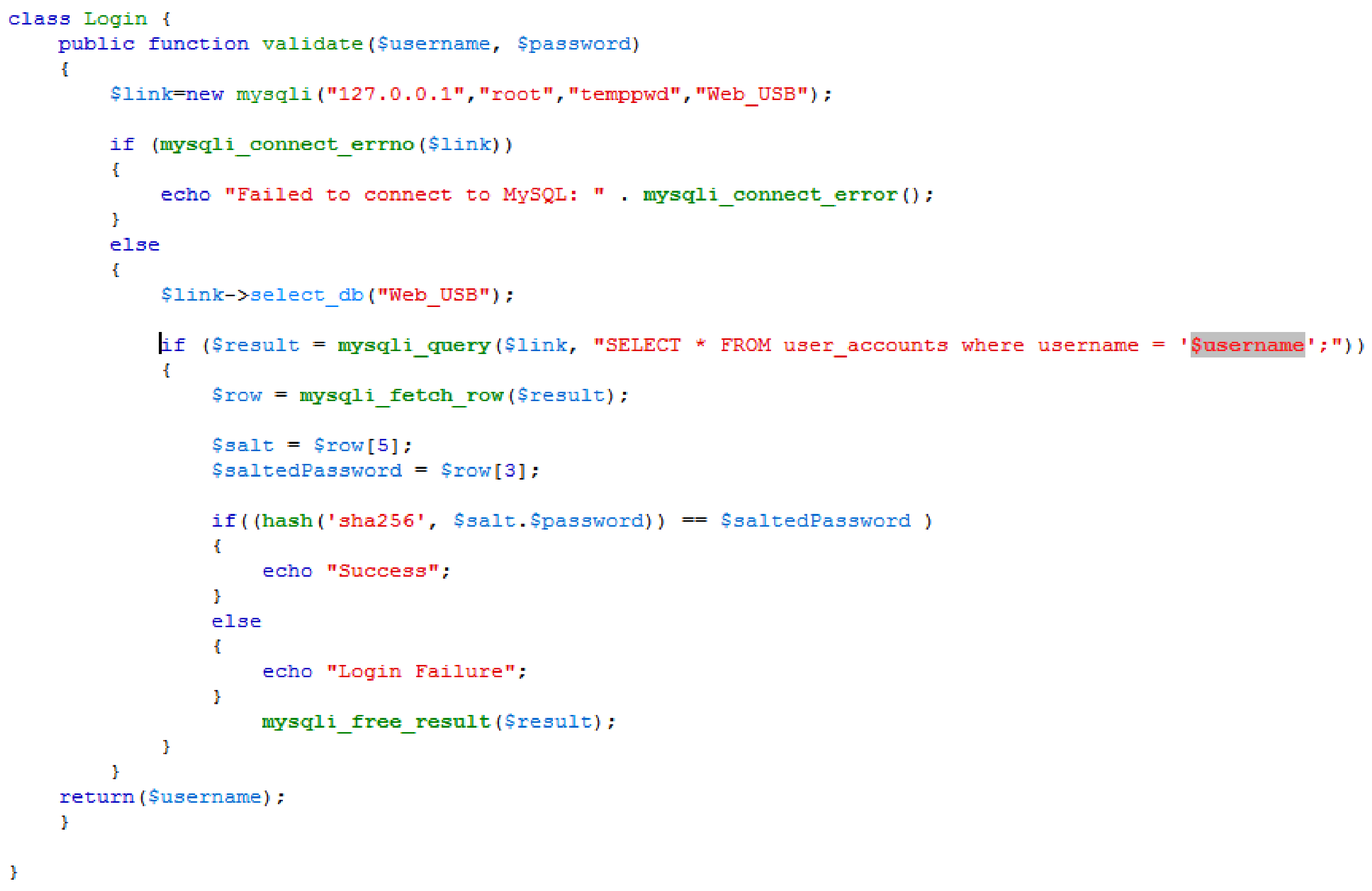
- NetBeans IDE

  NetBeans is the IDE and PHP editor that I will be using for this project. The software comes with the ability to autocomplete and highlight code sections along with built in FTP. This built in FTP allows me to develop my code locally on my machine and then synchronise the source code to the BeagleBone Black.

Partial Trial Implementation

Login Class

To test my technologies and software are working correctly together I decided to do a mock version of my Login class with a draft version of the front end. This implementation simply allows a user to login and if successful the page displays a success notification. Below is a screenshot of the PHP code.

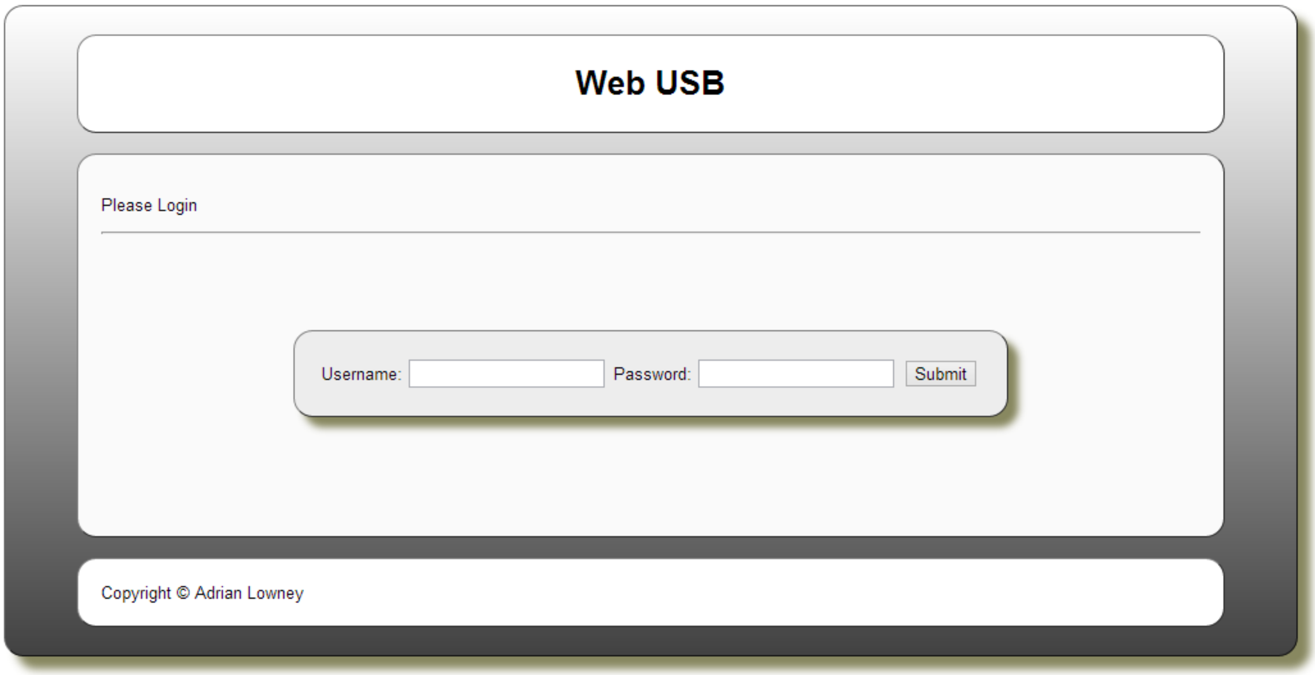


```php
class Login {
    public function validate($username, $password)
    {
        $link=new mysqli("127.0.0.1","root","temppwd","Web_USB");

        if (mysqli_connect_errno($link))
        {
            echo "Failed to connect to MySQL: " . mysqli_connect_error();
        }
        else
        {
            $link->select_db("Web_USB");

            if ($result = mysqli_query($link, "SELECT * FROM user_accounts where username = '$username';"))
            {
                $row = mysqli_fetch_row($result);

                $salt = $row[5];
                $saltedPassword = $row[3];

                if((hash('sha256', $salt.$password)) == $saltedPassword )
                {
                    echo "Success";
                }
                else
                {
                    echo "Login Failure";
                }
                    mysqli_free_result($result);
            }
        }
    return($username);
    }

}
```

The code above is executed when the login form requests a user to be validated. This is done by the Login class above. For this draft of the class the method validate($username, $password) takes two parameters as the code shows. These parameters are then checked against the database. The $username variable is used to find the row in the MySQL database containing the users information. The method then allocates each column for that username into an array $row. This array is then compared to the hash of the user's entered password with their salt from the database. If this hash matches the stored one from the database the user is permitted access.

Login Page

The image below shows the basic layout of the login page. This is simply a rough draft of how I may present the page. This was created simply to test the login class by posting the entered fields.

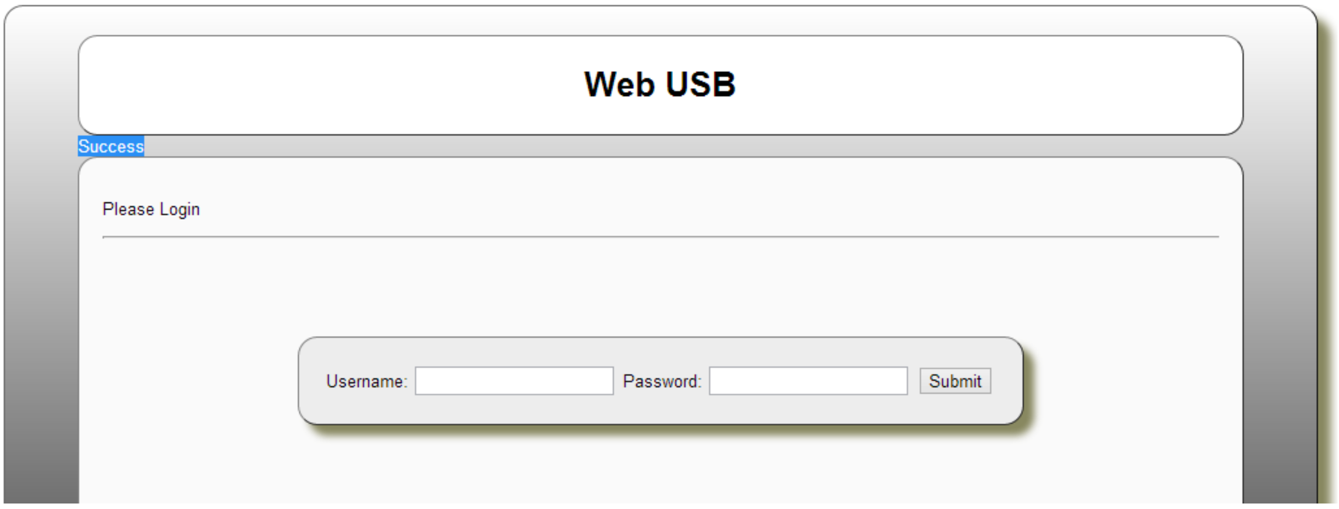

Web USB

Please Login

Username: Password: Submit

Copyright © Adrian Lowney

Though no registration page has been created yet I have inserted a test user account into a table named "user_accounts". In the image below is the result of entering the correct username and password for this test account. The "Success" text simply pops up when the correct username and password are entered. In the future this will obviously bring the user to their homepage.


Web USB

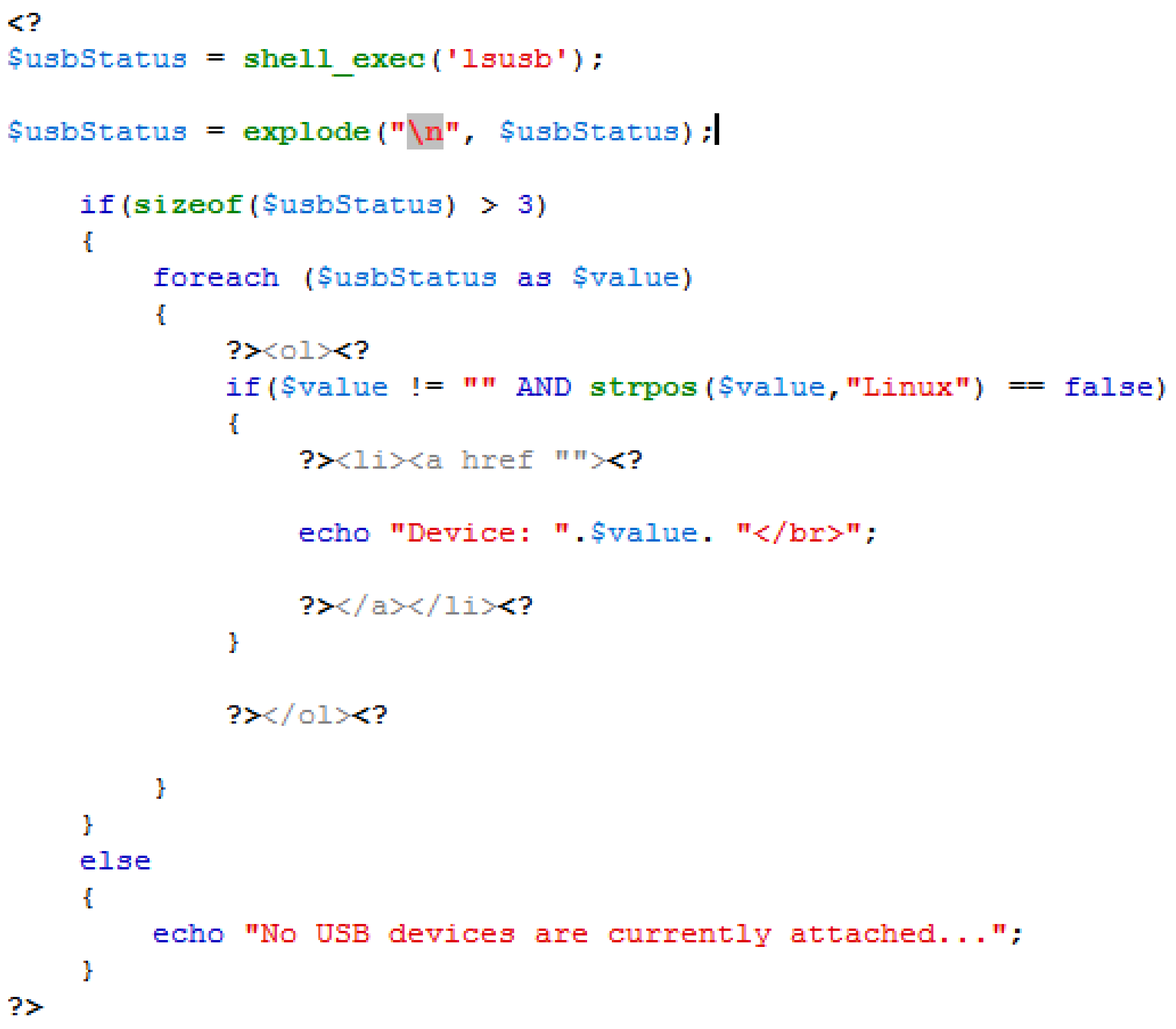
Success

Please Login

Username: Password: Submit

Using PHP to View USB Disks

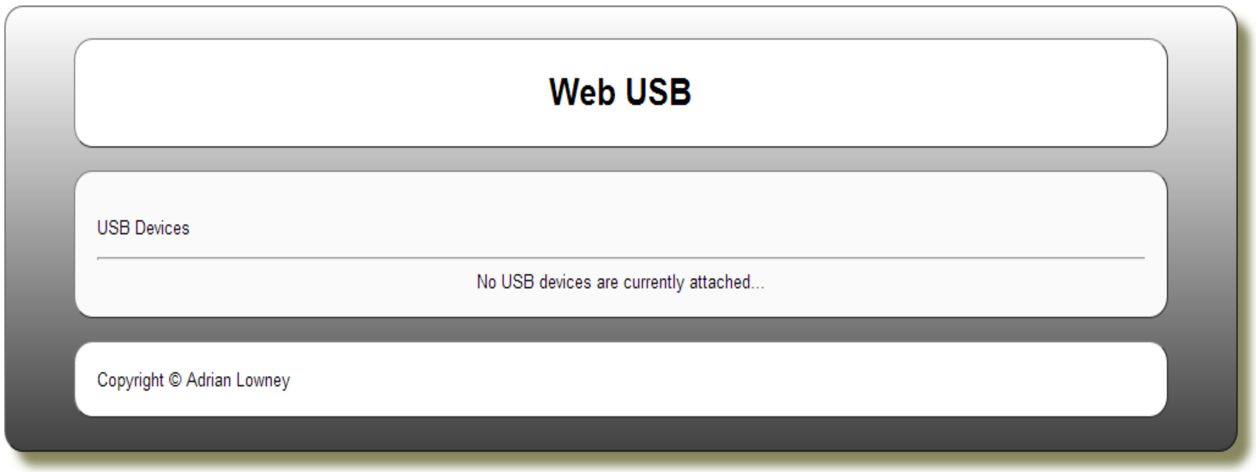
The code below is a draft snippet that simply displays the list of connected USB disk drives. This code uses a Linux system command to display the attached drives, the result of which command is sent into an array. This array is then printed to the screen as a list. The code below will not be present in the later iterations of the project as it is only used to test that PHP can collect system information correctly. A similar code segment will likely be in the later iterations of the USB class.



```
<?
$usbStatus = shell_exec('lsusb');

$usbStatus = explode("\n", $usbStatus);

    if(sizeof($usbStatus) > 3)
    {
        foreach ($usbStatus as $value)
        {
            ?><ol><?
            if($value != "" AND strpos($value,"Linux") == false)
            {
                ?><li><a href ""><?

                echo "Device: ".$value. "</br>";

                ?></a></li><?
            }

            ?></ol><?

        }
    }
    else
    {
        echo "No USB devices are currently attached...";
    }
?>
```

The code segment above simply uses the Linux system command "lsusb" to list attached USB devices. This is then read into an array and printed to the screen. The two built in USB controllers are filtered as the user cannot use these for storage.

When no USB disks are attached this piece of code essentially produces this view as seen below.

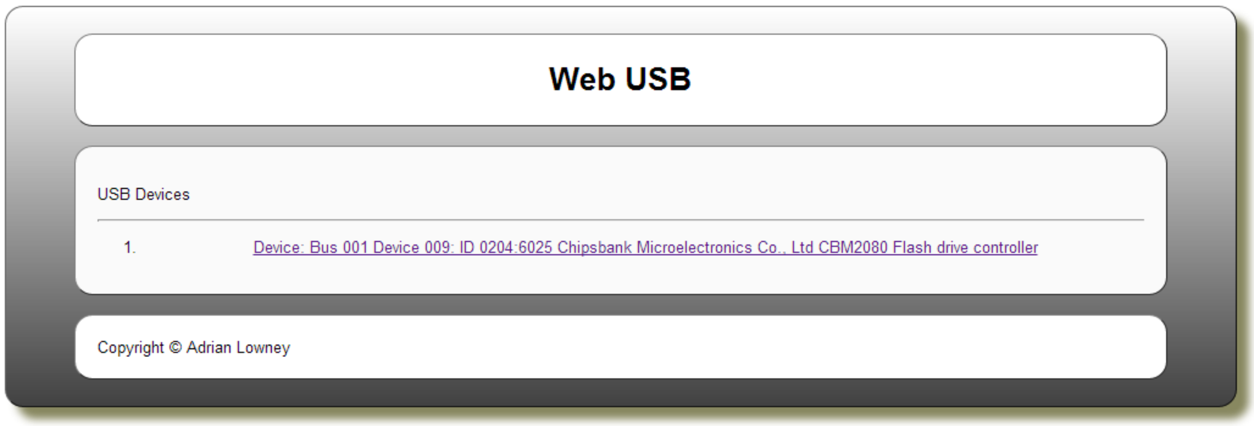

Web USB

USB Devices

No USB devices are currently attached...

Copyright © Adrian Lowney

However when a USB disk is attached to the BeagleBone black it will produce a list similar to this as below.


Web USB

USB Devices

1. Device: Bus 001 Device 009: ID 0204:6025 Chipsbank Microelectronics Co., Ltd CBM2080 Flash drive controller

Copyright © Adrian Lowney

This code segment combined with some draft HTML views show that the fundamental hardware building blocks and the use of PHP are capable of producing the necessary results.

Development

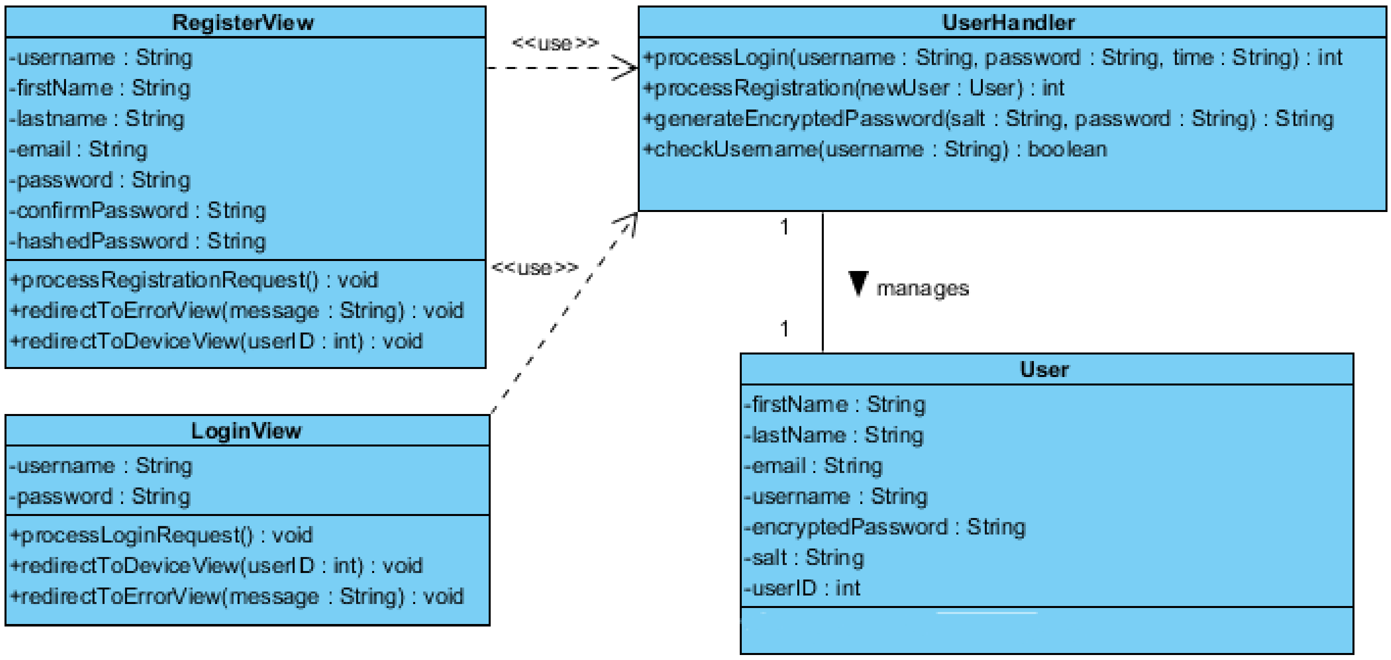
Development Schedule

I have devised a timetable of goals that will allocate each month of semester two to a group of goals. I have allotted around three to four goals per month in order to make room for other assignments but also to keep my work flow steady with the project without overwhelming myself. Certain areas of development will challenge me more such as the upload, download and move functions as these will require a relatively large amount of AJAX and JQuery compared to other sections of the project, I am less familiar with these languages and have spread much of the development in these topics over two months (February - March).

Month	Objectives
January	• Complete registration and login pages. • Begin development of user homepage. • Implement dynamic navigation bar into top of each view. • Compile monthly report for final submission
February	• Complete delete and rename functions on homepage. • Begin Development of basic move, upload and download functionality. • Compile monthly report for final submission
March	• Complete basic move, upload and download functionality. • Incorporate multiple file upload and download functionality. • Complete and test move, upload and download of files and folders. • Compile monthly report for final submission
April	• Incorporate the disk formatting functionality. • Incorporate IP address emailing functionality. • Compile monthly report for final submission
May	• Improve overall usability and styling with further JavaScript and CSS. • Test overall functionality and correct any newly found issues. • Compile monthly report and complete final report.

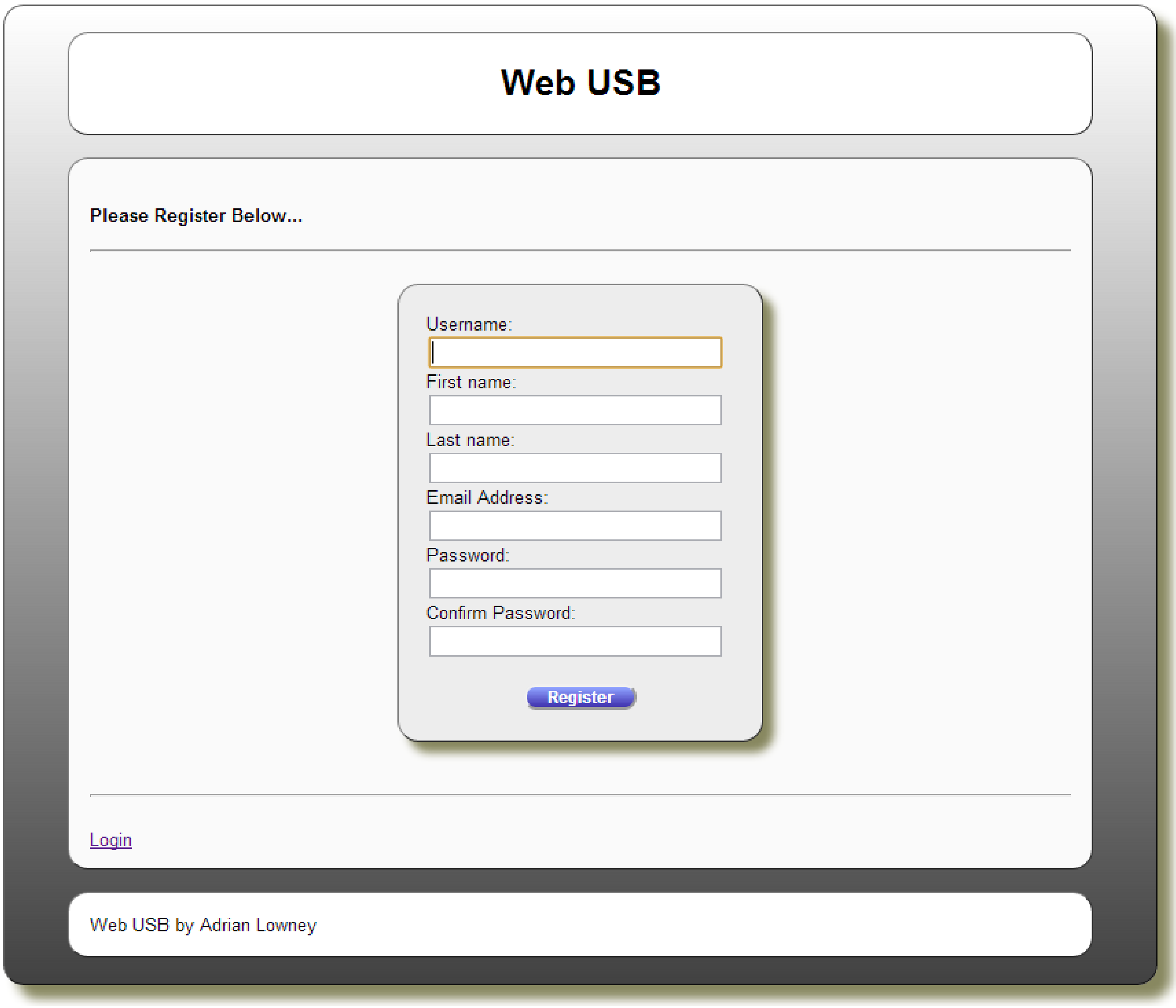
Development to Date

Development prior to January included the implementation of the registration and login functionality of the class diagram. Below is the section of the class diagram which has been implemented thus far.


RegisterView
-username : String
-firstName : String
-lastname : String
-email : String
-password : String
-confirmPassword : String
-hashedPassword : String
+processRegistrationRequest() : void
+redirectToErrorView(message : String) : void
+redirectToDeviceView(userID : int) : void
LoginView
-username : String
-password : String
+processLoginRequest() : void
+redirectToDeviceView(userID : int) : void
+redirectToErrorView(message : String) : void
<<use>>
<<use>>
UserHandler
+processLogin(username : String, password : String, time : String) : int
+processRegistration(newUser : User) : int
+generateEncryptedPassword(salt : String, password : String) : String
+checkUsername(username : String) : boolean
1
manages
1
User
-firstName : String
-lastName : String
-email : String
-username : String
-encryptedPassword : String
-salt : String
-userID : int

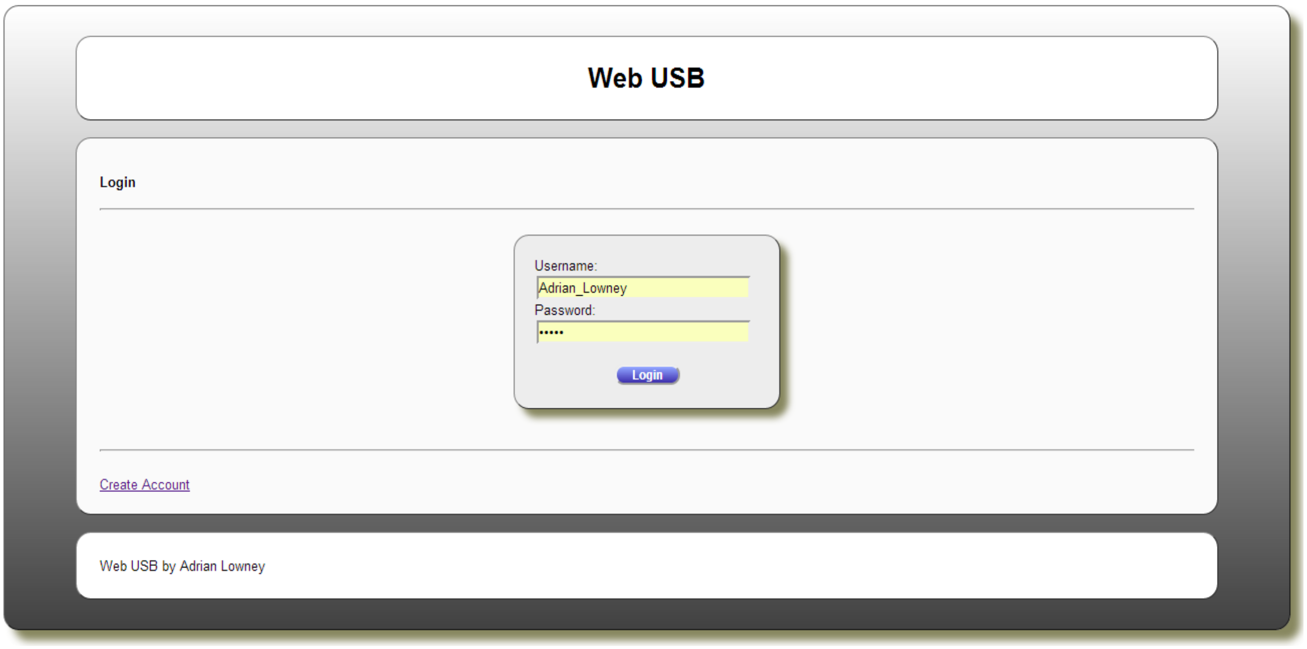
As above I have developed the registration view and login view class pages along with the two classes UserHandler and User. These alone cater for the functionality required to register and login a user. The implementation follows the class diagram exactly and the user interface (HTML and CSS) follows the layout in the wireframes bar the navigation bar which will be completed before further implementation.

RegisterView Class Implementation



The registration view above provides text fields for the unauthenticated user to complete in order to register. These fields are checked by PHP for correctness and then entered into the database if they are deemed acceptable. The code required to perform this is quite length so I have not included this in the document but it will of course be present in my final submission and upon request.

LoginView Class Implementation

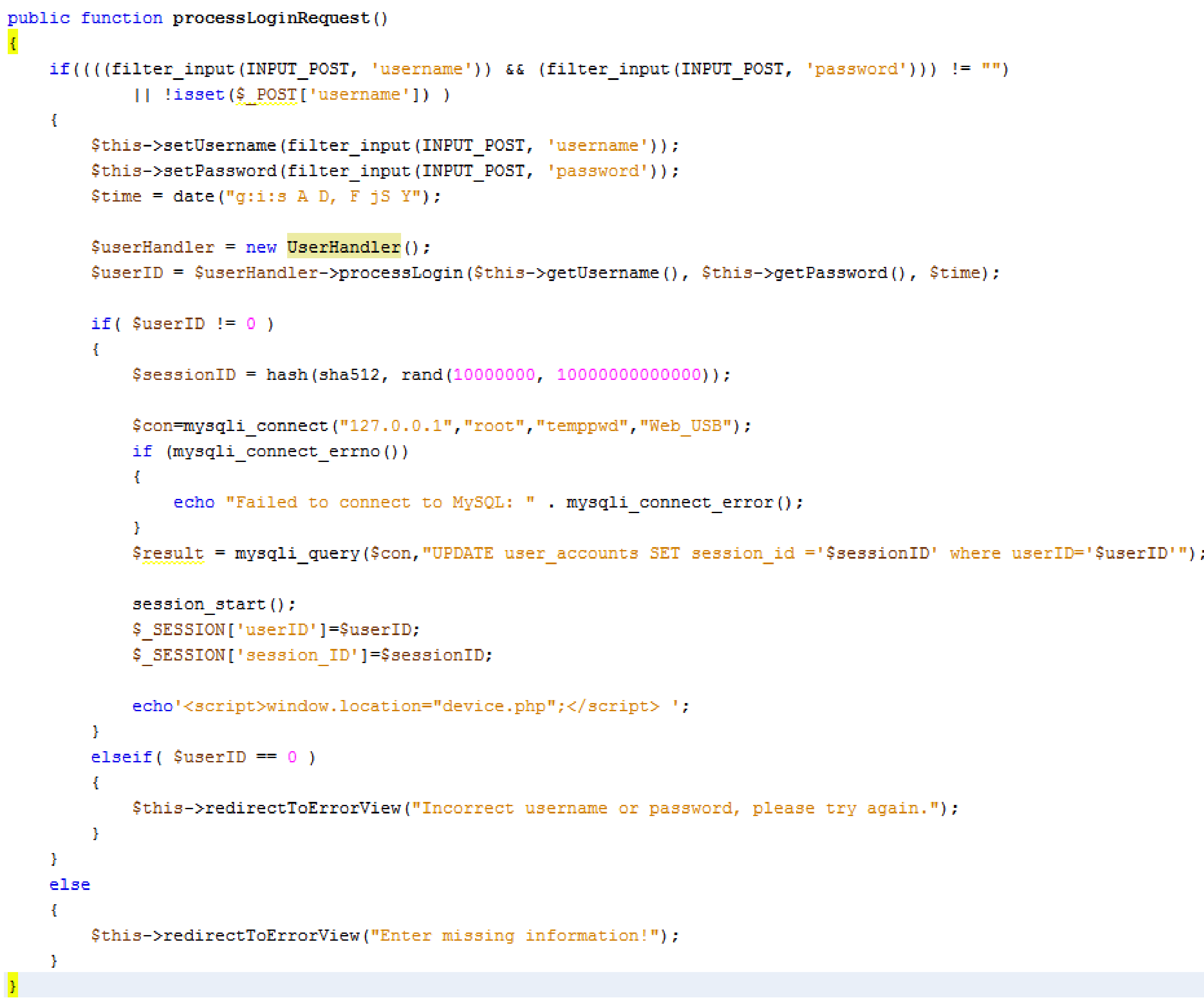

Web USB

Login

Username:
Adrian_Lowney
Password:
•••••

Web USB by Adrian Lowney

The login page above requires the unauthenticated user to enter their username and password before progressing with the application. The code implementation of the function takes the entered username, finds it in the database, creates a hash of the entered password along with the salt from the database and then compares it with the hashed password in the database. If the username exists and the password is correct then a session ID is generated and stored in the database for that user. This stops any user attempting to access the site without visiting the login page first. The session ID is updated every time the user successfully logs in. The functional implementation of this page's PHP code is provided on the following page.



```
public function processLoginRequest()
{
    if(((filter_input(INPUT_POST, 'username')) && (filter_input(INPUT_POST, 'password'))) != "")
            || !isset($_POST['username']) )
    {
        $this->setUsername(filter_input(INPUT_POST, 'username'));
        $this->setPassword(filter_input(INPUT_POST, 'password'));
        $time = date("g:i:s A D, F jS Y");

        $userHandler = new UserHandler();
        $userID = $userHandler->processLogin($this->getUsername(), $this->getPassword(), $time);

        if( $userID != 0 )
        {
            $sessionID = hash(sha512, rand(10000000, 10000000000000));

            $con=mysqli_connect("127.0.0.1","root","tempppwd","Web_USB");
            if (mysqli_connect_errno())
            {
                echo "Failed to connect to MySQL: " . mysqli_connect_error();
            }
            $result = mysqli_query($con,"UPDATE user_accounts SET session_id ='$sessionID' where userID='$userID'");

            session_start();
            $_SESSION['userID']=$userID;
            $_SESSION['session_ID']=$sessionID;

            echo'<script>window.location="device.php";</script> ';
        }
        elseif( $userID == 0 )
        {
            $this->redirectToErrorView("Incorrect username or password, please try again.");
        }
    }
    else
    {
        $this->redirectToErrorView("Enter missing information!");
    }
}
```

References

ELinux (2013). Beagleboard: BeagleBone Black. Available: http://elinux.org/Beagleboard:BeagleBoneBlack. Last accessed 10th Nov 2013.

ISTQB. (2012). What is the Agile model – advantages, disadvantages and when to use it? Available: http://istqbexamcertification.com/what-is-agile-model-advantages-disadvantages-and-when-to-use-it/. Last accessed 5th November 2013.

Kehoe, D. (2013). What is Ruby on Rails? Available: http://railsapps.github.io/what-is-ruby-rails.html. Last accessed 10th Nov 2013.

MacFarlane, N (2004). Rapid Application Development with Mozilla. New Jersey: Pearson Education. p24-25.

Taei, PJ. (2013). 10 Advantages of PHP over other Languages. Available: http://www.webnethosting.net/10-advantages-of-php-over-other-languages/. Last accessed 10th Nov 2013.