

Eth Photo

TEAM 10

# Contents

I	Software Requirement and Specifications	5
1	Introduction	6
1.1	Purpose . . . . .	6
1.2	Product Scope . . . . .	6
2	Overall Description	7
2.1	Product Perspective . . . . .	7
2.2	Product Functions . . . . .	7
2.3	Operating Environment . . . . .	8
2.4	Design and Implementation Constraints . . . . .	8
2.5	User Implementation . . . . .	8
2.6	Assumptions and Dependencies . . . . .	8
3	External Interface Requirements	9
3.1	User Interfaces . . . . .	9
3.2	Hardware Interfaces . . . . .	9
3.3	Software Interfaces . . . . .	9
3.4	Communication Interfaces . . . . .	9
3.5	Memory Constraints . . . . .	10
4	Functional Requirements	11
4.1	Add Image . . . . .	11
4.2	Get User Image . . . . .	11
4.3	Delete . . . . .	11
5	Other Non-functional requirements	12
5.1	Security and Privacy Requirements . . . . .	12
5.2	Reliability . . . . .	12

II	Installation Instructions	13
6	Installing dependencies	14
6.1	Aptitude dependencies . . . . .	14
6.2	npm dependencies . . . . .	14
6.3	source dependencies . . . . .	15
III	User Guide	16
7	Starting the Software	17
7.1	Graphical User Interface . . . . .	17
8	Main Window	19
8.1	Search bar . . . . .	19
8.1.1	Types of Search . . . . .	20
8.2	Zoom . . . . .	20
8.3	Add Photos . . . . .	21
8.3.1	How to upload . . . . .	21
8.4	My Photos . . . . .	22
IV	Algorithms	23
9	Front end	24
9.1	Initiate Geo-Location . . . . .	24
9.2	Initiate search loader . . . . .	24
9.3	Remove search loader . . . . .	24
9.4	Change location . . . . .	24
9.5	Add Image . . . . .	24
9.6	Show Info . . . . .	25
9.7	Auto Complete . . . . .	25
9.8	Create New Map . . . . .	25
9.9	Get New PhotoLayer . . . . .	25
9.10	Map Click . . . . .	25
9.11	Current Location . . . . .	25
9.12	Search Query . . . . .	25
10	Back End	26
10.1	contract SimpleStorage . . . . .	26
10.2	struct Image . . . . .	26
10.3	Mapping . . . . .	26

10.4 String Concatination . . . . .	26
10.5 Simple Storage . . . . .	26
10.6 Get Images . . . . .	27
10.7 Get User Images . . . . .	27
10.8 Upload . . . . .	27
10.9 Strings Equal . . . . .	27
10.10Delete Image . . . . .	27
 V Architecture	 28
11 Context Diagram	29
12 EthPhoto IPFS Architecture	31
13 EthPhoto peer to peer Architecture	32
14 Class Diagram	33
15 Sequence Diagram	34
16 Use Case Diagram	35
17 Activity Diagram	37
 VI Test Plan	 39
18 Introduction	40
19 Test Objective	41
20 Process Overview	42
21 Testing Strategy	43
21.1 Unit Testing . . . . .	43
21.1.1 White Box Testing . . . . .	43
21.1.2 Black Box Testing . . . . .	44
21.2 Integration Testing . . . . .	45
21.2.1 Incremental Testing . . . . .	45
21.3 System Testing . . . . .	45
21.3.1 Function Validation Testing . . . . .	46
21.3.2 Performance Testing . . . . .	46

22	Entry and Exit Criteria	47
22.1	Unit Testing . . . . .	47
22.1.1	Black Box Phase . . . . .	47
22.1.2	White Box Testing . . . . .	48
22.2	Integration Testing . . . . .	49
22.2.1	Integration Test Entry Criteria . . . . .	49
22.2.2	Integration Test Exit Criteria . . . . .	50
22.3	System Testing . . . . .	50
22.3.1	System Test Entry Criteria . . . . .	51
22.3.2	System Test Exit Criteria . . . . .	51
22.4	Shipping/Live Release . . . . .	51
22.4.1	Shipping/Live Release Entry Criteria . . . . .	51
22.4.2	Shipping/Live Release Exit Criteria . . . . .	52

# Part I

## Software Requirement and Specifications

# Chapter 1

## Introduction

### 1.1 Purpose

This SRS describes the software functional and non-functional requirements for release of EthPhoto v0.1. This software is a block-chain application designed to share photos between peers and store the data in the form of hashes in the block-chain, all requirements specified here are of high priority and committed for release v0.1.

### 1.2 Product Scope

This software consists of following major functions:

- Import photo from the user along with the latitude, longitude and a tag.
- Generating a hash of the photo and uploading the hash along with the latitude, longitude and tag in the block-chain.
- Viewing all the photos shared by the user
- viewing all the photos shared by all the users
- Deleting photos shared by the user

# Chapter 2

## Overall Description

### 2.1 Product Perspective

This software is a peer to peer photo sharing application .The user can upload an image with the latitude, longitude and any number of tags.This photo will be displayed in the Map at the specified location .All the data is stored in block-chain therefore all the data is with every user and there is no central server or storage, thus satisfying the requirement of decentralized server.The user can view any photo on the block-chain but can delete only the photos he has shared.

### 2.2 Product Functions

The set of functionalities supported by this software are as follows :

- *Add Image*: The user can upload an image along with the location co-ordinates and a Tag to the block-chain.
- *Get User Images* : View all the images shared by the user.
- *Delete images*: The user can navigate to 'My Photos' section to view all the photos uploaded, as well as delete any of them.



## 2.3 Operating Environment

This software is developed in Solidity 0.4.11, running on Debian-based Ubuntu 14.04 x64 Architecture. The system should have ethereum and IPFS in order to work. It should also be compatible with other Debian-Linux based x64 Operating Systems.

## 2.4 Design and Implementation Constraints

- The image must have a geo tag otherwise the user has to point the location on the map for co-ordinates.
- The user must enter a tag for the image.

## 2.5 User Implementation

A user manual in PDF format would be made available along with the software.

## 2.6 Assumptions and Dependencies

This software has been targeted at Debian-based x64 Operating Systems. It depends on Open Source tools like:

- Ethereum
- IPFS
- EmbarkJS

The required dependencies are all packaged with the installer, although super-user permissions might be necessary to install them. The software cannot be installed on any other operating system apart from Linux-Debian based x64 Architecture.

## Chapter 3

### External Interface Requirements

#### 3.1 User Interfaces

The user interface is simplistic. It consists of a MAP on which all the images are displayed at their specific locations. There is a search bar for searching locations on the map. There is a ADD button to add images to the block-chain. There is also a SHOW MY PHOTOS button to show all the photos shared by the user.

#### 3.2 Hardware Interfaces

No special hardware is needed for the functioning of this software. A computer with a monitor, a keyboard and a mouse suffices.

#### 3.3 Software Interfaces

The software consists of a multiple user block-chain system with a central node called ETHEREUM on which the contract is deployed using Embark Framework. All the users use MIST browser to open the website and connect the user node to central node .

#### 3.4 Communication Interfaces

Internet connection is necessary for the installation of the software and for its working.

## 3.5 Memory Constraints

The software is quite memory efficient as it only requires minimal space to store the uploaded hash of the images in the project directory which can later be freed by the wish of the user. Any temporary files created by the software are erased upon exit.

## Chapter 4

### Functional Requirements

#### 4.1 Add Image

Select the Add image button to browse for the image in the system to upload .If the image is not geo-tagged, a location on map must be selected for co-ordinates and atleast one tag is required to be entered for the image.

#### 4.2 Get User Image

This returns all the images shared by the user.

#### 4.3 Delete

The user can go to 'My Photos' section to browse all the photos uploaded by user and can choose to delete them.

# Chapter 5

## Other Non-functional requirements

### 5.1 Security and Privacy Requirements

All the data is hashed before uploading therefore all the data is secured. All the information is on the block-chain i.e all the users have all the data therefore the data will not be lost until the full block-chain is removed

### 5.2 Reliability

As this software only depends on good INTERNET connectivity for its processing it proves to be quite a reliable software.

## Part II

### Installation Instructions

# Chapter 6

## Installing dependencies

1. Open terminal. Press Ctrl + Alt + T on ubuntu to open it.
2. Execute `install.sh` file by typing `bash install.sh` and hitting enter.
3. Type Y and press enter whenever prompted during installation.

The dependencies are large and are installed by various package managers like aptitude, npm, brew and downloaded using wget. There is another tool called axel, which acts like a download manager and can be used to download the mist browser zip, in case of a slow internet.

The following packages are supposed to be installed :

### 6.1 Aptitude dependencies

- build-essential
- ethereum
- nodejs
- npm

### 6.2 npm dependencies

- embark
- ethereumjs-testrpc

## 6.3 source dependencies

- ipfs
- Mist browser



# Part III

## User Guide

# Chapter 7

## Starting the Software

### 7.1 Graphical User Interface

1. Run `nohup ipfs daemon &` in the terminal. `nohup` and `&` enables us to work on the same terminal and put the process in the background.
2. Run `ipfs swarm connect /ip4/<ip>/tcp/4001/ipfs/<ipfshash>`
3. Save the file `genesis.json` with the following content

```
{
  "nonce": "0x00000000000000042",
  "difficulty": "0x40000",
  "mixhash": "0x00000000000000000000000000000000",
  "coinbase": "0x00000000000000000000000000000000",
  "timestamp": "0x00",
  "parentHash": "0x00000000000000000000000000000000",
  "extraData": "0x",
  "gasLimit": "0x4c4b40000"
}
```

4. Run

```
$ geth --identity "Sukhi" --rpc --rpcport "8001" --rpccorsdomain "*"
--rpcaddr "0.0.0.0" --datadir "./node0" --port "30301" --ipcapi "ad-
min,db,eth,debug,miner,net,shh,txpool,personal,web3" --rpcapi "db,eth,net,web3"
--autodag --networkid 1900 --nat "any" init genesis.json
```

5. Run

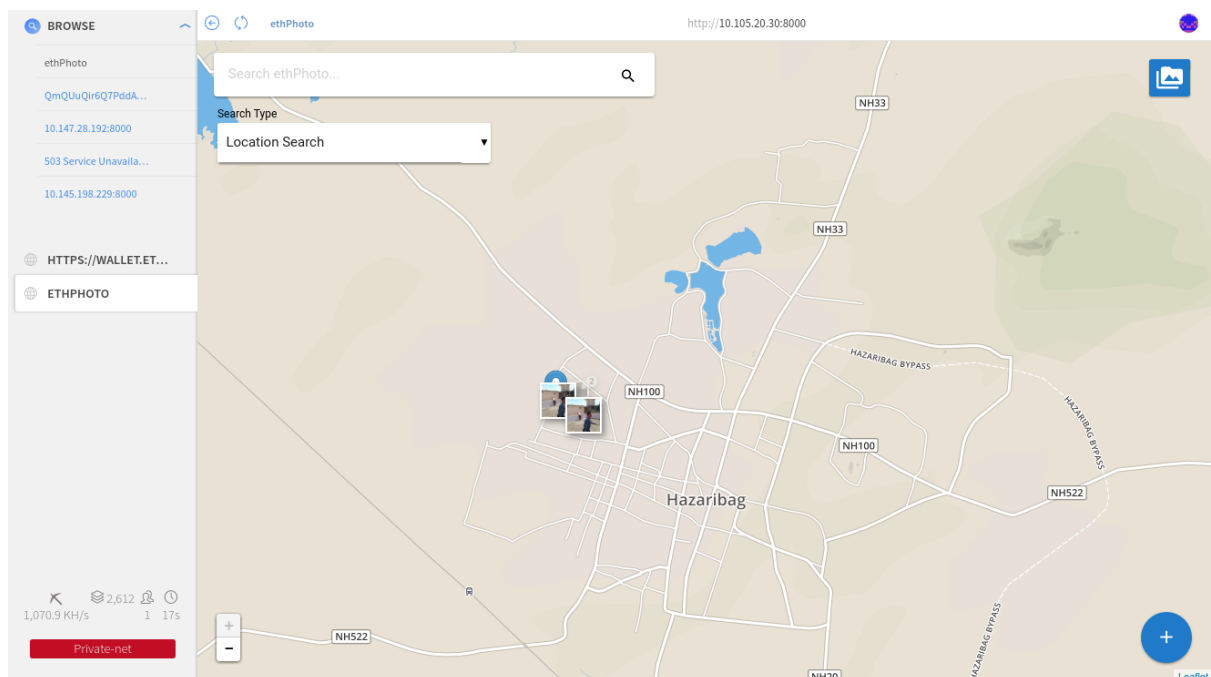
```
$ geth --identity "Sukhi" --rpc --rpcport "8001" --rpccorsdomain "*"
--rpcaddr "0.0.0.0" --datadir "./node0" --port "30301" --ipcapi "ad-
min,db,eth,debug,miner,net,shh,txpool,personal,web3" --rpcapi "db,eth,net,web3"
--autodag --networkid 1900 --nat "any" console on geth command
line
```

6. Run `admin.addPeer("enode://<enodehash>@<ip>:30301")`
7. Make an account in geth by executing `personal.newAccount("password")`
8. Now set networkx proxy (http and https) to `http://<ip>:1337`
9. Run `miner.start()`
10. Type `admin.addPeers` on geth command line
11. Run  
`$ mist -rpc node0/geth.ipc` in another terminal and open `http://<ip>:8000`  
in address bar

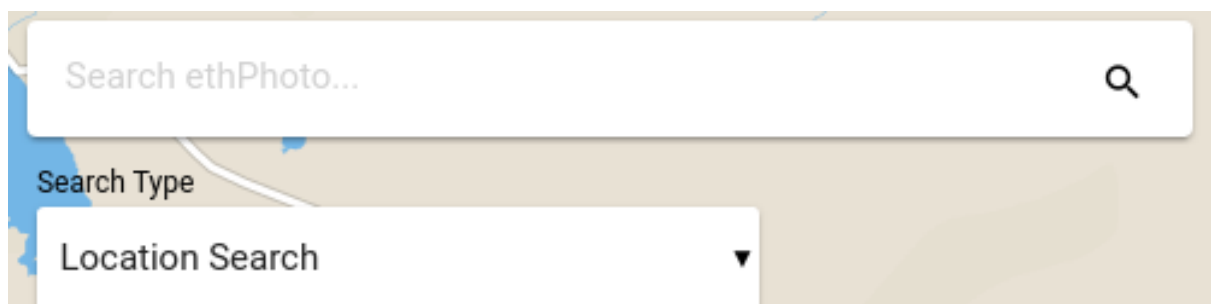
# Chapter 8

## Main Window

The main window is the mist browser .

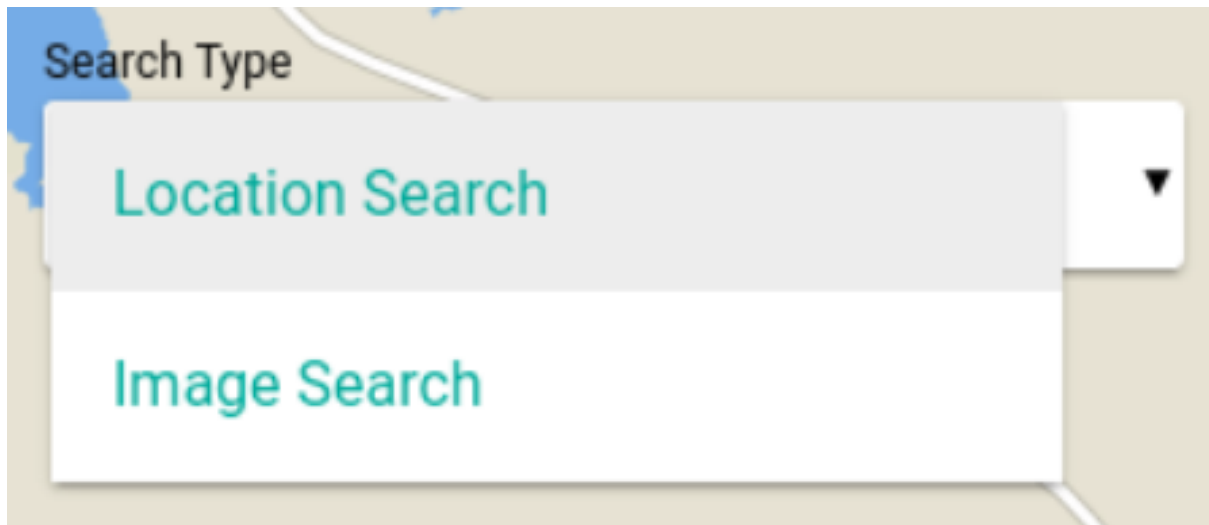


### 8.1 Search bar



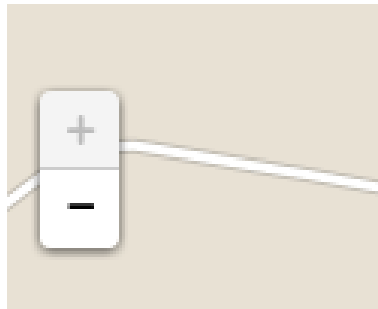
The search bar offers two types of search i.e. location and Image.

### 8.1.1 Types of Search



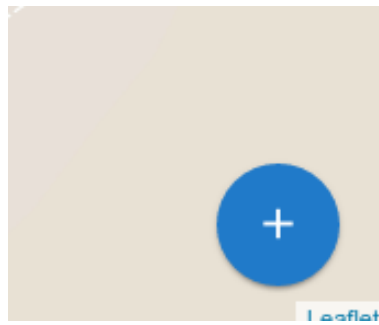
1. Location : The user has to enter the desired location where he/she wants to view the photos.
2. Image : The user enters the particular tag for which he/she wants to view the photos.

## 8.2 Zoom

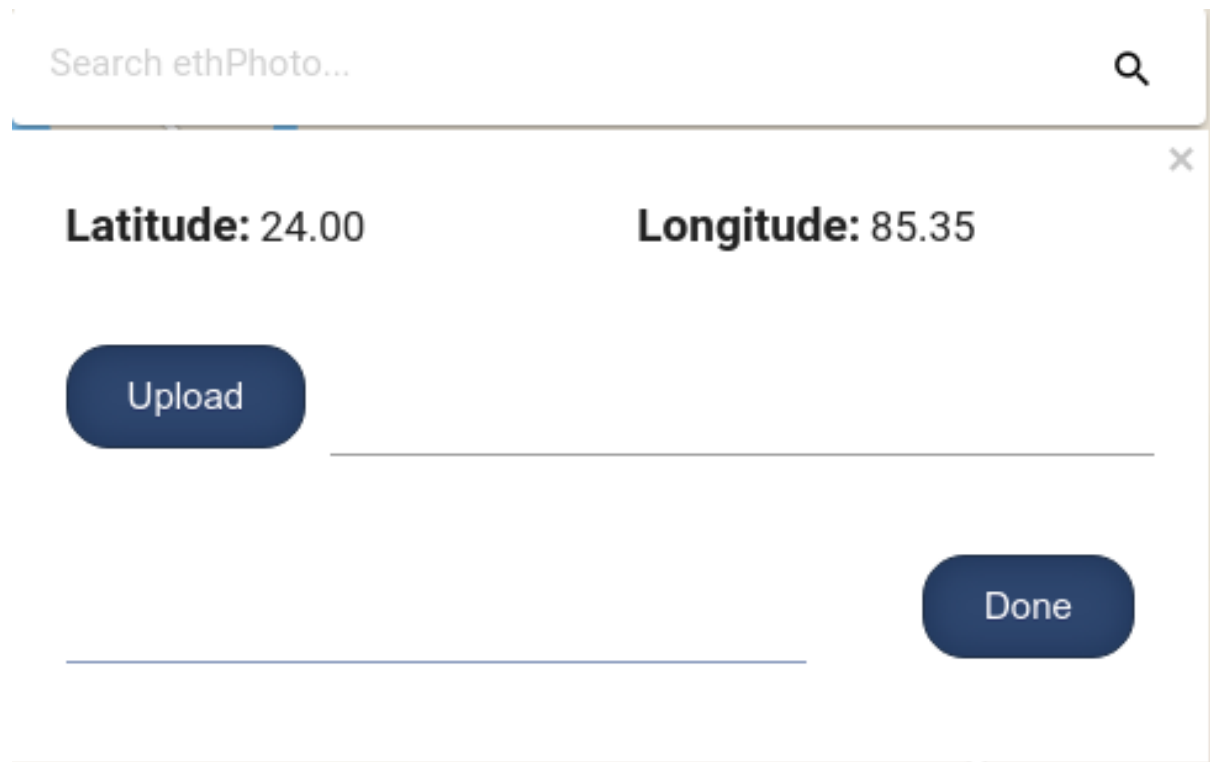


This button in the bottom left corner of the window enables the user to zoom in or zoom out of the map.

## 8.3 Add Photos



This button on the bottom right corner of the window enables the user to upload images . When the user clicks on the button ,this pop-up appears.

A screenshot of a pop-up form for uploading photos. The form has a light beige background and a thin border. At the top, there is a search bar with the placeholder text 'Search ethPhoto...' and a magnifying glass icon on the right. Below the search bar, there is a close button (X) in the top right corner. The form contains two fields: 'Latitude: 24.00' and 'Longitude: 85.35'. Below these fields, there is a blue rounded button labeled 'Upload' on the left and a blue rounded button labeled 'Done' on the right. There are also two horizontal lines for additional input or text.

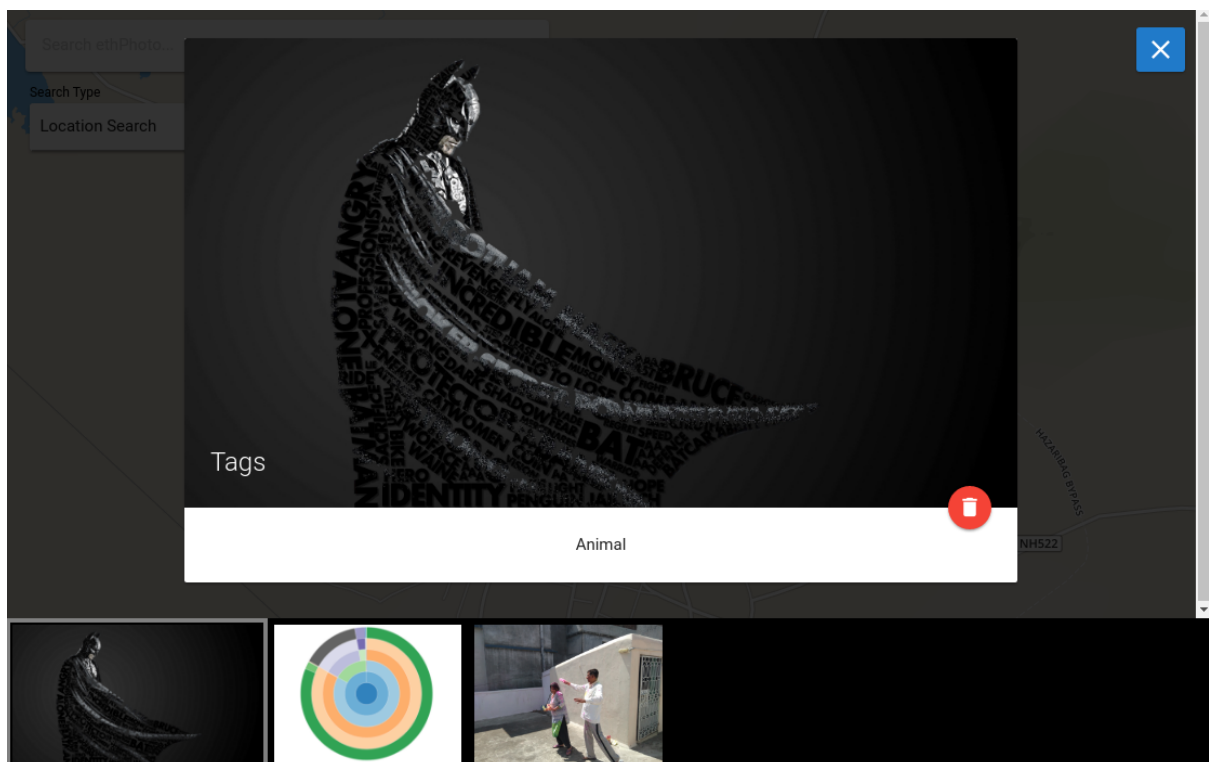
### 8.3.1 How to upload

1. Selects the file from the system.
2. Selects the desired location by either clicking on the map or using the geo-tag of the image.
3. Select one or more tags for the image and then click on Done.

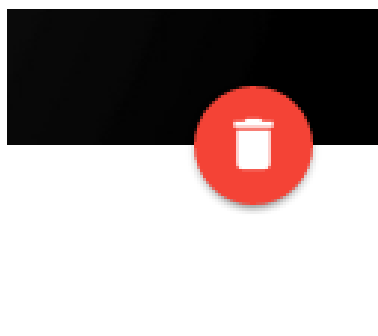
## 8.4 My Photos



This button on the upper right corner of the window shows the user all the images that he/she has uploaded.



The user can view all the images here and also has the option of deleting the images by clicking this button



on the bottom right corner of the image the user wants to delete.

# Part IV

## Algorithms



## Chapter 9

### Front end

#### 9.1 Initiate Geo-Location

This function checks if the browser supports location tracking. If the browser supports then the function calls `getCurrentLocation` function to initiate the current location in the program otherwise it shows an error statement "Sorry, your browser does not support geolocation services."

#### 9.2 Initiate search loader

This function initiates the loader in the search bar when a search is made.

#### 9.3 Remove search loader

This function removes the loader in the search bar when the search is finished.

#### 9.4 Change location

This function takes latitude and longitude as arguments and points to the corresponding location on the map.

#### 9.5 Add Image

This function bundles a photo with all its attributes and pushes it onto the photo layer which is a floating layer present over the map. Before uploading a photo it checks whether the user has added any tag with the photo and return an alert message if the photo does not contain any tag.

## 9.6 Show Info

This function displays the Latitude and Longitude of the the pointer on the map.

## 9.7 Auto Complete

This function predicts the possible location when the user starts typing in the search bar.

## 9.8 Create New Map

This function creates a new map with default settings. It has the location co-ordinates of our campus.

## 9.9 Get New PhotoLayer

This function displays the map and current photos uploaded. When the delete button for the photos is pressed, the whole photo layer is removed. The images which have to be deleted are removed from the array storing the images, and the remaining images are again uploaded to a new photo layer.

## 9.10 Map Click

When the user clicks on any point on the map, this function places the marker on that position, and displays it on the file upload menu.

## 9.11 Current Location

This function finds and stores the user's location via the web browser. When the user selects the upload image button, this location is used if the user does not provide any.

## 9.12 Search Query

When a user enters a location in the search bar, this function calls an API to find out the latitude and longitude of that location, which is displayed on the screen.

# Chapter 10

## Back End

### 10.1 contract SimpleStorage

This is an Ethereum Solidity Smart-Contract which contains a structure named Image.

### 10.2 struct Image

This structure contains 4 data types, namely ihash (image hash), lat(latitude of the photo taken), long(longitude of the photo taken) and tags (tags for the photo).

### 10.3 Mapping

It creates a public state variable named Images. Mappings are like hashtables which are virtually initialized such that every possible key exists and is mapped to a value whose byte-representation is all zeros.

### 10.4 String Concatination

It accepts 5 strings and concatenates them.

### 10.5 Simple Storage

This function pushes a new user to the UserIndex.

## 10.6 Get Images

This function returns all user's images with the location and tags.

## 10.7 Get User Images

This function returns the current user's images with the location and tags.

## 10.8 Upload

This function accepts the image hash, location and tags for an image and pushes it to the images array for the current user.

## 10.9 Strings Equal

This function compares two strings and returns true if they are equal and vice-versa.

## 10.10 Delete Image

This function accepts the image hash value from the user and searches for it in the images array. Once found, this image is removed and the total length of the array is reduced by one.

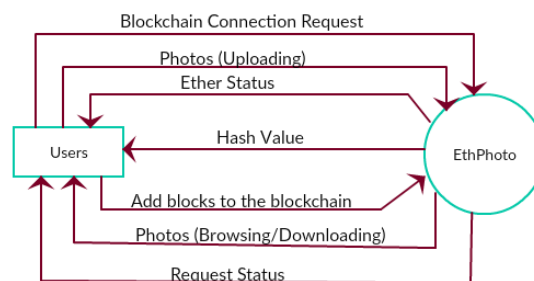
## Part V

### Architecture

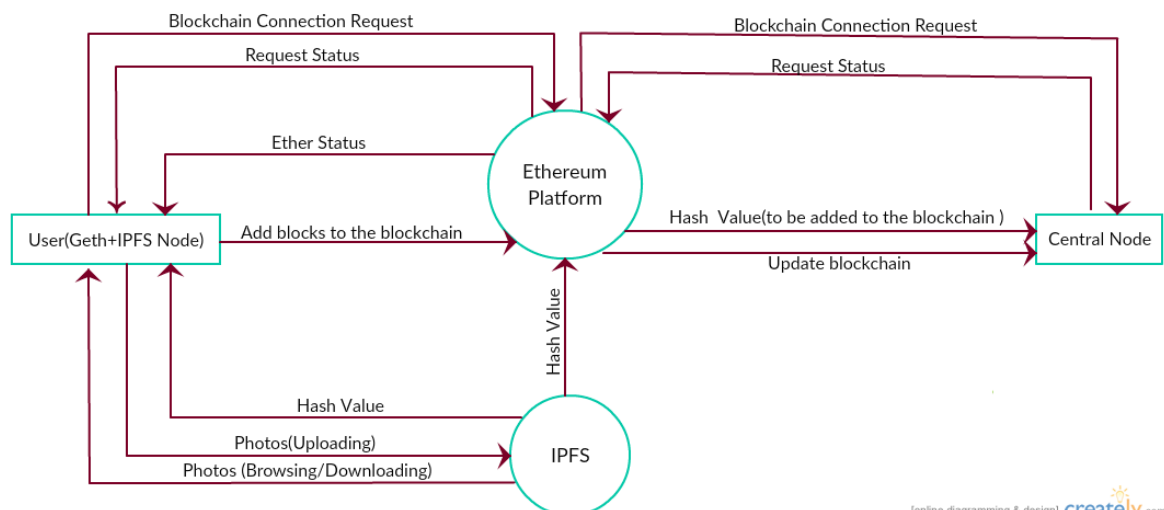
# Chapter 11

## Context Diagram

Context Level DFD



Level 1 DFD



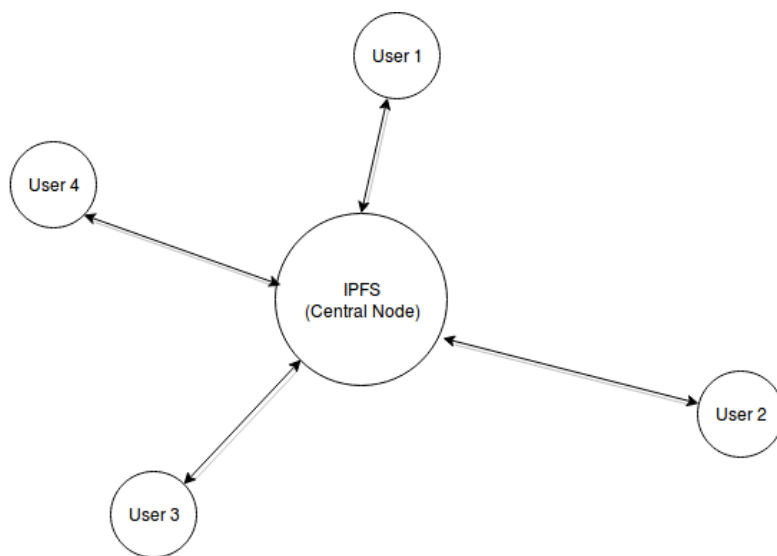
[online diagramming & design] [creately.com](https://www.creately.com)

The figure above provides an overview of the functionality of the whole software in a very crude term. It shows the distinct parts of the software and it interacts to effectively put the whole system working. The various parts of the system are explained below:

- EthPhoto: EthPhoto is the interactive and the visible part of the whole software which the user of the software interacts with. It just displays the actions performed by the ethereum based dapp.
- User: User is the object which represents the different user nodes present in the blockchain.

## Chapter 12

### EthPhoto IPFS Architecture

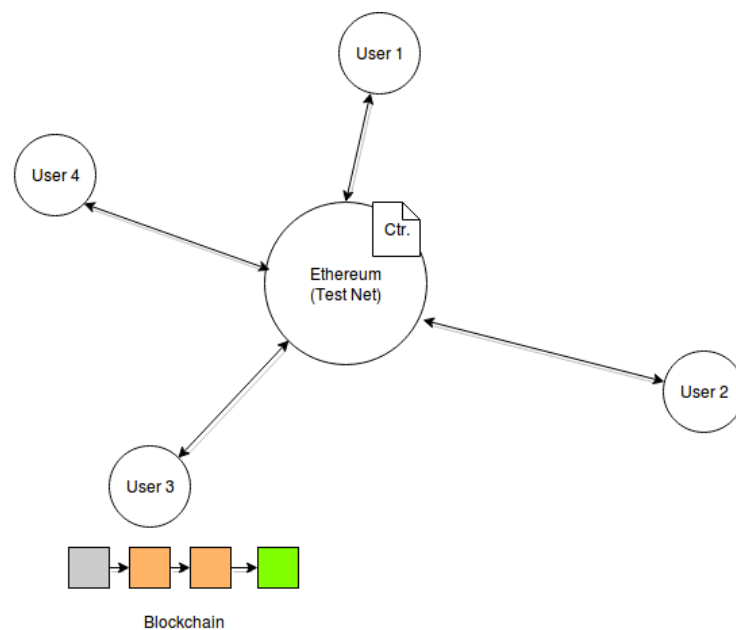


### EthPhoto IPFS Architecture



## Chapter 13

### EthPhoto peer to peer Architecture

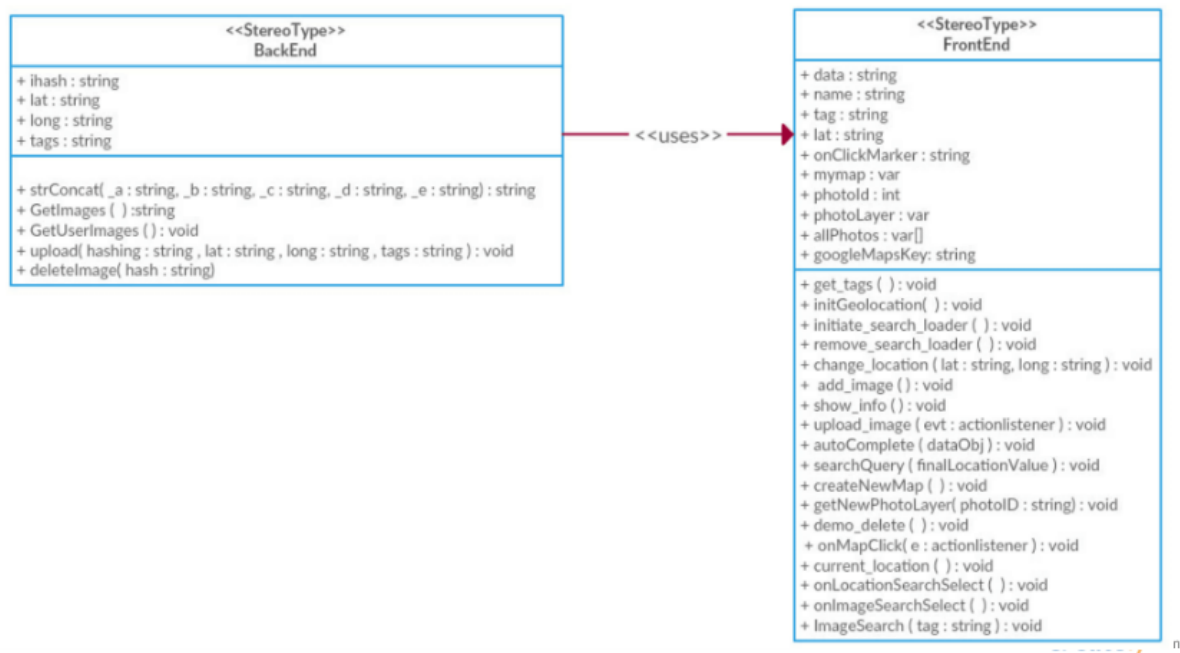


### EthPhoto Peer to Peer Architecture

# Chapter 14

## Class Diagram

### Class Diagram

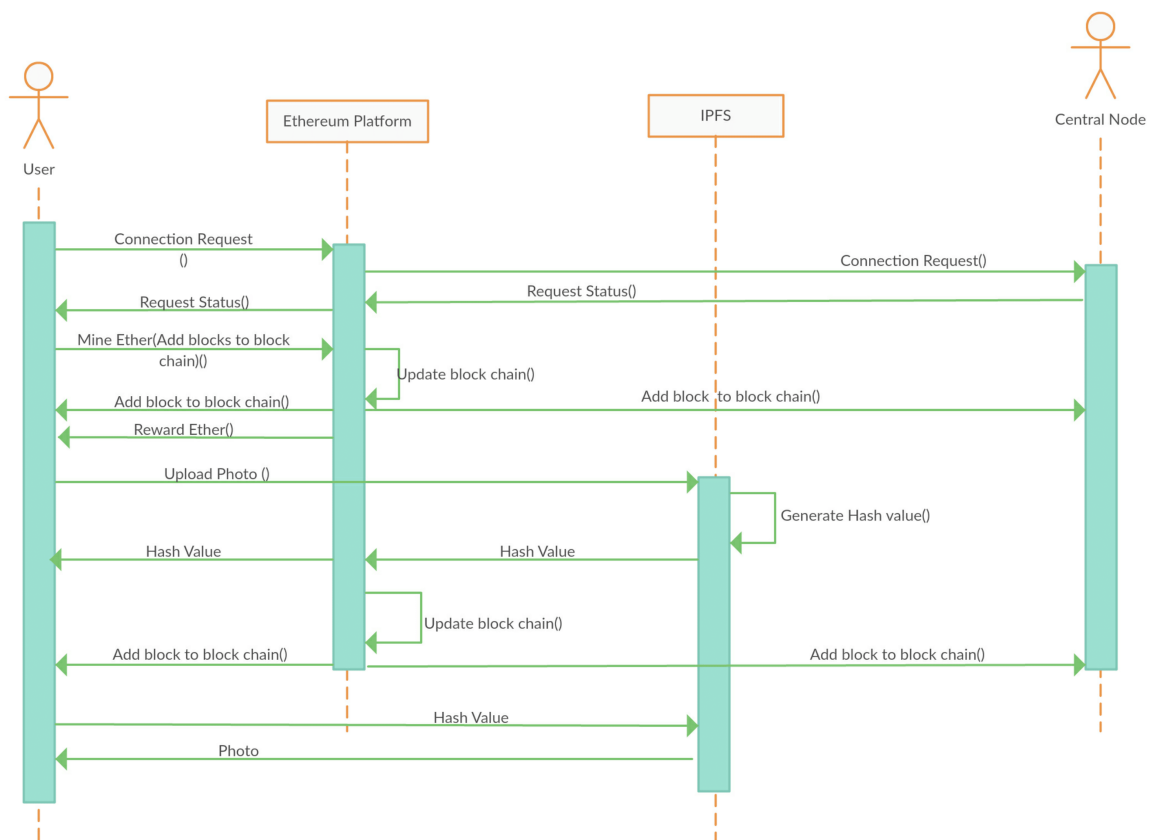


High resolution picture in images folder.

1. There are 2 main classes used in the software.
2. The relationships between the classes are as shown in the diagram.

# Chapter 15

## Sequence Diagram



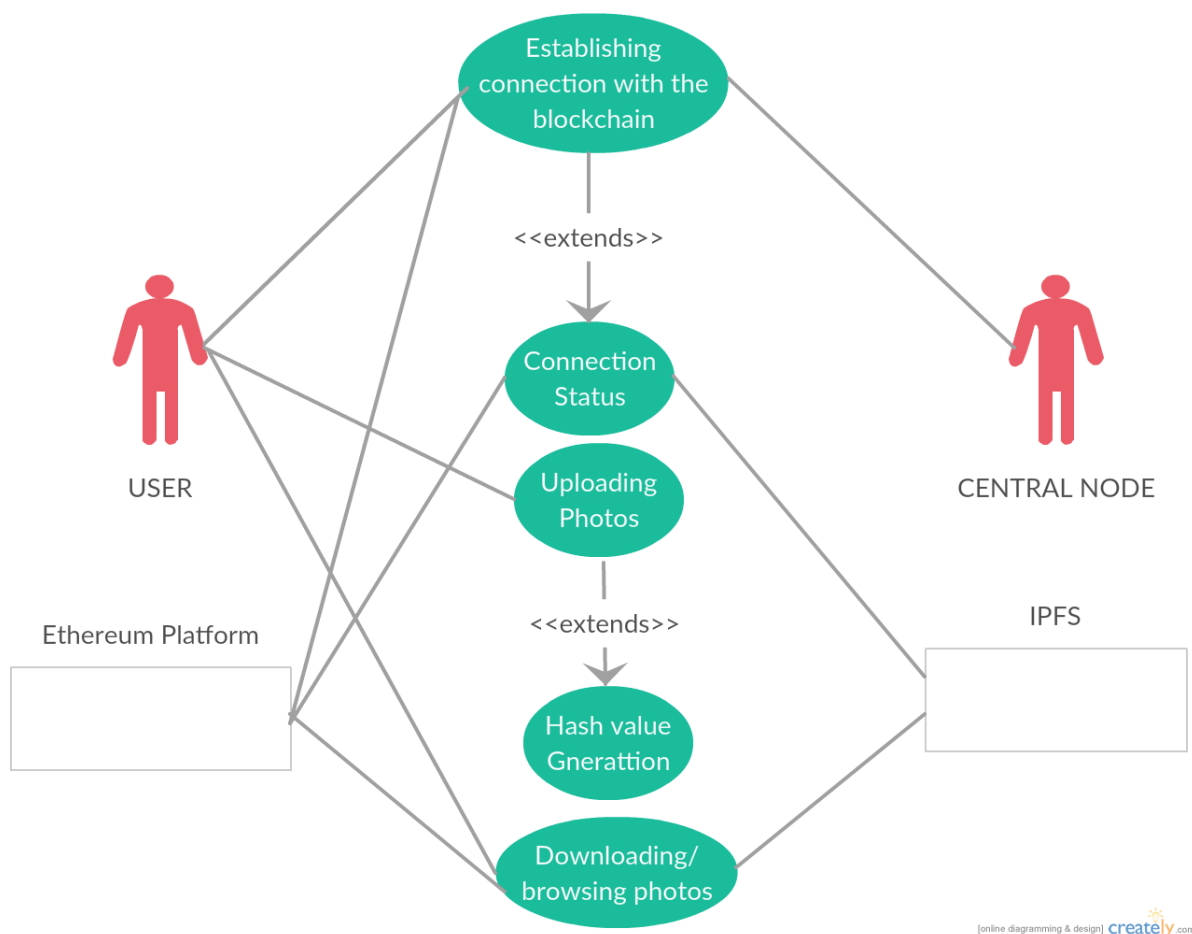
High resolution picture in images folder.

### Analysis

- 1.The sequence diagram above shows the time flow of the software.
- 2.The Program starts with instance of request for connection.
- 3.The request is then passed to the central node for authentication.
- 4.If the user does not have required amount of ether then it has to first mine it and then can upload photo.
- 5.The program then proceeds by adding the hash value obtained through IPFS to the existing blockchain.

# Chapter 16

## Use Case Diagram



High resolution picture in images folder.

Analysis :

1.The diagram shows the interaction between actors,other entities and usecases.

2.The software starts when the user establishes connection.

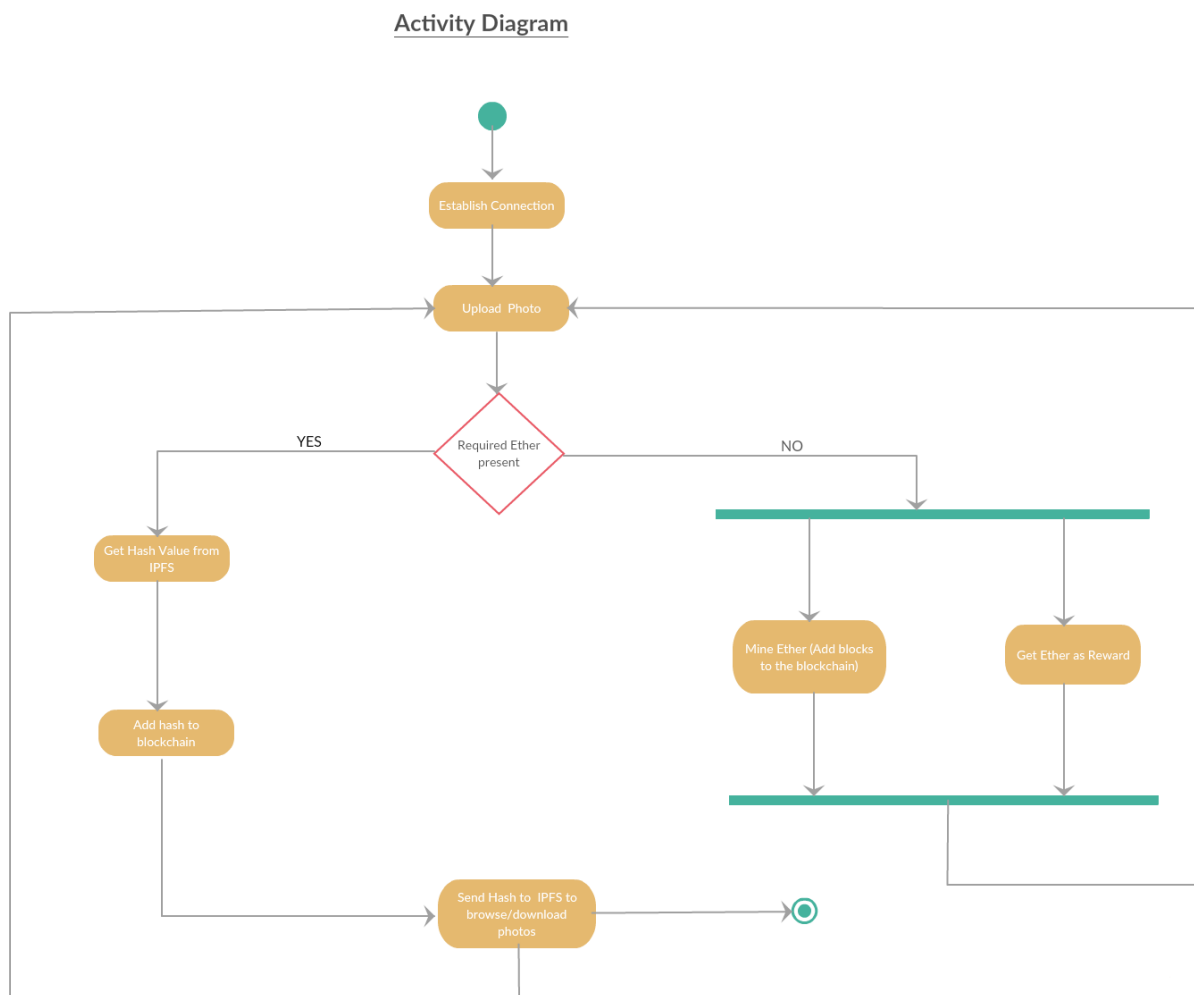
3.The Ethereum based platform helps implement deals as per contract, hence is an important entity in the system.

4.The IPFS acts a hash generator in the system where any user can upload

any photo and get a hash value in return which is then added to the blockchain by spending some ether.

# Chapter 17

## Activity Diagram



High resolution picture in images folder.

Analysis :

1.The diagram above shows the various activities involved in the execution of the software.

2.The software starts when the user establishes connection.

3.Then the user uploads photos by spending some ether .

- 4.If the user doesnot have required amount of ether then it has to first mine it and then can upload photo.
- 5.Then the photo is added to the blockchain and the blockchain is updated.
- 6.If the user wants to see any photo then it sends the Hash value to the IPFS and thus retrieves the photos.

## Part VI

### Test Plan



# Chapter 18

## Introduction

This document is a high-level overview defining testing strategy for the Eth-Photo. Its objective is to communicate project-wide quality standards and procedures. It portrays a snapshot of the project as of the end of the planning phase. This document will address the different standards that will apply to the unit, integration and system testing of the specified application. Testing criteria under the white box, black box, and system-testing paradigm will be utilized. This paradigm will include, but is not limited to, the testing criteria, methods, and test cases of the overall design. Throughout the testing process the test documentation specifications described in the IEEE Standard 829-1983 for Software Test Documentation will be applied.

# Chapter 19

## Test Objective

The objective of this test plan is to find and report as many bugs as possible to improve the integrity of our program. Although exhaustive testing is not possible, a broad range of tests will be exercised to achieve the goal. There will be following functions that can be performed by this application: Upload images, download images for a user, select a user to see their images, deleting our image. The user interface to utilize these functions is designed to be user-friendly and provide easy access to all the functions.

To check the integrity of the software and verify if it's working properly, go to the root of the project, and in Terminal, execute

```
$ npm test
```

```
.....
```

---

```
Ran 31 tests in 13.992s
```

OK

ava is a futuristic Javascript test runner and the library used for extensive unit testing and picking out code smells. Even though JavaScript is single-threaded, IO in Node.js can happen in parallel due to its async nature. AVA takes advantage of this and runs your tests concurrently, which is especially beneficial for IO heavy tests. In addition, test files are run in parallel as separate processes, giving you even better performance and an isolated environment for each test file. Switching from Mocha to AVA in Pageres brought the test time down from 31 to 11 seconds. Having tests run concurrently forces you to write atomic tests, meaning tests don't depend on global state or the state of other tests, which is a great thing!

# Chapter 20

## Process Overview

The following represents the overall flow of the testing process:

1. Identify the requirements to be tested. All test cases shall be derived using the current Program Specification.
2. Identify which particular test(s) will be used to test each module.
3. Review the test data and test cases to ensure that the unit has been thoroughly verified and that the test data and test cases are adequate to verify proper operation of the unit.
4. Identify the expected results for each test.
5. Document the test case configuration, test data, and expected results.
6. Perform the test(s).
7. Document the test data, test cases, and test configuration used during the testing process. This information shall be submitted via the Unit/System Test Report (STR).
8. Successful unit testing is required before the unit is eligible for component integration/system testing.
9. Unsuccessful testing requires a Bug Report Form to be generated. This document shall describe the test case, the problem encountered, its possible cause, and the sequence of events that led to the problem. It shall be used as a basis for later technical analysis.
10. Test documents and reports shall be submitted. Any specifications to be reviewed, revised, or updated shall be handled immediately.

# Chapter 21

## Testing Strategy

The following outlines the types of testing that were done for unit, integration, and system testing. It includes what will be tested, the specific use cases that determine how the testing is done.

### 21.1 Unit Testing

Unit Testing is done at the source or code level for language-specific programming errors such as bad syntax, logic errors, or to test particular functions or code modules. The unit test cases shall be designed to test the validity of the programs correctness.

#### 21.1.1 White Box Testing

In white box testing, the UI is bypassed. Inputs and outputs are tested directly at the code level and the results are compared against specifications. This form of testing ignores the function of the program under test and will focus only on its code and the structure of that code. Test cases are generated that not only cause each condition to take on all possible values at least once, but that cause each such condition to be executed at least once.

Testing the front-end module

A module `test_front-end` was implemented as a test bench for the front-end. The module contained the following methods:

- `test_intGeoLocation()` - Verifies the function `intGeoLocation()`.
- `test_search_loader()` - Verifies the function `initiate_search_loader()` and `remove_search_loader()`.

- `test_addImage()` - Verifies the function `addImage()` by checking whether the photo is uploaded.
- `test_showInfo()` - Verifies whether the location information is displayed.
- `test_createNewMap()` - Verifies whether the map is created and displayed.
- `test_getNewPhotoLayer()` - Verifies the deletion of an image.
- `test_onMapClick()` - Verifies whether the pointer of the location moves when any point on the map is clicked.
- `test_searchQuery()` - Verifies the backend API call when a location is entered.

### Testing the back-end module

A module `test_back-end` was created to test the back-end module which contains the following methods:

- `test_strConcat()` - This method asserts the concatenating of different strings into one.
- `test_GetImages()` - This method tests the function `GetImages()`
- `test_GetUserImages()` - This method tests the function `GetUserImages()`
- `test_upload()` - This method tests the function `upload()`

All the tests were rigorously scrutinized and were successful.

### 21.1.2 Black Box Testing

Black box testing typically involves running through every possible input to verify that it results in the right outputs using the software as an end-user would. We have decided to perform Error guessing and Boundary Value Analysis testing on our application.

## 21.2 Integration Testing

### 21.2.1 Incremental Testing

There are two primary modules that were needed to be integrated: the front-end and the back-end. The two components, once integrated form the complete EthPhoto Application. The following describes these modules as well as the steps that will need to be taken to achieve complete integration. We will be employing an incremental testing strategy to complete the integration.

**Module 1 - Front-end Module.** This module provides a simple GUI where the user can perform the different actions (functions). This module was tested separate from the back end to check if each interface (e.g. search button) is functioning properly, and in general, to test if the mouse-event actions were working properly. The testing was performed by writing a stub for each element in the interface.

**Module 2 - back-end Module** The back-end functions accept the input from the front-end to perform different functions like uploading or deleting the image. This module was tested separate from the front-end separately. In testing this module we followed the incremental testing method i.e. testing one function first and then keep adding additional function and test it again until all the required functions are tested.

When the front-end is combined with the back end module, we will have a complete EthPhoto application. To achieve complete integration of these two modules, we test each element in the GUI by replacing the stubs with the appropriate function from the back end. The results were displayed within the GUI instead of through the Console. In testing the combined modules, we followed the incremental testing method. Each stub will be replaced one at a time and tested. This was done until all stubs have been replaced by the appropriate functions from the back end.

## 21.3 System Testing

The goals of system testing are to detect faults that can only be exposed by testing the entire integrated system or some major part of it. Generally, system testing is mainly concerned with areas such as performance, security, validation, load/stress, and configuration sensitivity. But in our case we focused only on function validation and performance. And in both cases we used the black-box method of testing.

### 21.3.1 Function Validation Testing

The integrated EthPhoto was tested based on the requirements to ensure that we built the right application. In doing this test, we tried to find the errors in the inputs and outputs, that is, we tested each function to ensure that it properly implements the extraction and processing procedures, and that the results are expected. The behavior of each function are contained in the Software Requirement Specification.

Function	Expected Behavior
Upload Image	see Software Requirement Specification
Delete Image	see Software Requirement Specification
Show Image	see Software Requirement Specification
Delete Image	see Software Requirement Specification

In addition, we tested:

- The interfaces to ensure they are functioning as desired (i.e. check if each interface is behaving as expected, specifically verifying the appropriate action is associated with each mouse\_click event).
- The interaction between the front-end and the back end modules. In this case the images will be uploaded and checked if they are processed in the back end and stored.

### 21.3.2 Performance Testing

This test was conducted to evaluate the fulfillment of a system with specified performance requirements. It was done using black-box testing method. The following were tested:

- Taking input with large number of images and checking how much time the application takes.
- Taking input images which are from very far-away locations.

# Chapter 22

## Entry and Exit Criteria

This section describes the general criteria by which testing commences, temporarily stopped, resumed and completed within each testing phase. Different features/components may have slight variation of their criteria, in which case, those should be mentioned in the feature test plan. The testing phase also maps to the impact level definition when a defect is entered in the bug-tracking phase.

### 22.1 Unit Testing

Unit Testing is done at the source or code level for language-specific programming errors such as bad syntax, logic errors, or to test particular functions or code modules. The unit test cases shall be designed to test the validity of the programs correctness.

#### 22.1.1 Black Box Phase

Black box testing typically involves running through every possible input to verify that it results in the right outputs using the software as an end-user would. We will use Equivalence Partitioning and Boundary Value Analysis complexity metrics in order to quantifiably determine how many test cases needed to achieve maximum code coverage.

#### Black Box Entry Criteria

The Black Box Entry Criteria will rely on the component specification, and user interface requirements. Things that must be done on entry to the Black Box stage:



- All functions like Upload images, download images for a user, select a user to see their images, deleting our image must either be coded or stubs written.
- The type of Black Box testing Methods will be determined upon entry. We will use Error Guessing, and Boundary Value Analysis.
- Error Guessing included entering garbage string in search field, trying to add the same photo multiple times, closing the Internet connection and searching etc.
- Boundary Value Analysis included setting uploading a large number of files, opening large number of tabs and searching and uploading simultaneously.

### Black Box Exit Criteria

The Black Box Exit Criteria listed below explains what needs to be completed in-order to exit Black Box phase. To exit the Black Box phase 100% success rate must be achieved. Things that must be done upon exiting the Black Box stage:

- The application showed the message if no results are found in case of garbage string.
- The applications searched on a maximum of 12 - 15 tabs simultaneously beyond which a crash might occur which was handled.
- All code bugs that are exposed are corrected.

### 22.1.2 White Box Testing

The White Box criteria apply for purposes of focusing on internal program structure, and discover all internal program errors. Defects will be categorized and the quality of the product will be assessed.

#### White Box Entry Criteria

The White Box Entry Criteria will rely verifying that the major features work alone but not necessarily in combination; exception handling will not be implemented. The design and human interface are stable. Things that must be done on entry to the White Box stage:

- All functions upload images, download images for a user, select a user to see their images, deleting our image - must be coded.
- The type of White Box testing Methods was determined upon entry. We will use unit testing and test for memory leaks.
- Black Box Testing should be in its late stages.

### White Box Exit Criteria

The EthPhoto software in the White Box stage should have a generally stable feel to it. White Box testing continued until the Black Box or next milestone criteria were met. To exit the White Box phase 100% success rate must be achieved. The following describes the state of the product upon exit from the White Box Stage:

- All functions- Upload images, download images for a user, select a user to see their images, deleting our image- are implemented, operational and tested.
- All Branch Testing test cases were generated. The test cases will be generated from the Control Flow diagrams of all functions.
- The graphical interface has been reviewed and found to satisfactory and is stable, that is, no further changes to dialog boxes or other interface elements are planned. Minor changes are acceptable, but must be arranged with the Development and Test Engineers.
- All code bugs that are exposed are corrected.

## 22.2 Integration Testing

There are two modules that will be integrated for Integration Testing. The two modules are The front-end module and back-end module. The two components consists of a mixture of stubs, driver, and full function code. The following describes the entry and exit criteria for Integration testing.

### 22.2.1 Integration Test Entry Criteria

The Integration Test Entry Criteria will rely on both modules to be operational. The Ethphoto front-end and back-end must be stable. Things that must be done on entry to the Integration Test stage:

- All modules functions - Upload images, download images for a user, select a user to see their images, deleting our image - must either be coded and/or stubs created.
- The front-end must either be coded and/or a driver and stubs must be created. The driver is implemented to facilitate test case input and output values.
- Interfaces and interactions between the EthPhoto front-end and back-end must be operational.
- A bottom-up Integration Test Strategy will be conducted. The low level details of the front-end and back-end were integrated. A driver will be written to facilitate test case input and output values. The driver temporarily satisfied high-level details of the input and output values.

### 22.2.2 Integration Test Exit Criteria

The Integration Test Exit Criteria relied on both modules to be operational. The EthPhoto front-end and back-end must be stable. To exit the Integration Testing phase 100% success rate must be achieved. Things that must be done on exit from the Integration Test stage:

- All code bugs that were exposed were corrected.
- The front-end and back-end modules interacted together with complete accuracy, according to the System Specification Design. All discrepancies were corrected.
- Both Modules are ready for System Testing. Stubs and drivers are replaced with fully functional code.
- Black Box Testing was completed.

## 22.3 System Testing

The System Test criteria apply for purposes of categorizing defects and the assessing the quality level of the product. All elements of the EthPhoto - Front-end and Back-end modules are meshed together and tested as a whole. System test focuses on functions and performance, reliability, installation, behavior during special conditions, and stress testing.

### 22.3.1 System Test Entry Criteria

- Unit testing of all the modules has been completed.
- System test cases have been documented.
- Specifications for the product have been completed and approved.
- UI and functionalities to be tested should be frozen.
- Priorities bugs have been fixed.

### 22.3.2 System Test Exit Criteria

- Application meets all the document requirements and functionalities.
- Defects found during System testing should be fixed and closed.
- All the test cases for the system have been executed and passed.
- No critical defects have been opened which is found during system testing.

## 22.4 Shipping/Live Release

The back-end testing is scaled down and combines all phases of testing into two phases - Function Complete and Regression testing - and follows the release criteria.

### 22.4.1 Shipping/Live Release Entry Criteria

The installer of the software EthPhoto was rigorously tested by many users and it was found to be installed and executed without any found bugs. The users also tested the basic functionality implemented in the software and no found bugs were detected.

- All open product defects, regardless of fixed defects, documented, deferred, or otherwise addressed were identified.
- Regression testing on all product defects and the entire product has been completed was verified.

#### 22.4.2 Shipping/Live Release Exit Criteria

- No critical defects found.
- Business processes are working satisfactorily.