Software Requirements Specification

for

PyFamilyTree

**Version 1.0**

**Prepared by**

**Group Name: xxxxxx**

| **xxxxxx** | **xxxxxx** | **xxxxxx** |
| --- | --- | --- |
| **xxxxxx** | **xxxxxx** | **xxxxxx** |
| **xxxxxx** | **xxxxxx** | **xxxxxx** |
| **xxxxxx** | **xxxxxx** | **xxxxxx** |

| **xxxxxx:** | **xxxxxx** |
| --- | --- |
| **xxxxxx:** | **xxxxxx** |
| **xxxxxx:** | **xxxxxx** |

**Contents**

**CONTENTS II**

**REVISIONS II**

**1** **INTRODUCTION 1**

1.1 Document Purpose 1

1.2 Product Scope 1

1.3 Definitions, Acronyms and Abbreviations 1

1.4 References and Acknowledgments 2

1.5 Overview 3

**2** **OVERALL DESCRIPTION 3**

2.1 Product Perspective 3

2.2 Product Functions 4

2.3 User Characteristics 5

2.4 Design and Implementation Constraints 6

2.5 Assumptions and Dependencies 6

**3** **SPECIFIC REQUIREMENTS 9**

3.1 External Interfaces - Database 9

3.2 User Stories 10

3.3 Use Cases 15

3.4 Activity Diagrams 21

**4** **OTHER NON-FUNCTIONAL REQUIREMENTS 27**

4.1 Performance Requirements 27

4.2 Safety and Security Requirements 27

4.3 Usability Requirements 27

4.4 Software Quality Attributes 27

4.5 Maintainability Requirement 27

**Revisions**

| **Version** | **Primary Author(s)** | **Description of Version** | **Date Completed** |
| --- | --- | --- | --- |
| 1.0 | **xxxxxx**  **xxxxxx**  **xxxxxx**  **xxxxxx** | **xxxxxx** | **xxxxxx** |

# Introduction

## 1.1 Document Purpose

The purpose of this Software Requirements Specification (SRS) document is to clearly and accurately outline the functional and non-functional requirements of the PyFamilyTree web application. The document is intended to provide a comprehensive exploration of the application’s features, constraints, assumptions, and dependencies for all stakeholders involved in the development, testing, and deployment of the application. It serves as a guideline for the development team to ensure that the application is developed according to the requirements specified by all stakeholders, and as reference to ensure proper functionality through testing against the requirements. This document also facilitates communication between stakeholders, including clients & the PyFamilyTree development team.

## 1.2 Product Scope

PyFamilyTree is a solution for a web-based Family Tree. This solution is intended for anyone interested in documenting their family history and sharing that information with other family members. It will allow family members to join an existing family tree or create one with administrative approval. Once someone is added to a family tree, they can update their personal information, and family relationships, such as parents, spouse, and children. They will also be able to upload documents and images relevant to their family and have the ability to share them with others. Family members can search their family tree to learn more about their ancestors and living relatives. This creates an opportunity to document and share important family history.

This product aims to provide a solution for families to create and document a web-based, editable, and shareable family tree that is created collaboratively.

The goal of this solution is to enable families to learn more about themselves, and dig deeper into their family history.

## 1.3 Definitions, Acronyms and Abbreviations

**1.3.1 Super Administrator (Super Admin)**: A user who has complete privileges and access to all objects, folders, role templates, and groups across all family trees in our system. Assigns an administrator (admin) to manage a specific family tree directly.

**1.3.2. Administrator (Admin)**: A user with full privileges and access to all objects, folders, and role templates within a specific family tree.

**1.3.3 Member**: A user of the family tree app who has been approved by an admin or super admin and can access the features of the app for a given family tree.

**1.3.4 Direct Relationship**: A relationship between two individuals/members that can be traced via a direct lineage, like a spouse or a parent-child relationship. Direct relationships correspond to member ids for which the specific member has a direct connection in the appropriate database table.

**1.3.5 Indirect Relationship**: A relationship between two individuals that is not a direct lineage, such as cousins or siblings. Populating indirect relationships in query results would require multiple subqueries.

**1.3.6 Ancestor**: A person from whom a member is descended. An ancestor’s degree refers to the number of generations between the ancestor and the member.

**1.3.7 Descendant**: A person who is descended from a particular member. A descendant’s degree refers to the number of generations between the member and his ancestor.

**1.3.8 PostgreSQL (Postgres)**: A popular, free and open-source relational database management system (RDBMS), emphasizing extensibility and SQL compliance.

**1.3.9 Django**: A commonly-used free and open source Python-based framework for building scalable web applications. It follows the model-template-views architectural pattern.

**1.3.10 Python**: A popular dynamically typed, high-level, general-purpose programming language. Python emphasizes code readability, and supports numerous programming paradigms. Python is the primary programming language in our server side/ backend’s implementation.

**1.3.11 JavaScript (JS):** One of the core programming languages of the World Wide Web, especially in front end, alongside HTML and CSS. JS is the main component of our client side / frontend implementation.

**1.3.12 HyperText Markup Language (HTML)**: The most popular markup language for documents designed to be displayed in web browsers.

**1.3.13 Cascading Style Sheets (CSS)**: a style sheet language used for describing the presentation in a markup language (like HTML / XML).

## 1.4 References and Acknowledgments

The development of the PyFamilyTree web application is made possible with the help of the following references and resources:

**1.4.1 PostgreSQL documentation**: <https://www.postgresql.org/docs/> - The official documentation for PostgreSQL, the database management system used in the application.

**1.4.2 Django documentation**: <https://docs.djangoproject.com/en/3.2/> - The official documentation for Django, the chosen web platform for the application.

**1.4.3 Python documentation**: <https://docs.python.org/3/> - The official documentation for the Python programming language, which is the key programming language in the backend implementation.

**1.4.4 JavaScript documentation**: <https://javascript.info/> - The official documentation for JavaScript, the key programming language in the front-end of the application.

**1.4.5 CSS documentation**: <https://devdocs.io/css/> - The official documentation for CSS, the stylesheet language used for our HTML documents.

**1.4.6 HTML documentation**: <https://html.spec.whatwg.org/multipage/> - The official documentation for HTML, the Markup language used in the frontend’s design.

**1.4.7. Lucidchart documentation**: <https://www.lucidchart.com> - A diagramming and visual communication tool used in the design and planning of the database structure.

**1.4.8 Visio documentation***:* [*https://support.microsoft.com/en-us/visio*](https://support.microsoft.com/en-us/visio) - A diagramming and vector graphics

application used in the design and planning of use cases, and activity diagrams.

**1.4.9** [**https://www.researchgate.net/publication/245514229**](https://www.researchgate.net/publication/245514229) , Assumption management in software development.

We acknowledge the contributors of these resources for their contributions to the fields of web development and database management, which are invaluable for the development of this application.

## 1.5 Overview

PyFamilyTree is a web application that allows users to organize and share family tree data and content. The application’s backend is built using Python, Django, and PostgreSQL, while its frontend is built using Bootstrap, Javascript, HTML, and CSS. External content storage will be provided through Google Drive. The purpose of this software requirements specification document is to provide an overview of the application’s requirements, including both functional and non-functional specifications. The document is intended for all project stakeholders, including potential clients and the development team. This SRS provides a detailed description of the product scope, functionality, design and implementation constraints, underlying assumptions, and non-functional requirements, as well as detailed schematics regarding database design, user stories, use cases, and activity diagrams.

# Overall Description

## 2.1 Product Perspective

PyFamilyTree is an Open Source Project designed to provide a convenient and easily accessible way to organize family data and content. The application will be a one-stop-shop for accessing and sharing family information. High-level member information and family relationships will be stored in a Postgres database, which will point to the family-specific content store on Google drive.

PyFamilyTree’s main goal is to provide a user-friendly and intuitive platform for managing family information and content. The application’s functionality will be focused on organizing family trees, managing family member information, and storing family content securely. The application’s security measures will ensure that member data is kept confidential, and will be accessible only by the specific family’s administrator and other credentialed users.

PyFamilyTree will allow users to modify relationships, search for family members, and store documents and multimedia files in cloud-based storage. The application will support various types of content including documents (.docx, .pdf), images (.png, .jpeg), video (.mp4), and audio (.mp3, .wav) files. The app will be compatible with all modern web browsers.

## 2.2 Product Functions

### 2.2.1 Super Admin Functions:

**2.2.1.1 Create a New Family Tree**: The Super Admin can create a new family tree, automatically becoming its first admin.

**2.2.1.2 Assign Admin**: The super admin can assign admin roles to other users for a given family tree. Admins have the ability to manage family members and their information in their assigned family tree.

**2.2.1.3 Set Content Control**: The super admin can set content controls for all users - defined as the terms of service for interacting with the app and thus all family trees.

### 2.2.2 Admin Functions:

**2.2.2.1 Create Family Members**: The assigned admin can create family members belonging to their family tree. In creating a new member, the admin will be required to input necessary information, including name (first, middle, last), gender, relationships with existing members (Father ID, Mother ID, Spouse ID), contact information (email address, phone number, and address), and allocated space (for bio and personal cloud storage URL).

**2.2.2.2 Approve Member Account**: Only the super admin or assigned admin can approve member account requests. The approval process will require verification of the requestor's identity and admin approval of the account.

**2.2.2.3 Deny Member Account**: Only the super admin or assigned admin can deny member account requests.

**2.2.2.4 Blacklist Email Address**: Only the super admin or assigned admin can blacklist, and thus prevent future attempts to register with a given email address.

**2.2.2.5 Enable Content Control**: Only the super admin or assigned admin can enable and enforce content controls for all users - defined as acceptable content for the particular family tree.

**2.2.2.6 Assign Cloud Storage**: Only the super admin or assigned admin can assign limited storage on the shared Google Drive to members, either upon membership or through modification.

**2.2.2.7 Modify Media on Shared Cloud Storage**: The super admin or assigned admin can delete/share/unshare all existing content on the shared family storage.

### 2.2.3 Member Functions:

**2.2.3.1 Modify Settings / Edit One’s Own Record**: Members can edit their own personal information. This capability is also available for the assigned admin and super admin.

**2.2.3.2. Search Functionality**: Members can search for a target member (including themselves), to find family members belonging to their family tree. This functionality exists for both direct and indirect relationships.

**2.2.3.3 Cloud-Based Storage Access**: Members may store documents and multimedia files in Google Drive, with a limited storage capacity assigned to them by the admin (upon membership or through modification). Members can upload, download, or update their own files, as well as define their content as private or shareable. Shareable content can be accessed by other members of the same family tree.

**2.2.3.4 Invite to Family Tree**: Members may invite others to join their family tree. This invitation itself is a secure email invitation with a unique link.

### 2.2.4 Potential Family Member Functions:

**2.2.4.1 Self-Registration**: Any potential member may fill out a self-registration form and request to join a family tree.

**2.2.4.2 Accept Invitation to Family Tree**: Potential members can accept an email invitation from an existing family tree member to join their family tree.

## 2.3 User Characteristics

The PyFamilyTree web app is designed with a diverse set of users in mind. Prospective users would be interested in exploring their family history and creating a visual representation of their family tree. The application is designed to be intuitive and easy-to-use, including for members without much technological proficiency or experience with genealogy research. That said, some novices may require some guidance to make full use of the app’s features, and should review the demo videos highlighting the app’s key functionality. Other users could be experienced researchers familiar with genealogy research, for whom interacting with a family tree app should be fairly intuitive.

Since the PyFamilyTree web app will handle sensitive personal data (including dates of birth, marriage, and death, as well as additional private information), users of the application should be aware of the importance of data privacy and security, and refrain from sharing sensitive information with third parties. The application will include measures to protect user privacy and prevent unauthorized data access, through password protection and data encryption.

## 2.4 Design and Implementation Constraints

## The PyFamilyTree application may face several developmental constraints/ limitations which includes the following:

## 2.4.1 Compatibility: Compatibility is a major issue when integrating different technologies or databases for all applications. These issues could limit the options available in integrating many varied features, such as data import/export. The compatibility constraint could also restrict the ability to directly share data with other genealogical databases.

## 2.4.2 Development-Based Technical Limitations: Developers are limited by the technical capabilities of their available hardware and software platforms, as well as their knowledge and experience. As a result, the final application could be suboptimal with respect to certain aspects of performance.

## 2.4.3 User-Oriented Technical Limitations: End users are similarly limited in both their technical infrastructure and/or computer literacy. As a result, the experience of certain features would be suboptimal for specific users in such situations.

## 2.4.4 Budget constraints: A limited budget means certain features will not be implemented or would only be partially implemented for the first version. A larger budgetary investment could ameliorate this issue.

## 2.4.5 Regulatory compliance & privacy: Complying with various regulations governing data privacy (like GDPR), and user protection could limit the development options available in terms of data management, user access, and other features.

## 2.4.6 Scalability limitations: The scalability of the application could be tested provided the active user base would not grow beyond what is expected. Faster-than-expect growth could potentially throttle certain services.

## 2.4.7 Time constraints: The tight development timeline may complicate the implementation phase of the development process. This would lead to certain trade-offs being taken and suboptimal or even omitted features in the first rolled out version of the app.

## 2.4.8 User requirements: A lack of market research could lead to a mismatch between the developer application and the desired features for certain users. This could reduce the desirability or frequency of interaction with certain components/features of PyFamilyTree for such users.

## 2.4.9 User interface limitations: The technical limitations of a JS/Bootstrap/HTML/CSS-based front-end would restrict design choices. This could limit the intuitiveness of the design, as well as its flexibility and how user-friendly the app is.

## 2.5 Assumptions and Dependencies

### 2.5.1 Assumptions:

Assumptions are an integral part of the development of the PyFamilyTree application. By their nature and potential impact on the application, we’ve categorized them in the following categories:

**2.5.1.1 Control Assumptions** - These assumptions relate to the constant and unchanging variables in the development process. The software architect will continue to evaluate and examine their consistency with the application’s flow. Software updates and maintenance work will be performed carefully, as to prevent violations of these assumptions.

**2.5.1.2 Environment Assumptions** - These assumptions define the expectation from the environment in which the application will run, including (but not limited to):

* Compatible Server side OS
* Compatible Client side OS
* Network connectivity
* Client-side web browser compatibility
* Database schema
* Python interpreter and site package version
* Bootstrap/ JavaScript version and installed packages

**2.5.1.3 Data Assumptions** - These assumptions capture the assumption that user input data will be consistent with the database schema. The data assumptions are crucial because they will affect the reliability of the content presented by the application.

- Data distribution assumption - To successfully allocate enough resources for the app’s core functionality, an underlying assumption has been made regarding the relevant data’s distribution. A large deviation from this assumption would negatively impact performance.

- Homoscedasticity assumption - A core assumption of PyFamilyTree is the assumption that the variance between users for given numeric inputs will be well defined. This assumption is important because a highly anomalous value is likely to correspond to user entry error.

**2.5.1.4 Usage Assumptions** - These assumptions define the expected user behavior, as defined by the application’s terms of service. Core use-case functionality includes:

- Users interacting with the application to trace their family lineage and create a visual representation of their family tree

- Approved user addition/edit/deletion of family tree-related information

- Approved user sharing of information/media files with their family

- User expectation for intuitive design and ease-of-use.

- User variation in technological proficiency which may require different levels of support and guidance.

**2.5.1.5 Convention Assumptions** - These assumptions refer to the standard or common practices followed by the development team in the application’s implementation. These assumptions rely on conventions or established design principles of other family tree applications and generally similar software.

*-* Commonly used symbols and icons to ensure ease-of-use. This includes familiar user interface elements, such as buttons, menus, and navigation controls that are commonly used in similar applications.

- Standardized terminology (as is defined in the definitions section). These conventional genealogical terms include (but are not limited to) ancestor, descendant, spouse, etc.

- Appropriate security measures, including password protection & data encryption.

- Standard Family Tree representation/ visualization structure (i.e. the oldest generation at the top of the chart, and the youngest at the bottom)

**2.5.1.6 User Language Assumption** - This assumption relates to the language skills/capabilities of the users of the PyFamilyTree application. A lack of language fluency in supported languages would complicate interaction with the app.

**2.5.1.7 Feature Independence Assumption** - This assumption relates to the independence of software features, guided through modular design and allowing the development team to effectively test features in isolation.

### 2.5.2 Dependencies: The PyFamilyTree application has the following core dependencies:

**2.5.2.1 Database Dependencies** - The application relies on a specific database management system and a well-defined schema to store and retrieve family tree data. Indexing and additional optimization techniques will be utilized to ensure efficient querying.

**2.5.2.2 Operating System Dependencies** - The application is designed to run on the OSX and Ubuntu operating systems. The development team will release periodic updates to ensure compatibility with operating system updates. Attempting to run a local server using other operating systems is not supported by the development team.

**2.5.2.3 Web Framework Dependencies** - The application is web-based; so it will require a compatible web framework for the server to run. Django has been selected as the default web framework.

.

**2.5.2.4 Library Dependencies** - The application’s functionality depends on numerous third-party libraries, especially in the Python & JavaScript programming languages. These libraries will provide core functionality, such as data visualization and authentication. The usage of these libraries will be carefully tested to ensure correct compatibility.

**2.5.2.5. API Dependencies** - The application will support both internal and external API (Application Programming Interface) integrations, for both internal communication and interaction with additional services. Internal APIs will be thoroughly tested to ensure reliability.

# Specific Requirements

## 3.1 Interface Requirements - Database

## 

## 3.2 User Stories

The user stories below highlight system functionality in the context of value to our end users. Each user story represents specific functionality within the system. Success criteria is used to measure successful implementation of each story.

## 

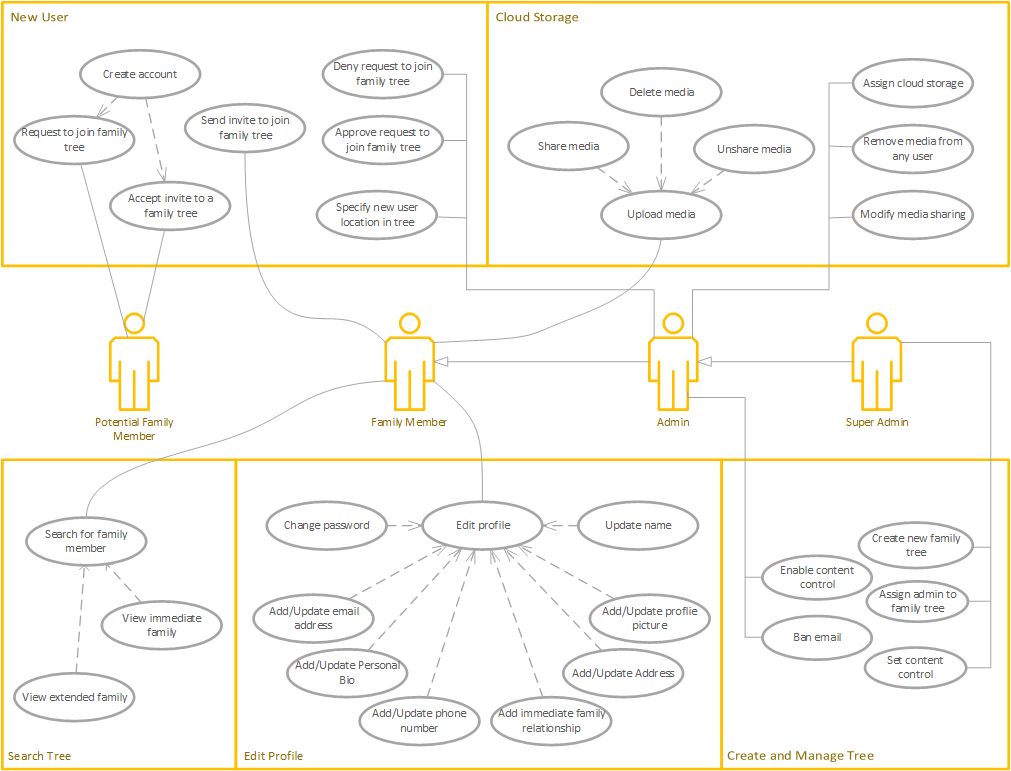
| User Story ID | User Story Name | As a <type of user> | I want to <perform some task> | So that I can <achieve some goal> | Success Criteria |
| --- | --- | --- | --- | --- | --- |
| 1 | New Family Tree Creation | Super admin | I want to create a new family tree | So that I can document my family history and share it with family members | Being able to create a new family - by entering it into the database and accessing it through the GUI |
| 2 | Assigning an Admin to an Existing family | Super admin | Assign an admin to my family tree | So that they can share the responsibility of approving new members within our tree | The admin is successfully assigned to a particular family, and has all necessary credentials/ permissions |
| 3 | Inviting My Family to Our Family Tree | Super admin or admin | Invite my family members to join our tree | So that they can view and contribute to our tree | The admin successfully sent out an email invitation to a new member. This email was received by the target prospective member. |
| 4 | Accepting an Invitation to a Family Tree | Potential member | Accept an invitation from my family tree’s admin to join our family tree | So I can sign up and access our family tree | The potential member was successfully forwarded to the signup page |
| 5 | Requesting to Join a Family Tree | Potential member | Request to join a particular family tree | So I can view and contribute to my family tree | Family tree admin is prompted with a join request |
| 6 | Approving a Request to Join a Family Tree | Super admin or admin | Approve requests from family members to join my family tree | So they can view and contribute to my family tree | A potential member’s request is successfully approved by the admin, adding them to the family tree. . |
| 7 | Denying a Request to Join a Family Tree | Super admin or admin | Deny request to join my family tree | So that only family members I want can view and contribute to our family tree. | A potential member’s request is denied by the admin, rejecting their request. |
| 8 | Blocking an Email Address from Requesting to Join a Family Tree | Super admin or admin | Block an email address from submitting requests to join a particular family tree | So that non-family members and bad actors would be successfully blocked from joining a family tree | A given email address is successfully added to a blocked list (existing in the database), and the email address cannot be used again to request to join. If this email address is used again, an alert is prompted. |
| 9 | Specify the Relationship of a New Member In My Family Tree | Super admin or admin | Specify where a new member belongs in my family tree | So that I can add them to the appropriate location within the tree | A new/existing member now has information in the “father”, “mother”, “spouse”, and “children” fields in the appropriate table, and can be shown correctly in the GUI |
| 10 | Update My Name | Member | Update my name for my record | So that I can contribute accurate information to the tree | The firstName, middleName and lastName fields in the database are updated, can be successfully queried |
| 11 | Adding a Profile Picture | Member | Add a profile picture to my the associate cloud storage and a pointer/link to the database | So family will recognize my record by face | The user’s row in the database successfully points to the correct cloud storage location, and the picture can be viewed by users with permissions to see it. |
| 12 | Adding My Family Members’ Information To My Record | Member | Enter information relating to each family member’s (parents, spouse, children) names into my record | So that I can accurately document my relationship to my family members. | the queried family member is highlighted (via the search), and is added to the user’s information |
| 13 | Changing An Email Address | Member | Change my email address | So that I will update my login information and other family members will have the information necessary to contact me by email. | The member can now use their new email address to log in, and users with permissions can see their new email address |
| 14 | Change Account Password | Member | Change family tree account password | So I could change my password if I’m concerned that it’s been shared or if I forgot my password. | The member’s password is updated, and can be used in a fresh log in. |
| 15 | Update A Temporary Password | Member | to be able to select my own password instead of the default password | So my password will be unique to me. | The member successfully logs in with their new password |
| 16 | Add Phone Number Personal Record | Member | Enter a phone number into my record | So that other family members can contact me by phone if they would like | Phone number field is correctly populated for the member in the database, passes an error check for the correct number of digits, and is viewable by other members of the family tree. |
| 17 | Add Address To Personal Record | Member | Add address | So that other family members have my address for mail, family gatherings, or emergencies. | Physical address is correctly populated for the member in the database, passes an error check for street, city, state, and zip code, and is viewable by other members of the family tree. |
| 18 | Update Personal Bio | Member | Add/update my bio on my record | To have a written description of myself and my relationship to our family | Bio field is updated in the database, and is viewable by other members of the Family tree. |
| 19 | Search for Immediate Family | Member | Find my immediate family | So I can look up their contact information | Immediate family members (mother, father, spouse, and children) are correctly queried and displayed. |
| 20 | Search for Extended Family | Member | Find my extended family and ancestors | So I can learn more about who I am related to. | Extended family members are correctly queried (by textbox/GUI) and displayed. |
| 21 | Search for Individual | Member | Search for a specific family member | So I can view their record and see their connections in the family tree | The individual is displayed correctly, allowing the searcher to notice if the target exists and, if it does, to identify their location in the family tree. |
| 22 | Assign Cloud Storage | Super admin / admin | Assign cloud storage to members | So they can add media to share with our family | Family member’s account includes 5 GBs of cloud storage. |
| 23 | Upload to cloud storage | Member | Upload media to my record | So I can create a record with photos, documents etc | An approved file (.img, .png, .mp4, .pdf, and .docx) is successfully uploaded to the user’s assigned storage. |
| 24 | Share Media through link | Member | Share my media with other family members through a sharable link | So other members can learn more about me through photos and documents. | Shared media is successfully accessed via shareable link by another Family member. Notification sent to the media owner. |
| 25 | Delete Media | Member | Delete my media | If I make a mistake or no longer want that media available to others | The media file has been successfully deleted. |
| 26 | Unshare Media | Member | Make my media unshareable | I am not ready to share with others yet but won’t want to delete | The media file is now unavailable to users other than the source and admins. |
| 27 | Admin update sharing | Admin | Modify sharing credentials for different media files | So the admin would be able to assist family members in correctly sharing their content | The file’s sharing option is successfully modified between share and unshare. |
| 28 | Update Database Directly | Database Administrator | Modify information in the database | So the DBA would be able to ensure data quality | The existing data in the database is modified in an expected behavior, per the DBA’s query. |
| 29 | Content Control | Super Admin | Define acceptable content in the app’s terms of service | To prevent “undesired” content (going against the App’s terms of service) | The application’s terms of service are successfully modified. This modification relates to acceptable content classes.  New and existing members are prompted to agree to the new terms of service. |
| 30 | Family-specific Content Control | Admin | Define acceptable family-specific “content classes” (family-friendly, violence, etc.) in a given family’s shared cloud / information | To prevent “undesired” content (the family’s requirements) in the shared cloud or in user information | A specific file is blocked for violating the family-specific content control rules. |
| 31 | Security-related Upload Filtering | Super Admin/ Admin | Restrict uploaded content to specific files (sizes, extensions, etc.) [no executables or uncompressed files] | To prevent viruses/malware or unnecessary memory bloat (at upload). | A specific file is blocked for violating the terms of the service of the app from a security standpoint. |
| 32 | Admin Removal of Media | Super Admin/ Admin | Delete existing content in a family’s shared cloud storage | To prevent viruses/malware or unnecessary memory bloat (ongoing) | The admin successfully removes a media file from a given user account. |
| 33 | Access Definition for a given Folder/File | Admin / Member | Determine what the other users can do with a particular shared folder or file | To prevent the removal/change of a user’s media | A member/admin successfully defines a folder or a file as read only/write for the rest of their family |

## 3.3 Use Cases

In the following use case sections a system diagram and written descriptions are used to illustrate different actors and their expected use of the system.

### 3.1 System Use Case Diagram

The system use case is divided into five different subsystems, these include New User, Cloud Storage, Search Tree, Edit Profile, and Create and Manage Tree. Each subsystem includes a specific functionality available to one or more actors. These subsystems collectively represent the entire system.



In the following ten sections, specific use cases are labeled and detailed. Use case descriptions include the related actors, a description of the use case, data used, how the use case begins, ends, and any relevant comments.

### Use Case #1 - User login

| Use Case | User login |
| --- | --- |
| ID | 01 |
| Actors | Super Admin, Admin, Family Member |
| Description | All users must login to the PyFamilyTree web application to access or manage Family Trees. |
| Data | The users username and password |
| Stimulus | The user visits the login webpage |
| Response | The user is successfully logged into the application, or if an incorrect username/password combination is entered then an error message is presented to the user. |
| Comments | The user must already have an account to login, otherwise they have the option to request to join a family tree. |

### Use Case #2 - Request to join a family tree

| Use Case | Request to join a family tree |
| --- | --- |
| ID | 02 |
| Actors | Potential Family Member |
| Description | A potential family member may request to join an existing family tree. If an invite is accepted by the admin, the potential family member can create a user account and contribute to the family tree. |
| Data | The requester's first name, last name, email, family tree, and a comment on their relationship to the requested family tree. |
| Stimulus | A request to join form is filled out and submitted by the requester |
| Response | The request to join is received by the family tree admins. Then the admins can decide to approve or deny the request. |
| Comments | The requester must know the name of the family tree they wish to join. |

* + 1. **Use Case #3 - Invite a potential family member to join your family tree**

| Use Case | Invite a potential family member to join your family tree |
| --- | --- |
| ID | 03 |
| Actors | Super Admin, Admin, Family Member |
| Description | A family member or admin can send an email invitation to join their family tree to a potential family member. Entering the family tree will consist of creating a user account and enabling the ability to update profile information that will contribute to the family tree. |
| Data | Potential family member's first name, last name, and email address |
| Stimulus | An admin or family member fills out an invitation form with the potential family member's name and email address |
| Response | The invitee receives a family tree join invitation via email. |
| Comments | A valid email address is required for the invitation to be successfully sent by an admin |

* + 1. **Use Case #4 - Search the family tree for immediate family members**

| Use Case | Search the family tree for immediate family members |
| --- | --- |
| ID | 04 |
| Actors | Super Admin, Admin, Family Member |
| Description | A family member can search for any target member in the family tree and then view queried member's immediate family. If an existing family member is searched, the results should include (if applicable) parents, self, children, in-laws, and grandchildren |
| Data | Family member profile with links to profiles of related family members. Self, parents, children, in-laws, and grandchildren |
| Stimulus | A search is entered into the family tree search box |
| Response | A successful or unsuccessful search result is returned. If the family member is found, their profile will be shown on the screen. If not, an error message stating that the family member could not be found in the tree. |
| Comments |  |

* + 1. **Use Case #5 - Edit profile information**

| Use Case | Edit profile information |
| --- | --- |
| ID | 05 |
| Actors | Super Admin, Admin, Family Member |
| Description | The user profile contains all pertinent information about an individual family member. Other family members can view the profile to learn more about a particular family member. A family member needs to be able to update their profile information so it is accurately reflected in the family tree. |
| Data | First name, middle name, profile picture, address, immediate family, phone number, bio, email address |
| Stimulus | Family members can select Edit profile from their personal profile page. |
| Response | After entering a valid update, changes are saved into the database and visible to any family member that views the updated profile. |
| Comments | Data validity rules apply to each of the fields that can be updated. |

* + 1. **Use Case #6 - Change password**

| Use Case | Change password |
| --- | --- |
| ID | 06 |
| Actors | Super Admin, Admin, Family Member |
| Description | All credentialed users must be able to change their password for security reasons or if they forgot their current password. They can change their password from within the PyFamilyTree application or request a link be sent to their email to change their password. |
| Data | Username and password |
| Stimulus | A user navigates to their profile and selects *change password*. Alternatively, if the user forgot their password, from the login screen they can request to have a request link sent to their email. |
| Response | User is prompted with a successful password change message with a link to go back to the login page. |
| Comments | If a valid email is not entered for the *forgot a password* function, the user will not receive a password reset email. |

* + 1. **Use Case #7 - Assign cloud storage to family members**

| Use Case | Assign cloud storage to family members |
| --- | --- |
| ID | 07 |
| Actors | Super Admin, Admin |
| Description | Admins can allocate cloud storage to family members. Storage can be used by family members to upload media and is directly accessible via their profile after a login |
| Data | Cloud storage space, cloud storage availability |
| Stimulus | Admin creates a new folder and sets appropriate permission so only the admins and assigned family members can modify and upload to the cloud storage space. |
| Response | The family member will see a cloud storage link the next time they log in to the application and view their profile. |
| Comments | Admins can increase storage space per request. |

* + 1. **Use Case #8 - Upload media to assigned cloud storage**

| Use Case | Upload media to assigned cloud storage |
| --- | --- |
| ID | 08 |
| Actors | Super Admin, Admin, Family Member |
| Description | Family members with assigned cloud storage can upload media files to their storage space. This includes photos, videos, and audio recordings. |
| Data | Media file, photo, video and audio files |
| Stimulus | A user opens their cloud storage and selects to upload a new media file |
| Response | The media file is uploaded to their cloud storage space if the selected file is valid. Otherwise, an error message explaining which file types are valid for upload is presented to the user. |
| Comments |  |

* + 1. **Use Case #9 - Edit sharing settings on uploaded media**

| Use Case | Edit sharing settings on uploaded media |
| --- | --- |
| ID | 09 |
| Actors | Super Admin, Admin, Family Member |
| Description | Family members can select individual media files and set sharing settings. Media files can either be private or shared with all family members. Shared media files are found in a family member's profile. |
| Data | Media file |
| Stimulus | A family member selects a media file from their shared storage and sets the sharing settings. |
| Response | The sharing settings are updated, and the visibility of the media file to other family members is immediately changed based on the new selection, share or unshare. |
| Comments |  |

* + 1. **Use Case #10 - Assign an admin to the family tree**

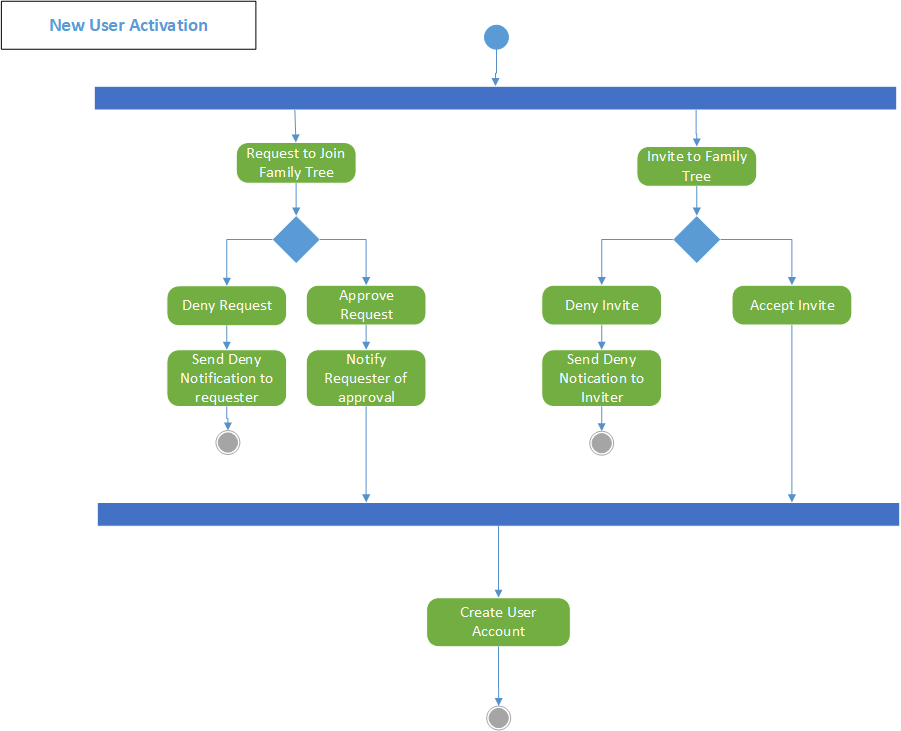
| Use Case | Assign an admin to the family tree |
| --- | --- |
| ID | 10 |
| Actors | Super Admin |
| Description | Super admins can assign admins to a family tree. An admin will have the ability to invite and remove users from a family tree. They can also assist with sharing/unsharing media content for individual family members. |
| Data | Username of family member to be promoted to admin |
| Stimulus | Super admin can select any standard user from the user administration page and promote them to an admin |
| Response | The promoted user receives an alert that they have be given admin access to the family tree. They now can login to the family tree admin page. |
| Comments | Admins are critical in ensuring the family tree can be updated and joined by any family members. |

### 

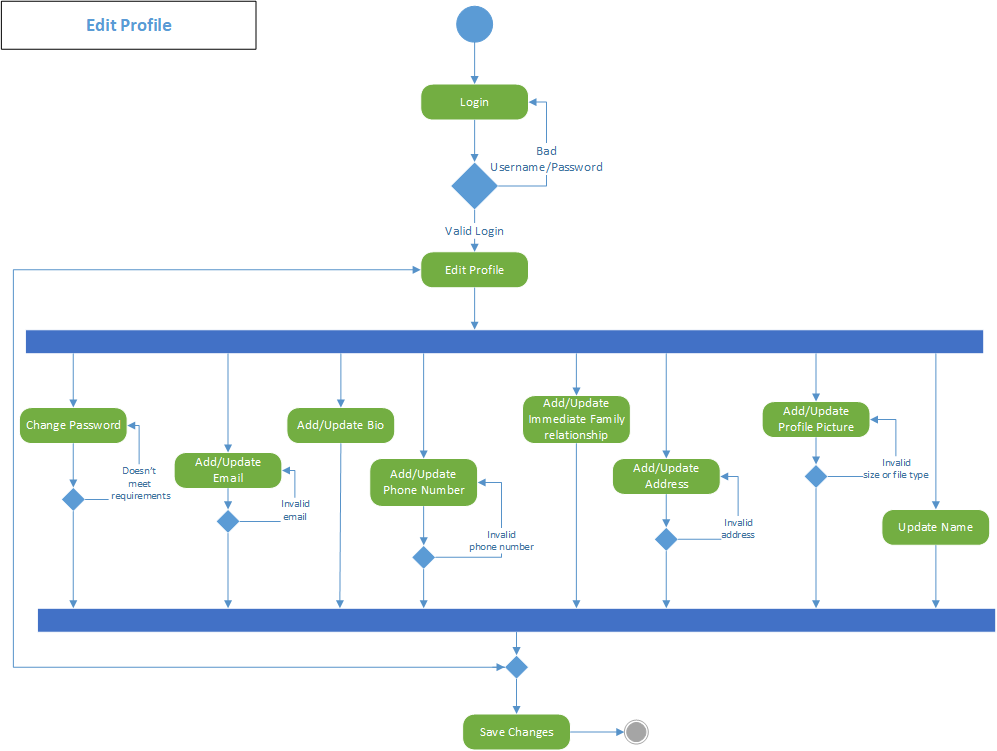
### 3.3 Activity Diagrams

The following activity diagrams visualize the major system functionally in the context of actions within the system. Actions are represented by a green icon and may be followed by another action or a decision. When a decision is reached, the system allows the flow by selecting one of the available actions.

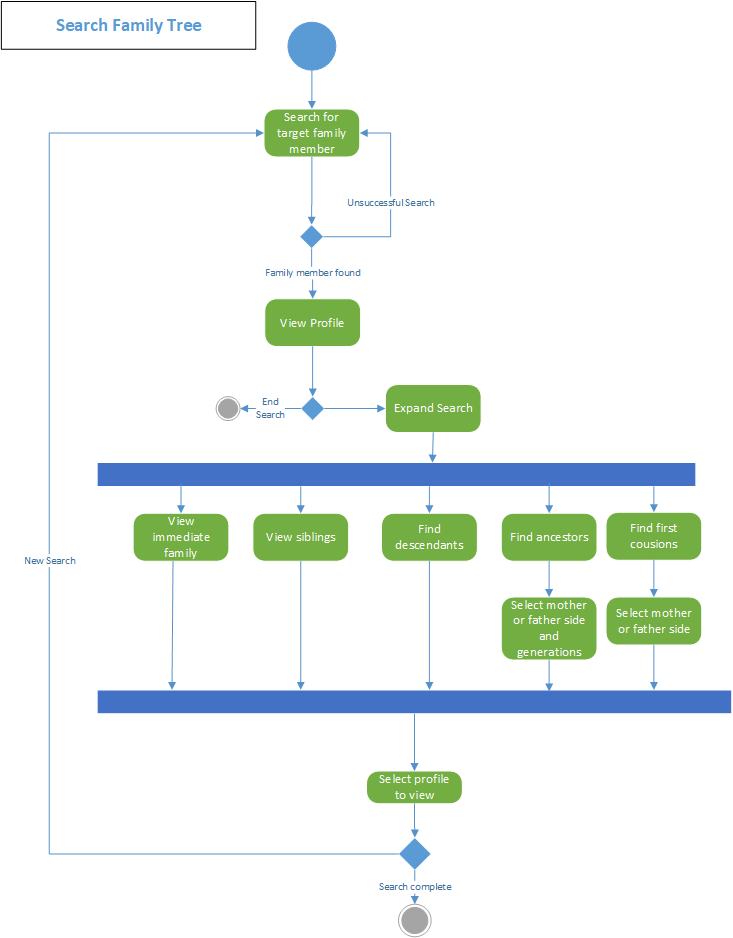
**3.3.1 New User Activation**

****

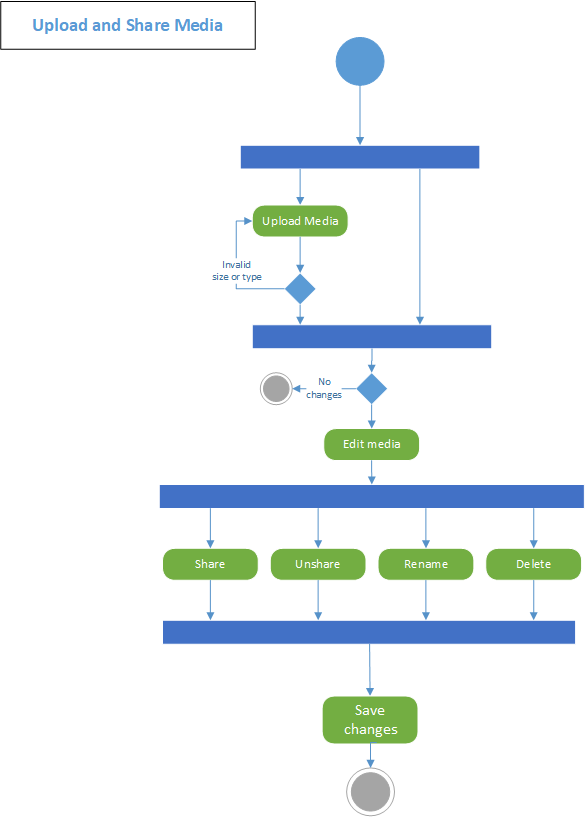
**3.3.2 Edit Profile**

****

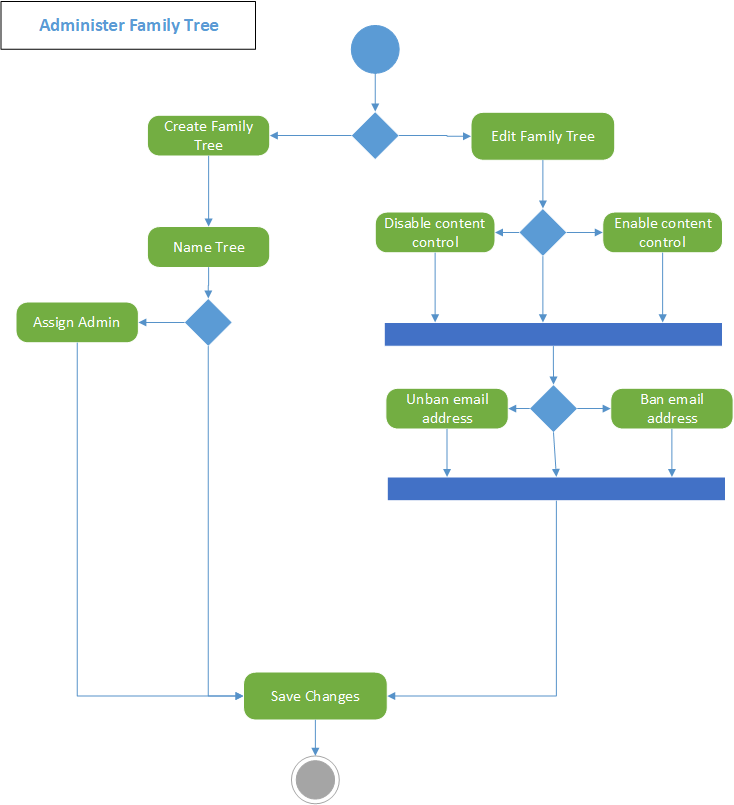
**3.3.2 Search Family Tree**

****

**3.3.3 Upload and Share Media**

****

**3.3.4 Administer Family Tree**

****

# Other Non-functional Requirements

## 4.1 Performance Requirements

The application must perform under high traffic conditions, and is designed to handle a large number of family trees, members, and their requests and queries. Load balancing, caching, and software optimization will be performed to ensure that the app can handle traffic without slowdown or timeout issues. Code and service test profiling and paging would ensure compliance of these performance requirements.

## 4.2 Safety and Security Requirements

PyFamilyTree is designed with security in mind, with a focus on preventing unauthorized access to private user data. This is performed via authentication and authorization mechanism to ensure only authorized users can access the app, and unique family tree data. Member data will be kept confidential and is not to be shared with any unauthorized third par ties. To ensure the security of member data, measures such as data encryption, secure storage of sensitive data, and secure communication protocols between app components will be taken.

## 4.3 Usability Requirements

The application is designed to be user-friendly and intuitive for all users. The user interface is designed to be straightforward to navigate, with clear and simple instructions. Moreover, a simple introductory video will be published shortly following the launch of the first version of the app. Thereafter, the application would be appropriate for users of varying levels of technical proficiency. User testing & surveying will ensure effective usability per this requirement.

## 4.4 Software Quality Attributes

PyFamilyTree will be reliable, scalable, and maintainable, and software decisions have been made to support that aim. Moreover, the app is designed to be easy to maintain and update via periodic patches. To that end, the code will be well-documented and will adhere to established coding standards - PEP8 for the Python backend and JSLint for the JS frontend. The app is also designed to be scalable and to accommodate new features. Underwriting software quality will be a Pytest-based test suite which will be developed alongside the software in a Test-Driven Development methodology, ensuring that the application will be of high quality and the standard defined in this SRS.

## 4.5 Maintainability Requirement

PyFamilyTree is designed with maintainability in mind, as new features and updates are planned to be released in the future., To ensure that the app will remain easy to maintain and update, the code will be well-organized and well-documented (see 4.4), with clear naming conventions and comments. Software modularity will be employed in development, allowing for modular updates without affecting the rest of the codebase. PyFamilyTree will undergo regular testing, including unit testing whenever a new build is pushed, to ensure that new code won’t negatively impact the existing code. Following the first launch, the development team will release a detailed user manual to help future developers understand the underlying code and architecture.