Software Requirements Specification

for

Audio Basket

Prepared by

Prosanto Deb ASH1925005M
Md. Armanur Rashid ASH1925013M
Md. Rokonuzzaman ASH1925018M
Arnab Dey ASH1925023M
Md. Rayhan Billah ASH1925028M

Institute of Information Technology
Noakhali Science and Technology University

10.04.2022

List of Figure

List of Tables

1. Introduction

The policy, scope, references, and summary of Software Requirements Specification (SRS) are all included in the SRS introduction. By presenting the problem statement in detail, the purpose of this document is to collect, evaluate, and provide a deeper understanding of the whole "Audio Basket" system. While it defines high-quality product features, it also focuses on the strengths that participants need and their needs. This page contains detailed "Audio Basket" information.

1.1 Problem Statement

There are different types of music players based on desktop or Android where everyone listens to audios or sees videos. Users can only listen to audio /watch video on existing players. Can't give friends access to files stored in their own player. Also, users can't share files with anyone without using any other app. If the music player is uninstalled or the mobile is lost, the files stored by the user can no longer be recovered. This is a matter of great concern for the user. Also, if any file is placed in the player, it takes up the storage of the user's device. Our application works as a solution to these problems. Users can "Compressed Size (90%)" when they store files in the application. Users can find these files even if the device is lost using their account and can share any file or give access to a folder to other accounts.

1.2 Purpose

The purpose of this document is to collect and analyze all the various ideas that have emerged in order to develop the program and its standards for alumni members. Also, to get a better idea of the project, describe the ideas that may be developed later, and write down ideas that are being considered but which can be eliminated as the system progresses, we will predict and plan how we expect this plan to be implemented.

In short, the purpose of this SRS document is to provide a comprehensive overview of our software product, including its specifications and objectives. This document describes the user interface, hardware, and software requirements, as well as its partners. Specifies how the system and its functions are viewed by administrators, alumni students, and general members. However, it assists any designer or developer in software delivery processes (SDLC).

1.3 Project Scope

This project traverses a lot of areas ranging from business concept to computing field and is required to perform several researches to be able to achieve the project objectives. The following are the scopes of work:

- Developing a pc and smartphone-based application as around 97% of users use smartphones nowadays.

- Studying the existing systems and learning their weakness hence developing a new system to cater for the challenges the local and world domains face when dealing with house rental issues

1.4 Glossary

This section provides definitions for all document names, acronyms, and abbreviations. The application domain's terms and concepts are defined.

GUI - Graphical User Interface
API – Application Programming Interface
SRS – Software Requirement Specification
UI – User Interface
SDLC – Software Development Life Cycle
MB – Megabytes
XML – Extensible Markup Language
RESTful – Representational State Transfer
HTML – Hyper Text Markup Language

1.5 References

IEEE. *IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications.* IEEE Computer Society, 1998.

1.6 Overview

Nowadays people spend their leisure time sitting at home by watching videos or listening to music. Also watch videos or hear music while traveling or for entertainment. Music has the ability to change people's emotions and feelings in a matter of seconds. It has the ability to reduce stress, discomfort, difficulty, and distraction while also bringing happiness and peace into our daily lives. Many people also listen or watch different files for academic or official work. For this, they use different existing music players. Users face many problems while using existing music players. Music is widely acknowledged as humanity's universal language. It has the ability to bring happiness and positivity into people's lives. Sometimes a user wants to share his favorite music files with his friend. The user can't share the file of his choice with his friends without using other apps. No folder of his choice can be given to his friend in a moment. Even a friend in a remote place can't hear the files on his mobile. If you put any file in the existing music player, it takes up the storage of the user's device and decreases the storage of the user's device. In addition to destroying the device's storage, if the user's device is damaged, or lost, or if the apps are uninstalled, the stored files can no longer be recovered. Various useful files can then be lost. Which can cause a lot of damage to a user.

By using our application, all problems will be solved. The user will be able to see all options beautifully after logging in. Users do not need to have any prior knowledge to use this application. No need to watch any tutorials to use this application. Users will be able to share files with other users without using any third-party apps. The user can give access to any folder to another user

through an access token. Sometimes if a user has uploaded videos of different academic courses to his account, then if another student has access to his folder, then other students will be able to see the videos kept in his account. Usually, the size of the video is much larger. When uploading large size videos, a lot of storage is lost. If someone uploads a video in our application, two options will come. Upload to original size or compressed upload. Compressing and uploading will reduce the file size by up to 80%. Even if the user's device is lost, there is no problem. If the user logs in to his account with any other device, he will get all the uploaded files.

2. Stakeholders and Characteristics

2.1 General User

We will have some general users in our application. General users will initially get 1GB storage. They will not be able to use more until they shift to the premium user. General users will compress the file up to 30-40% at the time of uploading the file.

2.2 Premium User

We will have some premium users in our application. Premium users will have unlimited storage. If they want, they can upload the file compressing 70-80%. Besides, they will get many more opportunities.

2.3 Developer

To be classified as stakeholders, persons who have a legitimate interest in the product or project must have legitimate interests. Because the developers have too much at stake, they are deemed stakeholders because they have a legitimate interest. As stakeholders, developers are responsible for software delivery and estimation on time.

3. Design and Implementation Constrains

We have employed design and implementation constraints to ensure the success of this project. It also refers to a tool that allows developers and testers to inspect and interact with the application's user interface (UI) elements.

3.1 JavaScript, JSX and React.js

The visual layout of the components that a user could interact with in a website or technical product is referred to as user interface design, or UI design. In other terms, it is a website's visual design.

3.1.1 Programming Language

JavaScript is an ECMAScript-compliant high-level, frequently just-in-time compiled language. It has first-class functions, dynamic typing, and prototype-based object orientation. It's multi-paradigm, allowing you to program in event-driven, functional, or imperative styles.
React is a front-end JavaScript toolkit for creating user interfaces using UI components that is free and open-source. Meta and a community of individual developers and businesses support it.
JavaScript XML is abbreviated as JSX. It's just a JavaScript syntactic extension. It allows us to create HTML directly in React (within JavaScript code). It is straightforward to generate a template in React using JSX, but it is not a simple template language; instead, it has all of JavaScript's capability.

It is faster than standard JavaScript because it optimizes when converting to standard JavaScript. Rather than dividing the markup and functionality in different files, React makes use of components.

3.1.2 CSS Framework

Cascading Style Sheets (CSS) is a language for specifying the appearance of a document written in a markup language like HTML. Along with HTML and JavaScript, CSS is a key component of the World Wide Web. Semantic UI is a website using UI component framework. Developers may use Semantic UI to create websites with quick and clear HTML, as well as a fully mobile responsive experience. Semantic UI offers a React-integrated version called Semantic UI React, which includes the following functionalities:

- jQuery Free.
- Declarative API.
- Augmentation.
- Shorthand Props.
- Sub Components.
- Auto Controlled State

3.2 Server-Side Technology

Server-side development refers to the actions that take place behind the scenes when an application is used. It primarily focuses on databases, scripting, website architecture, backend logic, APIs, and Servers.

3.2.1 Python, Django Rest API

Python is a dynamically semantic, interpreted, object-oriented high-level programming language. Its high-level built-in data structures, together with dynamic typing and dynamic binding, make it ideal for Rapid Application Development and as a scripting or glue language for connecting existing components. Python's concise, easy-to-learn syntax prioritizes readability, which lowers software maintenance costs.

Python is widely used to create an application's back end. It should come as no surprise that there are a variety of Python frameworks available to help with server-side programming. The Django REST Framework (DRF) is a popular, robust, and versatile framework for creating Web APIs.

To get data from our server to our application, we'll use the Django Rest API. Representational State Transfer is abbreviated as REST. Application Programming Interface is what API stands for.

3.2.2 Database Server

PostgreSQL is an advanced, enterprise class open source relational database that supports both SQL (relational) and JSON (non-relational) querying. PostgreSQL's speed, security and robustness make it suitable for 99% of applications, so it's a great starting place for our application. It is a dependable, powerful, and stable solution with sophisticated features such as the following:

- User-defined types.
- Table inheritance.
- Sophisticated locking mechanism.
- Foreign key referential integrity.
- Views, rules, subquery.
- Nested transactions (save points)
- Multi-version concurrency control (MVCC)
- Asynchronous replication.

3.2.3 Cloud Storage

Amazon S3 is a type of object storage that allows to store and retrieve any quantity of data from any location. It's a straightforward storage solution with industry-leading durability, availability, performance, security, and scalability. So we will use S3 to store audio files.

4. Requirement Specification

All the requirements based on the elicitation process are described in this section.

4.1 Functional Requirement

Functional requirements are those requirements that are used to illustrate the internal working nature of the system, the description of the system, and explanation of each subsystem. It consists of what task the system should perform, the processes involved, which data the system should hold and the interfaces with the user.

4.1.1 User login and register

FR-1	User Registration and Login to registered account.		
Description	User should register his/her account for the first time and be able to login to the account which was registered once. Already registered users will not face this stage.		
Stakeholders	General User, Premium User	**Priority**	High

4.1.2 Create new folders

FR-2	Create new folders to store files.		
Description	Users can create a new folder. It is a good practice that some users want to have individual files organized.		
Stakeholders	General User, Premium User	**Priority**	Low

4.1.3 Upload their audio files

FR-3	Upload files to the “Audio Basket” application.		
Description	Users need to upload files to get the benefits of file compression.		
Stakeholders	General User, Premium User	**Priority**	High

4.1.4 Play a previously uploaded audio file

FR-4	Play a previously uploaded audio file from the application.		
Description	Users can play any audio file at any time. Audio files may be synced or downloaded before or stored in the server.		
Stakeholders	General User, Premium User	**Priority**	Low

4.1.5 Search and filter audio files

FR-5	Search and filter audio files according to user preference		
Description	Users can search a specific audio file using a search box and the system will provide results for search results.		
Stakeholders	General User, Premium User	**Priority**	Medium

4.1.6 Download the audio files

FR-6	Download the audio files for offline use.		
Description	For ease of user involvement, users can download audio files to the local directory and play them whenever he/she want.		
Stakeholders	General User, Premium User	**Priority**	Medium

4.1.7 Buy more storage

FR-7	Buy more storage when storage limit exceeds		
Description	Users can buy more space if they want. Initially, the system will provide 1GB of server space to every user free of cost. Then a decent amount of currency will be charged for every extension package.		
Stakeholders	Premium User	**Priority**	High

4.1.8 User can see how much storage they have left

FR-8	Progress bar to indicate remaining or consumed space.		
Description	A progress bar will be shown along with the percentage of storage users have used to users consumed page.		
Stakeholders	General User, Premium User	**Priority**	High

4.1.9 Share a file

FR-9	Users can share a file.		
Description	A user can share a file in his account to another “Audio Basket” user.		
Stakeholders	General User, Premium User	**Priority**	High

4.1.10 Delete files or folders

FR-10	Delete single or multiple files.		
Description	A single or several files can be permanently deleted by the user. The user can place it in the "Trash Bin" without permanently removing it if he/she so desires. It will be permanently erased after 30 days if he/she does not restore it during that time frame.		
Stakeholders	General User, Premium User	**Priority**	Medium

4.1.11 Share folder access

FR-11	Users can give access to their folders.		
Description	Using an Access Token, a user can provide another "Audio Basket" user access to a certain folder on his account. The other user will be able to access the files in his account folder.		
Stakeholders	General User, Premium User	**Priority**	High

4.1.12 User logout from their account

FR-12	Log out from their account.		
Description	The user will be able to log out of his account at the end of his need. Users will need to login again for later use.		
Stakeholders	General User, Premium User	**Priority**	Medium

4.2 Data Requirement

For our application we have to store many audio files. As a result, we'll need a huge amount of server storage, which will cost a lot. To mitigate this storage costing we will compress user uploaded music files by a maximum of 80% (For example 100mb to 20mb) using lossless compression algorithms and then store them to the server.

4.2.1 Data compression

DR-1	Data compression using lossless compression algorithm		
Description	Users' uploaded music files will be compressed by using lossless compression algorithms and then stored on the server. This algorithm shrinks the audio file by removing frequencies that are not in human capacity.		
Stakeholders	General User, Premium User, Developers	**Priority**	High

4.3 Performance Requirement

It is important to maintain the performance of the software system. To ensure performance we maintain these steps:

4.3.1 Higher speed and low latency while playing

PR-1	File will play with less buffering and load faster.		
Description	File will load faster and play with less buffering, because of the compression the application uses while uploading files		
Stakeholders	General User, Premium User, Developers	**Priority**	High

4.3.2 Capacity Requirements

This system can load up to thousands of tenant's information and thousands of advertisements information.

4.3.3 Safety Critical Requirement

There are no safety critical requirements for our project.

4.3.4 Robustness or Fault-Tolerance Requirements

There are no **Safety Critical Requirement**s for our project.

4.4 Maintainability and Supportability

4.4.1 Maintenance Requirements

MR-1	Make the code maintainable.		
Description	Code must be developed so that it can be modified later and will be readable.		
Stakeholders	Developers	**Priority**	High

4.4.2 Supportability Requirements

This system meets Testability, Maintainability, Compatibility, Configurability, Serviceability, and install ability which are related to supportability requirements.

4.5 Security Requirements

Securing information is much more important for a system to get users dependability. Here are some of them:

4.5.1 Access Requirements

For accessing information, the system will use some authorization techniques to ensure that correct data is used by the correct user.

4.5.2 Integrity Requirements

Integrity requirements refer to a security system which ensures an expectation of data quality. It also ensures that all data of the system would never be exposed to malicious modification or accidental destruction. For preventing anonymous access to user passwords, the system will use an encryption technique called Hash Function for encrypting user passwords.

4.5.3 Privacy Requirements

Privacy requirements are enhanced to protect stakeholder's privacy. In this way, all data or a partial part of data is going to be disclosed according to the system's privacy policy. To ensure privacy, the central database should be protected by the anonymous. Users are permitted to get access to those data which are being associated with them which can be ensured by the user login system.

4.6 Usability and Human Integrity Requirements

This system will provide more user-friendly environment

4.6.1 Ease of Use Requirements

Our system will be easier to use by any type of people and they don't need any training to use the system.

4.6.2 Accessibility Requirements

To get access to the application, the application provides authorization/authentication. This application uses various modules.

SR-1	The system provides security strategies.
Description	The system is designed in a way that allows all modules to access a mechanism that provides security services.

4.7 Look and Feel Requirements

Look and feel requirements mainly refer to how the system will look.

4.7.1 Appearance Requirements

AR-1	Text color and font		
Description	Our system has to be different and attractive from other existing audio players using a better look and feel.		
Stakeholders	Developer, User	**Priority**	High

4.8 Style Requirements

There are no style requirements in our system.

4.8 Legal Requirements

Legal requirements normally refer to the terms and conditions or privacy policy of any organization. The terms and condition of our application is that, no third-party software or person is allowed to use our data for their business purpose.

5. Requirement Engineering Process

Requirements Engineering (RE) determines software requirements according to customer requirements or needs. Requirements engineering process includes requirements elicitation, needs modeling, requirements analysis, requirements assurance & validation, and requirements management.

5.1 Requirement Elicitation Techniques

Requirements elicitation is the practice of researching and finding system requirements for users, customers, and other stakeholders, also referred to as "requirement gathering". Requirement elicitation can be done by contacting participants directly or by doing some research, analysis and testing.

5.1.1 Hold Interviews

We hold discussions that can be held individually or with a small group of participants. They are an effective way to access services without spending a lot of time with participants because we meet with people to discuss only certain important requirements of this program. Negotiations are useful for obtaining individual requirements for members in organizing workshops where those members of the program come together to resolve any issues or conflicts. We mainly perform our interview based on some specific criteria.

- Short description about goals and objectives
- Registration process
- Searching Audio Files
- Storage system of each account
- Compression size of audio files

5.1.2 Perform Document Analysis

Existing documentation can help to show how systems are currently operating or what they are what I should do. Documents include written information about current programs, business processes, needs specifications, and competitor research. Review once textual analysis can help

determine which performance should remain and functionality that isn't in use. After existing document
In analysis, we found several problems with the existing system.

- Existing systems cannot perform file compression.
- A user cannot share a file with others.
- No cloud storage system is provided by the existing systems.

5.1.3 System Interface Analysis

The first thing to do is to identify which systems the system-to-be shall communicate with. It could be a server on the Internet, a piece of software on the same host as the system-to-be, some hardware or something completely different.

5.1.4 Distribute Questionnaires

The questionnaire is a useful way to investigate styles, changes in attitudes and users' ideas, and user satisfaction with priorities and preferences. Our lists of questions were as short as possible. The respondent may be tired or frustrated. Had a basic reason for all the questions as well as group the topic areas together for the respondent to focus on. The main advantage of this survey responses was that they were collected in the usual way. Information was summarized by a large number of people.

5.2 Requirement Validation

Requirement validation ensures that the requirements are correct and reflect the quality you want from this program. In the beginning, our requirements looked good but when we read them and tried to work with them they came out having ambiguities and gaps.

5.2.1 Review the Requirements

Negative peer review, especially the type of rigorous review called evaluation, is unique among the highest quality software processes available. We had a team of reviewers representing different perspectives and carefully examined written needs, analysis models, and related information on disability.

5.2.2 Test the Requirements

The test creates another view of the requirements. We also performed writing tests regarding assurance of whether the expected performance was found or not. Getting tested by the user needs to document the expected product behavior under specified conditions.

5.2.3 Simulate the requirements

To stimulate requirements, trading tools are available that we have used to simulate a proposed system in place or to add details of written requirements. The simulation takes prototyping to the next level.

6. Use Case Diagram

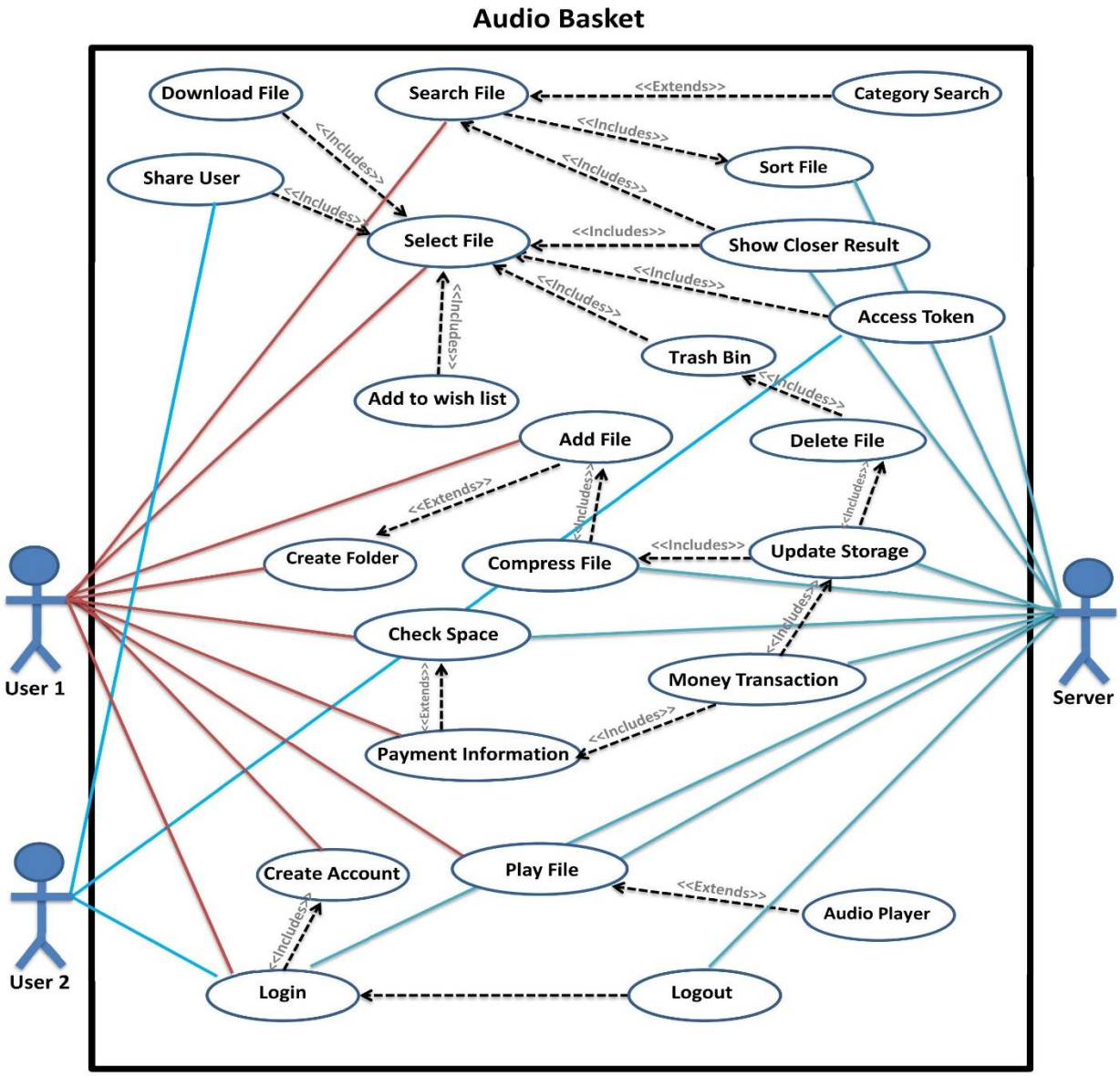

Figure 1: Usecase Diagram

7. Use Case Description

Table 1: Create Account

Use Case	Create Account	
Goal <a longer statement of the goal in context if needed>	User wants to create an account in the "Audio Basket" application.	
Preconditions <what we expect is already the state of the world>	N/A	
Success End Condition <the state of the world upon successful completion>	A user account is created.	
Failed End Condition <the state of the world if goal abandoned>	User account is not created.	
Primary Actors: **Secondary Actors:**	User Application Server	
Trigger <the action upon the system that starts use case>	"Create Account" button needs to be clicked.	
Main Success Flows <the steps of the scenario from trigger to goal delivery and any clean up after>	**Step**	**Action**
	1	User opens the "Audio Basket" application.
	2	User clicks the "Create Account" button.
	3	User provides username, email, password, Profile Picture, payment information (Optional).
	4	Server checks information and provides confirmation.
	5	Account is created.
Alternative Flows <a: condition causing branching> <a1: action or name of sub-use case>	**Step**	**Branching Action**
	2a	User does not have an account
	2a1	User needs to create one.
	2b	User does not login with an existing account.
	2b1	User needs to create another account
	4a	Server shows that information invalid or used before.
	4a1	User needs to change provided information.
Quality Requirements	**Step**	**Requirement**
	3	Username and email must be unique.

Table 2: Log in

Use Case	Login	
Goal <a longer statement of the goal in context if needed>	User login to “Audio Basket” application.	
Preconditions <what we expect is already the state of the world>	User Account.	
Success End Condition <the state of the world upon successful completion>	Successfully login to “Audio Basket” application.	
Failed End Condition <the state of the world if goal abandoned>	Unable to login.	
Primary Actors: **Secondary Actors:**	User Application Server	
Trigger <the action upon the system that starts use case>	“Login” Button needs to be clicked.	
Main Success Flows <the steps of the scenario from trigger to goal delivery and any clean up after>	**Step**	**Action**
	1	User opens the “Audio Basket” application.
	2	User provides username and password.
	3	Server confirms the password with that username.
	4	User login successful
Alternative Flows <a: condition causing branching> <a1: action or name of sub-use case>	**Step**	**Branching Action**
	2a	User does not have an account.
	2a1	User clicks the “Create Account” button to create an account.
Quality Requirements	**Step**	**Requirement**
	2	Users must have an account.

Table 3: Search File

Use Case	Search File or folder	
Goal <a longer statement of the goal in context if needed>	User wants to search a file or folder.	
Preconditions <what we expect is already the state of the world>	N/A	
Success End Condition <the state of the world upon successful completion>	User gets his/her desired file or folder.	
Failed End Condition <the state of the world if goal abandoned>	The application server is unable to provide the desired file or folder and shows "Not Found"	
Primary Actors: **Secondary Actors:**	User Application server	
Trigger <the action upon the system that starts use case>	The "Search box" needs to be clicked.	
Main Success Flows <the steps of the scenario from trigger to goal delivery and any clean up after>	**Step**	**Action**
	1	The user clicks the search box and provides desired keywords.
	2	Application server shows maximum possible results according to the keyword.
	3	User looks for his/her desired file or folder.
Alternative Flows <a: condition causing branching> <a1: action or name of sub-use case>	**Step**	**Branching Action**
	3a	Application server failed to show the desired result
	3a1	User provides another keyword
	3a2	User stops searching
Quality Requirements	**Step**	**Requirement**
	3	Users must provide appropriate keywords.

Table 4: Select Category

Use Case	Select Category	
Goal <a longer statement of the goal in context if needed>	User wants to select a category.	
Preconditions <what we expect is already the state of the world>	N/A	
Success End Condition <the state of the world upon successful completion>	User selected a category while searching.	
Failed End Condition <the state of the world if goal abandoned>	User unable to select category.	
Primary Actors: **Secondary Actors:**	User Application server	
Trigger <the action upon the system that starts use case>	"Select Category" dropdown box needs to be clicked.	
Main Success Flows <the steps of the scenario from trigger to goal delivery and any clean up after>	**Step**	**Action**
	1	The user clicks the search box.
	2	User selects a category and provides a search keyword.
	3	User's desired category is selected.
Alternative Flows <a: condition causing branching> <a1: action or name of sub-use case>	**Step**	**Branching Action**
	N/A	N/A
Quality Requirements	**Step**	**Requirement**
	2	Application must contain some category.

Table 5: Sort File

Use Case	Sort File	
Goal <a longer statement of the goal in context if needed>	Application server wants files or folders sorted.	
Preconditions <what we expect is already the state of the world>	Search or select files or folders.	
Success End Condition <the state of the world upon successful completion>	Files or folders get sorted.	
Failed End Condition <the state of the world if goal abandoned>	Application server shows error and files could not be sorted.	
Primary Actors: **Secondary Actors:**	Application Server N/A	
Trigger <the action upon the system that starts use case>	User searches for a file or selects some files.	
Main Success Flows <the steps of the scenario from trigger to goal delivery and any clean up after>	**Step**	**Action**
	1	User clicks the search box and provides desired keywords or selects some files.
	2	Application server sorts possible results by name.
Alternative Flows <a: condition causing branching> <a1: action or name of sub-use case>	**Step**	**Branching Action**
	N/A	N/A
Quality Requirements	**Step**	**Requirement**
	3	Users must provide appropriate keywords,
	3	Users must select some files to be sorted.

Table 6: Show Closer Result

Use Case	Show Closer Result	
Goal <a longer statement of the goal in context if needed>	Application server shows closer results according to the keyword from the search box.	
Preconditions <what we expect is already the state of the world>	Search for a file or category.	
Success End Condition <the state of the world upon successful completion>	Application server provides closer results.	
Failed End Condition <the state of the world if goal abandoned>	Application server fails to provide closer results.	
Primary Actors: **Secondary Actors:**	Application Server N/A	
Trigger <the action upon the system that starts use case>	User provides keywords in the search box.	
Main Success Flows <the steps of the scenario from trigger to goal delivery and any clean up after>	**Step**	**Action**
	1	User clicks the search box and provides desired keywords.
	2	Application server provides the possible result.
Alternative Flows <a: condition causing branching> <a1: action or name of sub-use case>	**Step**	**Branching Action**
	N/A	N/A
Quality Requirements	**Step**	**Requirement**
	3	Users must provide appropriate keywords.

Table 7: Select File

Use Case	Select File	
Goal <a longer statement of the goal in context if needed>	User selects any file.	
Preconditions <what we expect is already the state of the world>	Login Search a file Enter into any folder	
Success End Condition <the state of the world upon successful completion>	Specific files or folders get selected.	
Failed End Condition <the state of the world if goal abandoned>	Files or folders failed to get selected.	
Primary Actors: **Secondary Actors:**	User Application Server	
Trigger <the action upon the system that starts use case>	User clicks on a file	
Main Success Flows <the steps of the scenario from trigger to goal delivery and any clean up after>	**Step**	**Action**
	1	User clicks any file.
	2	File gets selected.
Alternative Flows <a: condition causing branching> <a1: action or name of sub-use case>	**Step**	**Branching Action**
	N/A	N/A
Quality Requirements	**Step**	**Requirement**
	2	Clicked element should be a file.

Table 8: Share user

Use Case	Share User	
Goal <a longer statement of the goal in context if needed>	User wants to share a particular directory/file with others	
Preconditions <what we expect is already the state of the world>	A directory/file must be selected	
Success End Condition <the state of the world upon successful completion>	User shared his/her directory/file with the desired person.	
Failed End Condition <the state of the world if goal abandoned>	The desired person's account was not found.	
Primary Actors: **Secondary Actors:**	User Server	
Trigger <the action upon the system that starts use case>	User typed a username in the select user input box.	
Main Success Flows <the steps of the scenario from trigger to goal delivery and any clean up after>	**Step**	**Action**
	1	User types username in the user input box.
	2	The desired account comes up with a name and profile picture.
	3	User clicks on that account.
	4	A pop up comes up for confirmation.
	5	The user clicks on "yes".
	6	The desired account will be associated with that directory/file.
Alternative Flows <a: condition causing branching> <a1: action or name of sub-use case>	**Step**	**Branching Action**
	1a	The desired account was not found.
	1a1	User provides another username.
	2a	User does not confirm the pop up.
	2a1	User goes back.
Quality Requirements	**Step**	**Requirement**
	1	The desired account comes up within a second.

Table 9: Download File

Use Case	Download File	
Goal <a longer statement of the goal in context if needed>	User downloads a file	
Preconditions <what we expect is already the state of the world>	Select a file	
Success End Condition <the state of the world upon successful completion>	User gets the desired file to his/her local storage	
Failed End Condition <the state of the world if goal abandoned>	Download process was interrupted. User does not have the file.	
Primary Actors: **Secondary Actors:**	User Application Server	
Trigger <the action upon the system that starts use case>	Clicks the "Download" button	
Main Success Flows <the steps of the scenario from trigger to goal delivery and any clean up after>.	**Step**	**Action**
	1	Click the download button.
	2	User is asked where (directory location) the file would be downloaded.
	3	User chooses a directory path.
	4	Download starts
	5	User has the file.
Alternative Flows <a: condition causing branching> <a1: action or name of sub-use case>	**Step**	**Branching Action**
	4a	Downloading file.
	4a1	User pauses downloading.
	4a2	User resumes downloading.
Quality Requirements	**Step**	**Requirement**
	1	Users must have the internet with a decent download speed.

Table 10: Access Token

Use Case	Access Token	
Goal <a longer statement of the goal in context if needed>	User wants to share his/her folder for a certain time.	
Preconditions <what we expect is already the state of the world>	login	
Success End Condition <the state of the world upon successful completion>	User shares his/her folder's directory	
Failed End Condition <the state of the world if goal abandoned>	User2 can't access User1's folder's directory	
Primary Actors: **Secondary Actors:**	User1 User2	
Trigger <the action upon the system that starts use case>	User1 requests to generate an access token	
Main Success Flows <the steps of the scenario from trigger to goal delivery and any clean up after>	**Step**	**Action**
	1	User1 clicks get access token button
	2	User1 gets the token
	3	User1 shares the directory link with his/her desired person.
	3.1	User1 share the token with his/her desired person
	4	User2 browses the directory link.
	4.1	User2 pastes the token into a inputs box
	5	User2 has the access of the directory/file .
Alternative Flows <a: condition causing branching> <a1: action or name of sub-use case>	**Step**	**Branching Action**
	4a	Directory link is broken or directory is deleted by user1.
	4.1a	The token is not valid.
	4.1b	User requests for another token
Quality Requirements	**Step**	**Requirement**
	N/A	N/A

Table 11: Add to wish list

Use Case	Add to Wish List	
Goal <a longer statement of the goal in context if needed>	User is willing to add a file to the favorite list	
Preconditions <what we expect is already the state of the world>	Select a file	
Success End Condition <the state of the world upon successful completion>	File is added to the list.	
Failed End Condition <the state of the world if goal abandoned>	N/A	
Primary Actors: **Secondary Actors:**	User Application server	
Trigger <the action upon the system that starts use case>	User clicks “Add to wish list” button	
Main Success Flows <the steps of the scenario from trigger to goal delivery and any clean up after>	**Step**	**Action**
	1	Select a file
	2	Add to Wishlist button comes.
	3	User Clicks the button.
	4	Fill is added to the wish list.
Alternative Flows <a: condition causing branching> <a1: action or name of sub-use case>	**Step**	**Branching Action**
	N/A	N/A
Quality Requirements	**Step**	**Requirement**
	N/A	N/A

Table 12: Trash Bin

Use Case	Trash Bin	
Goal <a longer statement of the goal in context if needed>	Remove a file.	
Preconditions <what we expect is already the state of the world>	Select specific files.	
Success End Condition <the state of the world upon successful completion>	File removed but stored in Trash Bin.	
Failed End Condition <the state of the world if goal abandoned>	N/A	
Primary Actors: **Secondary Actors:**	User Application Server	
Trigger <the action upon the system that starts use case>	"Trash Bin" button needs to be clicked	
Main Success Flows <the steps of the scenario from trigger to goal delivery and any clean up after>	**Step**	**Action**
	1	Users select one or more files.
	1.1	Click the "ok" button.
	2 2.1	Server remove the files from the location Server stored file in Trash
Alternative Flows <a: condition causing branching> <a1: action or name of sub-use case>	**Step**	**Branching Action**
	N/A	N/A
Quality Requirements	**Step**	**Requirement**
	2	Delete file permanently after 30 days and account storage updated.

Table 13: Delete File

Use Case	Delete File	
Goal <a longer statement of the goal in context if needed>	Delete file permanently	
Preconditions <what we expect is already the state of the world>	N/A	
Success End Condition <the state of the world upon successful completion>	File deleted permanently.	
Failed End Condition <the state of the world if goal abandoned>	File deleted automatically after 30days.	
Primary Actors: **Secondary Actors:**	User Application Server	
Trigger <the action upon the system that starts use case>	"Delete file" button needs to be clicked	
Main Success Flows <the steps of the scenario from trigger to goal delivery and any clean up after>	**Step**	**Action**
	1	From trash bin user select one or more file
	2	User click Delete file
	3	Server deletes the specific files permanently.
	4	Server update storage.
Alternative Flows <a: condition causing branching> <a1: action or name of sub-use case>	**Step**	**Branching Action**
	3.a1 3.a2 3.a3	User did not select Delete file Server did not delete files. Server delete files automatically after 30days
Quality Requirements	**Step**	**Requirement**
	1	File must be kept in the trash bin.

Table 14: Create Folder

Use Case	Create Folder	
Goal <a longer statement of the goal in context if needed>	Create folders so that files can be kept.	
Preconditions <what we expect is already the state of the world>	User Login	
Success End Condition <the state of the world upon successful completion>	A new folder will be created in the application.	
Failed End Condition <the state of the world if goal abandoned>	N/A	
Primary Actors: **Secondary Actors:**	User Application Server	
Trigger <the action upon the system that starts use case>	"Create Folder" button needs to be clicked.	
Main Success Flows <the steps of the scenario from trigger to goal delivery and any clean up after>	**Step**	**Action**
	1	User click in "Create Folder" button
	2.1 2.2 2.3	Server ask for the name of the folder User name the folder User click the ok button
	3	A new folder will be created.
Alternative Flows <a: condition causing branching> <a1: action or name of sub-use case>	**Step**	**Branching Action**
	N/A	N/A
Quality Requirements	**Step**	**Requirement**
	3.2a	User must provide a valid name.

Table 15: Add File

Use Case	Add File	
Goal <a longer statement of the goal in context if needed>	Adding a new file.	
Preconditions <what we expect is already the state of the world>	Logs into application	
Success End Condition <the state of the world upon successful completion>	New file will be added	
Failed End Condition <the state of the world if goal abandoned>	File will not be added.	
Primary Actors: **Secondary Actors:**	User Application Server	
Trigger <the action upon the system that starts use case>	"Add file" button needs to be clicked	
Main Success Flows <the steps of the scenario from trigger to goal delivery and any clean up after>	**Step**	**Action**
	1	Select "Add file" option
	2 2.1 2.2	Server asks to select the file from a local storage. User browse and select a file from local storage. User selects the "add" button.
	3 3.1 3.2	Server shows two options Add in original size Compressed and add
	4	User choose an option
	5	Server adds the file.
Alternative Flows <a: condition causing branching> <a1: action or name of sub-use case>	**Step**	**Branching Action**
	2.2a 2.2b 3	User's storage is full or will exceed the limit. Buy storage
	4a	User didn't choose any option
	5a	Server will not add files.
Quality Requirements	**Step**	**Requirement**
	1	Select the correct format file.

Table 16: Compress File

Use Case	Compress File	
Goal <a longer statement of the goal in context if needed>	Reduce the file size	
Preconditions <what we expect is already the state of the world>	Select a file.	
Success End Condition <the state of the world upon successful completion>	File size will reduce up to 90%	
Failed End Condition <the state of the world if goal abandoned>	Remain in original size.	
Primary Actors: **Secondary Actors:**	Application Server N/A	
Trigger <the action upon the system that starts use case>	"Compress File" button needs to be clicked	
Main Success Flows <the steps of the scenario from trigger to goal delivery and any clean up after>	**Step**	**Action**
	1	User add a file
	2 2.1 2.2	Server shows two options Add in original size Compressed and added
	3.1 3.2	User select Compress file Server compressed the file.
Alternative Flows <a: condition causing branching> <a1: action or name of sub-use case>	**Step**	**Branching Action**
	2.1a 2.1b	User select original file option. Server add the file in original size
	3.2a	File was not compressed.
Quality Requirements	**Step**	**Requirement**
	1	Compress file within decent time.

Table 17: Check Space

Use Case	Check Space	
Goal <a longer statement of the goal in context if needed>	User want to check how much storage is available.	
Preconditions <what we expect is already the state of the world>	User Login	
Success End Condition <the state of the world upon successful completion>	User will know how much storage is available	
Failed End Condition <the state of the world if goal abandoned>	Application server was unable to show storage information.	
Primary Actors: **Secondary Actors:**	User Application Server	
Trigger <the action upon the system that starts use case>	"Check Space" button needs to be clicked	
Main Success Flows <the steps of the scenario from trigger to goal delivery and any clean up after>	**Step**	**Action**
	1	User clicks "Check space"
	2	Server will show the user how much space is available
Alternative Flows <a: condition causing branching> <a1: action or name of sub-use case>	**Step**	**Branching Action**
	N/A	N/A
Quality Requirements	**Step**	**Requirement**
	N/A	N/A

Table 18: Payment Information

Use Case	Payment Information	
Goal <a longer statement of the goal in context if needed>	See different packages. Pay for preferred packages and increase storage.	
Preconditions <what we expect is already the state of the world>	User Login	
Success End Condition <the state of the world upon successful completion>	User will get information about recent packages.	
Failed End Condition <the state of the world if goal abandoned>	N/A	
Primary Actors: **Secondary Actors:**	User Credit card company, bank	
Trigger <the action upon the system that starts use case>	"Payment Information" button needs to be clicked	
Main Success Flows <the steps of the scenario from trigger to goal delivery and any clean up after>	**Step**	**Action**
	1	User clicks "Payment Information" button
	2	Server gives user information on different packages.
	3	User see the details of each package
	4	User choose a package.
Alternative Flows <a: condition causing branching> <a1: action or name of sub-use case>	**Step**	**Branching Action**
	3a 3a1	User see the details of each package User did not prefer any package.
	4a	User did not select any package.
Quality Requirements	**Step**	**Requirement**
	N/A	N/A

Table 19: Money Transaction

Use Case	Money Transaction	
Goal <a longer statement of the goal in context if needed>	Pay for preferred packages and increase storage.	
Preconditions <what we expect is already the state of the world>	Choose any package as needed.	
Success End Condition <the state of the world upon successful completion>	Storage Increase Admin has money for increasing storage.	
Failed End Condition <the state of the world if goal abandoned>	Server will not increase the storage User has not spent the money.	
Primary Actors: **Secondary Actors:**	User Application Server	
Trigger <the action upon the system that starts use case>	"Money Transaction" button needs to be clicked	
Main Success Flows <the steps of the scenario from trigger to goal delivery and any clean up after>	**Step**	**Action**
	1	User choose a package
	2 2.1 2.2 2.3 2.4	Server shows payment method Payment via Bkash Payment via MasterCard Payment via Visa card etc.
	3	User chooses an option for payment.
	4	User provides the information as needed for the method for payment.
	5	User request for update storage
	6.1 6.2	Server ask user for confirmation User confirms the order
	7	Server sent an email as his/her order was confirmed.
	8	Server Update user account storage.
Alternative Flows <a: condition causing branching> <a1: action or name of sub-use case>	**Step**	**Branching Action**
	4a1 4a2 4a3	Information provided by the user is incorrect. Server alerts the user and gives a notification of error. Server asked the user to re-enter the information.
	5a1 5a2 5a3 5a4	There is not enough balance in the account. Server sends an alert notification Server cancels users request. Storage is not increased.
Quality Requirements	**Step**	**Requirement**
	1	User have to Provide all correct information
	2	The user must have money in the account he provided.

Table 20: Update Account Storage

Use Case	Update Account Storage	
Goal <a longer statement of the goal in context if needed>	User wants to increase storage.	
Preconditions <what we expect is already the state of the world>	Add new files Delete files Buying storage	
Success End Condition <the state of the world upon successful completion>	Storage will be increased or decreased.	
Failed End Condition <the state of the world if goal abandoned>	Storage will remain same.	
Primary Actors: **Secondary Actors:**	User Application Server	
Trigger <the action upon the system that starts use case>	Clicks on “Add file” or “delete file” or “Buy Storage”	
Main Success Flows <the steps of the scenario from trigger to goal delivery and any clean up after>	**Step**	**Action**
	1 1.1	User add a file Storage increase
	2 2.1	Delete files Storage decreased
	3 3.1	Buy storage Storage capacity increased.
Alternative Flows <a: condition causing branching> <a1: action or name of sub-use case>	**Step**	**Branching Action**
	N/A	N/A
Quality Requirements	**Step**	**Requirement**
	1	Update storage without delay.

Table 21: Play File

Use Case	Play File	
Goal <a longer statement of the goal in context if needed>	User will play and listen to the file.	
Preconditions <what we expect is already the state of the world>	Enter any specific folder and select a file.	
Success End Condition <the state of the world upon successful completion>	Users can play and listen to files.	
Failed End Condition <the state of the world if goal abandoned>	User will not able to play desired file	
Primary Actors: **Secondary Actors:**	User Application Server	
Trigger <the action upon the system that starts use case>	Select a specific file and the "Play" option needs to be clicked.	
Main Success Flows <the steps of the scenario from trigger to goal delivery and any clean up after>	**Step**	**Action**
	1	User selects a file
	2	User selects the "Play file" option.
	3	Application opens "Audio Player" of "Audio Basket"
	4	Application plays the file.
	5	User selects an option while playing the file.
Alternative Flows <a: condition causing branching> <a1: action or name of sub-use case>	**Step**	**Branching Action**
	3a	User does not select the "Play File" option.
	3a1	User selects other options or goes back.
Quality Requirements	**Step**	**Requirement**
	1	User has to select a file.

Table 22: Run Audio Player

Use Case	Run "Audio Player"	
Goal <a longer statement of the goal in context if needed>	Play a specific file in "Audio Player" using different options.	
Preconditions <what we expect is already the state of the world>	Enter any specific folder and select a file.	
Success End Condition <the state of the world upon successful completion>	Audio player of the "Audio Basket" application starts playing the selected file using different options.	
Failed End Condition <the state of the world if goal abandoned>	Audio player is unable to play the file.	
Primary Actors: **Secondary Actors:**	User Audio Player	
Trigger <the action upon the system that starts use case>	Select a specific file and the "Play" option needs to be clicked.	
Main Success Flows <the steps of the scenario from trigger to goal delivery and any clean up after>	**Step**	**Action**
	1	User selects a file
	2	User selects the "Play file" option.
	3	Application opens "Audio Player" of "Audio Basket"
	4	Application plays the file.
	5	User selects an option while playing the file.
Alternative Flows <a: condition causing branching> <a1: action or name of sub-use case>	**Step**	**Branching Action**
	5a	User selects any option.
	5a1	User selects the "Pause" button and the audio file pauses.
	5a2	User selects the "Resume" button and the audio file resumes.
	5a3	User selects the "Stop" button and the audio file stops playing
	5a4	User selects the "Next" button and the next audio file that starts playing.
	5a5	User selects the "Previous" button and the previous audio file that starts playing.
Quality Requirements	**Step**	**Requirement**
	1	User has to select a file.
	1	User has to select a button in "Audio Player".

Table 23: Log Out

Use Case	Log out	
Goal <a longer statement of the goal in context if needed>	Logout from “Audio Basket” application.	
Preconditions <what we expect is already the state of the world>	User login	
Success End Condition <the state of the world upon successful completion>	Successfully logout from “Audio Basket” application.	
Failed End Condition <the state of the world if goal abandoned>	Remain in a login state.	
Primary Actors: **Secondary Actors:**	User Application Server	
Trigger <the action upon the system that starts use case>	“Logout” Button needs to be clicked.	
Main Success Flows <the steps of the scenario from trigger to goal delivery and any clean up after>	**Step**	**Action**
	1	User using the “Audio Basket” application after login
	2	User clicks the “Logout” button.
	3	User logout successful.
Alternative Flows <a: condition causing branching> <a1: action or name of sub-use case>	*Step*	*Branching Action*
	N/A	N/A
Quality Requirements	**Step**	**Requirement**
	1	Users must login to logout.

8. Activity Diagram

(Create Account)

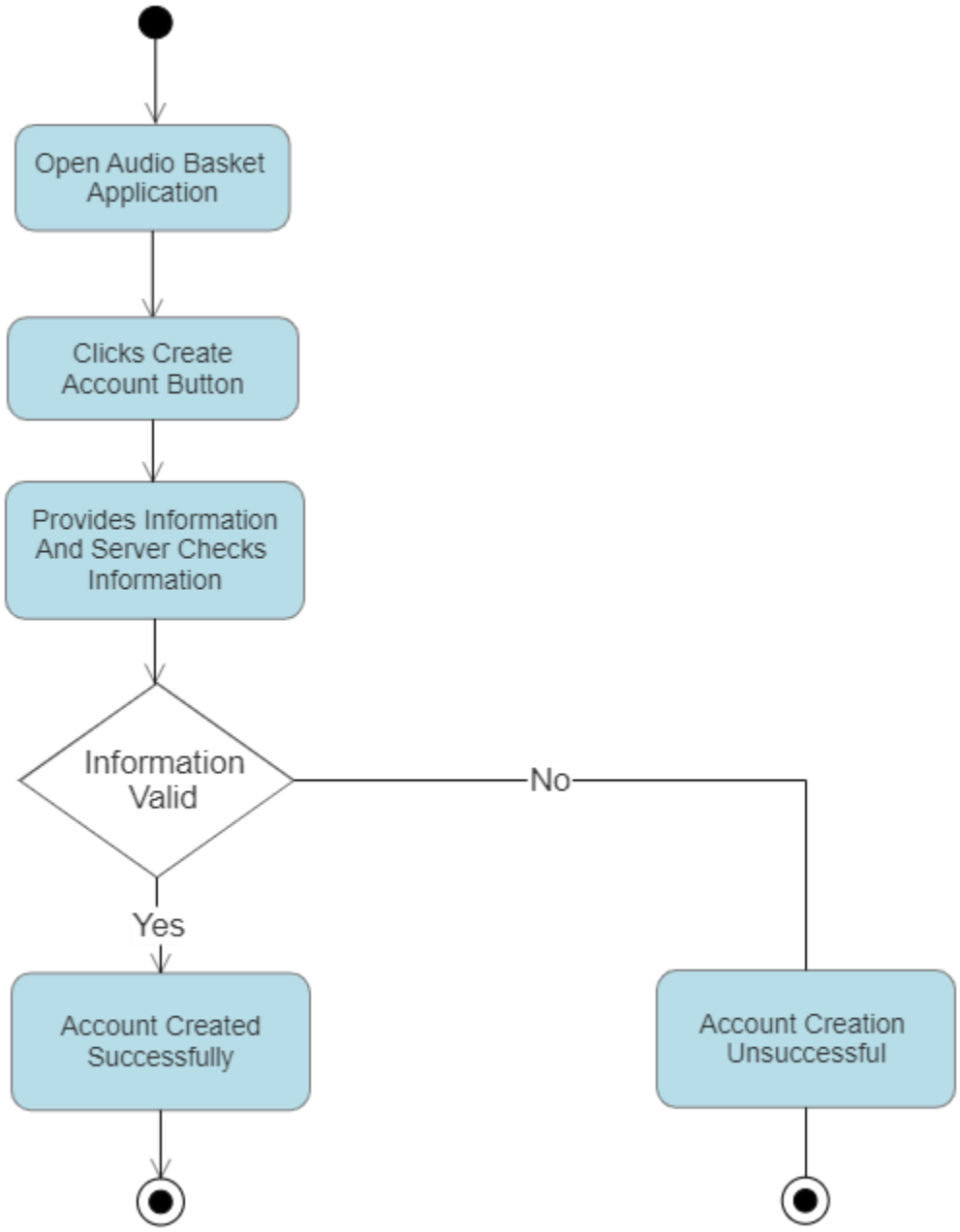

Figure 2: Create Account

Activity Diagram (Log in)

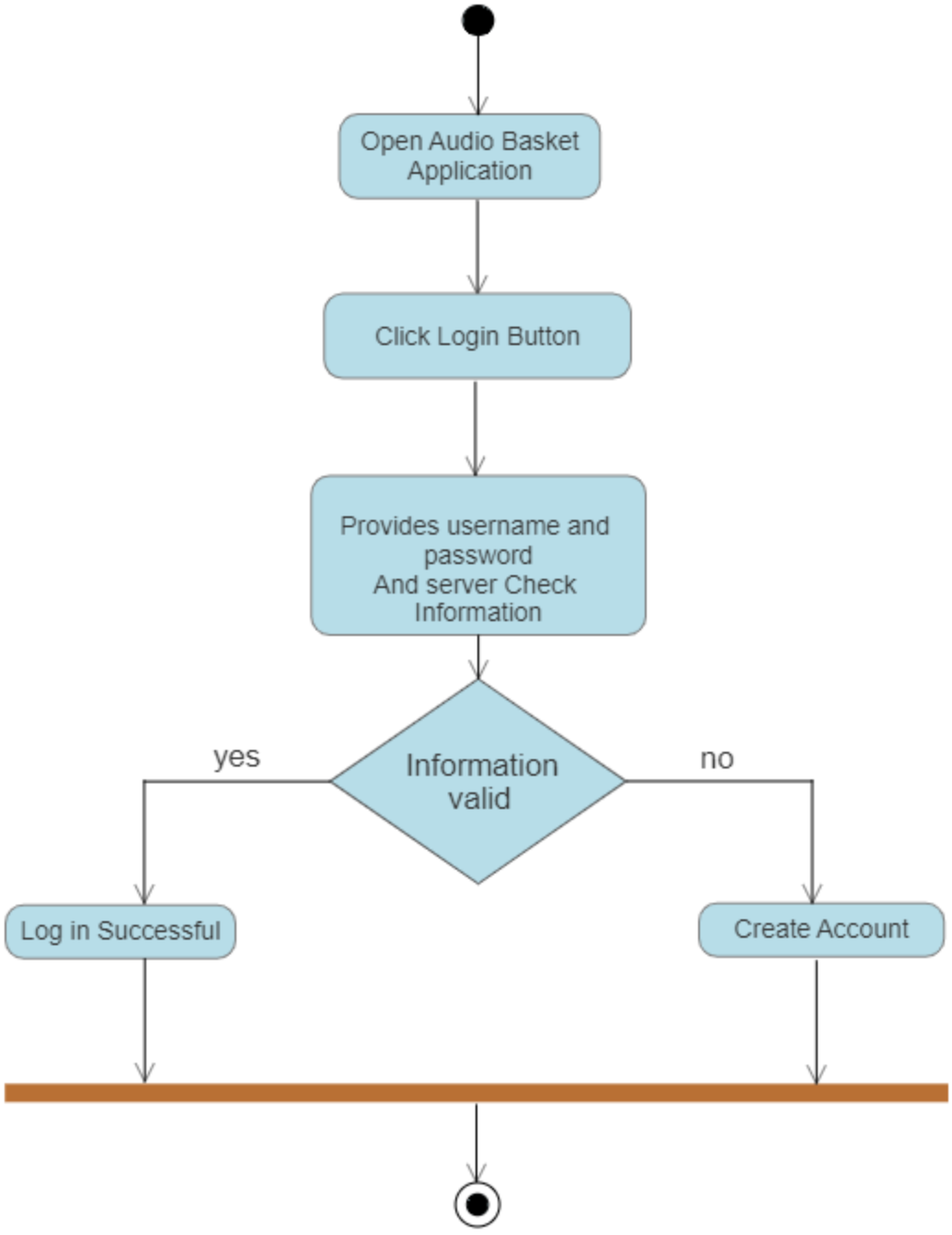

Figure 3: Log in

Activity Diagram (Search File)

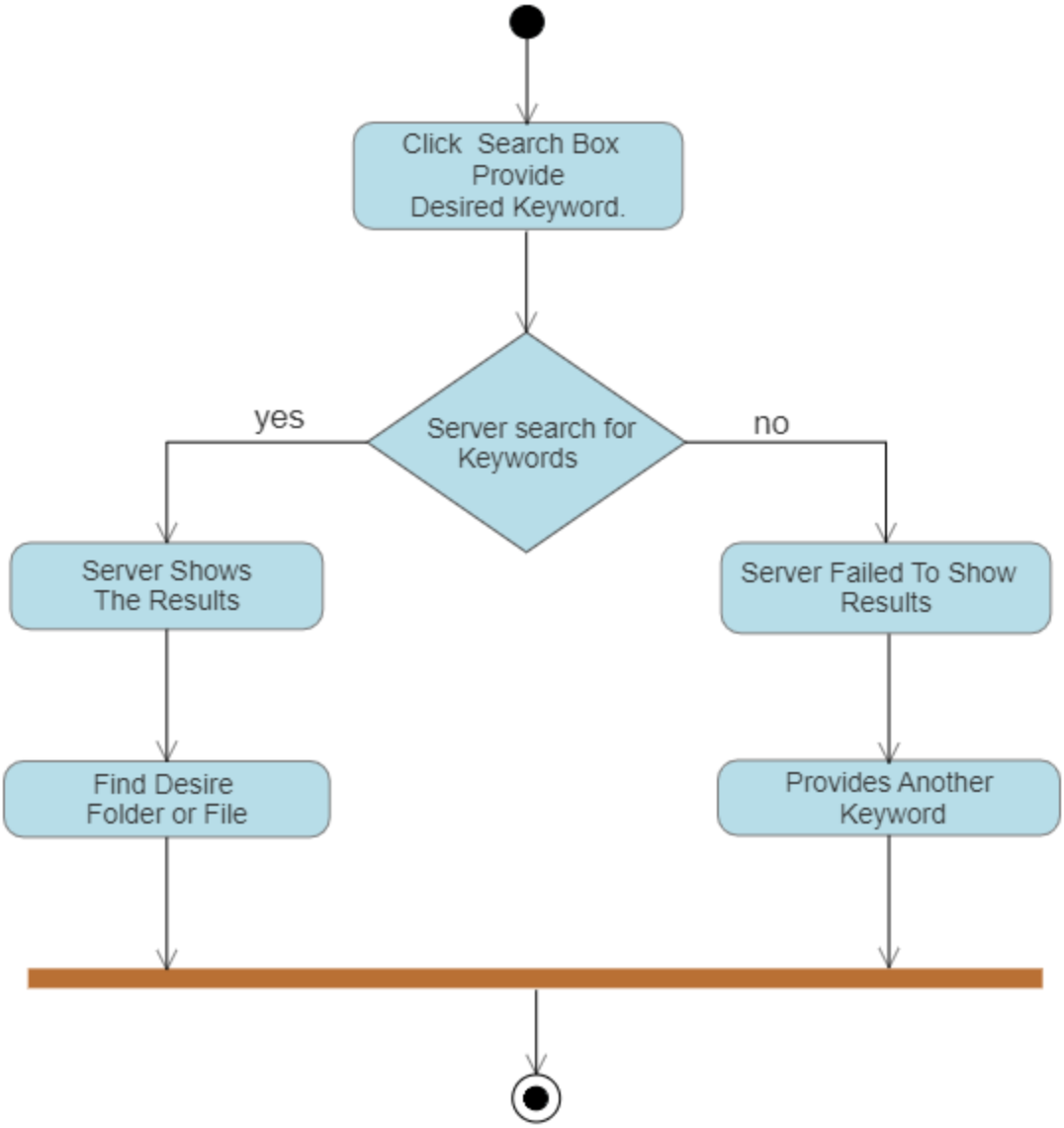

Figure 4: Search File

Activity Diagram (Search Category)

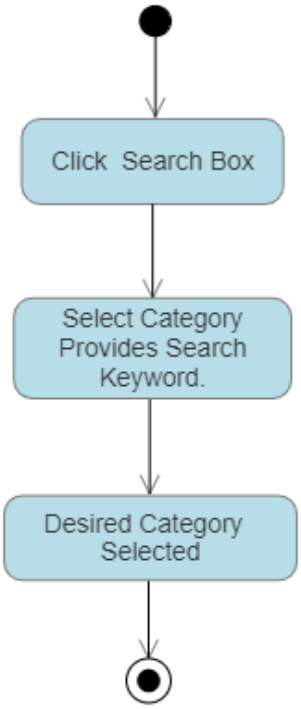

Figure 5:Search Category

Activity Diagram (Sort File)

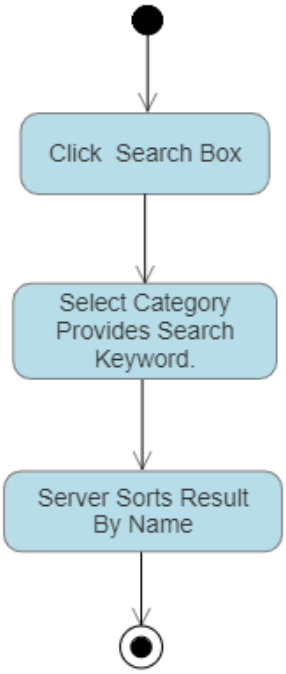

Figure 6: Sort File

Activity Diagram (Show Closer Result)

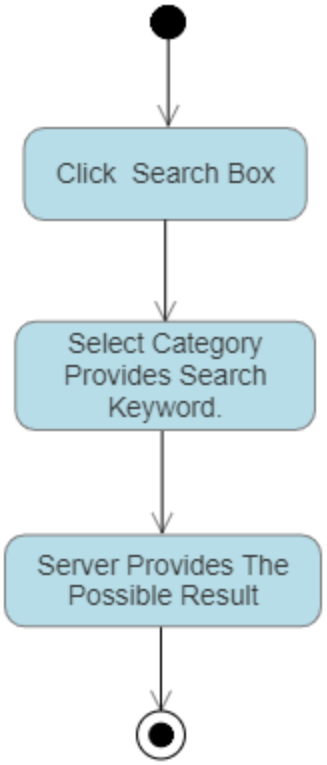

Figure 7: Show Closer Result

Activity Diagram (Select File)

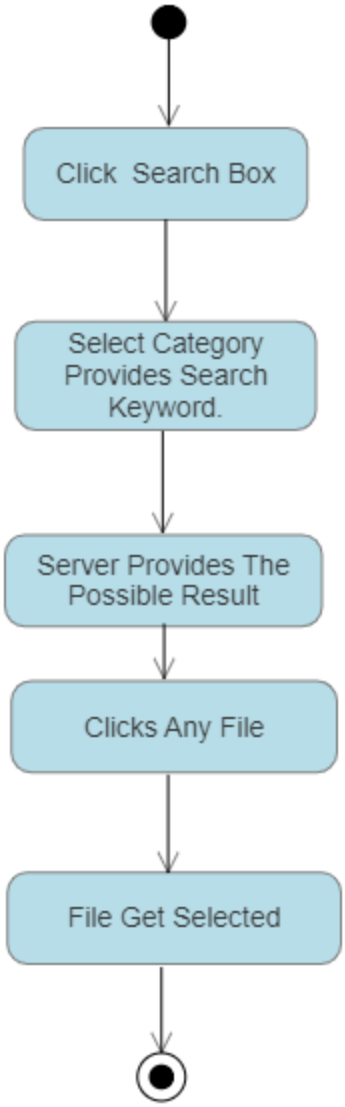

Figure 8: Select File

Activity Diagram (Share File)

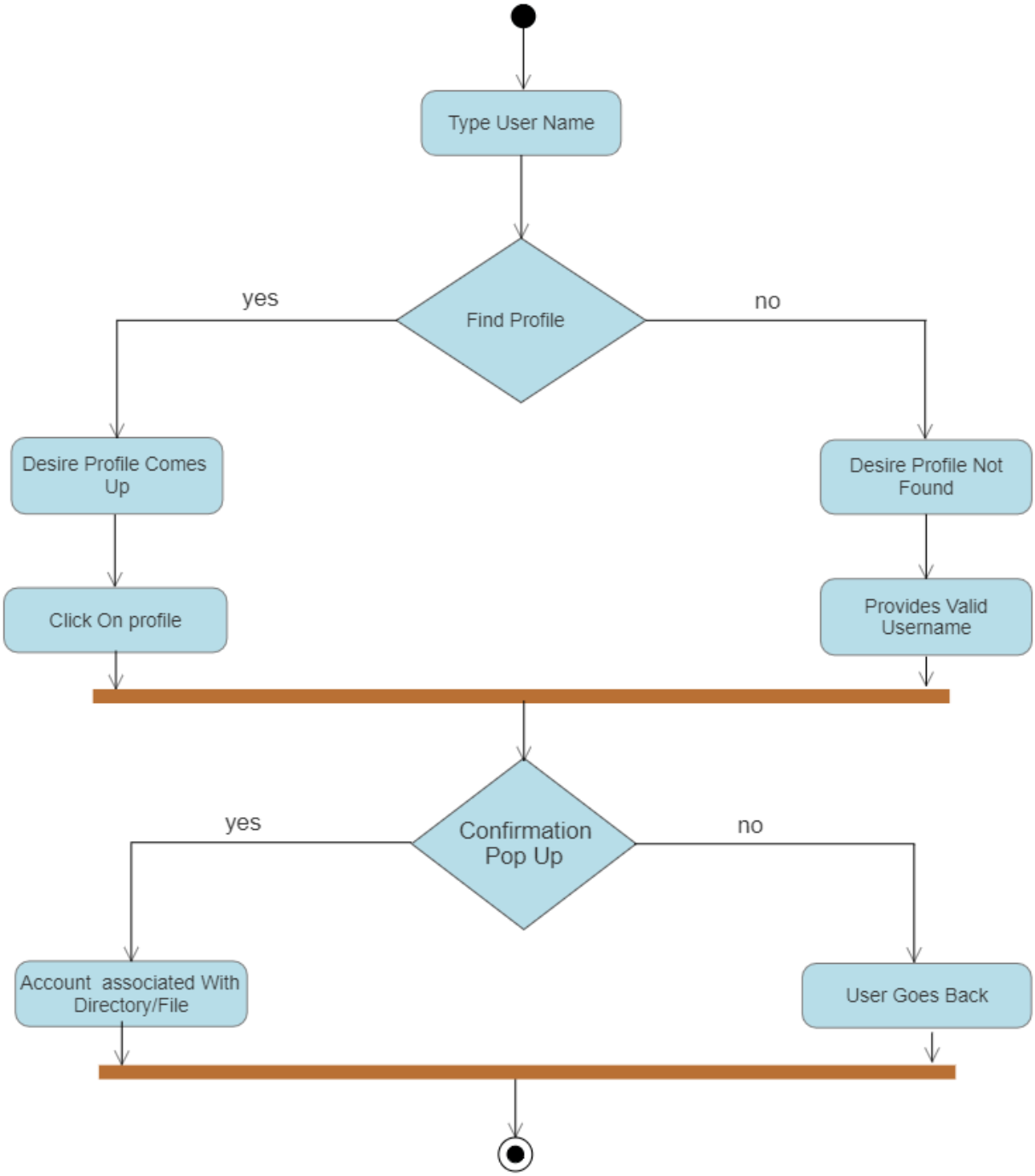

Figure 9: Share File

Activity Diagram (Download File)

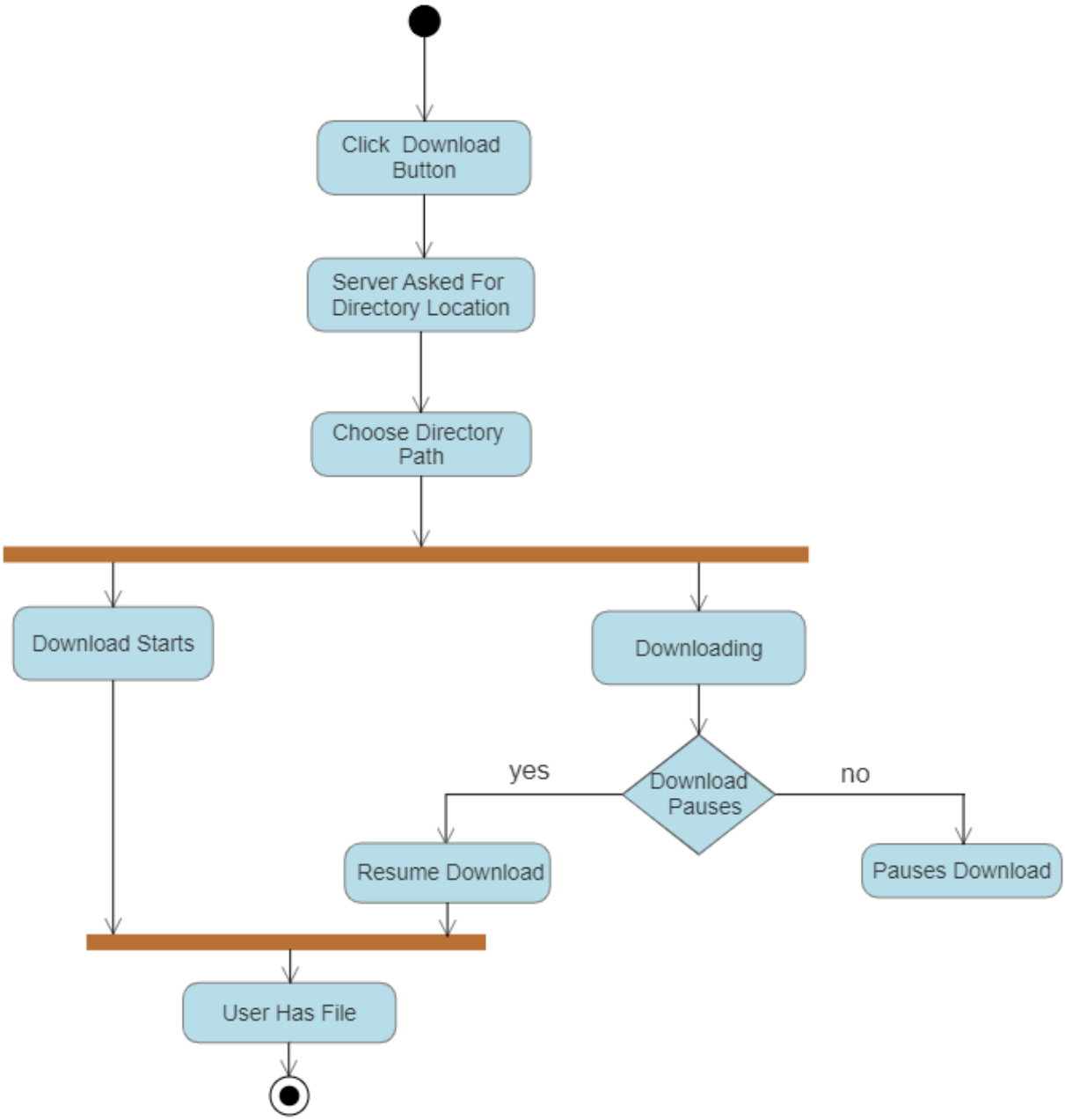

Figure 10: Download File

Activity Diagram (Access Token)

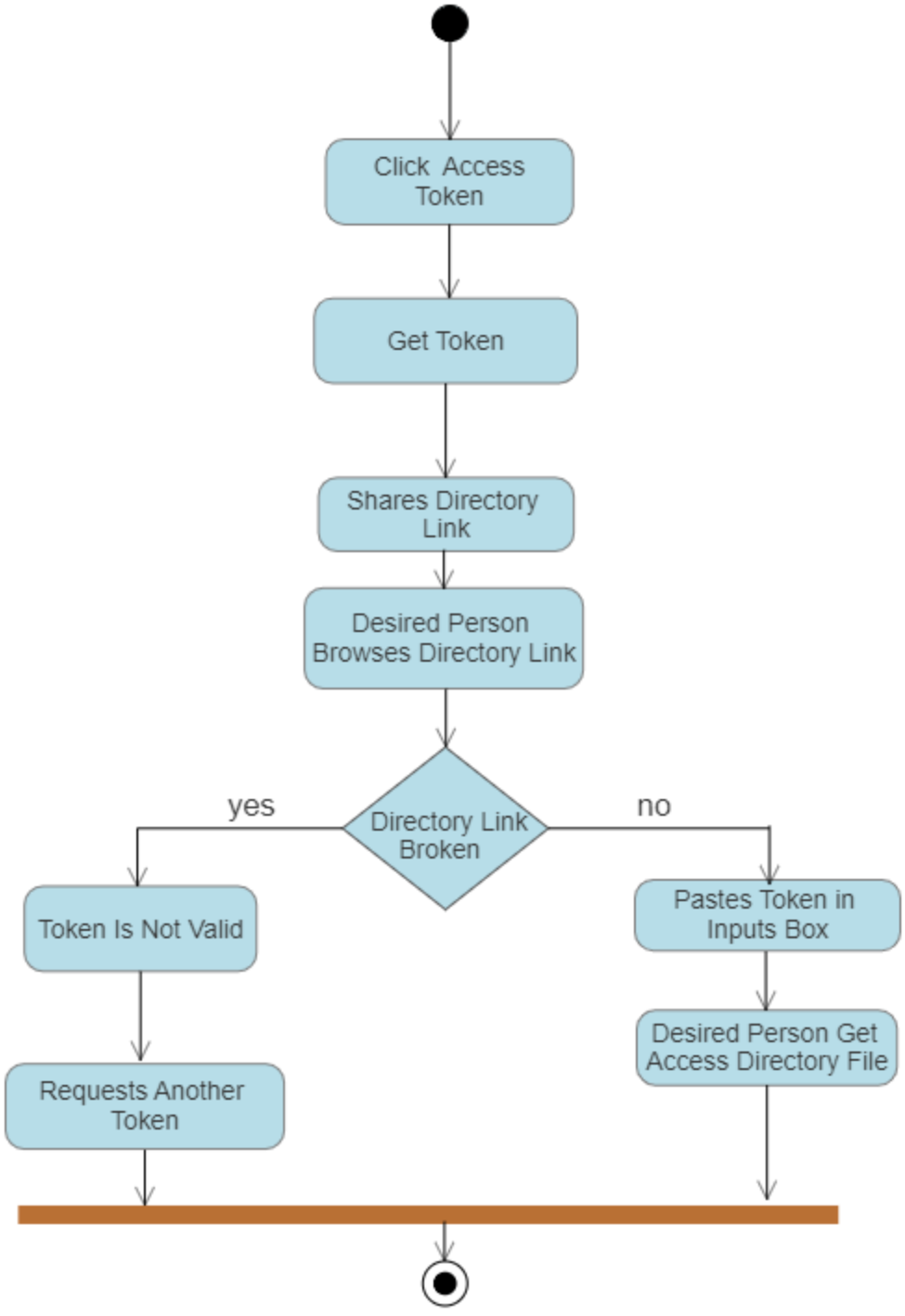

Figure 11: Access Token

Activity Diagram (Add to wishlist)

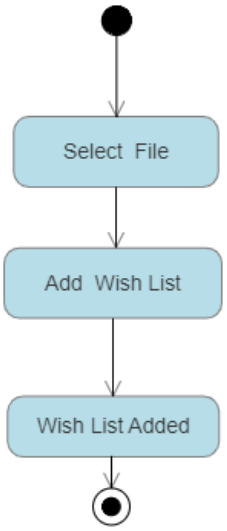

Figure 12: Add to wish list

Activity Diagram (Trash Bin)

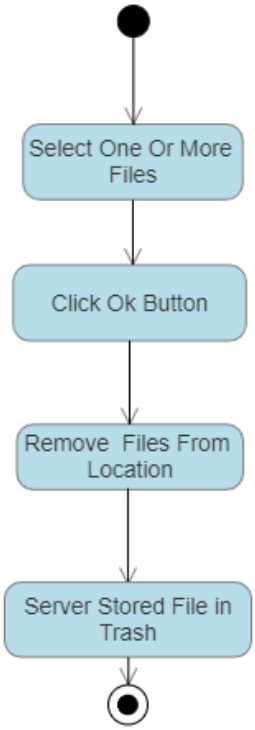

Figure 13: Trash Bin

Activity Diagram (Delete File)

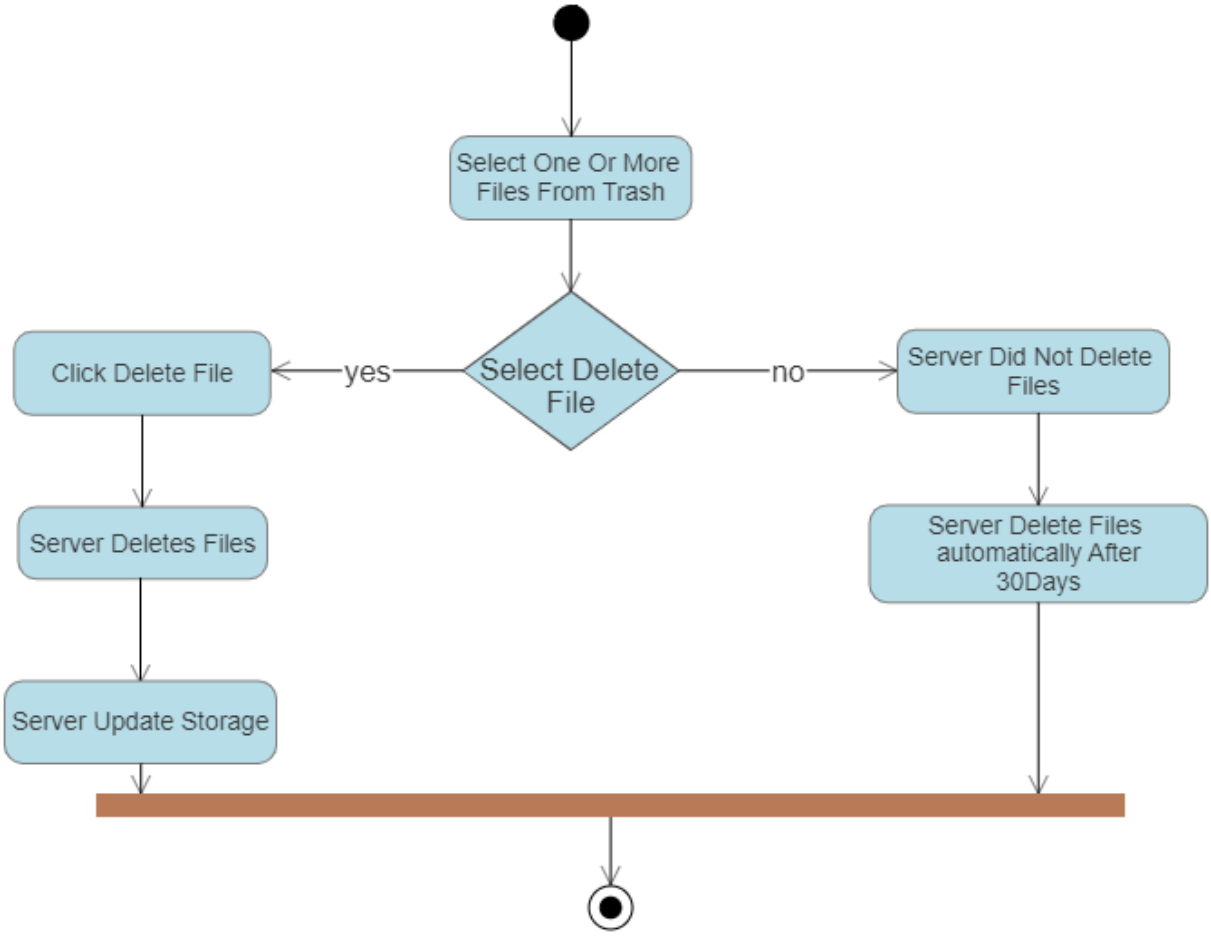

Figure 14: Delete File

Activity Diagram (Create Folder)

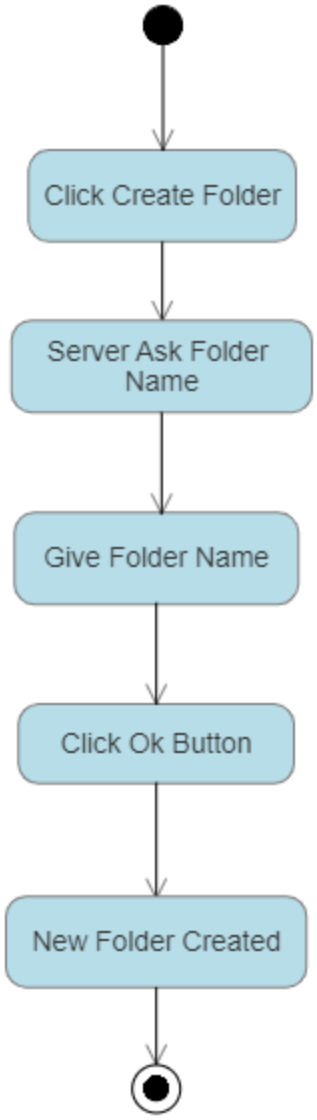

Figure 15: Create Folder

Activity Diagram (Add file)

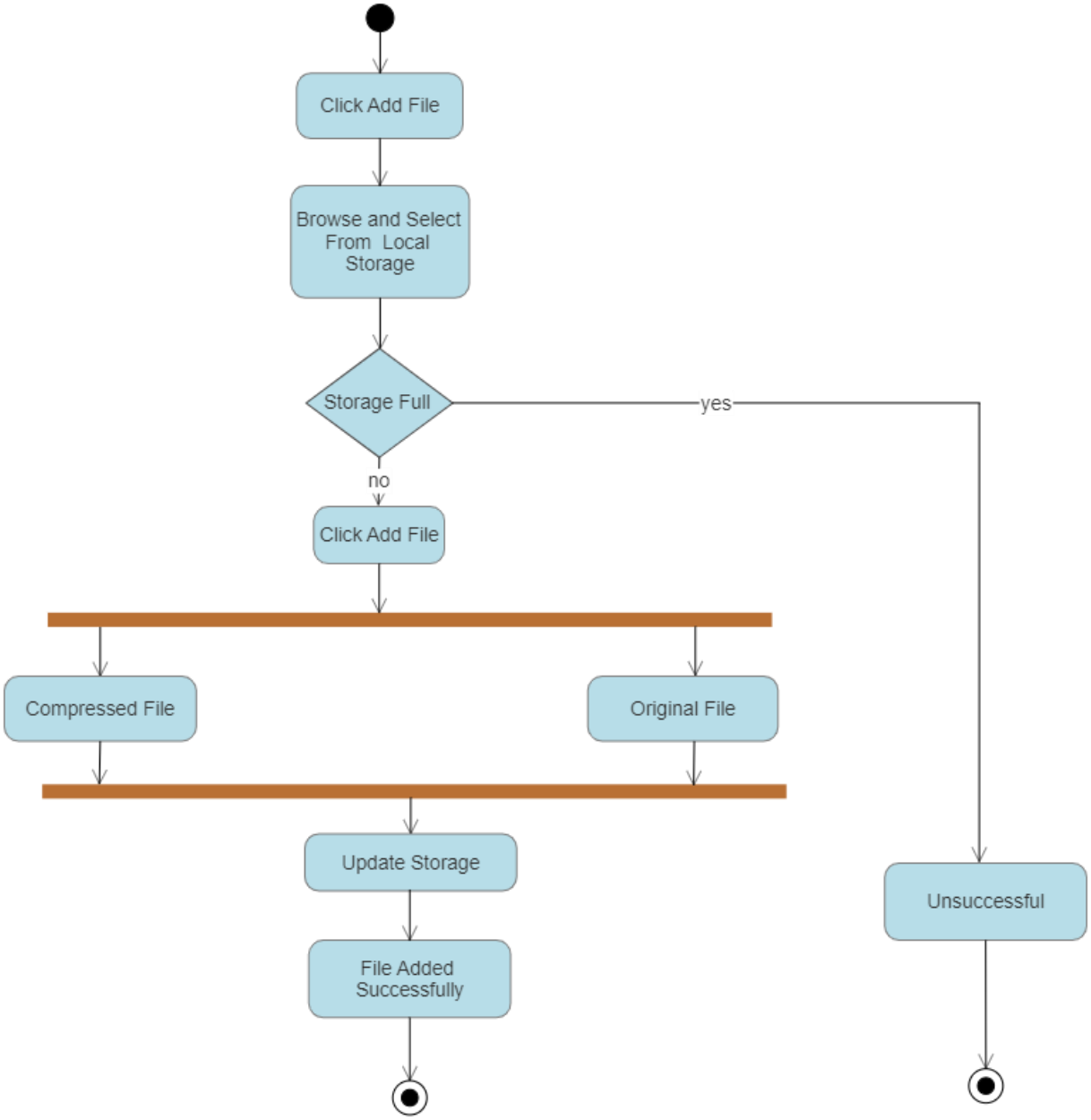

Figure 16: Add File

Activity Diagram (Compress File)

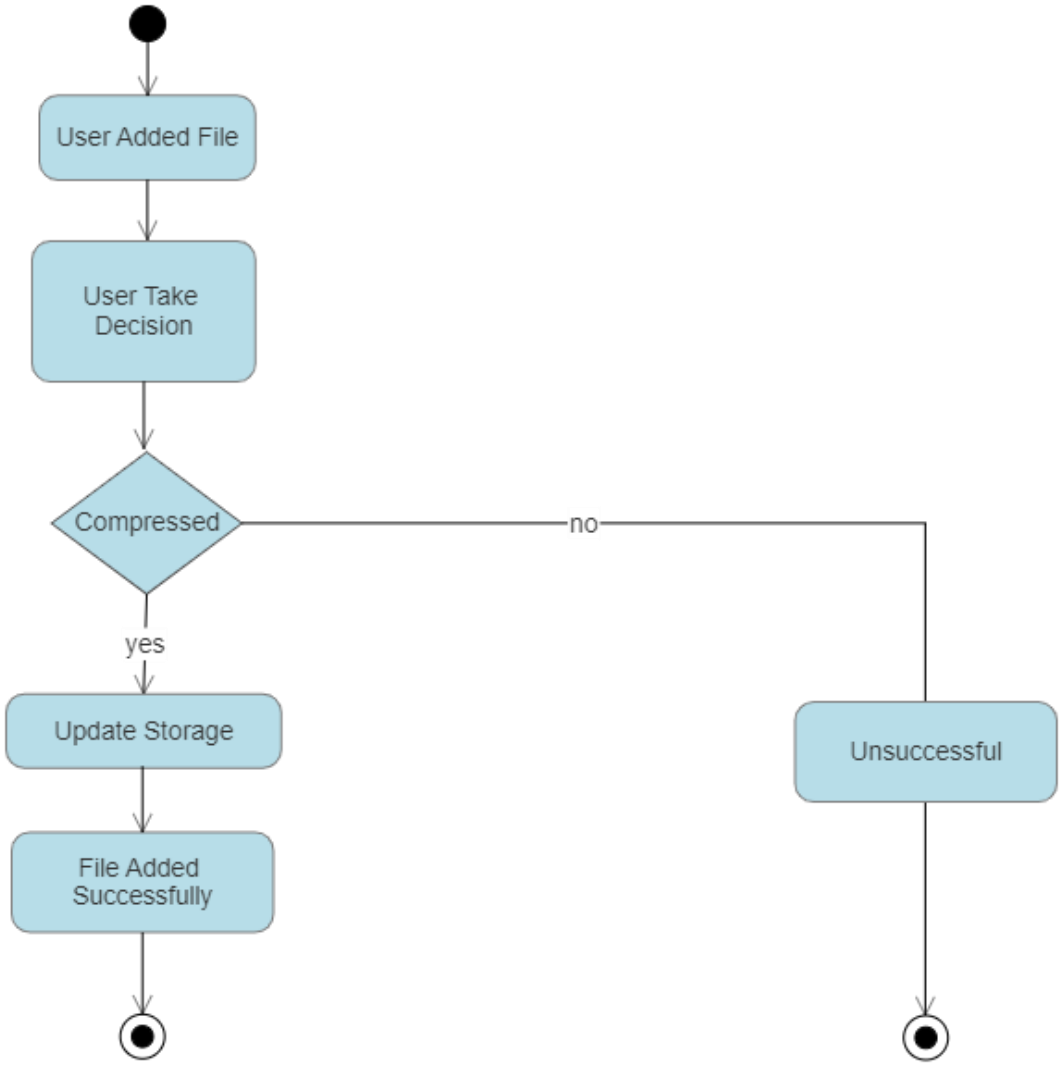

Figure 17: Compress File

Activity Diagram (Check Space)

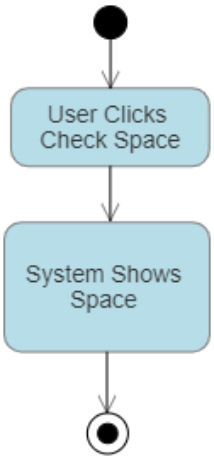

Figure 18: Check Space

Activity Diagram (Payment Information)

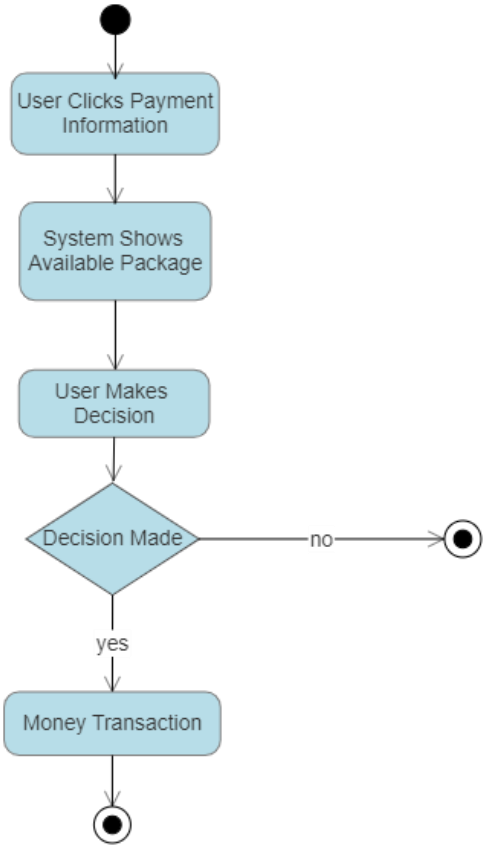

Figure 19: Payment Information

Activity Diagram (Money Transaction)

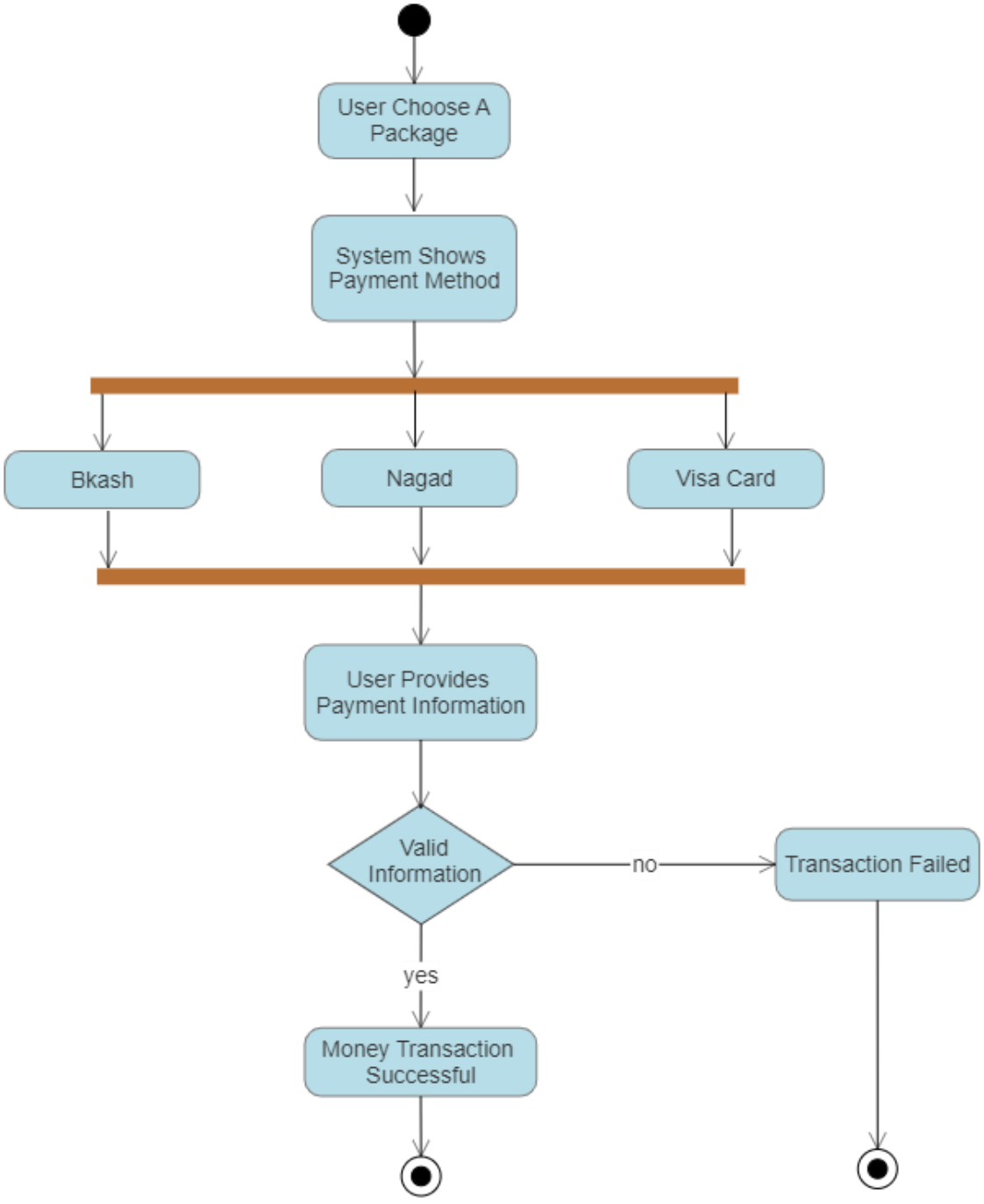

Figure 20: Money Transaction

Activity Diagram (Update account storage)

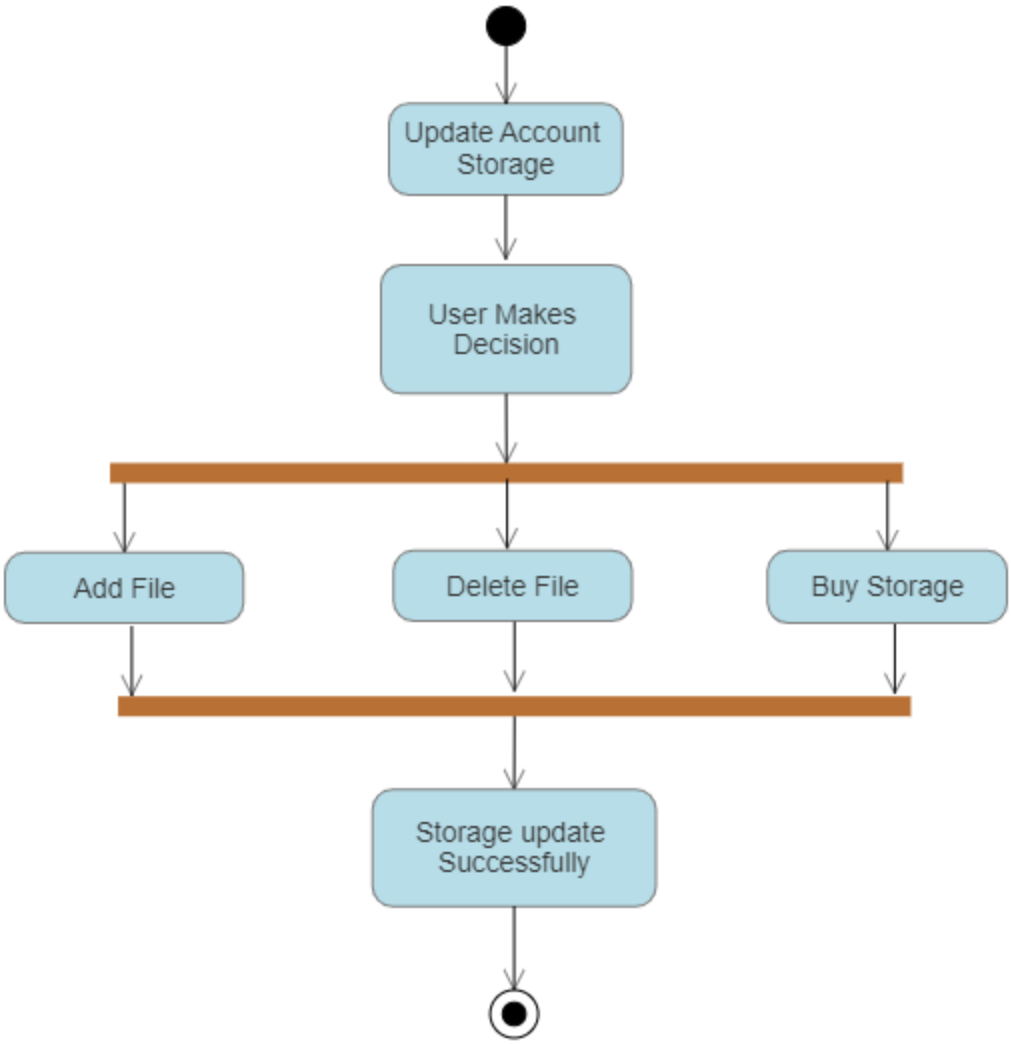

Figure 21: Update Account Storage

Activity Diagram (Play File)

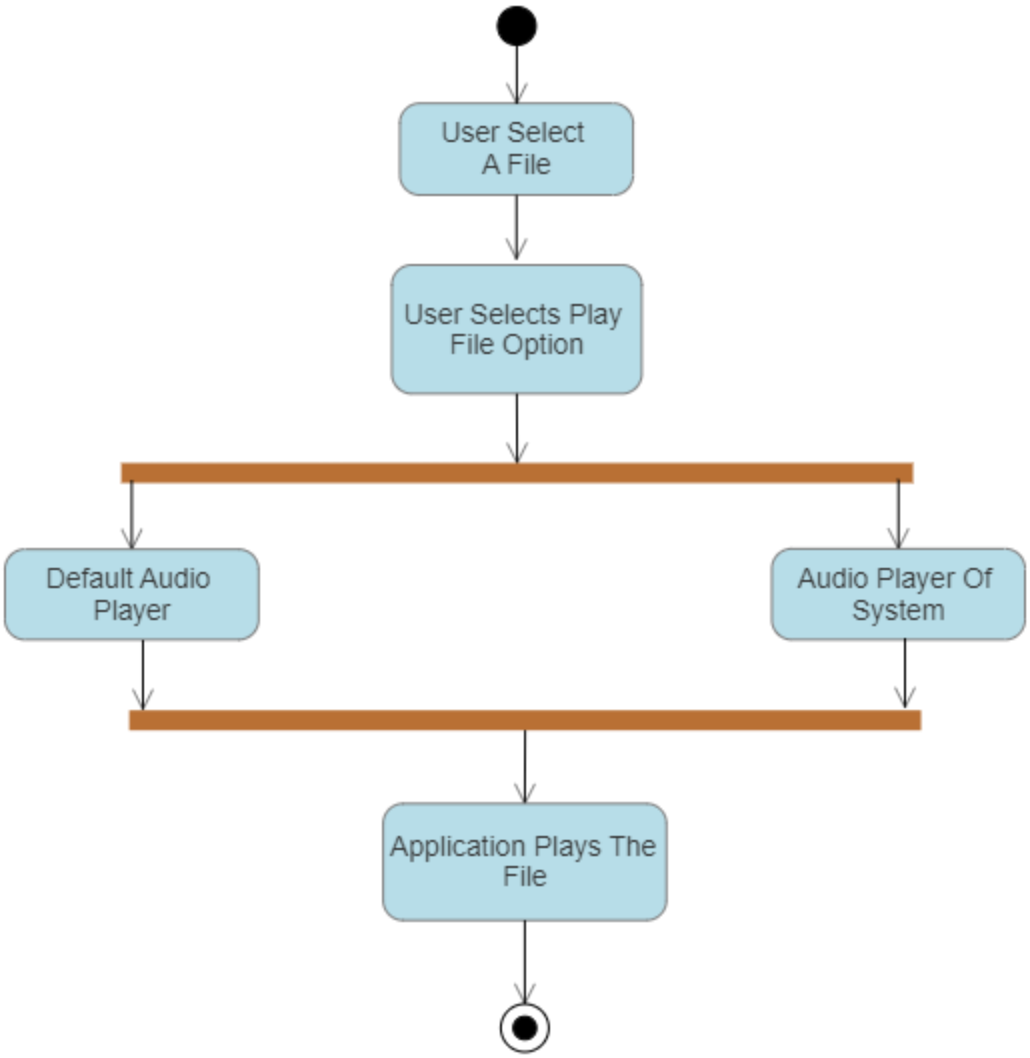

Figure 22: Play File

Activity Diagram (Run Audio Player)

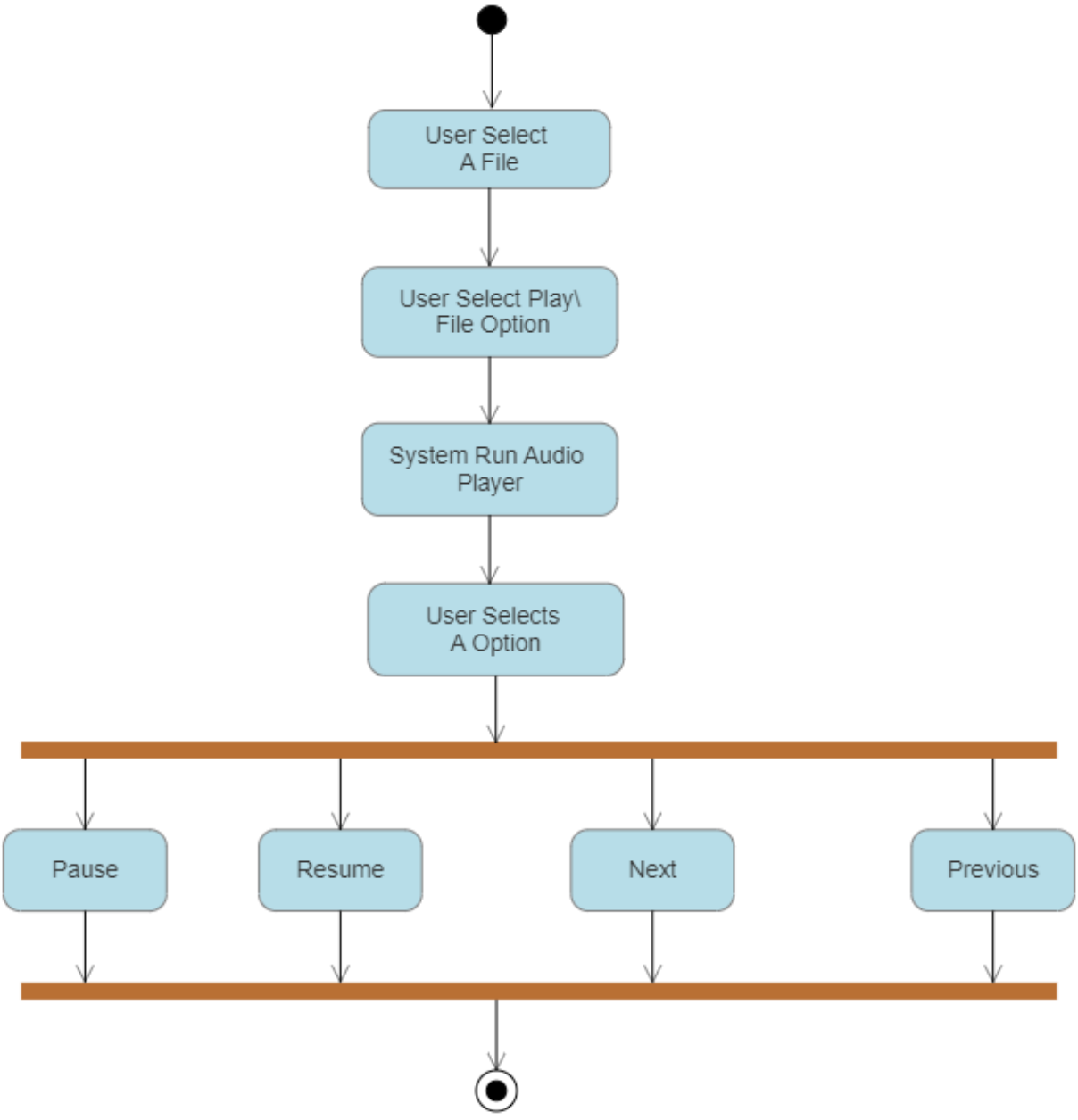

Figure 23: Run Audio Player

Activity Diagram (Log out)

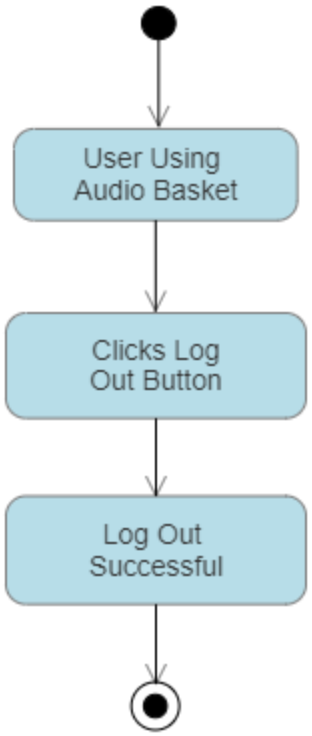

Figure 24: Log out

9. Requirement Traceability Matrix

A traceability matrix is a document, usually in the form of a table, used to assist in determining the completeness of a relationship by correlating any two baselined documents using a many-to-many relationship comparison. It is often used with high-level requirements (these often consist of marketing requirements) and detailed requirements of the product to the matching parts of high-level design, detailed design, test plan, and test cases.

Requirements Traceability Matrix							
Project Name	**Audio Basket**	**Business Area**			**Global**		
Project Manager	**Armanur Rashid**	**Business Analyst Lead**			**Prosanto Deb**		
QA Lead	**Arnab Dey**	**Target Implementation Date**					
Category / Functional Activity	**Requirement Description**	**Use Case Reference**	**Design Document Reference**	**Code Module / Reference**	**Test Case Reference**	**User Acceptance Validation**	**Comments**
FR1	User should register his account for the first time and be able to login to the account which was registered once.	UC21, UC11			TC1	Pass	
FR2	Users can create new folders to store their songs.	UC24, UC19			TC2	Pass	

FR3	Users can upload their audio files to the "Audio Basket" application.	UC14			TC3	Verified	
FR4	Users can play a previously uploaded audio file from the application.	UC9					
FR5	Users can search and filter audio files.	UC19			TC4	Verified	
FR6	Users can download the audio files for offline use.	UC23			TC5	Verified	
FR7	Users can buy more space if they want. Initially 1GB of server space will be given to every user free of cost.	UC4			TC8	Pass	
FR8	A progress bar will be shown along with the percentage	UC15			TC9	Verified	

	of storage users have used.						
FR9	Users can share a file which is previously uploaded to the "Audio Basket" application.	UC11			TC10	Pass	
FR10	Users can delete one or multiple files or folders.	UC17,UC26			TC4	Pass	
FR11	Users can give access to their folders, to other "Audio Basket" users.	UC5			TC9	Pass	
FR12	Users can log out from their previously logged in account.	UC3,UC4			TC13	Verified	

10. Appendix

10.1 Prioritization of requirements

We've prioritized the functional requirements by following Three-level Scale technique.

10.1.1 Three-level Scale

When a Business Analyst categorizes the requirements in any of the ordering or ranking scale, it is subject to the analyst's understanding of the business. Many analysts suggest that this method has some drawbacks and advocate methods that have more than one scale.

10.1.2 Prioritization of the requirements of Audio Basket

FR1 – High priority: Its essential requirement for our system. User registration is obvious to run this application.

FR2 – Low priority: It is not always necessary to create a folder and then operate. User can execute on existing folders to.

FR3 – High priority: Users need to upload files to get the benefits of file compression.

FR4 – Low priority: Play a previously uploaded audio file from the application. It is fully user's personal wish

FR5 – Medium priority: Users can search a specific audio file using a search box and the system will provide results for search results.

FR6 – Medium priority: For ease of user involvement, users can download audio files to the local directory and play them whenever he/she want.

FR7 – High priority: Users can buy more space if they want. Initially, the system will provide 1GB of server space to every user free of cost. Then a decent amount of currency will be charged for every extension package.

FR8 – High priority: A progress bar will be shown along with the percentage of storage users have used to users consumed page.

FR9 – High priority: This is one of the best requirements in our system. A user can share a file in his account to another "Audio Basket" user.

FR10 – Medium priority: A single or several files can be permanently deleted by the user. The user can place it in the "Trash Bin" without permanently removing it if he/she so desires. It will be permanently erased after 30 days if he/she does not restore it during that time frame.

FR11 – High priority: Using an Access Token, a user can provide another "Audio Basket" user access to a certain folder on his account. The other user will be able to access the files in his account folder.

FR12 – Medium priority: The user will be able to log out of his account at the end of his need. Users will need to login again for later use.

DR1 – High priority: Users' uploaded music files will be compressed by using lossless compression algorithms and then stored on the server. This algorithm shrinks the audio file by removing frequencies that are not in human capacity.

PR1 – High priority: File will load faster and play with less buffering, because of the compression the application uses while uploading files.

MR1 – High priority: Code must be developed so that it can be modified later and will be readable.