# Software Design Document

# for

# Expense Tracking System

# Author: Victor Guzman

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Reason |
| 1.0 | 3/3/25 | Steve Smiley | Initial Document Created |
| 1.1 | 4/14/25 | Victor Guzman | Description and Systems to be Updated Added |
| 1.2 | 4/21/25 | Victor Guzman | Systems Architecture, Hardware Need, Software Need Added |
| 1.3 | 4/25/25 | Victor Guzman | User Interfaces Added |
| 1.3.1 | 4/26/25 | Victor Guzman | Backend Interfaces Flowchart and Images Added |
| 1.3.2 | 4/27/25 | Victor Guzman | Interfaces Flowcharts/Images Updated (Password Recovery) |
| 1.3.3 | 4/28/25 | Victor Guzman | Expanded User Panel Section |
| 1.3.4 | 4/28/25 | Victor Guzman | Added Data Law Information to Interfaces |
| 1.3.5 | 4/28/25 | Victor Guzman | Added Assumptions Section to Hardware Needs |
| 1.3.6 | 4/28/25 | Victor Guzman | Added Bootstrap.js to Software Needs |
| 1.3.7 | 4/28/25 | Victor Guzman | Updated Terms (AWS, Token) |
| 1.4 | 5/5/25 | Victor Guzman | Added Data Dictionary |
| 1.5 | 5/12/25 | Victor Guzman | Added Software Architecture and Structure |
| 1.5.1 | 5/12/25 | Victor Guzman | Removed AWS Lambda from Systems Architecture Section |
| 1.5.2 | 5/12/25 | Victor Guzman | Removed AWS Lambda from Hardware Needs Section |
| 1.5.3 | 5/12/25 | Victor Guzman | Removed AWS Lambda from Software Needs Section |
| 1.5.4 | 5/12/25 | Victor Guzman | Added AWS SDK for Javascript to Software Needs Section |
| 1.5.5 | 5/19/25 | Victor Guzman | Added Use Cases |
| 1.5.6 | 6/11/25 | Victor Guzman | Minor spelling/corrections to all sections. |
| 1.5.7 | 6/11/25 | Victor Guzman | Updated Manage Screen Output examples to match implementation. |
| 1.5.7 | 6/11/25 | Victor Guzman | Updated AWS Console screens to match implementation. |
| 1.5.8 | 6/12/25 | Victor Guzman | Updated Use Case section with testing data. |
| 1.6 | 6/13/25 | Victor Guzman | Added Deployment plan |
| 1.6.1 | 6/13/25 | Victor Guzman | Updated references |
| 1.6.2 | 6/13/25 | Victor Guzman | Updated New/Edit swimlanes to better reflect implementation |
| 1.6.3 | 6/13/25 | Victor Guzman | Updated Manage/Edit Screens to better reflect implementation |
| 1.6.4 | 6/13/25 | Victor Guzman | Updated UML diagrams/class descriptions to match implementation |
| 1.6.5 | 6/13/25 | Victor Guzman | Updated data dictionary to better reflect implementation |

# TABLE OF CONTENTS

[Project Description 4](#_Toc200747921)

[Current System or Processes to be Updated Overview 5](#_Toc200747922)

[Systems/Software Architecture with a description of how the materials will be used 6](#_Toc200747923)

[Hardware Needs (General Description) 8](#_Toc200747924)

[Software Needs 9](#_Toc200747925)

[User Interfaces – INPUT AND output 12](#_Toc200747926)

[Data Dictionary and Implementation Considerations 32](#_Toc200747927)

[Software Architecture and Structure 32](#_Toc200747928)

[Test plan and Use Cases 43](#_Toc200747929)

[Deployment plan 97](#_Toc200747930)

[References used 103](#_Toc200747931)

## Project Description

This proposal outlines the development of a system which will enable an organization to collect and store expense data generated by users during their work. The system will largely consist of two parts: a user application and an administrative system. The user application will let users quickly submit simple digital reports and evidence of expenses, such as pictures of receipts, to a central storage location where they can be reliably and securely accessed as necessary. Users will access the user application from any internet-capable device, such as a smartphone. The user application will focus on being easy to use and accessible to users with a low level of technical proficiency. The administrative system will allow privileged users provision new users and manage user access to the system. Administrative users will require a greater level of technical proficiency and a comprehensive understanding of the system. The new system is intended to replace existing expense-tracking solutions in two categories: manual solutions (i.e. keeping physical receipts), and existing substitutes (i.e. applications currently on the market).

This new system is necessary to overcome the drawbacks and vulnerabilities of the two aforementioned solutions (manual solutions and existing substitutes), while preserving advantages of each. Manual solutions involve the storage and processing of physical documents, which are vulnerable to wear and misplacement, but are relatively low friction for most users, as they simply have to remember to keep documents such as receipts in a safe place. The proposed system will keep friction low by simplifying the process of reporting expenses, while eliminating physical documents as early in the process as possible to reduce the risk of loss or damage. Existing substitutes involve using a third-party application to store expense data online by scanning documents such as receipts with the camera on a mobile device, requiring users to pay a subscription fee to maintain access to their data. The proposed system will maintain the advantages of allowing users to record expense data on a mobile device, while eliminating complications such as document scanning errors and complicated interfaces.

The project is to be completed by June 13, 2025, and will consist of two major phases: documentation and development. The documentation phase will be focused on drafting of this SDD, which will outline in detail the requirements, development, and functions of the proposed system. The development phase will take place over four weeks, consisting of individual one-week development cycles. These cycles will be concerned with deploying the architecture needed to host the system, development of the user application, integrating the user application with the administrative backend, and a final testing cycle.

Project Goals:

* Securely store expense data in a digital format.
* Increase custody of data.
* Reduce dependency on third-party solutions for expense tracking.
* Minimize human error introduced while processing expense documentation.
* Reduce the use of physical documents.

Project Terms:

|  |  |
| --- | --- |
| **Term** | **Definition** |
| Friction | The amount of effort it takes to complete an action, such as submitting a report. |
| Metadata | Data which contains information about other data, such as the location an image was taken or the time at which a report was submitted. |
| Data Custody | The management, storage, and protection of data. |
| OCR | Optical Character Recognition. The extraction of text data from images or documents, typically done by processing an image of a document taken with a smartphone. |
| Serverless | Cloud technologies provided by companies such as Amazon in which customers do not have to manage their own servers. |
| Webapp | Web application. An application that is created with web technologies and runs in a web browser. |
| AWS | Amazon Web Services, a leading cloud technology provider. |
| Token | A unique sequence of characters that can be used to identify a user for access to a system. |

## Current System or Processes to be Updated Overview

This project is primarily targeted at replacing two different types of systems which may be currently implemented and fall into two categories: existing substitutes and manual solutions.

**Existing substitutes:**

Existing substitutes are third-party systems which are currently available and would generally fulfill the same functions as the proposed system. These systems allow users to install a proprietary mobile application (iOS/Android) which can scan expense documentation using the device’s built-in camera. These scanned documents are automatically converted into expense data using OCR and are sent to a third-party server for storage. Full access to the data requires paying a subscription fee. The data can be managed on the platform directly by using the mobile app or be exported in one of many popular formats for applications such as Excel or QuickBooks.

The steps for submitting expense data are as follows:

1. A user receives a receipt.
2. The user opens the mobile app for the system and scans their receipt.
3. The user disposes of the receipt.
4. The app uses OCR to convert the scanned image into expense data.
5. Expense data is sent to a third-party server for storage.

Pros:

* Handling of physical documents is minimized, as receipts and the like may be disposed of once scanned.
* Expense data can be accessed and worked with immediately, as it is stored digitally.
* Expense data can be accessed from multiple different locations.

Cons:

* We must pay a subscription fee to maintain access to expense data.
* Users must download and install a mobile app to report expenses.
* Users may implicitly trust technology like OCR to accurately capture data from scanned documents.
* A third-party organization is responsible for the safety and security of our data.
* If the system is discontinued or otherwise compromised, access to our data may be lost.
* Good lighting conditions are required for OCR to work properly.

**Manual solutions:**

Manual solutions are systems which largely consist of the storage and manual processing of physical expense documents. These solutions are typically low-tech and only involve computer systems at the final step. Expense documents such as receipts are collected by users as they work and delivered to a central office for storage. Documents are stored there until they are ready for batch-processing by data input personnel. Data is input on a workstation machine (Microsoft Windows) to an application such as Excel or QuickBooks. The data is saved locally on workstation machines and access is limited to the user of that workstation or anyone they send the data to.

The steps for submitting expense data are as follows:

1. A user receives a receipt.
2. The user physically stores the receipt in a safe place alongside other receipts.
3. The user submits their receipts to the central office.
4. Central office stores the receipts.
5. Stored receipts are processed in batches at regular intervals.
6. Processed data is stored locally on workstation machines.

Pros:

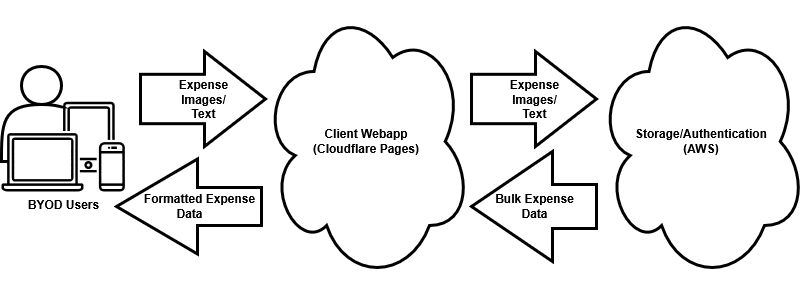
* Full ownership of all data.
* System is low friction for most users, as they simply store physical documents as they’re received.

Cons:

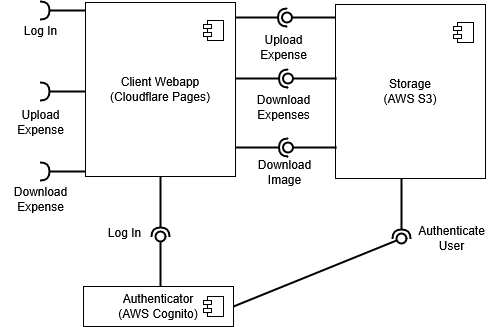
* Data must be manually processed before it can be meaningfully worked with.
* Physical documents are prone to loss, damage, and wear.
* System is extremely high friction for users tasked with processing expense data.
* Data is stored in a single location and vulnerable to loss.
* Data cannot be easily accessed by multiple users.
* Data may not maintain parity if worked on by multiple users.

## Systems/Software Architecture with a description of how the materials will be used

The system will consist of two major components: a webapp which will provide the user interface, and a backend consisting of several components provided by Amazon Web Services which will provide authentication and data storage. The following diagram provides a high-level overview of the basic components of the system and how they interact with each other.



Users will use a web browser to access a web application, which will provide an interface to upload new expense data (images and text) and download expense data (CSV files). The web application will interface with API endpoints exposed by an AWS backend which will allow it to securely upload and download data and provide user authentication functions. A more detailed component diagram is provided below, which gives more context to the specific interfaces the components have and their providers.



As stated previously, the interfaces exposed by the webapp will be accessed by user devices, which are not considered part of this system. It is assumed that users will use the devices which are most convenient for them to access the webapp (BYOD). Letting users use the devices they are familiar with lowers training and equipment costs and makes accessing the system more convenient for users.

**Why a webapp?**

The primary advantage of web applications is their portability. Users will not be required to install any specific software on their devices, and they will be able to seamlessly switch between devices by simply accessing the website and logging in onto a new device. This also provides a level of futureproofing, as it means that there are far fewer restrictions on which devices this application can run on. A web application also lowers development overhead, as the code written will be far more likely to be immediately compatible with multiple platforms versus code written for a native application.

**Why go serverless?**

Serverless computing offers three major advantages: high-availability, a pay-as-you-go pricing structure, and a high degree of scalability. With serverless architecture we can leverage the high availability of providers such as Amazon Web Services, without having to worry about deploying and maintaining the large amount of hardware that would be required to attempt to match their availability. As these serverless technologies are pay-as-you-go, we only have to pay for the bandwidth and storage that we use, and nothing more. Technologies like AWS Cognito and AWS S3 mean that we’re never paying for hardware while it’s sitting idle. Finally, serverless technologies are highly scalable, which adds a basic level of futureproofing to this system.

**Why Amazon Web Services?**

AWS is a leading cloud technology provider, which makes it is both highly reliable and well documented. Developing this application using AWS technologies means that any functions which are provisioned through them will be more easily maintained and expanded upon by future developers. Using multiple AWS services together also means we can leverage their strong integration, leading to lower development times and more reliable systems.

## Hardware Needs (General Description)

Development hardware will consist of a mid-range Windows 11 computer and access to several Amazon Web Services. These services will also make up the operational expenditures of the system as outlined below.

**Development Hardware Required**

**Desktop Computer** – This is the machine which will be used to develop the code for the system and access the cloud components onto which the system will be deployed. The desktop computer is expected to include all necessary internal components to function and come provisioned with a licensed copy of Windows 11. The expected cost of this computer is $500.

**Computer Mouse** – To be used with the development computer and not expected to exceed a price of $30.

**Computer Keyboard** – To be used with the development computer and not expected to exceed a price of $60.

**Computer Monitor** – Any display with a resolution of at least 1080p will work for this application and is not expected to exceed a cost of $100.

**Apple Macbook Air** – Will be used to test the application on MacOS devices. The expected cost is $999.

**Apple iPhone 15** – Will be used to test the application on iOS devices. The expected cost is $699.

**Samsung Galaxy A15** – Will be used to test the application on Android devices. The expected cost is $199.

**Amazon Cognito** – Amazon Cognito offers free access for applications provisioning up to 10,000 monthly active users, which is more than enough for our application.

**Amazon S3** – Amazon S3 will be used to store the data for this project. The S3 Archive Instant Access tier will be used, which is priced at $0.004 per GB per month. The costs to store text data are negligible, thus the bulk of the storage used will be in image storage. A 3MB image limit will be imposed on the system, so if we assume even a single terabyte of S3 storage ($4.01 per month) we would be able to store ~350,000 receipt images. We will consider this the worst case over the long term and use a terabyte of S3 storage for our high-end estimate.

**Cloudflare Pages** – Static hosting provided via Cloudflare. While there are several pricing tiers available, the free tier provides unlimited bandwidth for static hosting, so there is no associated cost with this item.

**Usage Hardware Required**

**Web Capable Device** – Users are expected to use their preferred devices (personal or existing at work) to access the application.

**Maintenance Hardware Required**

The hardware required for maintaining this system is expected to be the same as that which is required to develop the system. The expected monthly cost of this application will vary by usage but is not expected to exceed the cost of one terabyte of Amazon S3 storage in the Archive Instant Access Tier ($4.01), bringing the expected monthly cost of the application to $4.01 at the high end.

|  |  |  |  |
| --- | --- | --- | --- |
| **Final Hardware Cost Summary** | | | |
| Hardware Name | Free | Capital Expenses | Operational Expenses (Monthly) |
| Desktop Computer |  | $500 |  |
| Computer Mouse |  | $30 |  |
| Computer Keyboard |  | $60 |  |
| Computer Monitor |  | $100 |  |
| Apple Macbook Air |  | $999 |  |
| Apple iPhone 15 |  | $699 |  |
| Samsung Galaxy A15 |  | $199 |  |
| Amazon Cognito | X |  |  |
| Amazon S3 |  |  | Up to $4.01 |
| Cloudflare Pages | X |  |  |
| Final Expected Hardware Costs |  | $2587 | Up to $4.01 |

**Assumptions**

It is assumed that the organization will allow users to access the application through their own devices or provide access to devices themselves. It is assumed that the organization has consistent access to an internet connection and will be maintaining access in the future. These assumptions are outside the scope of this project.

## Software Needs

The software needs for this project are relatively light, as most of the heavy lifting will be done by AWS. There are no operational expenditures associated with the software requirements of this project, and capital expenditures are limited to software costs which are bundled with hardware costs, such as operating system licenses. Like the hardware, the software needs are also split into three categories.

**Development Software Required**

This software is required for the development and initial deployment of the application. There is no paid software in this category, however some of the software is included or only available as part of a hardware purchase, which is outlined in the hardware section above. Some of this software will be required for the maintenance of the application, so it will be listed separately in that section as well.

**Windows 11** – The operating system used for development. Windows 11 is paid software, however it will be included in the cost of the development desktop computer in the hardware section.

**Visual Studio Code** – Integrated development environment used for creating the client website and Lambda functions. VSCode is free software.

**Internet Browsers** – The client application should be tested on all popular browsers, so they’ll be required for development. Currently, there are no paid versions of any popular browser, and many are installed by default on the operating systems we’ll be testing on. Major browsers in this list are Google Chrome, Mozilla Firefox, Microsoft Edge, Apple Safari, Brave, and Opera. See the usage software section for a specific list of operating system/browser combinations to be tested.

**MacOS** – MacOS will be used for testing the application on Apple PC platforms. This operating system is included with the purchase of an Apple computer at no extra cost.

**iOS** – iOS will be used for testing the application on Apple mobile platforms. This operating system is included with the purchase of an Apple mobile device at no extra cost.

**Android OS** – Android will be used for testing the application on mobile platforms running the Android operating system. This operating system is included with the purchase of an Android-compatible phone.

**Bootstrap.js** – Bootstrap.js will primarily be used to enable the developer to quickly implement responsive user interfaces. Bootstrap.js is free.

**AWS SDK for Javascript v2** – The AWS SDK will primarily be used to streamline calls to the AWS S3 REST API.

**Development Software Skills Overview**

**Web Development** – The developer of this system will need to be familiar with frontend web development using JavaScript, HTML, and CSS. They will need to be able to create a functional user experience and be able to interface with Amazon API endpoints to upload and download data.

**JavaScript** – The developer of this system will need to be able to effectively use JavaScript to implement the needed functionality into the AWS Lambda functions for this application.

**Amazon Web Services** – The developer of this application will need to have a general familiarity with AWS, including the ability to deploy and manage services using either a web browser or terminal application. A terminal application and web browser are available by default on the Windows 11 system used to develop this application. The developer will need specific proficiencies in the following AWS services:

* AWS Cognito – Used to authenticate users accessing the application.
* AWS S3 – Used to store system data.

**Usage Software Required**

The software interfaces for this application will be accessed entirely through a web browser, thus only a modern web browser and the respective operating system for the hardware platform the site is being accessed from will be required. The operating systems and browsers the application will be tested on are listed below:

**Windows 11 Browsers:**

* Microsoft Edge
* Google Chrome
* Mozilla Firefox
* Brave
* Opera

**MacOS Browsers:**

* Apple Safari
* Google Chrome
* Mozilla Firefox
* Brave
* Opera

**iOS Browsers:**

* Apple Safari
* Google Chrome

**Android Browsers:**

* Google Chrome

**Usage Software Skills Overview**

**Basic Web Browsing** – Users will need to have a basic proficiency in using the device of their choice to access the web page via a web browser.

**Basic File Handling** – Users will need to understand how to access files such as images on their device when uploading image data. Users downloading expense data will need to understand how to download files on their web browser.

**Maintenance Software**

This application will not require regular maintenance, however emergency maintenance due to critical system flaws or other errors may be required. Situations in which all or part of the application are to be reengineered or rewritten should also be considered maintenance. The software required for maintenance will largely be the same as the software required for initial development, as such costs associated with emergency upgrades or maintenance should be estimated to match the cost of the initial development of this project.

**Maintenance Software Skills Overview**

As with the maintenance software requirements, the maintenance skill requirements are largely expected to be the same as those required for development. Naturally this does not apply to situations in which maintenance requires reimplementing a large portion of the system, which may have requirements which fall outside the scope of this document.

|  |  |  |  |
| --- | --- | --- | --- |
| **Final Software Cost Summary** | | | |
| Software Name | Free | Included with Hardware | Paid |
| Windows 11 |  | X |  |
| Visual Studio Code | X |  |  |
| Internet Browsers | X |  |  |
| MacOS |  | X |  |
| iOS |  | X |  |
| Android OS |  | X |  |
| Bootstrap.js | X |  |  |
| Final Expected Software Cost | $0 | | |

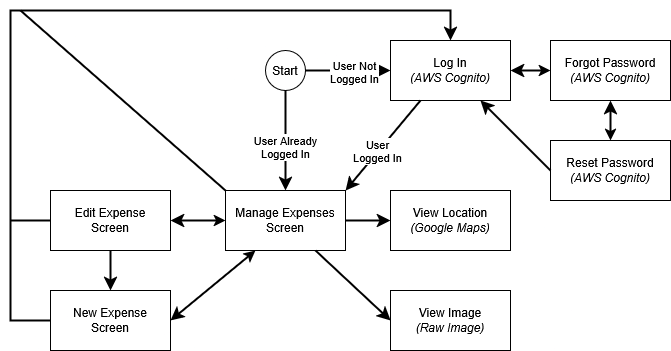
## User Interfaces – INPUT AND output

The user interface for managing expense data in this application can be broadly split into three general categories: the log in screens, the new expense record creation screen, and the expense record management screen. The log in screens are integrated with AWS Cognito and don’t need to be created as part of this project. As user account creation is handled through the AWS Cognito console, there is also no need to create an interface for creating new user accounts. The system will have four different outputs: images attached to expense data, Google Maps URLs created from location data, expense data in a QuickBooks compatible 3-column CSV format, and detailed expense data in CSV format.

The interfaces for adding and managing users are accessed via the AWS console. A Cognito user will be provisioned with limited access to the Cognito interface and will be able to use said interface for administrative tasks involving the management of users and their permissions.

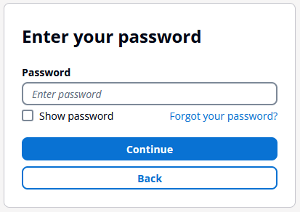
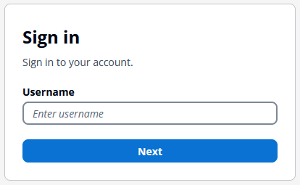
Generally speaking, the only laws which apply to the data transmitted by this system are those which require individuals to be notified if their personal information (i.e. names, emails, phone numbers) are leaked through a data breach. These laws are present in all 50 states. Otherwise, the only laws which exist for the type of information processed by this system do not apply in this case as the scale of the data storage/processing is not large enough. The system does not have any features to detect data breaches and provides no mechanism for notifying users of such an occurrence. Security for elements that could potentially compromise the security of this system, such as user emails, is outside of the scope of this project. Appropriate parties should be made responsible for monitoring employee security practices as part of a separate effort. As account management will be done through AWS, any data breach on their end will be their responsibility to handle.

The flow between the interfaces for managing expense data is as follows:



**Expense Data Management Interfaces**

**Log In**



The first screen a user will see is the log in AWS Cognito login screen. This screen will not be displayed for users with a currently valid login session, and they will instead be redirected automatically to the Manage Expenses screen.

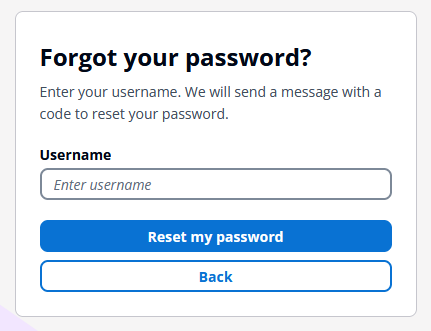
Users who are not currently logged in will be prompted to provide a username and password, after which they will be successfully logged in and automatically redirected to the Manage Expenses screen. The log in screen images are representative of the final interface, as no development needs to be done to implement the login interface of AWS Cognito.

Inputs:

* Username – The username provided to the user by the account administrator.
* Password – The user’s password.

Outputs: None

**Forgot Password**



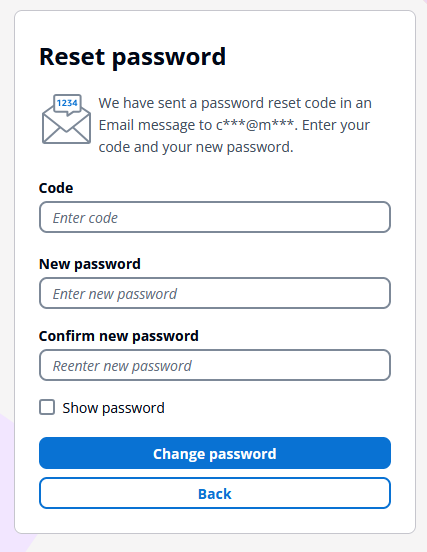
Clicking on the “Forgot your password?” link in the Log In interface will display this interface, which first asks a user for their username.

Inputs:

* Username – The username provided to the user by the account administrator.

Outputs:

* Recovery Email – An email sent to the email account associated with the username entered containing a six-digit security code.



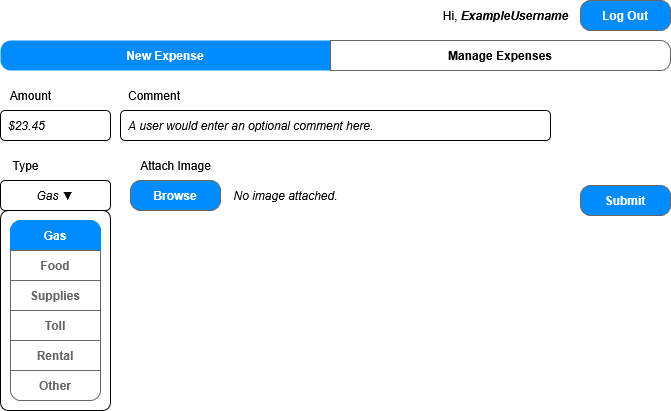
The user is then sent an email containing a six-digit security code, which they can provide at this interface to confirm their identity and set a new password.

Inputs:

* Code – A six-digit security code found in the recovery email sent to the user’s email address.
* Password – The user’s new password.
* Confirm Password – The same password again for confirmation.

Outputs: None

**New Expense**



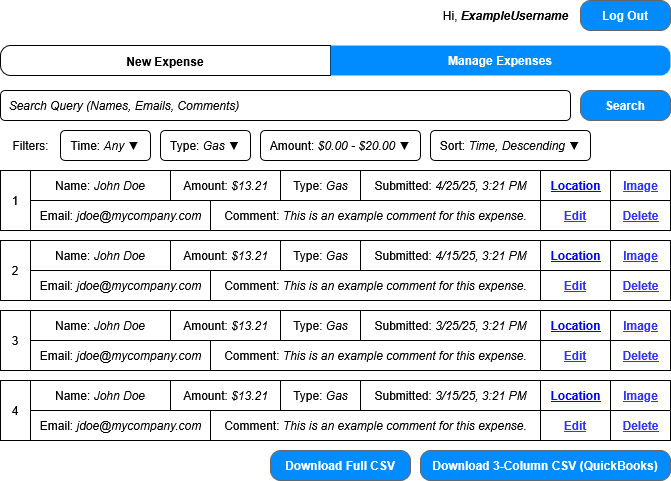
The New Expense screen allows users to input expense information to add to the database. Data input into the fields will be validated to ensure it meets the format requirements. The data submitted on this screen will be submitted through an AWS API (which will validate the users identity token) and be stored in S3 storage. The Log Out button can be used to return to the Log In screen.

Inputs:

* Amount – Dollar cost of the expense, represented as a positive number.
* Comment – Optional text string with additional information on the expense.
* Type – The type of expense selected from pre-defined categories.
* Image – Optional image relevant to the expense, such as a receipt. 3MB max size in JPG, PNG, or BMP format.
* Time – A hidden value generated by the device detailing the time in which the report is submitted. Unix epoch format.
* Location – Latitude and longitude provided (if possible) by the device.
* Token – An AWS Cognito authentication token representing the user’s identity.

Outputs: None

**Manage Expenses**



The Manage Expenses screen is used to view information about existing expenses. Users can search for and filter expenses, and then output the displayed expense data in one of two CSV formats. Users have links for viewing the location of the expenses in Google Maps and links for viewing the images associated with the expenses directly. Clicking the Delete button will bring up a confirmation prompt for deleting the relevant expense record. Clicking the edit button will open the Edit Expense interface.

The interface for the filter dropdowns and the delete confirmation prompt are showcased at the end of this section.

Inputs:

* Search Query – Text string used for searching expense data. The string will be matched against names, emails, and comments.
* Time – A time and date range used to filter the displayed results. Dates are validated by the dropdown form showcased at the end of this section.
* Type – An expense type to filter expense data by.
* Amount – A currency range used to filter the displayed results. Amounts are validated by the dropdown form showcased at the end of this section.
* Sort – One of four sort types used to sort the displayed expense data.
* Token – An AWS Cognito authentication token representing the user’s identity.

Outputs:

* QuickBooks CSV – A CSV containing the displayed expense data formatted in a QuickBooks compatible CSV format.
* Full CSV – A CSV file containing all displayed non-image expense data.
* Image – A direct link to the expense image for that record. Depending on the device the image may either be displayed directly in the browser or downloaded directly.
* Location – A Google Maps URL for the expense location based on the geolocation data generated with the expense, generated by appending the latitude and longitude of the expense with a Google Maps URL prefix.

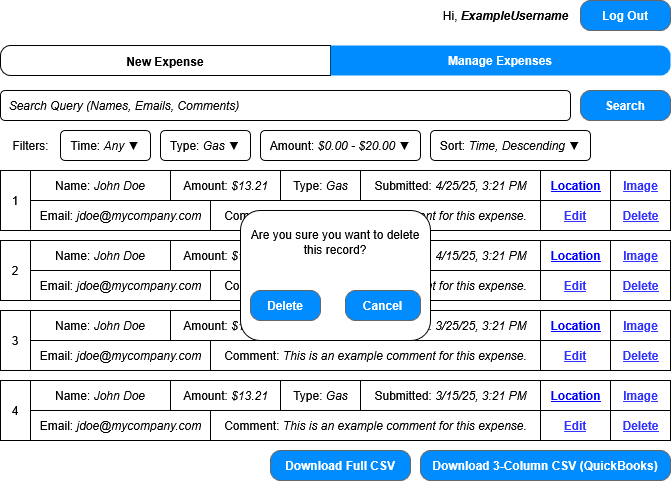
Output Examples:

|  |  |  |
| --- | --- | --- |
| **QuickBooks Compatible 3-column CSV** | | |
| Date (dd/mm/yyyy) | Description (Type + Comment) | Amount (Negative) |
| **Example** | | |
| Date,Description,Amount 4/24/2025,Supplies: Tape measure,-13.24 | | |

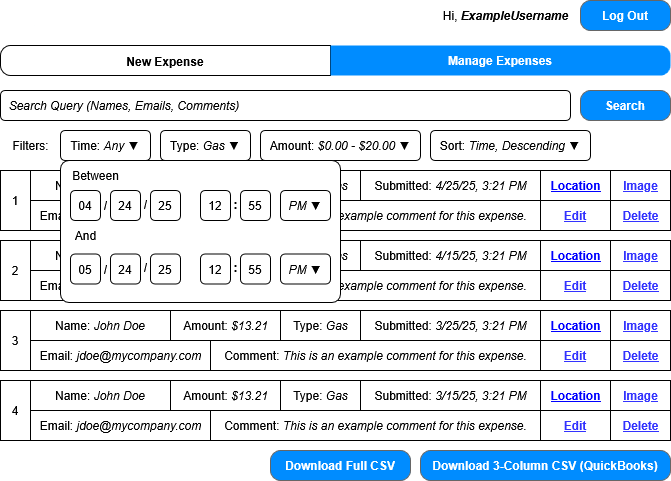
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Full Expense Data CSV** | | | | | | | | | |
| UserID | FirstName | LastName | Email | Amount | Type | Time | Location | ImageName | Comment |
| **Example** | | | | | | | | | |
| ID,FullName,Amount,Type,Time,Email,Comment,Latitude,Longitude,ImageID  1749809038311,John Doe,13.24,Supplies,1749809038311,jdoe@email.com,Tape measure,46.7379,-119.9028,1749809038311.png | | | | | | | | | |

|  |
| --- |
| **Google Maps Location URL** |
| https://www.google.com/maps?q=46.7379,-119.9028 |

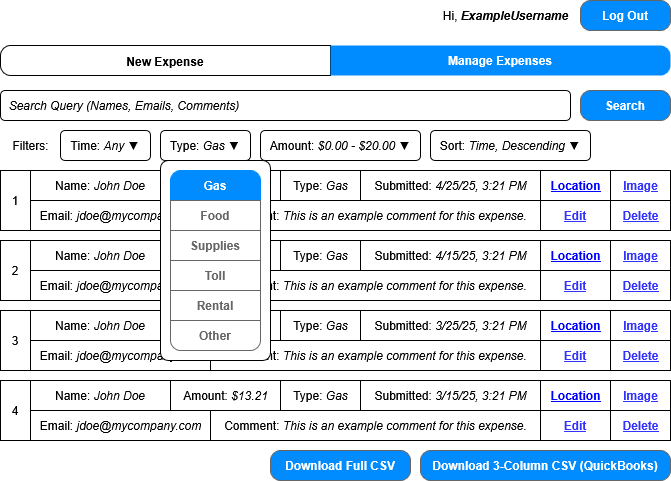
**Delete Confirmation Prompt**

****

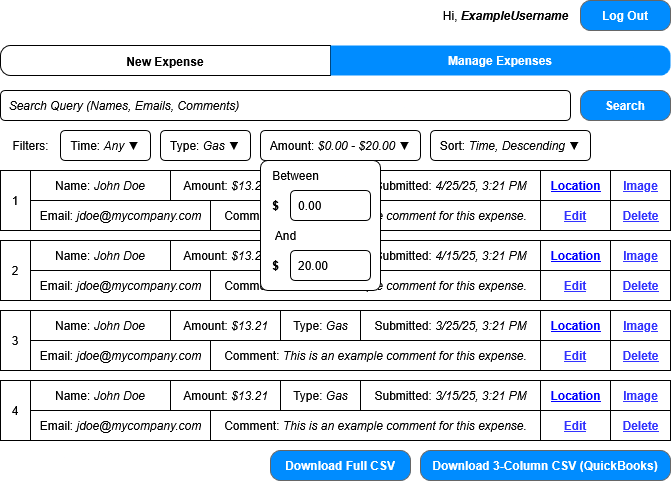
**Time Filter Dropdown**

****

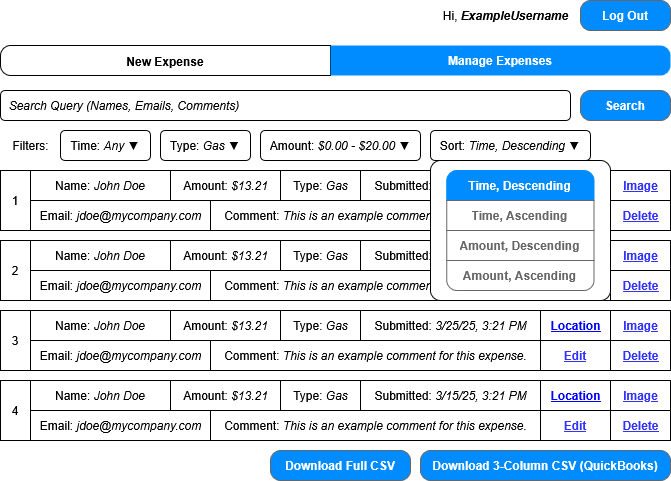
**Type Filter Dropdown**

****

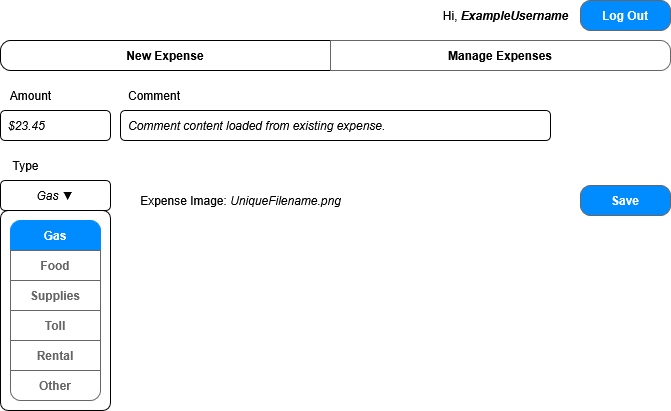
**Amount Filter Dropdown**

****

**Sort Filter Dropdown**

****

**Edit Expense**



The Edit Expense interface is nearly identical to the New Expense interface. The only major difference is that this interface will come pre-loaded with the existing expense data for the record being edited, which means that certain information (such as time and location data) will not be sent or updated.

Inputs:

* Amount – Dollar cost of the expense, represented as a positive number.
* Comment – Optional text string with additional information on the expense.
* Type – The type of expense selected from pre-defined categories.
* Image – Optional image relevant to the expense, such as a receipt. 3MB max size in JPG, PNG, or BMP format.
* Token – An AWS Cognito authentication token representing the user’s identity.

Outputs: None

**View Location**

Location data does not have an in-app interface and instead clicking on the Location button in the Manage Expenses screen will load a Google Maps URL generated from the geolocation data associated with the relevant expense record.

Inputs: None

Outputs: None

**View Image**

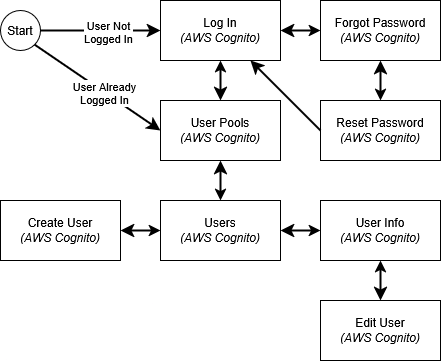
Images do not have a formal interface and will instead either be viewed directly in the browser or downloaded to be opened in third-party image software.

Inputs: None

Outputs: None

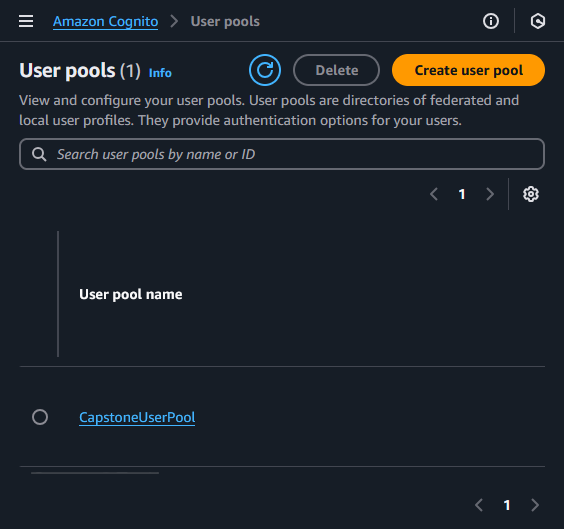
**User Management Interfaces (AWS Cognito Console)**

The flow between the interfaces for managing user accounts is as follows:

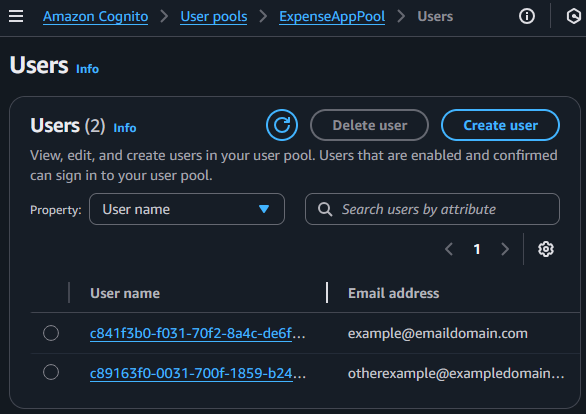


Note: The Log In, Forgot Password, and Reset Password interfaces are identical to those shown previously.

**User Pools**



Users given AWS Cognito management privileges can use the same Log In interface shown previously to access the AWS Cognito management console. The first interface they are shown will allow them to choose a user pool, of which only one will be available. Importantly, despite the “Create user pool” button being available, no users will not have permissions to create a user pool and thus will receive an access denied message. Creating new user pools is outside of the intended functionality of this system.

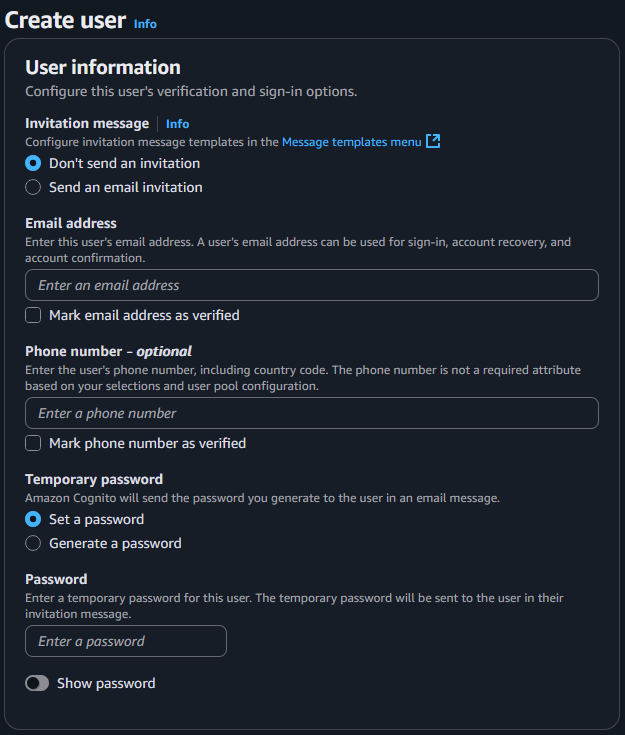


Upon clicking the only available user pool, users will be able to see the user management interface, which will display all available users. Users can click the “Create user” button to create a new user, or click on a User name to manage an existing user. Selecting the user radio button to the left of their name will enable the “Delete user” button which will display a confirmation prompt to delete the user.

Inputs: None

Outputs: None

**Create User**



This interface allows users to create a new user. An email address must be provided. An email will be sent to the provided address with a temporary password, which the user will use in the Log In screen shown previously. Unlike a regular log in, this log in will redirect the user to the First Time User Log In screen, which will ask them to change their password and provide other basic details.

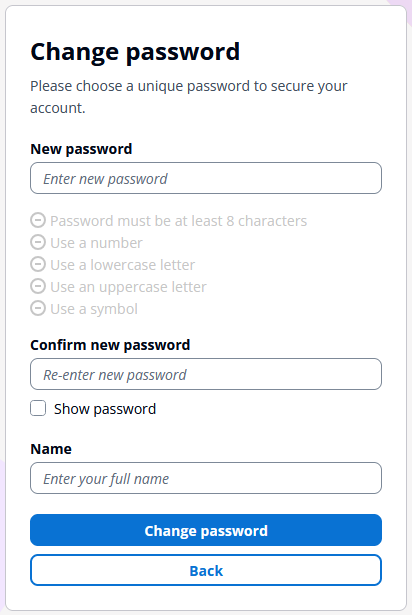
Inputs:

* Email Address – A valid email address the user will use to receive their temporary password and for resetting their password if forgotten.
* Phone Number – An optional phone number that can also be used for receiving a temporary password or resetting the user password.

Outputs:

* New User Email – An email will be sent to the email address provided for the user which will contain a username and temporary password for the user to log in for the first time.

**First Time User Log In**



Accessing the Log In page (shown previously) will allow the new user to use their temporary password to log in for the first time, after which they will be asked to change their password and enter their name. Administrative users can freely edit user’s names as necessary and these names are only used as a means of helping identify submitted expenses, and not for authentication. Once their password is changed the user will be automatically signed in and redirected back to the main application page.

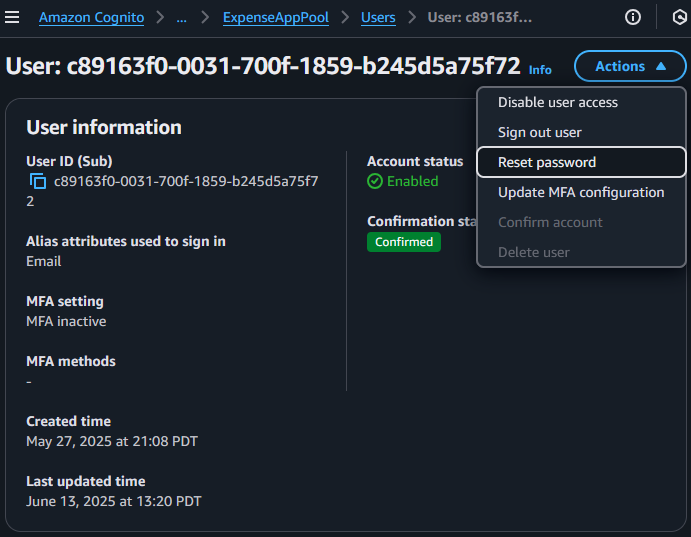
Inputs:

* New Password – A new password meeting the validation criteria.
* Confirm New Password – Confirmation of the new password.
* Full Name – The user’s full name.

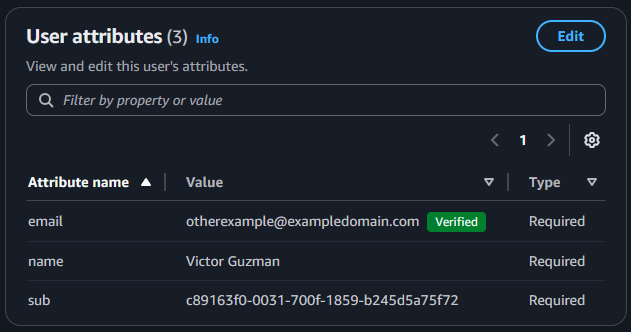
Outputs: None

**User**

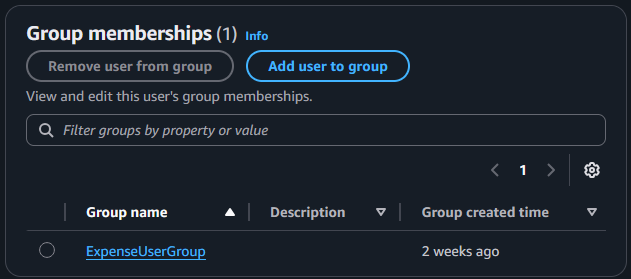
Clicking on a username in the Users interface will open the User interface, which gives access to features such as password resets, group membership, and user editing.



The first panel is the “User information” panel, which displays basic user information and provides access to features for disabling the user account, signing the user out, resetting their password, updating their access credentials, and deleting their account. The “Confirm account” feature is unused as accounts are created in a confirmed state.

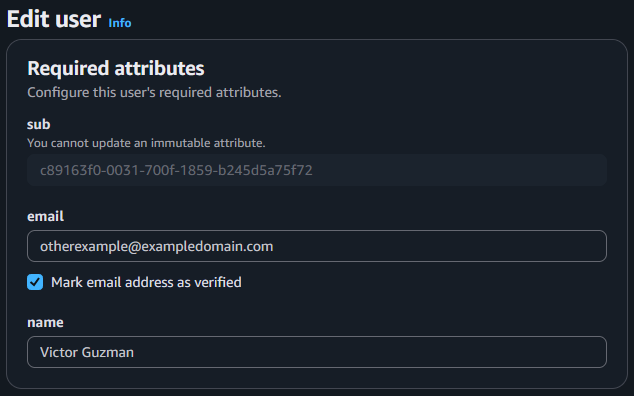


The “User attributes” panel is displayed below and shows the user attribute information which includes details like their name and email. The Edit button on the top right leads to the Edit User interface.



Finally, the “Group memberships” panel allows a user to manage group data for the selected user. An existing group can be removed by clicking the radio button next to it’s name, and a user can be added to a new group by clicking the “Add user to group” button.

**Edit User**



The Edit User interface allows a user to edit attributes for the selected user.

Inputs:

* email – A valid email address the user will use to receive their temporary password and for resetting their password if forgotten.
* name – The user’s full name.

Outputs: None

## Data Dictionary and Implementation Considerations

The data dictionary is contained in the embedded Excel spreadsheet below. Double clicking on the icon will open the sheet in a new Excel window. If this fails please see the data dictionary spreadsheet uploaded alongside this document instead.



## Software Architecture and Structure

The software architecture for this system is relatively simple. Javascript will be the main language used, and code will only be written for the client application. The AWS SDK will be used to streamline access to the AWS S3 API for uploading and downloading expense data. Bootstrap.js will be used to develop the interface, as it provides an easy way to create responsive user interfaces.

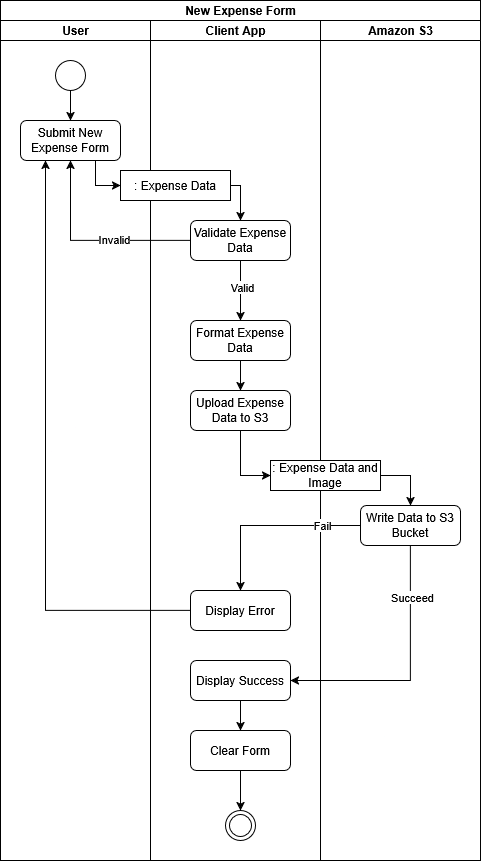
The backend software for this application will consist solely of AWS S3 and AWS Cognito. The pre-built login interface for AWS Cognito will be used to authenticate users, so the client application will only need to be able to detect that a user is not logged in and redirect the user to the login URL. AWS S3 REST API will be accessed via the AWS SDK Javascript library mentioned above. In short, very little programming will have to be done to impement the backend features of the application. The only validation performed by the backend will be a configuration setting in S3 to not allow files to be overwritten by default, with the JSON file used to store expense data being an exception.

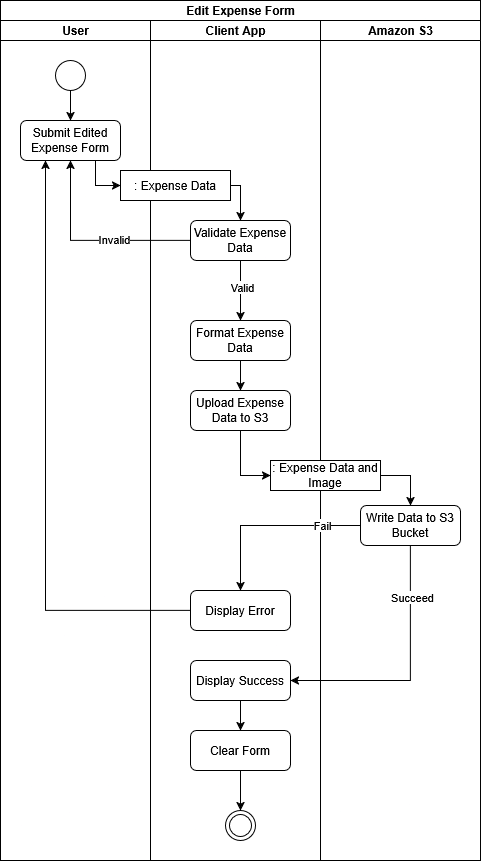
As mentioned, the frontend web application will largely be written in Javascript. HTML and CSS will be used as necessary to develop the interface, but most of the styling will be done by invoking the built-in Bootstrap.js HTML classes. Application logic will be written entirely in Javascript.

**New/Edit Expense Diagram**

The diagrams for these two systems are largely identical as their logic and implementation will be highly similar. It is important to note, however, that code for rendering these two interfaces will not be shared, as certain elements, such as the file picker element for image uploads will be handled differently.

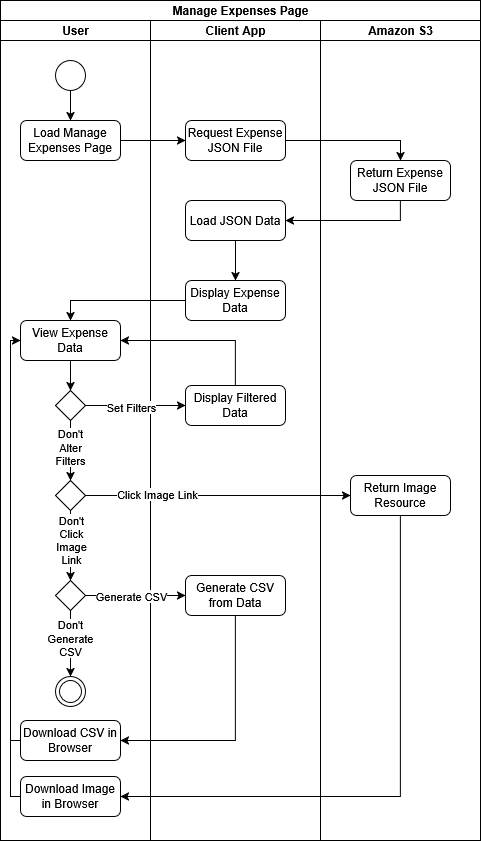
The starting state of the Edit Expense/New Expense diagrams is that the user has navigated to the Edit Expense/New Expense page respectively. The end state is that the user has successfully uploaded their validated data to the AWS S3 backend.





**Manage Expenses Diagram**

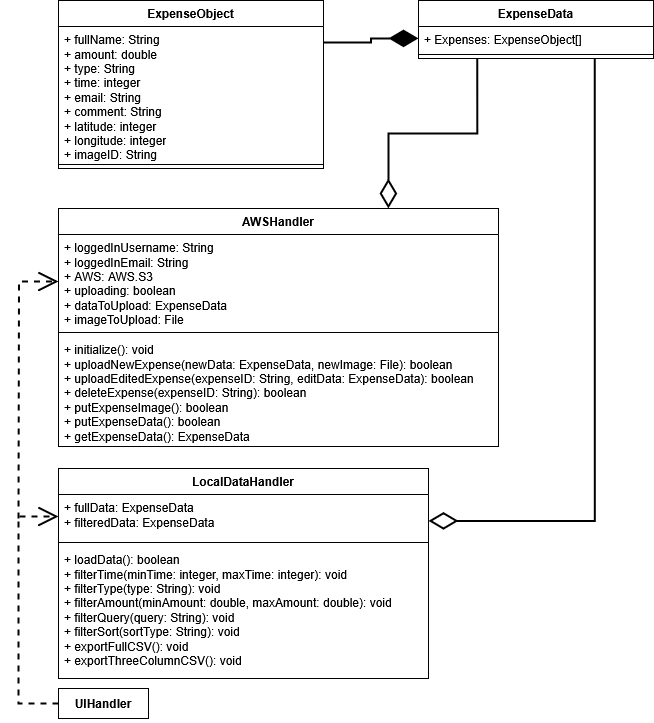
The Manage Expenses interface has more interactive UI elements than the previous two, so the diagram is a bit more involved. The starting state for the application is that the user has accessed the Manage Expenses page, but in this case it’s represented in the diagram as an action to better demonstrate where the action is initiated. The end state for the diagram is that the user has done all the actions they required (Viewing expense data, viewing expense images, downloading CSVs) and has no more need to interact with this interface.



**UML Class Diagrams**

The following are the implementation details for the data handling classes for the application, followed by the classes for the user interface. Each UML diagram will be followed by class descriptions, including a short description of each class attribute and method. It is important to note, that these diagrams may not be 1:1 with the code implemented, as their function is to demonstrate the intended logic of the application in a clear manner. For example, an Interface is used to denote the methods the UIHandler class will use when interfacing with the UI classes, but there does not exist a formal way to implement an interface in Javascript. This document assumes the developer will not implement logic to enforce the functionality of an interface, but these implementation details will be left up to them.

**Data Handling Classes**



*Note: UIHandler is described in the User Interface Classes section below.*

**ExpenseObject**

This object has no methods and simply acts as a container for a single expense record. The data in this object is defined in the data dictionary of this document, so it will be omitted from this section.

**ExpenseData**

This object simply contains a series of ExpenseObjects, stored as key value pairs, where the key is used as a unique expense record ID and the value is an ExpenseObject for that record. An ExpenseData object is typically meant to represent the full contents of the expense record database, but it may also be used to store partial expense data, as in cases where the data is being filtered.

**AWSHandler**

This class is meant to handle all functions related to interfacing with AWS Cognito and AWS S3. The AWS Cognito interfacing is largely concerned with exposing the logged in username to the user interface rendering functions, so the class is mainly concerned with interfacing with AWS S3 via an instance of the AWS.S3 interface object from the AWS SDK library. Descriptions for AWSHandler attributes and methods are listed below:

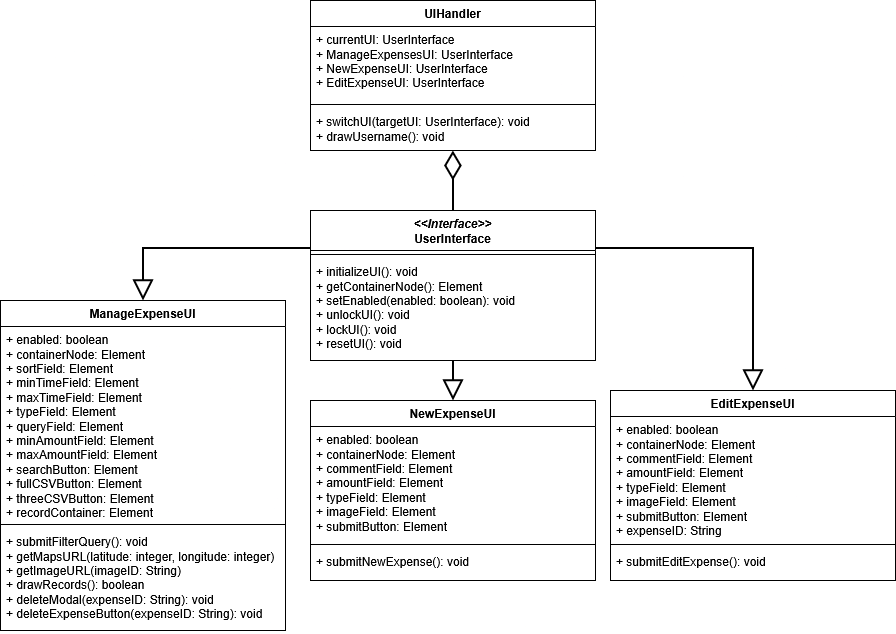
* loggedInUsername – A string containing the full name of the logged in user, used by user interface methods.
* loggedInEmail – A string containing the email of the logged in user, used by user interface methods.
* AWS – An instance of the AWS.S3 object used to streamline access to the S3 REST API.
* uploading – Whether the AWSHandler class is currently handling a file upload.
* dataToUpload – The ExpenseData object that will be uploaded, used to validate the data was uploaded correctly.
* imageToUpload – The image that will be uploaded.
* initialize() – Initialization method called when the page is first loaded, instantiates AWS and loggedInUsername.
* uploadNewExpense() – Upload a new expense. Downloads a fresh version of the expense data from S3 and merges that data with the new expense. Stores the data to dataToUpload and imageToUpload then attempts to upload the data to S3, validating the uploaded data against dataToUpload and imageToUpload once done.
* UploadEditedExpense() – Similar to uploadNewExpense(), but takes the edited data and updates the ExpenseObject at the expenseID key instead. Stores the data to dataToUpload then attempts to upload the data to S3, validating the uploaded data against dataToUpload once done.
* deleteExpense() – Downloads a fresh version of the expense data from S3 and removes the key at expenseID, then reuploads it. Stores the data to dataToUpload then attempts to upload the data to S3, validating the uploaded data against dataToUpload once done.
* putExpenseImage() – A utility function used to call the uploading functions of the AWS object when uploading an image.
* putExpenseData() – A utility function used to call the uploading functions of the AWS object when uploading a JSON file.
* getExpenseData() – A utility function used to call the downloading functions of the AWS object when downloading a JSON file.

**LocalDataHandler**

This class is primarily concerned with manipulating the expense data stored in memory by the client application. The class has methods for loading the data, filtering it, and exporting it. Descriptions for LocalDataHandler attributes and methods are listed below:

* fullData – The full expense data object obtained from S3 via the AWSHandler class.
* filteredData – The result data converted to an array via Object.entries() which holds filtered data that results from filtering methods below.
* loadData() – Load or reload fresh data from AWS S3 via AWSHandler.
* filterTime() – Filter the expense data in filteredData by a minimum and maximum time.
* filterType() – Filter the expense data in filteredData by type.
* filterAmount() – Filter the expense data in filteredData by a minimum and maximum amount.
* filterQuery() – Filter the expense data in filteredData by matching a string query against the name, comment, and email fields.
* filterSort() – Sort the data ascending/descending based on either amounts or times.
* exportFullCSV() – Convert the contents of filteredData to a CSV file and trigger a browser download.
* exportThreeColumnCSV() – Convert the contents of filteredData to a three-column CSV expense format and trigger a browser download.

**User Interface Classes**

****

**UIHandler**

This class manages the state of the UI as the user switches between different interfaces. This is accomplished by adding and removing HTML Elements containing full interfaces to and from the DOM. The UIHandler class is relatively simple, and is mainly used as a way to collect the user interface classes and ensure they are properly enabled and disabled as they are switched between. Descriptions for UIHandler attributes and methods are listed below:

* currentUI – A reference to the currently displayed UI. Used to disable the current UI when switching off of it.
* ManageExpensesUI – A reference to the ManageExpensesUI class.
* NewExpenseUI – A reference to the NewExpenseUI class.
* EditExpenseUI – A reference to the EditExpenseUI class.
* switchUI() – This method first disables the current interface, then replaces the current interface with the interface passed it, and then enables that interface.
* drawUsername() – Fetches the username from AWSHandler and draws it to the page header.

**UserInterface**

This interface defines the methods shared by the three user interface functions below which are used by UIHandler to manage switching between interfaces. As mentioned previously, Javascript does not have built-in support for interface objects, thus the code of the application itself is not expected to directly reflect the structure of this diagram, but to simply implement the interface methods to all UI functions. Descriptions for UserInterface attributes and methods are listed below:

* initializeUI() – Initializes the user interface for the first time, doing things like setting references to DOM elements and setting default values.
* getContainerNode() – Returns a reference to the node containing this interface.
* setEnabled() – Sets enabled to true or false and handles any actions needed when switching between the two states to prevent conflict between different interfaces.
* unlockUI() – Unlocks the UI by enabling all user input elements.
* lockUI() – Locks the UI by disabling all user input elements.
* resetUI() – Resets all applicable input elements back to their default state.

**ManageExpenseUI**

This is the interface object used for drawing and managing the Manage Expenses interface.

* enabled – Whether this interface is currently enabled or not. When this is false, all class methods will immediately return false when called.
* All Element types – All Element types listed in the UML are references to the inputs used by this interface.
* submitFilterQuery() – Filters expense data via the LocalDataHandler class based on the values of the input elements of this interface.
* getMapsURL() – Converts a latitude and longitude value into a Google Maps URL for those coordinates.
* getImageURL() – Converts an imageID to a URL used to download the relevant image from S3.
* drawRecords() – Draw record data onto the screen.
* deleteModal() – Prime and display the deletion confirmation modal.
* deleteExpenseButton() – Delete a record. Attached to a deletion button.

NewExpenseUI

This is the interface object used for drawing and managing the New Expense interface.

* enabled – Whether this interface is currently enabled or not. When this is false, all class methods will immediately return false when called.
* All Element types – All Element types listed in the UML are references to the inputs used by this interface.
* expenseID – The ID of the expense currently being edited.
* submitEditExpense() – Starts the submission process for an edited expense by validating the data input, then sending that data to AWSHandler for further processing.

EditExpenseUI

This is the interface object used for drawing and managing the New Expense interface.

* enabled – Whether this interface is currently enabled or not. When this is false, all class methods will immediately return false when called.
* All Element types – All Element types listed in the UML are references to the inputs used by this interface.
* submitNewExpense() – Starts the submission process for a new expense by validating the data input, then sending that data to AWSHandler for further processing.

## Test plan and Use Cases

The following use cases are split into two general categories denoted by their IDs. Use cases with “UI” in their ID are primarily concerned with user interface processing and generally operate at a higher level. Use cases with “M” in their ID deal with lower-level functionality, such as the behavior of specific class methods or their attributes. The use cases are given one of the following four priorities based on their importance and side effects should they fail.

**Priority 1** – This use case is critical to the basic functions of the system. Failure will likely cause significant data loss or system failure.

**Priority 2** – This use case serves an important function, but other features may still be functional should it fail. Failure may not be safe.

**Priority 3** – Some major system functions may be reduced or unavailable if this use case fails, but failure is generally safe.

**Priority 4** – The failure of this use case has limited consequences and cannot cause system failure or significant data loss.

The Frequency of Use fields are populated with some common values listed below. Fields may have values not listed below which describe a more specific frequency. The estimates are skewed towards a realistic higher end, with the assumption that users will make input mistakes and spend time navigating back and forth between interfaces.

**Daily** – Used roughly once or twice per day per user.

**Frequently** – Used a few times per hour per user.

**Constantly** – Used several times a minute while the application is in use.

**Rarely** – Only used in specific circumstances, typically with several weeks of time between each use.

|  |  |
| --- | --- |
| **ID:** | **UI-1** |
| **Title:** | Log In |
| **Description:** | The user accesses the log in interface and logs in with their Cognito username and password by entering them in their respective fields and submitting the form. |
| **Primary Actor:** | User |
| **Preconditions:** | There is no user currently logged in and the user has initiated loading on the website. |
| **Postconditions:** | The user is logged in and on the Manage Expenses page. |
| **Main  Success Scenario:** | 1. The web page redirects to the login screen for AWS Cognito. 2. The user enters their email in the email field. 3. The user enters their password in the password field. 4. The user clicks the submit button. 5. The page redirects to the Manage Expenses page with authentication data in URL. |
| **Extensions:** | 1a. Redirection is blocked or otherwise fails.  1a1. The page will display minimal content.  1a2. The user will not be able to log in or otherwise access the system.  4a. User submits form with incorrect username/password.  4a1. The form will display an error.  4a2. The user will not be logged in.  4a3. The user will be able to correct their inputs and try again. |
| **Frequency of Use:** | Daily |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 1 |

|  |  |
| --- | --- |
| **ID:** | **UI-2** |
| **Title:** | Recover Password |
| **Description:** | The user initiates the password recovery process from the log in screen and uses a code sent to their email to authenticate themselves and set a new password. |
| **Primary Actor:** | User |
| **Preconditions:** | There is no user currently logged in and the log in page is currently displayed. |
| **Postconditions:** | The user has logged in with a new password and is on the Manage Expenses page. |
| **Main  Success Scenario:** | 1. The user enters their email in the email field. 2. The user clicks the Next button. 3. The interface displays a prompt for a password. 4. The user clicks the “Forgot your password?” link. 5. The interface displays another prompt for the username. 6. The user enters their email in the email field. 7. The user clicks the “Reset my password” button. 8. The interface displays the “Reset password” prompt. 9. The user receives an email with a reset code at their registered email address. 10. The user enters the code from the email in the Code field. 11. The user enters a new password in the “New password” field. 12. The user enters the same new password in the “Confirm new password” field. 13. The user clicks Change password. 14. AWS Cognito registers the new password. |
| **Extensions:** | 6a. User submits a password change request with an incorrect email.  6a1. No email will be sent.  6a2. The user will be told no such email exists.  10a1. Interface will display an error as the user types.  10a2. The user can correct their password in the field.  11a. New password does not meet requirements.  11a1. Interface will display an error as the user types.  11a2. The user can correct their password in the field.  12a. Confirm password does not match New password.  12a1. Interface will display an error as the user types.  12a2. The user can correct their password in the field.  13a. The code entered is not correct.  11a1. Interface will display an error.  11a2. The user can go back to the log in screen and reinitiate the process. |
| **Frequency of Use:** | Rarely |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 2 |

|  |  |
| --- | --- |
| **ID:** | **UI-3** |
| **Title:** | User Clicks Text Field |
| **Description:** | The user clicks/touches any of the text fields displayed on the New Expense, Edit Expense, or Manage Expenses interfaces. |
| **Primary Actor:** | User |
| **Preconditions:** | The user is on one of the following interfaces:   * New Expense * Edit Expense * Manage Expenses |
| **Postconditions:** | A cursor is placed in the field and the user can now enter text input. |
| **Main  Success Scenario:** | 1. The user clicks on one of the following text fields:  * New Expense – Amount * New Expense – Comment * Edit Expense – Amount * Edit Expense – Comment * Manage Expenses – Search Query * Manage Expenses – Amount Query (2 fields)  1. The text input cursor is placed inside the field. 2. The user may not input text into the field. |
| **Extensions:** | 2a. The user is on a mobile device.  2a1. A virtual keyboard is displayed. |
| **Frequency of Use:** | Constantly |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 1 |

|  |  |
| --- | --- |
| **ID:** | **UI-4** |
| **Title:** | User Clicks Log Out |
| **Description:** | The user clicks/touches the Log Out button on any of the pages in which it is displayed. |
| **Primary Actor:** | User |
| **Preconditions:** | The user is on one of the following interfaces:   * New Expense * Edit Expense * Manage Expenses   The user is logged in. |
| **Postconditions:** | The user is logged out and the log in interface is displayed. |
| **Main  Success Scenario:** | 1. The user clicks on the log out button. 2. The page redirects to the AWS Cognito log in interface. 3. The user can choose to log out or log back in. |
| **Extensions:** | N/A |
| **Frequency of Use:** | Daily |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 2 |

|  |  |
| --- | --- |
| **ID:** | **UI-5** |
| **Title:** | Invalid Amount |
| **Description:** | The user enters an invalid amount into the Amount field on either the New Expense or Edit Expense interface. |
| **Primary Actor:** | User |
| **Preconditions:** | The user is on one of the following forms:   * New Expense * Edit Expense   The user has not input an amount into the form. |
| **Postconditions:** | An error is displayed informing the user that their input is invalid. |
| **Main  Success Scenario:** | 1. The user inputs one of the following into an Amount field:  * Symbols other than a single period. * Letters * Negative numbers  1. The interface will disallow input or display an error directly underneath the Amount field when the user clicks submit. 2. The form will not submit. |
| **Extensions:** | N/A |
| **Frequency of Use:** | Frequently |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 2 |

|  |  |
| --- | --- |
| **ID:** | **UI-6** |
| **Title:** | Invalid Comment |
| **Description:** | The user enters an invalid comment into the Comment field on either the New Expense or Edit Expense interface. |
| **Primary Actor:** | User |
| **Preconditions:** | The user is on one of the following forms:   * New Expense * Edit Expense   The user has not input a comment into the form. |
| **Postconditions:** | A comment of 255 characters will be retained in the comment field, but further input will be disallowed. |
| **Main  Success Scenario:** | 1. The user attempts to input more than 255 characters into the Comment field. 2. The field will disallow further input. |
| **Extensions:** | N/A |
| **Frequency of Use:** | Rarely |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 2 |

|  |  |
| --- | --- |
| **ID:** | **UI-7** |
| **Title:** | Show Type Dropdown |
| **Description:** | The user clicks on the Type button and a dropdown is displayed. |
| **Primary Actor:** | User |
| **Preconditions:** | The user is on one of the following forms:   * New Expense * Edit Expense * Manage Expenses |
| **Postconditions:** | A dropdown selector is displayed showing the available Type options. |
| **Main  Success Scenario:** | 1. The user clicks on the type button. 2. A type dropdown is displayed showing all available Type options. |
| **Extensions:** | N/A |
| **Frequency of Use:** | Constantly |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 1 |

|  |  |
| --- | --- |
| **ID:** | **UI-8** |
| **Title:** | Select Type |
| **Description:** | The user clicks on the Type button and a dropdown is displayed. |
| **Primary Actor:** | User |
| **Preconditions:** | The user is on one of the following forms:   * New Expense * Edit Expense * Manage Expenses   The Type dropdown generated in UI-NEWEDIT-3 is displayed. |
| **Postconditions:** | The type dropdown is hidden and a new type is displayed in the Type button. The type is also set as a string to the value element of the HTML for the Type dropdown button. |
| **Main  Success Scenario:** | 1. The user clicks on a type from the displayed list. 2. The selected type is set to the text of the Type dropdown button. 3. The selected type is set to the value field of the Type dropdown button. 4. The type dropdown is hidden. |
| **Extensions:** | 1a. The user clicks an area outside of the Type dropdown.  1a1. The type dropdown is hidden.  1a2. No new type is displayed or set internally. |
| **Frequency of Use:** | Constantly |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 1 |

|  |  |
| --- | --- |
| **ID:** | **UI-9** |
| **Title:** | Submit Filled New/Edit Form |
| **Description:** | The user clicks on the submit button of the New Expense or Edit Expense form with all required fields filled. |
| **Primary Actor:** | User |
| **Preconditions:** | The user is on one of the following forms:   * New Expense * Edit Expense   No errors are currently displayed.  The user has filled the following required inputs:   * Amount * Type |
| **Postconditions:** | The expense data is reflected in the S3 backend. The user is redirected to the Manage Expenses page. |
| **Main  Success Scenario:** | 1. The user clicks on the Submit button. 2. All input elements are disabled. 3. Data begins uploading to the S3 backend via the S3 API. 4. Data is successfully uploaded. 5. The user is redirected to the Manage Expenses page. |
| **Extensions:** | 3a. No image is attached.  3a1. The data is uploaded without an image.  3b. The comment field is empty.  3b1. The data is uploaded without a comment.  4a. There is an error uploading the data.  4a1. All input elements are enabled.  4a2. The user can attempt to upload again by clicking Submit. |
| **Frequency of Use:** | Daily |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 1 |

|  |  |
| --- | --- |
| **ID:** | **UI-10** |
| **Title:** | Submit Unfilled New/Edit Form |
| **Description:** | The user clicks on the submit button of the New Expense or Edit Expense form with one or more required fields not filled. |
| **Primary Actor:** | User |
| **Preconditions:** | The user is on one of the following forms:   * New Expense * Edit Expense   The user has not filled the following required inputs:   * Amount * Type   The user clears one of the following required inputs:   * Amount * Type |
| **Postconditions:** | The Submit button remains disabled. |
| **Main  Success Scenario:** | 1. The user clicks the submit button. 2. The fields with required inputs display an error. 3. No data is processed. 4. The form is not submitted. |
| **Extensions:** | N/A |
| **Frequency of Use:** | Rarely |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 2 |

|  |  |
| --- | --- |
| **ID:** | **UI-11** |
| **Title:** | Attach Image |
| **Description:** | The user clicks the Browse button on the New Expense or Manage Expense form and their device displays it’s native file picker. The user selects an image and it is loaded into the image element of the form. |
| **Primary Actor:** | User |
| **Preconditions:** | The user is on the New Expense form. |
| **Postconditions:** | An image is loaded into the image element of the form. |
| **Main  Success Scenario:** | 1. The user clicks the Browse button. 2. The user navigates to the image they want to upload. 3. The user selects that image. 4. The image is loaded into the image element of the input form. |
| **Extensions:** | 3a. The image is greater than 3MB in size.  3a1. The image is loaded into the image element.  3a2. The user clicks submit.  3a3. The field displays an error that the image is too large.  3a4. The user can attempt to load a different image by clicking Browse again. |
| **Frequency of Use:** | Daily |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 1 |

|  |  |
| --- | --- |
| **ID:** | **UI-12** |
| **Title:** | New Expense Navigation |
| **Description:** | The user clicks the New Expense navigation button at the top of the page and is redirected to the New Expense interface when the current interface is not already New Expense. |
| **Primary Actor:** | User |
| **Preconditions:** | The user is logged in. |
| **Postconditions:** | The New Expense page is displayed. |
| **Main  Success Scenario:** | 1. The user clicks the New Expense button. 2. The current interface is disabled. 3. The current interface is hidden. 4. The New Expense interface is enabled. 5. The New Expense interface is displayed. |
| **Extensions:** | 1a. The page is already New Expense  1a1. The page is not changed.  1a2. No data is changed or displayed. |
| **Frequency of Use:** | Frequently |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 1 |

|  |  |
| --- | --- |
| **ID:** | **UI-13** |
| **Title:** | Manage Expense Navigation |
| **Description:** | The user clicks the Manage Expenses navigation button at the top of the interface and is redirected to the Manage Expenses interface when the current interface is not already Manage Expenses. |
| **Primary Actor:** | User |
| **Preconditions:** | The user is logged in. |
| **Postconditions:** | The Manage Expenses page is displayed. |
| **Main  Success Scenario:** | 1. The user clicks the Manage Expenses button. 2. The current interface is disabled. 3. The current interface is hidden. 4. The Manage Expenses interface is enabled. 5. The Manage Expenses interface is displayed. |
| **Extensions:** | 1a. The page is already Manage Expenses  1a1. The page is not changed.  1a2. Expense data is reloaded and displayed. |
| **Frequency of Use:** | Frequently |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 1 |

|  |  |
| --- | --- |
| **ID:** | **UI-14** |
| **Title:** | Edit Expense Button |
| **Description:** | The user clicks the Edit button next to an expense record in the Manage Expenses screen. |
| **Primary Actor:** | User |
| **Preconditions:** | The Manage Expenses screen is displayed. There is at least one record displayed. |
| **Postconditions:** | The Edit Expense screen is displayed with it’s fields pre-filled with the relevant data for that record. |
| **Main  Success Scenario:** | 1. The user clicks the Edit button next to a record. 2. The current interface is disabled. 3. The current interface is hidden. 4. The Edit Expense interface is displayed. 5. The Edit Expense interface is enabled. 6. All displayed fields are pre-filled with data for that record. 7. The Submit button is enabled. |
| **Extensions:** | N/A |
| **Frequency of Use:** | Daily |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 3 |

|  |  |
| --- | --- |
| **ID:** | **UI-15** |
| **Title:** | Edit Expense Navigation |
| **Description:** | The user clicks the Manage Expenses or New Expense navigation button at the top of the page while on the edit page and is redirected to the respective interface. |
| **Primary Actor:** | User |
| **Preconditions:** | The user is logged in. |
| **Postconditions:** | The Edit Expense page is displayed. |
| **Main  Success Scenario:** | 1. The user clicks the Manage Expenses or New Expense button. 2. The current interface is disabled. 3. The current interface is hidden. 4. The selected interface is enabled. 5. The selected interface is displayed. |
| **Extensions:** | N/A |
| **Frequency of Use:** | Daily |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 3 |

|  |  |
| --- | --- |
| **ID:** | **UI-16** |
| **Title:** | Show Manage Expenses Dropdowns |
| **Description:** | The user clicks on one of many dropdown buttons on the Manage Expenses page. The dropdown displays it’s respective interface. |
| **Primary Actor:** | User |
| **Preconditions:** | The user is on the Manage Expenses page. |
| **Postconditions:** | The interface for the dropdown that was clicked on is displayed. |
| **Main  Success Scenario:** | 1. The user clicks on one of the following dropdowns.  * Time * Type * Amount * Sort  1. The interface for that dropdown is displayed. |
| **Extensions:** | N/A |
| **Frequency of Use:** | Frequently |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 3 |

|  |  |
| --- | --- |
| **ID:** | **UI-17** |
| **Title:** | Hide Manage Expenses Dropdown |
| **Description:** | The user clicks away from a dropdown currently displayed on the manage expenses screen, causing it to disappear. |
| **Primary Actor:** | User |
| **Preconditions:** | The user is on the Manage Expenses page with a dropdown currently displayed. |
| **Postconditions:** | The previously displayed dropdown is now hidden. |
| **Main  Success Scenario:** | 1. The user clicks anywhere outside of the area of the dropdown. 2. The dropdown is hidden. |
| **Extensions:** | N/A |
| **Frequency of Use:** | Frequently |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 3 |

|  |  |
| --- | --- |
| **ID:** | **UI-18** |
| **Title:** | Search Query |
| **Description:** | The user initiates a search based on the Search Query field and filters. The displayed expense record data is filtered and displayed based on the query and filters. |
| **Primary Actor:** | User |
| **Preconditions:** | The user is on the Manage Expenses page. |
| **Postconditions:** | The filtered query results are displayed. |
| **Main  Success Scenario:** | 1. The user inputs a search query into the Search Query field. 2. The user clicks the Search button. 3. The expense record data is filtered based on the search query input. 4. The expense record data is filtered based on any set filters. 5. The filtered expense record data is displayed. |
| **Extensions:** | 1a. The Search Query field is empty.  1a1. The search executes as normal but skips filtering based on Search Query.  2a. Any filter option is unset.  2a1. The search executes as normal but skips filtering based on that filter. See use cases for the Manage Expenses filters for specific behavior on each filter element.  2a. No records match the query/filters.  2a1. No data is displayed. |
| **Frequency of Use:** | Daily |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 3 |

|  |  |
| --- | --- |
| **ID:** | **UI-19** |
| **Title:** | Time Filter |
| **Description:** | The user inputs a time into the time filter dropdown interface and initiates a search. The data is filtered and displayed based on the input. |
| **Primary Actor:** | User |
| **Preconditions:** | The user is on the Manage Expenses page. The time filter dropdown is displayed. |
| **Postconditions:** | The filtered query results are displayed. |
| **Main  Success Scenario:** | 1. The user inputs a time range into the time inputs. 2. The user clicks the Search button. 3. The time dropdown is hidden. 4. The expense data is filtered based on any present search query and any set filters. 5. The expense data is displayed. |
| **Extensions:** | See Search Query use case for a general search case.  1a. Either time field is empty.  1a1. The search executes as normal but with no bounds for the empty field. |
| **Frequency of Use:** | Frequently |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 3 |

|  |  |
| --- | --- |
| **ID:** | **UI-20** |
| **Title:** | Amount Filter |
| **Description:** | The user inputs an amount into the amount filter dropdown interface and initiates a search. The data is filtered and displayed based on the input. |
| **Primary Actor:** | User |
| **Preconditions:** | The user is on the Manage Expenses page. The amount filter dropdown is displayed. |
| **Postconditions:** | The filtered query results are displayed. |
| **Main  Success Scenario:** | 1. The user inputs an amount range into the amount inputs. 2. The user clicks the Search button. 3. The time dropdown is hidden. 4. The expense data is filtered based on any present search query and any set filters. 5. The expense data is displayed. |
| **Extensions:** | See Search Query use case for a general search case.  1a. Either amount field is empty.  1a1. The search executes as normal but with no bounds for the empty field.  1b. The user inputs an invalid amount into either field and submits.  1a1. An error is displayed immediately under the field.  1a2. The form does not submit. |
| **Frequency of Use:** | Frequently |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 3 |

|  |  |
| --- | --- |
| **ID:** | **UI-21** |
| **Title:** | Sort Filter |
| **Description:** | The selects a sort option in the sort dropdown and clicks the search button. A search executes based on any set filters and the search query. |
| **Primary Actor:** | User |
| **Preconditions:** | The user is on the Manage Expenses page. The sort filter dropdown is displayed. |
| **Postconditions:** | The filtered query results are displayed in the set sort order. |
| **Main  Success Scenario:** | 1. The user selects a sort order in the dropdown. 2. The dropdown is hidden. 3. The filtered expense data is displayed in the set sort order. |
| **Extensions:** | See Search Query use case for a general search case. |
| **Frequency of Use:** | Frequently |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 4 |

|  |  |
| --- | --- |
| **ID:** | **UI-22** |
| **Title:** | Location Link |
| **Description:** | The user clicks the Location link next to an expense record and is redirected to a Google Maps page based on the geolocation data contained in the record. |
| **Primary Actor:** | User |
| **Preconditions:** | The user is on the Manage Expenses page. |
| **Postconditions:** | The user is redirected to a Google Maps page with the expense location. |
| **Main  Success Scenario:** | 1. The user clicks the Location button next to a record. 2. The user’s browser loads a Google Maps page with the expense location. |
| **Extensions:** | 1a. There is no location data for a record.  1a1. The Location link is not displayed.  1a2. The user can not access location data for this record.  2a. The user opens the link in a new tab by any means.  1a1. The Google Maps page is displayed as normal in a new tab.  1a2. The Manage Expenses page is unaffected. |
| **Frequency of Use:** | Daily |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 4 |

|  |  |
| --- | --- |
| **ID:** | **UI-23** |
| **Title:** | Image Link |
| **Description:** | The user clicks the Image link next to an expense record and is redirected to a page showing the image in their browser. |
| **Primary Actor:** | User |
| **Preconditions:** | The user is on the Manage Expenses page. |
| **Postconditions:** | The image for the respective expense record is displayed. |
| **Main  Success Scenario:** | 1. The user clicks the Image button next to a record. 2. The user’s browser loads the associated image for that record from AWS S3. |
| **Extensions:** | 1a. There is no image data for a record.  1a1. The Image link is not displayed.  1a2. The user can not access image data for this record.  2a. The browser is configured to download links or the user initiates a download by any means.  1a1. The image is not displayed in the browser and is instead downloaded.  2b. The user opens the link in a new tab by any means.  1b1. The image is displayed as normal in a new tab.  1b2. The Manage Expenses page is unaffected. |
| **Frequency of Use:** | Daily |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 4 |

|  |  |
| --- | --- |
| **ID:** | **UI-24** |
| **Title:** | Delete Record |
| **Description:** | The user clicks the delete button next to a record, confirms their choice, and deletes the record. |
| **Primary Actor:** | User |
| **Preconditions:** | The user is on the Manage Expenses page. |
| **Postconditions:** | The respective record is deleted. |
| **Main  Success Scenario:** | 1. The user clicks the delete button next to a record. 2. A prompt is displayed asking the user to confirm their choice with Delete and Cancel buttons. 3. The user clicks the Delete button. 4. The confirmation prompt is hidden. 5. The record is removed from the expense data and the updated data is uploaded to S3. 6. The new data is displayed to the user. |
| **Extensions:** | 3a. The user clicks the Cancel button.  1a1. The confirmation prompt is hidden.  1a2. No other actions are initiated and the record remains intact.  5a. The upload fails or is rejected by S3.  1a1. The expense data is not updated and the record is not deleted. |
| **Frequency of Use:** | Rarely |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 2 |

|  |  |
| --- | --- |
| **ID:** | **UI-25** |
| **Title:** | Download CSV |
| **Description:** | The user clicks either of the CSV download buttons. The displayed expense record data is converted to CSV format and a browser download is initiated. |
| **Primary Actor:** | User |
| **Preconditions:** | The user is on the Manage Expenses page. |
| **Postconditions:** | A CSV in the correct format based on the selected button is downloaded. The formats for each button are detailed earlier in this document. |
| **Main  Success Scenario:** | 1. The user clicks one of the following buttons.  * Full CSV * 3-Column CSV  1. The currently displayed expense data is converted into CSV format. 2. The CSV data is packed into a file. 3. A browser download is initiated for the file. 4. The file is downloaded. |
| **Extensions:** | 2a. No expense data is displayed.  1a1. A CSV file with only headings is downloaded. |
| **Frequency of Use:** | Daily |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 2 |

|  |  |
| --- | --- |
| **ID:** | **UI-26** |
| **Title:** | Missing/Corrupted Fields in Record Data |
| **Description:** | A record is displayed in the Manage Expenses page with no data in an associated field, either due to intentional design (no comment) or data corruption. |
| **Primary Actor:** | User |
| **Preconditions:** | The user is on the Manage Expenses page, which is about to begin displaying record data, either due to initial loading or a new search query. The data of one or more records has missing/corrupted fields. |
| **Postconditions:** | All available data is displayed. The missing/corrupted fields are left blank. |
| **Main  Success Scenario:** | 1. The interface begins displaying record data. 2. The interface detects that a field is missing/corrupted. 3. The interface does not attempt to display the data. 4. The interface loads a blank string into the display value for that field. 5. The interface displays all other available data as normal. |
| **Extensions:** | N/A |
| **Frequency of Use:** | Rarely |
| **Status:** | Main success scenario passed, however extremely corrupted data may still cause display errors and crashes. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 3 |

|  |  |
| --- | --- |
| **ID:** | **M-1** |
| **Title:** | AWSHandler initialize() |
| **Description:** | The initialize method is called on the AWSHandler class. The class initializes the AWS and loggedInUsername attributes. |
| **Primary Actor:** | AWSHandler |
| **Preconditions:** | The AWSHandler initialize() method has not been called. |
| **Postconditions:** | AWSHandler has initialized AWS and loggedInUsername. |
| **Main  Success Scenario:** | 1. The AWSHandler initialize() method is called by the main client script. 2. AWSHandler initializes AWS with a new instance of the AWS.S3 class. 3. AWSHandler reads the AWS Cognito username from the API into loggedInUsername. |
| **Extensions:** | 2a. The AWS.S3 instance fails to instantiate.  1a1. Execution is aborted.  2a2. The application cannot be used and will display a mostly empty screen.  3a. The username cannot be read.  3a1. The application assumes the user is logged out.  3a2. The application throws an error which triggers a redirect to the login screen. |
| **Frequency of Use:** | Frequently |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 1 |

|  |  |
| --- | --- |
| **ID:** | **M-2** |
| **Title:** | AWSHandler uploadNewExpense() |
| **Description:** | The uploadNewExpense() method is called on the AWSHandler class. Expense data is passed along with an optional image. The data is validated and uploaded to AWS S3. |
| **Primary Actor:** | AWSHandler |
| **Preconditions:** | The AWSHandler uploading variable is false. |
| **Postconditions:** | The new expense data has been uploaded to S3. |
| **Main  Success Scenario:** | 1. The AWSHandler uploadNewExpense() method is called by the submitNewExpense() method. 2. uploading is set to true. 3. The passed ExpenseData object is set to dataToUpload. 4. The passed image data is set to imageToUpload. 5. putExpenseData() is called. 6. putExpenseImage() is called. 7. The data is uploaded to AWS S3. 8. uploadNewExpense() returns true. |
| **Extensions:** | 2a. The ExpenseData object fails validation.  2a1. uploading is set to false.  2a2. The method immediately returns false.  3a. There is no image file passed.  3a1. imageToUpload is set to null. |
| **Frequency of Use:** | Daily |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 1 |

|  |  |
| --- | --- |
| **ID:** | **M-3** |
| **Title:** | AWSHandler uploadEditedExpense() |
| **Description:** | The uploadEditedExpense() method is called on the AWSHandler class. Edited expense data is passed along with the matching expense ID. The data is validated and uploaded to AWS S3. |
| **Primary Actor:** | AWSHandler |
| **Preconditions:** | The AWSHandler uploading variable is false. |
| **Postconditions:** | The edited expense data has been uploaded to S3. |
| **Main  Success Scenario:** | 1. The AWSHandler uploadEditedExpense() method is called by the submitEditedExpense() method. 2. uploading is set to true. 3. The passed ExpenseData object is set to dataToUpload. 4. putExpenseData() is called. 5. The data is uploaded to AWS S3. 6. uploading is set to false. 7. uploadEditedExpense() returns true. |
| **Extensions:** | 2a. The ExpenseData object fails validation.  2a1. uploading is set to false.  2a2. The method immediately returns false. |
| **Frequency of Use:** | Daily |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 1 |

|  |  |
| --- | --- |
| **ID:** | **M-4** |
| **Title:** | AWSHandler deleteExpense() |
| **Description:** | The deleteExpense() method is called on the AWSHandler class. Fresh expense data is downloaded from S3 and the relevant record ID is removed from it. The data is uploaded again. |
| **Primary Actor:** | AWSHandler |
| **Preconditions:** | The AWSHandler uploading variable is false. |
| **Postconditions:** | The expense data has been uploaded to S3 with the relevant record removed. |
| **Main  Success Scenario:** | 1. The AWSHandler deleteExpense() method is called by the event handler for the Delete button in the Manage Expenses interface. 2. uploading is set to true. 3. The record is removed from the latest expense data via getExpenseData() 4. putExpenseData() is called. 5. The data is uploaded to AWS S3. 6. uploading is set to false. 7. deleteExpense() returns true. |
| **Extensions:** | N/A |
| **Frequency of Use:** | Rarely |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 1 |

|  |  |
| --- | --- |
| **ID:** | **M-5** |
| **Title:** | AWSHandler putExpenseImage() |
| **Description:** | The putExpenseImage() method is called on the AWSHandler class. The image loaded into imageToUpload is uploaded to AWS S3 with a filename based on the current Unix epoch timestamp. |
| **Primary Actor:** | AWSHandler |
| **Preconditions:** | The AWSHandler uploading variable is false. |
| **Postconditions:** | The expense image has been uploaded to S3 and uploading is set to false. |
| **Main  Success Scenario:** | 1. The AWSHandler putExpenseImage() method is called. 2. The imageToUpload attribute is checked to make sure it’s not null. 3. The image in imageToUpload is uploaded to S3 via the AWS attribute of the AWSHandler object. 4. putExpenseImage() returns true. |
| **Extensions:** | 2a. The image is null.  2a1. An upload is not initiated.  2a2. The method immediately returns false.  3a. The upload to AWS fails.  3a1. The method immediately returns false. |
| **Frequency of Use:** | Daily |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 1 |

|  |  |
| --- | --- |
| **ID:** | **M-6** |
| **Title:** | AWSHandler putExpenseData() |
| **Description:** | The putExpenseData() method is called on the AWSHandler class. The data loaded into dataToUpload is uploaded to AWS S3 and replaces the existing data. |
| **Primary Actor:** | AWSHandler |
| **Preconditions:** | The AWSHandler uploading variable is false. |
| **Postconditions:** | The expense data has been uploaded to S3 and uploading is set to false. |
| **Main  Success Scenario:** | 1. The AWSHandler putExpenseData() method is called. 2. The data in dataToUpload is uploaded to S3 via the AWS attribute of the AWSHandler object. 3. putExpenseData() returns true. |
| **Extensions:** | 3a. The data is null.  3a1. An upload is not initiated.  3a2. The method immediately returns false.  3b. The upload to AWS fails.  3b1. The method immediately returns false. |
| **Frequency of Use:** | Daily |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 1 |

|  |  |
| --- | --- |
| **ID:** | **M-7** |
| **Title:** | AWSHandler getExpenseData() |
| **Description:** | The getExpenseData() method is called on the AWSHandler class. The latest expense data is downloaded from S3 via the AWS attribute and returned. |
| **Primary Actor:** | AWSHandler |
| **Preconditions:** | The AWSHandler uploading variable is false. |
| **Postconditions:** | The latest expense data has been returned by getExpenseData(). |
| **Main  Success Scenario:** | 1. The AWSHandler getExpenseData() method is called. 2. The data is downloaded from S3 via the AWS attribute of the AWSHandler object. 3. The JSON data is converted to a Javascript object file and loaded into an ExpenseData object. 4. The new ExpenseData object is returned. |
| **Extensions:** | 3a. The download from AWS fails.  3a1. The method immediately returns false. |
| **Frequency of Use:** | Constantly |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 1 |

|  |  |
| --- | --- |
| **ID:** | **M-8** |
| **Title:** | LocalDataHandler loadData() |
| **Description:** | The loadData() method is called, which downloads the latest expense data from S3 via getExpenseData() and loads it into fullData and filteredData. |
| **Primary Actor:** | LocalDataHandler |
| **Preconditions:** | None. |
| **Postconditions:** | The latest expense data is in fullData. |
| **Main  Success Scenario:** | 1. The loadData() method is called. 2. loadData calls getExpenseData(). 3. getExpenseData returns a new ExpenseData object. 4. loadData sets that object to fullData. 5. filteredData is set to fullData using Object.entries() 6. loadData returns true. |
| **Extensions:** | 3a. getExpenseData returns false.  3a1. The method immediately returns false. |
| **Frequency of Use:** | Constantly |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 1 |

|  |  |
| --- | --- |
| **ID:** | **M-9** |
| **Title:** | LocalDataHandler filterTime() |
| **Description:** | The filterTime() method is called, which removes any entries in filteredData which do not match it’s time range. |
| **Primary Actor:** | LocalDataHandler |
| **Preconditions:** | filterQuery() has been called. |
| **Postconditions:** | The data in filteredData is now filtered by the time criteria passed to filterTime(). |
| **Main  Success Scenario:** | 1. The filterTime() method is called by submitFilterQuery(). 2. filterTime() loops through all entries in filteredData. 3. Any entries lower than minTime or higher than maxTime are removed. |
| **Extensions:** | 3a. minTime or maxTime are null.  3a1. Entries are not filtered by minimum times or maximum times respectively. |
| **Frequency of Use:** | Daily |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 3 |

|  |  |
| --- | --- |
| **ID:** | **M-10** |
| **Title:** | LocalDataHandler filterType() |
| **Description:** | The filterType () method is called, which removes any entries in filteredData which do not match it’s type. |
| **Primary Actor:** | LocalDataHandler |
| **Preconditions:** | filterQuery() has been called. |
| **Postconditions:** | The data in filteredData is now filtered by the type criteria passed to filterType (). |
| **Main  Success Scenario:** | 1. The filterType () method is called by submitFilterQuery(). 2. filterType () loops through all entries in filteredData. 3. Any entries which do not match the type passed to filterType() are removed. |
| **Extensions:** | N/A |
| **Frequency of Use:** | Daily |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 3 |

|  |  |
| --- | --- |
| **ID:** | **M-11** |
| **Title:** | LocalDataHandler filterAmount() |
| **Description:** | The filterAmount () method is called, which removes any entries in filteredData which do not match it’s amount range. |
| **Primary Actor:** | LocalDataHandler |
| **Preconditions:** | filterQuery() has been called. |
| **Postconditions:** | The data in filteredData is now filtered by the amount criteria passed to filterAmount(). |
| **Main  Success Scenario:** | 1. The filterAmount() method is called by submitFilterQuery(). 2. filterAmount() loops through all entries in filteredData. 3. Any entries lower than minAmount or higher than maxAmount are removed. |
| **Extensions:** | 3a. minAmount or maxAmount are null.  3a1. Entries are not filtered by minimum amount or maximum amount respectively. |
| **Frequency of Use:** | Daily |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 3 |

|  |  |
| --- | --- |
| **ID:** | **M-12** |
| **Title:** | LocalDataHandler filterQuery() |
| **Description:** | The filterQuery() method is called, which loads fresh data then filters it based on all search and filter criteria, loading the results into filteredData(). |
| **Primary Actor:** | LocalDataHandler |
| **Preconditions:** | None |
| **Postconditions:** | The data in filteredData is now updated and filtered by all currently set filters. |
| **Main  Success Scenario:** | 1. The filterQuery() method is called by submitFilterQuery(). 2. filterQuery() loops through filtered data. 3. Any entries not matching it’s query in either their name, email, or comment fields are removed. |
| **Extensions:** | N/A |
| **Frequency of Use:** | Daily |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 3 |

|  |  |
| --- | --- |
| **ID:** | **M-13** |
| **Title:** | LocalDataHandler exportFullCSV() and exportThreeColumnCSV() |
| **Description:** | Either the exportFullCSV() or exportThreeColumnCSV() method is called. The data in filteredData is converted to the proper CSV format and packed into a file, which is sent to the browser for download. |
| **Primary Actor:** | LocalDataHandler |
| **Preconditions:** | None |
| **Postconditions:** | A download with the desired CSV of the data in filteredData has initiated. |
| **Main  Success Scenario:** | 1. Either the exportFullCSV() or exportThreeColumnCSV() method is called. 2. The method called loops through the data in filtered data. 3. Every record is converted into a string in CSV format. 4. The strings are appended together with newlines separating them. 5. The string is converted into a file using the File() constructor. 6. A download for the file is initiated in the browser. |
| **Extensions:** | 2a. filteredData is empty.  2a1. filterQuery() immediately returns false. |
| **Frequency of Use:** | Daily |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 3 |

|  |  |
| --- | --- |
| **ID:** | **M-14** |
| **Title:** | UIHandler switchUI() |
| **Description:** | The switchUI() method is called which deactivates the currently displayed interface and hides it, and activates the new interface and displays it. |
| **Primary Actor:** | UIHandler |
| **Preconditions:** | None. |
| **Postconditions:** | The new display will be activated and visible. |
| **Main  Success Scenario:** | 1. switchUI() is called. 2. setEnabled(false), lockUI(), and resetUI() are called for currentUI. 3. currentUI is set to targetUI. 4. setEnabled(true), unlockUI(), and resetUI() are called for currentUI. |
| **Extensions:** | 1a. targetUI is null.  1a1. The method returns immediately.  2a. currentUI is null.  2a1. Nothing is called.  2a2. Execution continues to step 3 as normal. |
| **Frequency of Use:** | Constantly |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 1 |

|  |  |
| --- | --- |
| **ID:** | **M-15** |
| **Title:** | UIHandler drawUsername() |
| **Description:** | The drawUsername() method is called which accesses the username HTML element and sets it’s value to the username of the currently logged in user. |
| **Primary Actor:** | UIHandler |
| **Preconditions:** | None. |
| **Postconditions:** | The username will be visible on the top right side of the screen. |
| **Main  Success Scenario:** | 1. drawUsername() is called. 2. The username HTML element is loaded. 3. The text value of that element is set to the value of the AWSHandler.loggedInUsername string. |
| **Extensions:** | 1a. username HTML element cannot be loaded.  1a1. An error is thrown and displayed in the console.  1a2. The username element remains unchanged. |
| **Frequency of Use:** | Daily |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 4 |

|  |  |
| --- | --- |
| **ID:** | **M-16** |
| **Title:** | UserInterface initializeUI() |
| **Description:** | The initializeUI() method is called which primarily loads all relevant HTML elements into the UI object. The UI objects which contain this method are:   * ManageExpenseUI * NewExpenseUI * EditExpenseUI |
| **Primary Actor:** | A user interface object. |
| **Preconditions:** | None. |
| **Postconditions:** | All associated HTML elements will be loaded into class attributes for easy access. |
| **Main  Success Scenario:** | 1. initializeUI() is called. 2. enabled is set to false. 3. All associated HTML elements are loaded into class attributes via getElementByID(). |
| **Extensions:** | 3a. An HTML element cannot be loaded.  3a1. An error is thrown into the console.  3a2. Execution continues as normal. |
| **Frequency of Use:** | Daily |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 1 |

|  |  |
| --- | --- |
| **ID:** | **M-17** |
| **Title:** | UserInterface getContainerNode() |
| **Description:** | The getContainerNode() method is called which returns the containerNode element. The UI objects which contain this method are:   * ManageExpenseUI * NewExpenseUI * EditExpenseUI |
| **Primary Actor:** | A user interface object. |
| **Preconditions:** | None. |
| **Postconditions:** | The containerNode element is returned. |
| **Main  Success Scenario:** | 1. getContainerNode() is called. 2. The containerNode element is returned. |
| **Extensions:** | None. |
| **Frequency of Use:** | Constantly |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 1 |

|  |  |
| --- | --- |
| **ID:** | **M-18** |
| **Title:** | UserInterface setEnabled() |
| **Description:** | The setEnabled() method is called which sets the enabled attribute of it’s respective UI object to the argument passed into it. The UI objects which contain this method are:   * ManageExpenseUI * NewExpenseUI * EditExpenseUI |
| **Primary Actor:** | A user interface object. |
| **Preconditions:** | None. |
| **Postconditions:** | The enabled attribute of the UI element is set to the argument passed into setEnabled(). |
| **Main  Success Scenario:** | 1. setEnabled() is called. 2. enabled is set to the value passed into setEnabled. |
| **Extensions:** | 2a. The value passed is not a boolean.  2a1. The value is interpreted as a boolean.  2a2. Enabled is set to the interpreted value. |
| **Frequency of Use:** | Constantly |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 1 |

|  |  |
| --- | --- |
| **ID:** | **M-19** |
| **Title:** | UserInterface unlockUI() |
| **Description:** | The unlockUI() method is called which enables all HTML elements of it’s respective UI. The UI objects which contain this method are:   * ManageExpenseUI * NewExpenseUI * EditExpenseUI |
| **Primary Actor:** | A user interface object. |
| **Preconditions:** | None. |
| **Postconditions:** | The associated HTML elements of the object are enabled for user input. |
| **Main  Success Scenario:** | 1. unlockUI() is called. 2. All HTML elements stored in this object’s attributes are set to be enabled for user input. |
| **Extensions:** | N/A |
| **Frequency of Use:** | Constantly |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 1 |

|  |  |
| --- | --- |
| **ID:** | **M-20** |
| **Title:** | UserInterface lockUI() |
| **Description:** | The lockUI() method is called which disables all HTML elements of it’s respective UI. The UI objects which contain this method are:   * ManageExpenseUI * NewExpenseUI * EditExpenseUI |
| **Primary Actor:** | A user interface object. |
| **Preconditions:** | None. |
| **Postconditions:** | The associated HTML elements of the object are disabled for user input. |
| **Main  Success Scenario:** | 1. lockUI() is called. 2. All HTML elements stored in this object’s attributes are set to be disabled for user input. |
| **Extensions:** | N/A |
| **Frequency of Use:** | Constantly |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 1 |

|  |  |
| --- | --- |
| **ID:** | **M-21** |
| **Title:** | UserInterface resetUI() |
| **Description:** | The resetUI() method is called which clears the values of all user inputs. The UI objects which contain this method are:   * ManageExpenseUI * NewExpenseUI * EditExpenseUI |
| **Primary Actor:** | A user interface object. |
| **Preconditions:** | None. |
| **Postconditions:** | The values of associated HTML elements of the object are cleared, except for buttons and containerNode. |
| **Main  Success Scenario:** | 1. resetUI() is called. 2. All HTML elements stored in the class have their value cleared, except for buttons and containerNode. |
| **Extensions:** | N/A |
| **Frequency of Use:** | Constantly |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 3 |

|  |  |
| --- | --- |
| **ID:** | **M-22** |
| **Title:** | ManageExpenseUI submitFilterQuery() |
| **Description:** | This method filters displayed data by calling the following methods of LocalDataHandler:   * filterQuery() * filterTime() * filterType() * filterAmount() * filterSort() |
| **Primary Actor:** | ManageExpenseUI |
| **Preconditions:** | ManageExpenseUI has been initialized. |
| **Postconditions:** | LocalDataHandler.filteredData contains only elements which passed the filters. |
| **Main  Success Scenario:** | 1. submitFilterQuery() is called. 2. When their relevant fields are not a default value the following methods are called:  * filterQuery() * filterTime() * filterType() * filterAmount()  1. Data is sorted by calling filterSort() |
| **Extensions:** | 2a. One of the relevant fields is a default value.  2a1. The respective filter method is not called.  2a2. Execution continues as normal. |
| **Frequency of Use:** | Daily |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 4 |

|  |  |
| --- | --- |
| **ID:** | **M-23** |
| **Title:** | ManageExpenseUI getMapsURL() |
| **Description:** | This method takes a latitude and longitude as an argument and returns a Google Maps URL as a string for the specified coordinates. |
| **Primary Actor:** | ManageExpenseUI |
| **Preconditions:** | None. |
| **Postconditions:** | A Google Maps URL for the appropriate location is returned. |
| **Main  Success Scenario:** | 1. getMapsURL() is called. 2. The latitude and longitude integers are converted to strings. 3. The latitude and longitude are given a Google Maps URL prefix. 4. The string is returned. |
| **Extensions:** | 1a. Either latitude or longitude values are null.  1a1. A generic google link is returned. |
| **Frequency of Use:** | Constantly |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 4 |

|  |  |
| --- | --- |
| **ID:** | **M-24** |
| **Title:** | ManageExpenseUI getImageURL() |
| **Description:** | This method takes an imageID and generates an S3 link to access the image directly. |
| **Primary Actor:** | ManageExpenseUI |
| **Preconditions:** | None. |
| **Postconditions:** | An AWS S3 URL for the appropriate image is returned. |
| **Main  Success Scenario:** | 1. getImageURL() is called. 2. The imageID is appended to an S3 bucket prefix. 3. The modified string is returned. |
| **Extensions:** | 1a. The imageID is null.  1a1. A link with the S3 bucket prefix is returned. |
| **Frequency of Use:** | Constantly |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 3 |

|  |  |
| --- | --- |
| **ID:** | **M-25** |
| **Title:** | NewExpenseUI submitNewExpense() |
| **Description:** | This method collects expense data input into the HTML elements contained within NewExpenseUI and sends them added to all expense data to AWSHandler.uploadNewExpense(). |
| **Primary Actor:** | NewExpenseUI |
| **Preconditions:** | NewExpenseUI has been initialized. |
| **Postconditions:** | AWSHandler.uploadNewExpense() has been called with the new expense data. |
| **Main  Success Scenario:** | 1. submitNewExpense() is called. 2. lockUI() is called. 3. The values of commentField, amountField, and typeField are packed into a new ExpenseObject. 4. The time and location reported by the browser are added to the new ExpenseObject. 5. The name of the user and email loaded in AWSHandler are added to the new ExpenseObject. 6. An imageID based on the current time and the file extension of the file loaded into imageField are added to the new ExpenseObject. 7. The new ExpenseObject is passed to AWSHandler.uploadNewExpense() along with the value in imageField. 8. uploadNewExpense() returns true. 9. UIHandler.switchUI(UIHandler.ManageExpensesUI) is called |
| **Extensions:** | 3a. amountField or typeField are empty.  3a1. unlockUI() is called.  3a2. Execution ends.  4a. The time or location are not available.  4a1. The fields in ExpenseObject are left empty.  6a. imageField is empty.  6a1. imageID is set to null.  8a. uploadNewExpense() returns false.  8a1. switchUI() is not called  8a2. unlockUI() is called. |
| **Frequency of Use:** | Daily |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 1 |

|  |  |
| --- | --- |
| **ID:** | **M-26** |
| **Title:** | EditExpenseUI submitEditExpense() |
| **Description:** | This method collects expense data input into the HTML elements contained within EditExpenseUI and replaces the appropriate data in the expense data and calls AWSHandler.uploadNewExpense(). |
| **Primary Actor:** | EditExpenseUI |
| **Preconditions:** | EditExpenseUI has been initialized. |
| **Postconditions:** | AWSHandler.uploadNewExpense() has been called with the new expense data. |
| **Main  Success Scenario:** | 1. submitEditExpense() is called. 2. lockUI() is called. 3. The ExpenseObject with the expenseID key is loaded. 4. The amount, type, and comment are replaced with the values in amountField, typeField, and commentField. 5. The new ExpenseObject is passed to AWSHandler.uploadEditedExpense() along with the expenseID 6. uploadEditedExpense() returns true. 7. switchUI(UIHandler.ManageExpensesUI) is called. |
| **Extensions:** | 3a. amountField or typeField are empty.  3a1. unlockUI() is called.  3a2. Execution ends.  3a. An ExpenseObject matching expenseID cannot be found.  3a1. A new ExpenseObject is created.  3a2. Execution ends.  7a. uploadNewExpense() returns false.  7a1. switchUI() is not called.  7a2. unlockUI() is called. |
| **Frequency of Use:** | Daily |
| **Status:** | Main success scenario and extensions passed. |
| **Owner:** | Victor Guzman |
| **Priority:** | Priority 1 |

## Deployment plan

**Overview**

Deployment for this application is relatively straightforward, but involves extensive navigation of the S3, IAM, and Cognito AWS consoles. Application clients, user pools, identity pools, IAM roles, and more must be configured to allow users to authenticate themselves with the application and successfully access data from S3. The client application is hosted on Cloudflare via their Cloudflare Pages service, and this client application redirects back and forth with the login page provided by AWS Cognito to provide authentication.

An optional step is included for creating an IAM role that can be applied to any Cognito user to allow them to manage users. This should be done in cases where user management privileges need to be given to a user without granting them access to more powerful tools that may compromise the application.

A verification and troubleshooting section is included at the end, which outlines the basic steps that should be taken to ensure the application was configured correctly.

Below is a step by step guide to full deployment detailing the process from start to finish. Deployment of the live and testing environments is identical, and two instances of this application can be deployed simultaneously so long as all components (user pools, app clients, S3 buckets, etc) are replicated.

**Download Required Files**

Github link here

**Create CloudFlare Pages Project**

1. Access the CloudFlare Pages site at <https://pages.cloudflare.com/>
2. Create an account as necessary
3. Click on the Pages tab
4. Under “Use direct upload” click the “Get started” button
5. Select a suitable project name, such as “expense-app”
6. Save the deployment URL you are provided (i.e. expense-app-22c.pages.dev)
7. The project is not yet deployed as no assets have been uploaded. We will return to CloudFlare in a later step.

**Configure Cognito App and User Pool**

1. Create or log into the AWS account for this application.
2. Access the Cognito console by using the search bar or other navigation.
3. Create a user pool by clicking the Create user pool button in Amazon Cognito > User pools
4. Select “Single-page application”
5. Give the application an appropriate name, such as “expense-app”
6. Check “Email” under “sign-in identifiers”
7. Select “name” as a required attribute
8. Enter the CloudFlare deployment URL from the previous section into the Return URL field
9. Click “Create user directory”
10. Navigate to the new user pool under Amazon Cognito > User pools
11. Click the app client under the Applications > App clients section
12. Under the “Login pages” tab click Edit on the “Managed login pages configuration” section
13. Add “Implicit grant” and remove “Authorization code grant” from “OAuth 2.0 grant types”
14. Add “Profile” and “aws.cognito.signin.user.admin” to “OpenID Connect scopes”
15. Click “Save changes”
16. Navigate to Amazon Cognito > User pools, and click on the newly created user pool
17. Click “Sign-up” under the Authentication section
18. Click Edit on the “Self-service sign-up” section
19. Uncheck “Enable self-registration”
20. Click “Save changes”

**Create Identity Pool and IAM Role**

1. Under Amazon Cognito > Identity pools click “Create identity pool”
2. Select the “Authenticated access” checkbox
3. Select the “Amazon Cognito user pool” checkbox
4. Click Next
5. Select “Create a new IAM role”
6. Enter an appropriate name such as “expense-iam-role”
7. Click Next
8. In the “User pool ID” drop down, select the user pool created in the previous section
9. In the “App client ID” drop down, select the app client for this application
10. Click Next
11. Enter an appropriate identity pool name, such as “expense-id-pool”
12. Click Next
13. Click “Create identity pool

**Configure IAM Role**

1. Access the IAM console via the AWS console search or other navigation.
2. Under IAM > Roles click the role created in the previous section
3. Click “Create inline policy” under the “Add permissions” dropdown
4. Click the JSON tab
5. Delete the contents of the policy editor
6. Paste the contents of the “S3Access Policy” box below this section into the policy editor
7. Click Next
8. Give the policy an appropriate name, such as “S3Access”
9. Click “Create policy”

|  |
| --- |
| **S3Access Policy** |
| {  "Version": "2012-10-17",  "Statement": [  {  "Effect": "Allow",  "Action": [  "s3:DeleteObject",  "s3:GetObject",  "s3:ListBucket",  "s3:PutObject",  "s3:PutObjectAcl"  ],  "Resource": [  "\*"  ]  }  ]  } |

**Create a User and Group**

1. Access the application user pool via Amazon Cognito > User pools
2. Under User management > Groups click “Create group”
3. Enter a group name such as “expense-users”
4. Select the IAM role created in the previous section in the “IAM role” dropdown
5. Click “Create group”
6. Under User management > Users click “Create user”
7. Input a valid email for the user.
8. Send an email invitation and temporary password as necessary.
9. Click “Create user”
10. Select the newly created user in the “Users” section
11. Click “Add user to a group”
12. Select the group created in steps 2-5
13. Click Add

**Configure AWS S3**

1. Access the S3 Console via the AWS Console search or other navigation
2. Click “Create bucket”
3. Choose an appropriate bucket name, such as “expense-app-bucket”
4. Uncheck “Block all public access”
5. Click “Create bucket”
6. Click the newly created bucket
7. Under the permissions tab click “Edit” on the “Bucket policy” section
8. Paste the contents of the “S3 Bucket Policy” box at the end of this section into the policy editor
9. Change the “BUCKET-NAME” text to the exact name of the bucket
10. Click “Save changes”
11. Scroll down to the CORS settings and click Edit
12. Paste the contents of the “CORS Configuration” box at the end of this section into the editor
13. Click “Save changes”
14. Click on the “Objects” tab
15. Upload the file “Expenses.json” to the bucket from the “s3” folder in the downloaded project directory
16. Back on the Objects tab click “Create folder”
17. Name the folder “img” and click “Create folder”
18. Navigate inside the img folder in the S3 bucket by clicking on it
19. Click “Upload” and then upload the image contained in the “img” folder of the downloaded project directory here

|  |
| --- |
| **S3 Bucket Policy** |
| {  "Version": "2012-10-17",  "Statement": [  {  "Sid": "AddPerm",  "Effect": "Allow",  "Principal": "\*",  "Action": "s3:GetObject",  "Resource": "arn:aws:s3:::vg-deployed-expense-bucket/img/\*"  }  ]  } |

|  |
| --- |
| **CORS Configuration** |
| [  {  "AllowedHeaders": [  "\*"  ],  "AllowedMethods": [  "HEAD",  "GET",  "PUT",  "POST",  "DELETE"  ],  "AllowedOrigins": [  "\*"  ],  "ExposeHeaders": [  "ETag"  ]  }  ] |

**Configure params.js**

1. Open the “params.js” file in the “https” folder of the project directory in a text editor
2. Put the bucket name inside the quotes on the “bucketName” line
3. Put the bucket region found on the Properties tab of the bucket inside the quotes on the “bucketRegion” line
4. Put the identity pool ID found in “Amazon Cognito > Identity pools” inside the quotes on the “identityPoolID” line
5. Click on the identity pool and click on the “User access” tab
6. Put the string in the “Identity Provider String” box at the end of this section inside the quotes on the “idProvider” line
7. Replace the text “PROVIDER” with the identity provider listed in the “Identity providers” section (i.e. us-west-2\_XXXXXXXXX)
8. Replace the text “REGION” with the section of the identity provider that corresponds to a region (i.e. us-west-2)
9. Go to Amazon Cognito > User pools and click on the user pool created earlier
10. Click on “App clients” and select the app client created earlier
11. Copy the URL of the “View login page” button by right clicking and using the “Copy Link” command, or clicking the button and copying the URL from the URL bar
12. Put the URL inside the quotes on the “cognitoLoginURL” line
13. Save and close the file

|  |
| --- |
| **Identity Provider String** |
| cognito-idp.REGION.amazonaws.com/PROVIDER |

**Deploy Site on CloudFlare**

1. Access the CloudFlare interface from before, log in if necessary
2. On the left navigation click “Compute (Workers) then click the page created earlier
3. Click “Upload assets”
4. In a file browser, navigate to the downloaded project directory and open the “https” file
5. Zip the contents of the “https” folder into a file called “expensesite.zip” (Note: do not add the “https” folder itself to the zip file, just the contents)
6. On CloudFlare, click “select from computer” and “Upload zip”
7. Select the file for upload in the file picker UI and confirm
8. Click “Deploy site”

**Verify Deployment**

1. Go to the CloudFlare deployment URL for the project
2. You should briefly see the project page load, then be redirected to a Cognito sign in screen
3. Sign in using the credentials for the user created in a previous section
4. Change the password and name when prompted
5. You should now be redirected to the Manage Expenses page
6. You should see an expense with the comment “Hello from your AWS S3 bucket!”
7. Click the image link on that expense
8. You should be given a stylized image of a receipt
9. Click on the New Expense tab and attempt to upload an expense
10. The submission should succeed, and the new expense should show up in the Manage Expenses tab
11. Delete that expense, it should disappear
12. Click “Log Out” on the top right corner, you should be redirected to the Cognito login screen
13. If all of these functions work, then the app has been deployed successfully

If the application fails to work correctly, double check all configuration files and ensure that all information is accurate and free of typos. Generally speaking, most issues are going to be caused by incorrect deployment. The developer console may give some idea as to what the issue is, as it can signal the specific step in the communication process with the AWS API that is causing the problem.

**[Optional] Create an IAM Role to Manage User Accounts**

1. Navigate to the IAM console and go to IAM > Roles
2. Click “Create role”
3. Select Web Identity and choose “Amazon Cognito” as the identity provider
4. Enter the Identity Pool ID from the Identity Pool created earlier
5. Do not add any permissions, click Next
6. Enter a role name such as “IAMUserManager”
7. Click Create Role
8. Click on the newly created role
9. In the “Add permissions” dropdown click “Create inline policy”
10. Click the JSON tab
11. Paste the contents of the “Cognito User Policy” box at the end of this section into the policy editor
12. Replace the “REGION” and “USERPOOLID” text with the appropriate values from the user pool created in this section
13. Replace the “ACCOUNTID” text with the AWS account ID of the root user (typically found by clicking the account name on the top right)
14. Click Next
15. Give the policy an appropriate name, such as “CognitoUserManager”
16. Click “Create policy”
17. Go to Amazon Cognito > User pools and select the user pool created previously
18. Click Groups then Create group
19. Give the group a name such as “UserManager”
20. Under IAM role add the role that was just created
21. Click Create group
22. This group can now be added to users to enable them to manage other users via the AWS Cognito console with minimal permissions to ensure system stability and security

|  |
| --- |
| **Cognito User Policy** |
| {  "Version": "2012-10-17",  "Statement": [  {  "Sid": "FullUserControlInSpecificUserPool",  "Effect": "Allow",  "Action": [  "cognito-idp:AdminCreateUser",  "cognito-idp:AdminDeleteUser",  "cognito-idp:AdminDisableUser",  "cognito-idp:AdminEnableUser",  "cognito-idp:AdminGetUser",  "cognito-idp:AdminUpdateUserAttributes",  "cognito-idp:AdminResetUserPassword",  "cognito-idp:AdminSetUserPassword",  "cognito-idp:ListUsers",  "cognito-idp:ListUsersInGroup",  "cognito-idp:AdminAddUserToGroup",  "cognito-idp:AdminRemoveUserFromGroup",  "cognito-idp:AdminListGroupsForUser"  ],  "Resource": "arn:aws:cognito-idp:REGION:ACCOUNTID:userpool/USERPOOLID"  }  ]  } |

## References used

Amazon. (2023, May 15). *Configure and confirm Cors in Amazon S3*. Amazon Web Services, Inc. <https://repost.aws/knowledge-center/s3-configure-cors>

Amazon. (n.d.-a). *Amazon S3: Allows Amazon Cognito users to access objects in their bucket - AWS Identity and Access Management*. AWS Docs. <https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_examples_s3_cognito-bucket.html>

Amazon. (n.d.-b). *Amazon S3 pricing - cloud object storage - AWS*. AWS S3. <https://aws.amazon.com/s3/pricing/>

Amazon. (n.d.-c). *Authentication service customer iam (CIAM) – amazon cognito pricing – AWS*. AWS Cognito. <https://aws.amazon.com/cognito/pricing/>

Amazon. (n.d.-d). *AWS SDK for JavaScript*. AWS Docs. <https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/>

Amazon. (n.d.-e). *Uploading photos to Amazon S3 from a browser - AWS SDK for JavaScript*. AWS Docs. <https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/s3-example-photo-album.html>

Apple. (n.d.-a). *Buy iphone 15 and iphone 15 plus*. <https://www.apple.com/shop/buy-iphone/iphone-15>

Apple. (n.d.-b). *Buy MacBook Air*. <https://www.apple.com/shop/buy-mac/macbook-air>

Bundrick, H. M. (2025, March 14). *Best apps to track expenses*. NerdWallet. <https://www.nerdwallet.com/p/best/finance/expense-tracker-apps>

Cloudflare. (n.d.). *Workers & Pages Pricing*. <https://www.cloudflare.com/plans/developer-platform/>

Samsung. (n.d.). *Buy galaxy A15 5G (unlocked) | samsung us*. Samsung US. <https://www.samsung.com/us/smartphones/galaxy-a15/buy/galaxy-a15-5g-128gb-unlocked-sm-a156ulbdxaa/>

StatCounter. (n.d.). *Browser market share worldwide*. StatCounter Global Stats. <https://gs.statcounter.com/browser-market-share>

Thornton, J., & Otto, M. (n.d.). *Get started with bootstrap*. Bootstrap v5.3. <https://getbootstrap.com/docs/5.3/getting-started/introduction/>