SALN Digitalization App

Lucas Miguel V. Agcaoili, Aeron Dann P. Peñaflorida, Miguel A. Guiang

LAMig

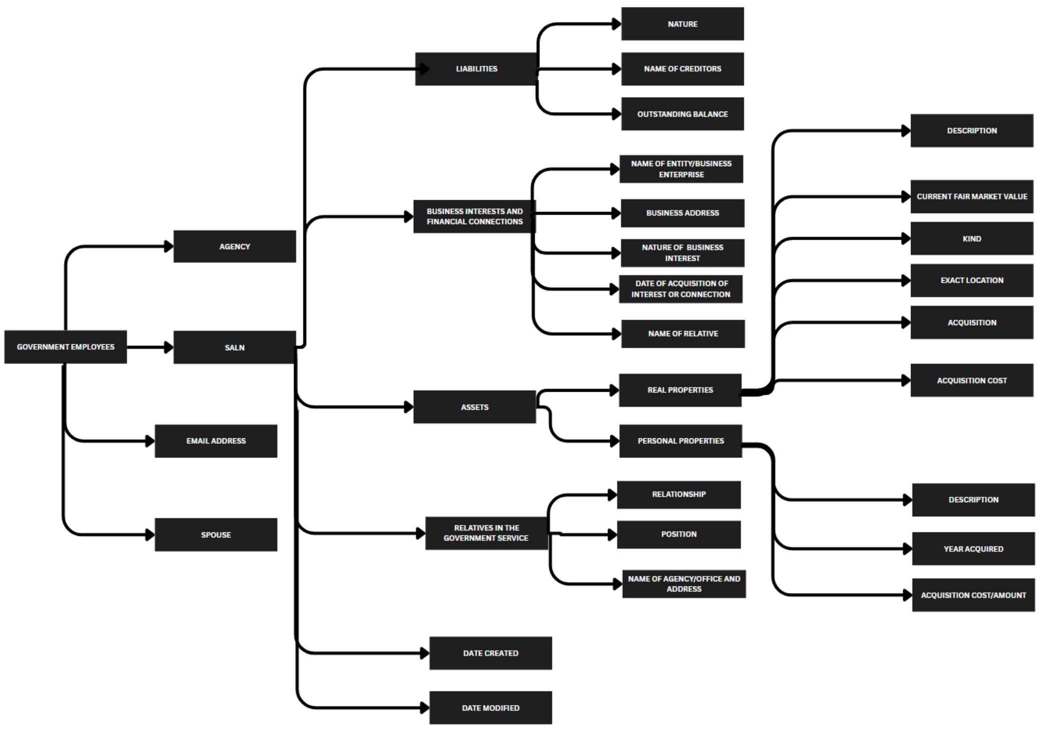
Client: Associate Prof. Rommel Feria

Group Blog: https://shirokamiqq.github.io/cs191-LAMig-GroupBlog/

Course Details: CS 191 WFR1 2526A - Associate Prof. Ligaya Leah Figueroa

WEB APP REQUIREMENTS MODEL

Content Model



The core feature of the app is the SALN form, which government employees use to disclose their financial connections, assets (both real and personal), and corresponding liabilities. To maintain data security, metadata such as creation and modification dates are also recorded, ensuring that SALN entries are automatically deleted after five days.

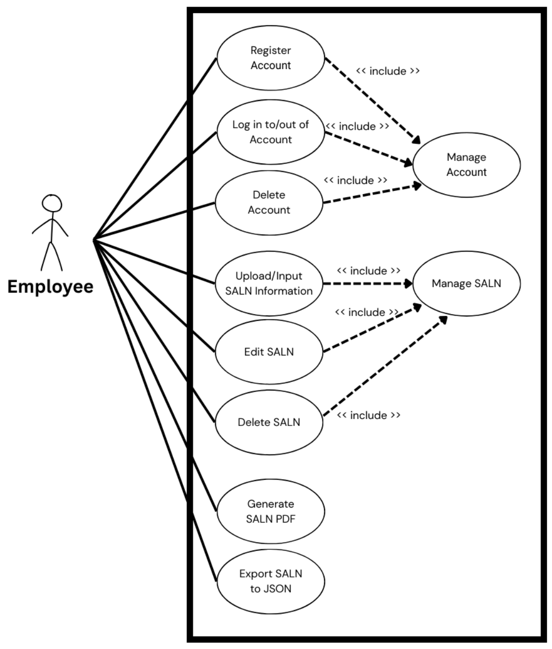
Interaction Model

Actors and Goals

The initiating actors of the system are the employees who need to fill up their SALN form. Their goal is to accomplish their SALN form, save it to their account, and export it to a printable format via PDF or a machine-readable format via JSON.

The participating actors of the system are the browser and DigitalOcean, and administrators. The browser's role is to support offline mode in the app, and to store user data via built-in APIs localStorage, cacheStorage, and IndexedDB. The role of DigitalOcean is to act as the cloud server to host the web server and the remote database. Lastly, the role of the admins is to maintain the site quality, fix any bugs, and implement updates to the SALN should the format change.

Use Cases



The main user of our app will be an employee who needs to fill in their SALN.

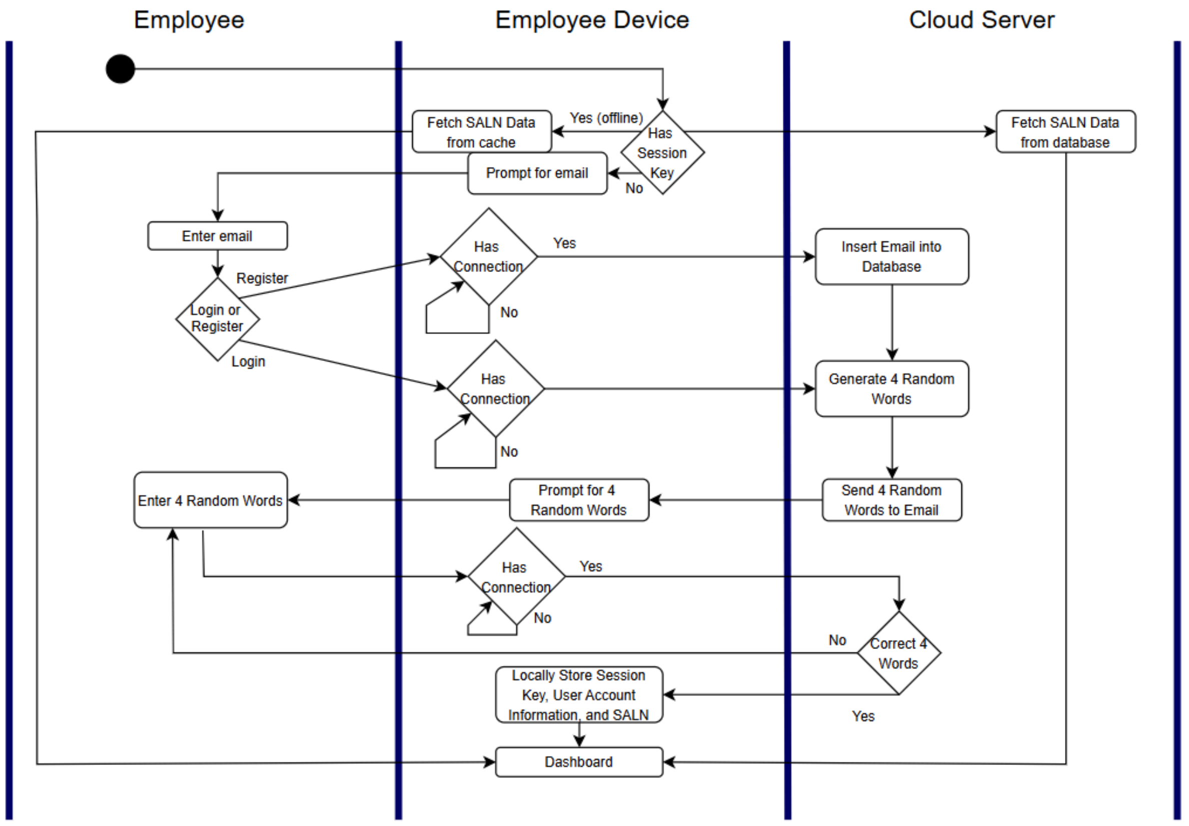
The use cases can be divided into account management, SALN management, and SALN exportation.

Use cases related to account management include registration, logging in to, and logging out of your account. These processes will be secured using a One-Time Password two-factor authentication sent via email.

Use cases related to SALN management include generating a new SALN form through either filling it up in the app or uploading a JSON. Once it's finished, the user may either edit the form or delete it if they so desire.

Lastly, the app comes with a feature to export the completed SALN to either PDF format or JSON format.

System Sequence Diagrams

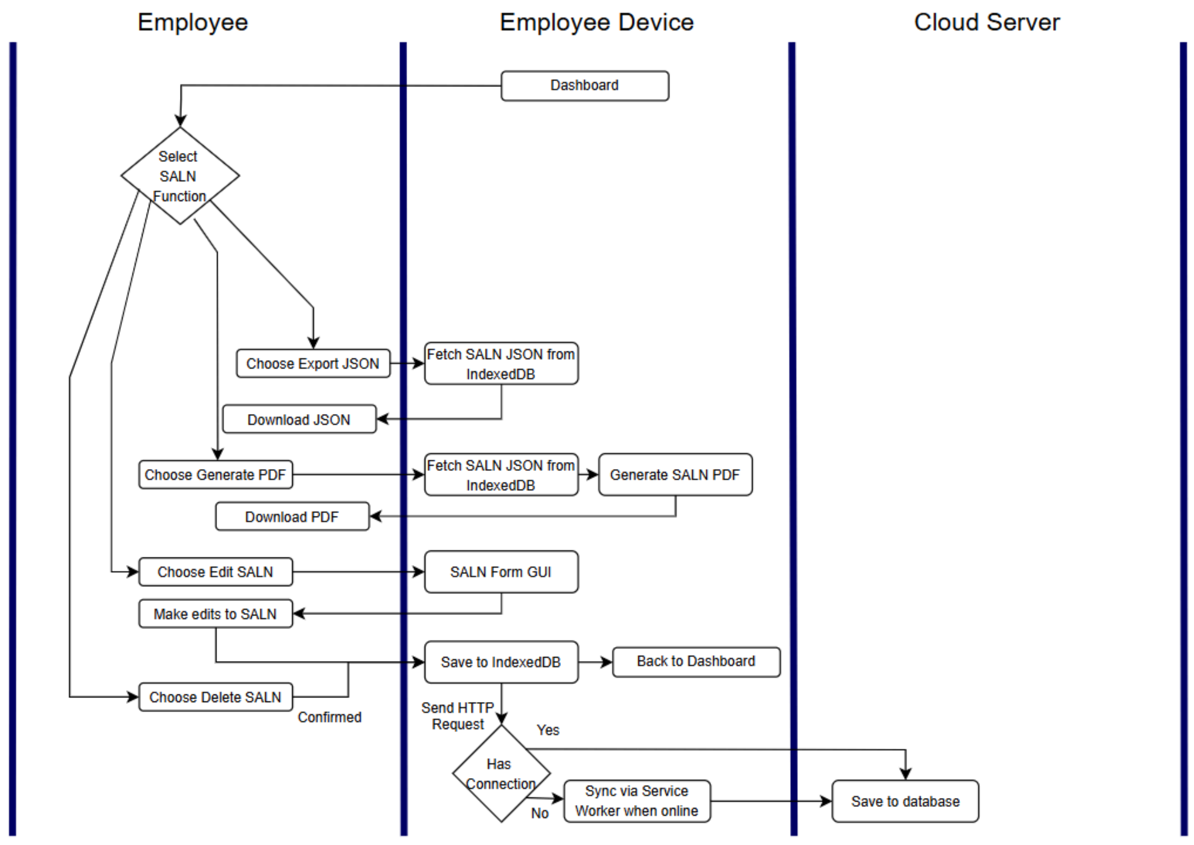


The first part of the swimlane diagram of the whole system is the login. The SALN app first checks if the user has a session key stored in IndexedDB. If so, he will immediately be logged in and be sent to the dashboard. If he was automatically logged in offline, he will use the latest SALN data that was cached. If he was automatically logged in online, he will use SALN data from the remote database; This would also allow him to sync his SALN data with the latest SALN data from the remote database.

If the user has no session key in his browser's IndexedDB, that means that it may have expired due to long inactivity (a week), or this is the user's first time on the app. The app will prompt the user for an email, and the user will select if they plan to register that email or just log in. Upon pressing either button, a registration/login HTTP request would be sent to the web server. But, before the web server would receive that request, the app would have its service worker intercept that request and check if there is a connection. If not, the service worker would make sure that the request won't fail and would resume it once the user is back online.

Once the web server has received the email, if the user's email is not yet registered in the database, then the web server will insert it into the database, then it will generate four random words for authentication. If the user's email was already registered (user chose to log in), then the web server would go straight to generating the four random words. The web server would then send those words to the email and the app will prompt the user for those four words. Once the user enters four words, the web server will cross check those words.

Should those words end up verified, the browser will then store data like session key, user account, and current SALN information locally via built-in browser APIs. Otherwise, the prompt for the four words would repeat. Then, the user will finally be sent to the dashboard.

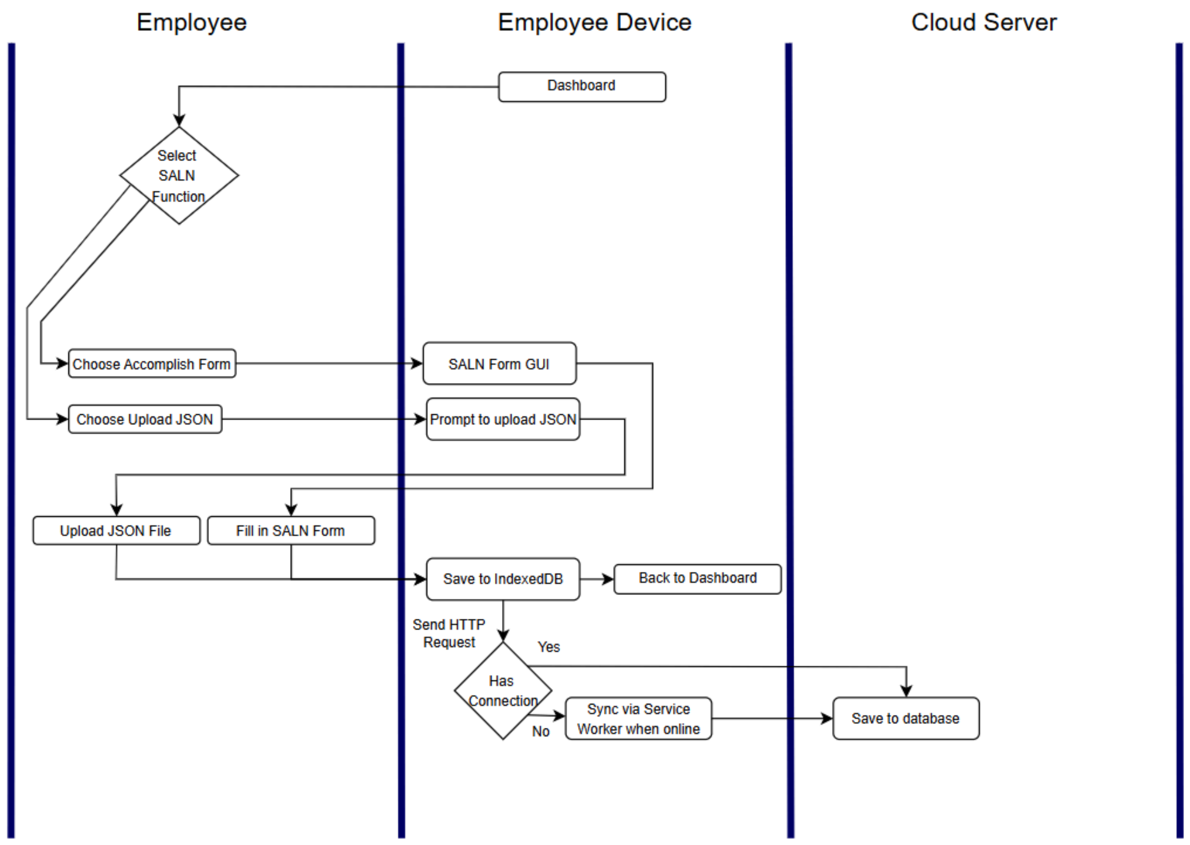


The second swimlane diagram shows use cases in which we are exporting or modifying SALN data from the app. For every SALN form that the user has, he has the option to download the SALN data in JSON format, to generate a PDF version of the SALN form, to edit that SALN form, and to delete that SALN form.

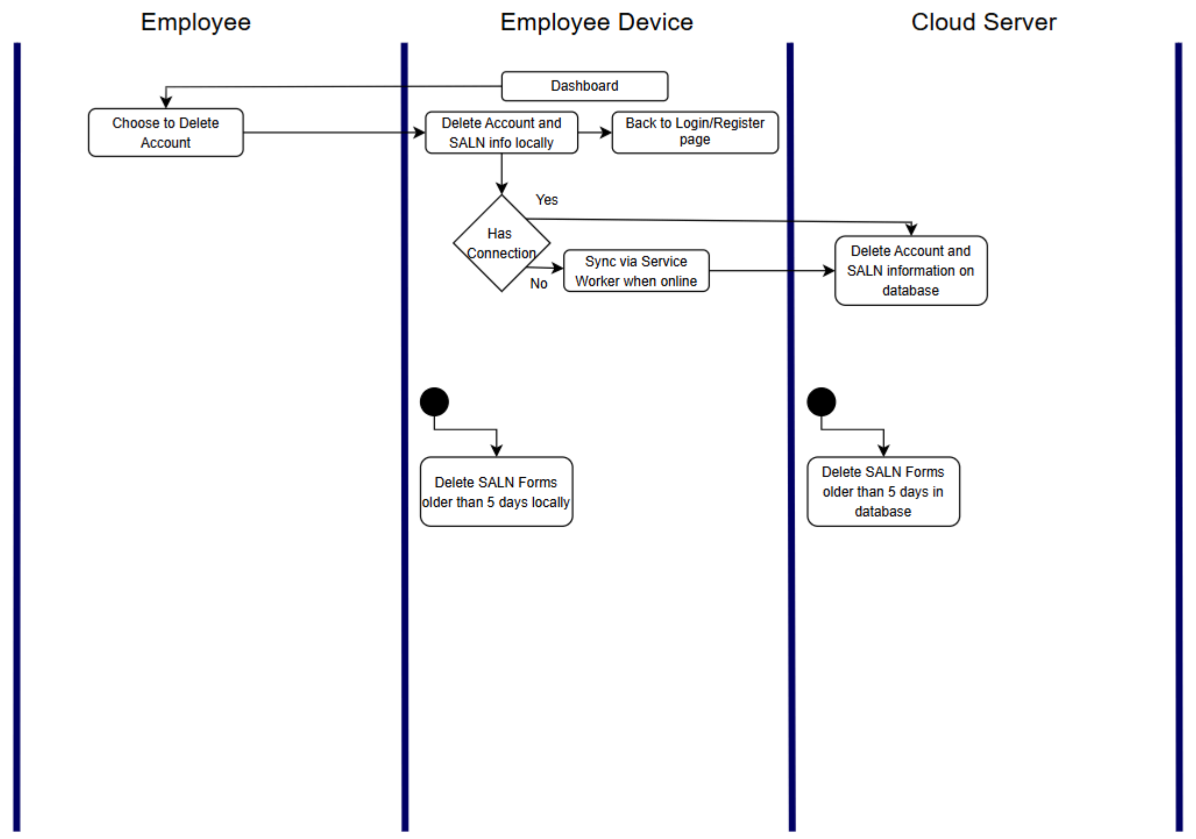
Upon looking at the Export JSON and Generate PDF options, fetching the SALN data is done from IndexedDB. This means that exporting the user's SALN data could be done completely offline.

Upon choosing to edit the SALN form, the user will be sent to the SALN Form GUI. There, he could edit his SALN form data. After that, the app would save those locally in IndexedDB. This allows the user to have an offline update. Then the user would go back to the dashboard. Additionally, the app would send an HTTP request to update the database with the new SALN data. If the user is offline, the Service Worker would save the HTTP Request to IndexedDB and sync it to the database once the user is back online.

Upon choosing to delete the SALN form, the app would ask the user for confirmation. Once the user gives confirmation, it will go through a similar process as the edit function. The local data gets updated, then an HTTP Request is sent to the web server to update the remote database. If the user is offline, then the HTTP Request gets stored into IndexedDB and gets synced once the user is back online via the Service Worker.



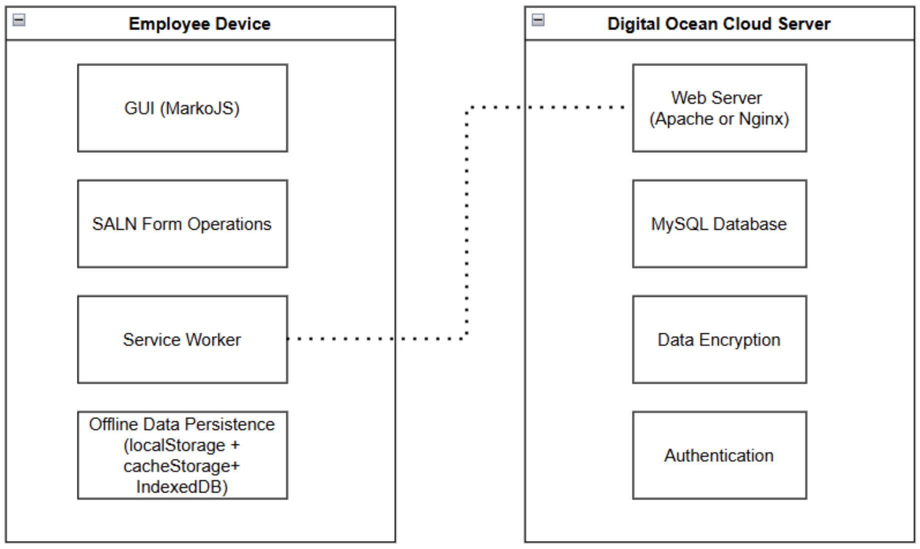
The third swimlane diagram contains SALN functions about uploading new SALN data. The user has the option to accomplish the SALN form digitally, or to upload a JSON file containing SALN data. Upon clicking either option, the app will send the user to the appropriate page that will prompt the user for data input. Once the user finishes inputting data, the progressive web app will make a local copy by saving it to IndexedDB and then send the user back to the dashboard. Similar to the previous functions, the Service Worker will also intercept the HTTP request for adding this new data into the remote database.


Employee
Employee Device
Cloud Server
Dashboard
Choose to Delete Account
Delete Account and SALN info locally
Back to Login/Register page
Yes
Has Connection
No
Sync via Service Worker when online
Delete Account and SALN information on database
Delete SALN Forms older than 5 days locally
Delete SALN Forms older than 5 days in database

This last swimlane diagram is about data deletion. Aside from the SALN functions in the previous swimlane diagrams, the user will also have the option to delete his account. Doing so has a similar workflow compared to the previous functions. Account information and the SALN data attached to that account will be deleted locally, and then the app will send the user back to the login/registration page. Furthermore, the Service Worker will intercept the HTTP Request for this deletion in the remote database.

Lastly, though this is not a function that could be done by the user, it is still an important function. The app will automatically delete SALN Forms that are older than 5 days, locally and remotely.

Configuration Model


Employee Device
GUI (MarkoJS)
SALN Form Operations
Service Worker
Offline Data Persistence (localStorage + cacheStorage+ IndexedDB)
Digital Ocean Cloud Server
Web Server (Apache or Nginx)
MySQL Database
Data Encryption
Authentication

As a Progressive Web Application, the app could be installed onto the browser's cache, allowing for use whilst online or offline. This installation would include the GUI of the app, SALN Form Operations, Service Worker code, and Offline Data Persistence code. This means that the user should be able to access the GUI and also make use of the SALN Form operations regardless of whether he is online or offline.

The Service Worker code will make use of the Service Worker API, which is provided by the user's browser. Basically, this module works on the logic for syncing the data for offline and online use. Additionally, the offline data persistence module will make use of built-in browser APIs such as localStorage, cacheStorage, and IndexedDB. localStorage will be used to store user account information, like email and user ID. cacheStorage would be used to store the app's assets such as HTML, CSS, and JS files for the frontend, and also the SALN template. IndexedDB acts as a local database, which could store the user's SALN entry, API requests, and session and refresh tokens.

The system will make use of a cloud server by Digital Ocean. For the configuration of the cloud server:

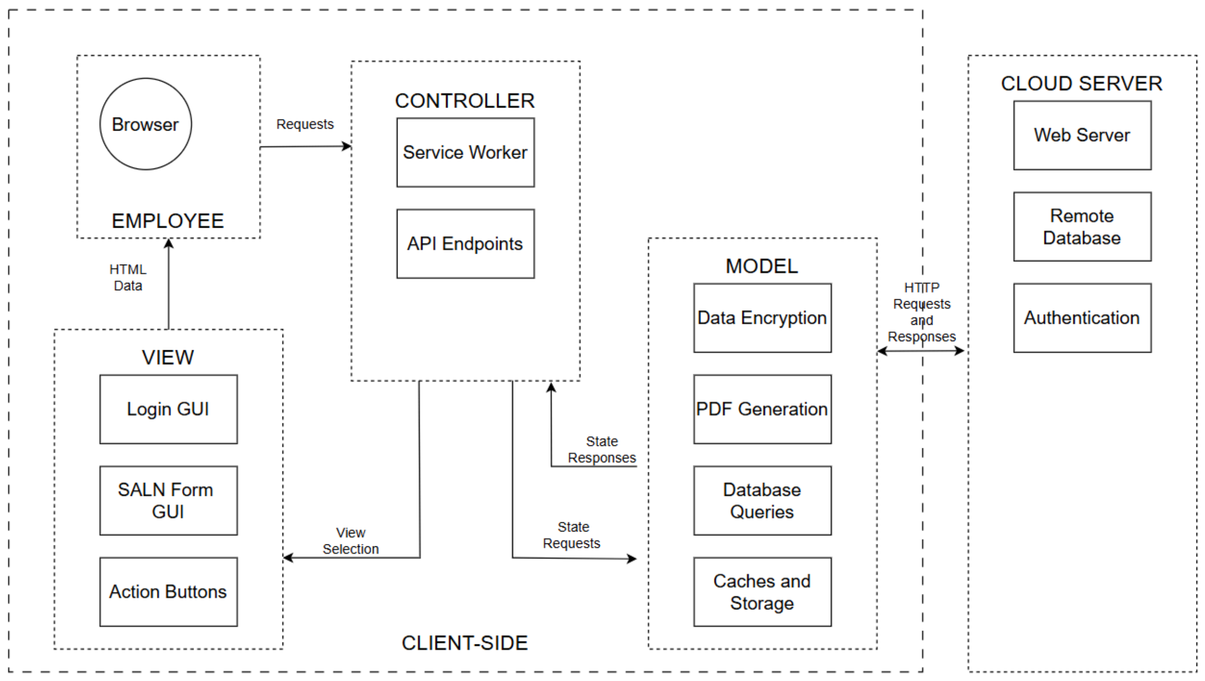
- OS Image: Ubuntu 24.04 (LTS) x64
- Disk Type: SSD
- 1 GB / 1 CPU
- 25 GB SSD Disk
- 1000 GB Transfer

The cloud server would make use of Apache acting as a web server receiving HTTP Requests from clients. It would also make use of MySQL for Database management. Data Encryption would be done using the SubtleCrypto interface of the WebCrypto API. Additionally, authentication logic for log-in and registration will be done on Laravel. A session token could then be saved, and made persistent, in the user's browser's localStorage.

WEB APP ARCHITECTURE AND DESIGN

Architecture Design

Architectural Style



The architectural style that this progressive web application will use is the Model-View-Controller (MVC) Architecture. In the View, we have modules on how to present our data and actions in the Employee's Browser, in the form of HTML data. These include modules on how to present the Login GUI, the SALN Form GUI, and the action buttons for the current app state.

The Controller contains API Endpoints that the Employee could use to interact with the View and the Model. In a sense, it is the endpoint on how the Employee could manage his data through making requests. Additionally, the Controller has the Service Worker, which will determine if the app will be using offline data stored locally or online data stored remotely. The Controller also has the ability to select what the View will be presenting based on the current states of the application in terms of authentication and SALN accomplishment.

The Model contains the backend functionality of the app. It has the logic of determining whether the Employee is authenticated. It also contains methods of fetching the employee's SALN data, either from the remote database or the local cache. The model also encapsulates request handling logic, which includes Database Queries, Queries to the local Cache and Storage, Data Encryption, and PDF Generation. It also has a connection to the external Cloud Server, which encapsulates the Web Server, Remote Database, and Authentication.

It is also worth noting that in order to have persistent offline functionality, everything encapsulated by the model, view, and controller is all on the client side, meaning the code for these would be cached.

Addressing Technical Attributes

Usability. The architecture makes use of GUIs from the View to be displayed on the Employee Browser to interact with functions from the Model in an abstracted way, such that it is understandable what the result of an input would be. Additionally, since this is a progressive web app, responsiveness will be handled by the View component.

Functionality. The Controller will determine what pages the user is allowed to navigate into based on the state of the Model. These allowed navigation are then sent to the View, and then the View would show the necessary GUI elements on the Employee Browser for those navigations. The Model is in charge of processing data from user inputs by storing them locally and in the remote database.

Reliability. To avoid errors related to fetching and updating data in the remote database, the Controller has a Service Worker to intercept requests just in case a connection is unable to be established. For user inputs, the Controller is in charge of user input validation before the data could enter the model to manipulate the database. Additionally, with the MVC Architecture, the practice of separations of concerns is in play. So an error in one of the components shouldn't affect the other components.

Efficiency. Since this is a Progressive Web App, files for the GUI are already stored in cache. This means that the GUI is already available offline and wouldn't need to be fetched from a remote server every time the app is used. Additionally, some other functionality is also kept locally, such as PDF generation. Also, SALN operations have their offline counterparts since the Service Worker in the Controller intercepts requests and then just syncs data on a separate thread.

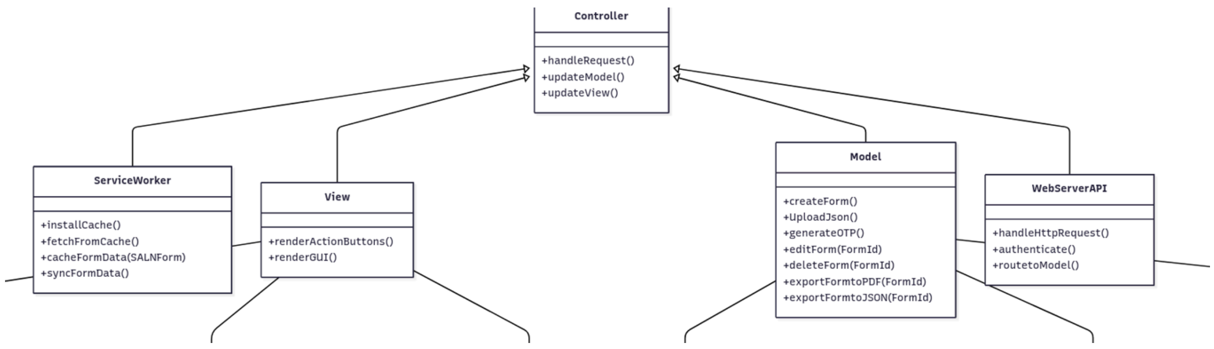
Maintainability. The separation of concerns via the MVC Architecture allows us to maintain different components separately. If we want to modify backend functionality, then we would just need to edit the Model. If we want to modify how the GUI looks, then we would just need to edit the View.

Architectural Decisions

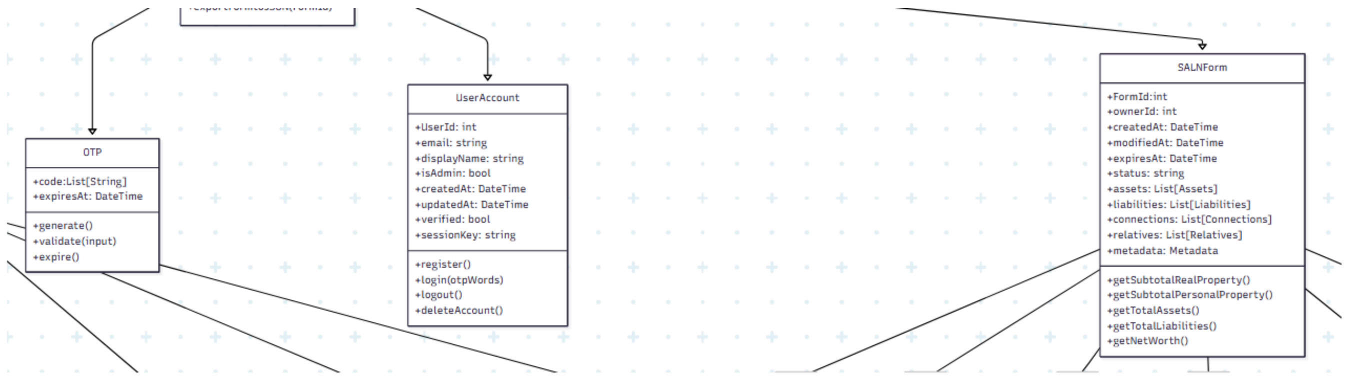
Design Issue	SALN Progressive Web App must have offline functionality. Additionally, it must sync remote SALN data with the local SALN data once back online.
Resolution	Have a Service Worker intercept Requests before they are sent to the Web Server. This way, if the Employee is offline, the Service Worker will store this request and then fetch it once back online. The progressive web app would also maintain a local version of the database via IndexedDB. This way, even in offline mode, the Employee still has access to their latest SALN data. Offline functionality for PDF generation and data encryption would be supported by fully offline dependencies.
Category	Integration
Assumptions	The employee must have his latest SALN data available offline. The Employee must be able to generate PDF offline.
Constraints	Service Workers, IndexedDB, cacheStorage, and localStorage are available built-in browser tools.
Alternatives	Make a native app version alongside the website. This was rejected because the effort needed to create both native app and website would be more. Additionally, the project client requested that no binary be installed for offline functionality.
Argument	From a developer perspective, a progressive web app would be simpler for data syncing because of built-in browser tools like IndexedDB, cacheStorage, and Service Worker API. From a user perspective, this allows the user to perform functions much quicker since everything would just be in cache. The online Requests would be performed on a separate thread.
Implications	Requests would be intercepted by a Service Worker. Database updates would be reflected in local IndexedDB. Code for needed offline functionality would be cached locally.
Related Decisions	None.
Related Concerns	None.
Work Products	Controller: particularly the Service Worker. Model: it contains local cache and storage.
Notes	None.

Content (Data) Design

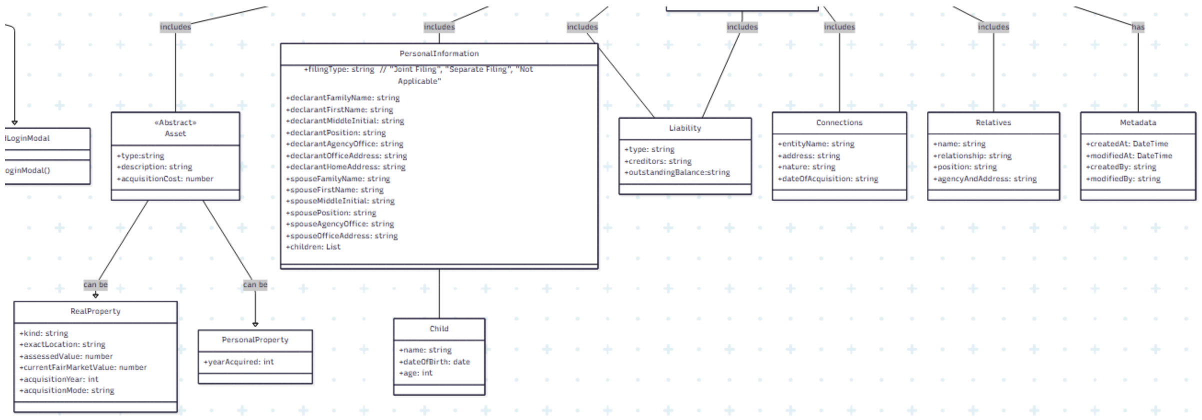
Class Diagrams



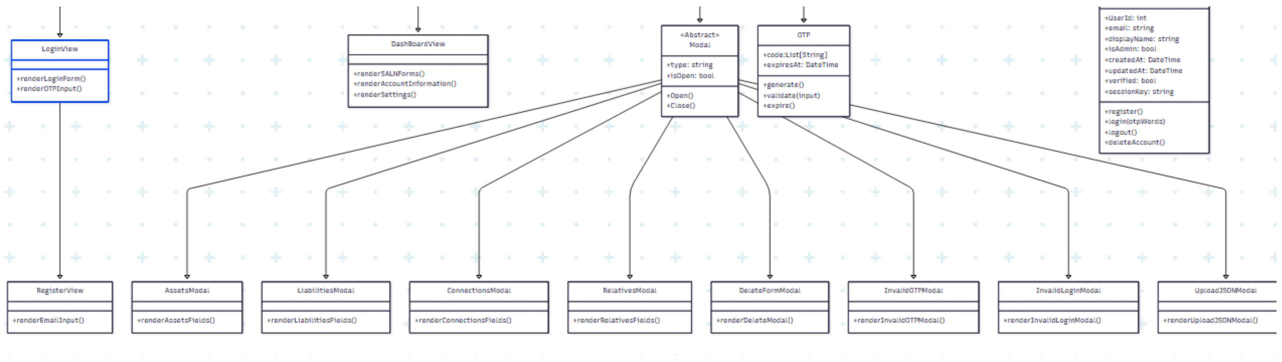
The UML class diagram of the SALN Management System is built on the **Model-View-Controller (MVC) framework**. The **Controller** coordinates requests, updates the Model, and refreshes the View. The **Model** encapsulates business logic, handling SALN forms, data exports, and authentication through OTP generation. The **View** renders the interface (the GUI), while the **WebServerAPI** supports web requests and routing. The **ServiceWorker** complements this by enabling caching, synchronization, and offline access.



Under the Model, in which we handle the state and the data of the Webapp, we have the **OTP**, the **SALN Form**, and the **User Account.** The OTP class contains the OTP itself and its needed details. The SALNForm class encapsulates its attributes, such as assets, liabilities, connections, relatives, and metadata, while also providing the operations to compute subtotals and net worth. The UserAccount class manages registration, login, session handling, and account verification, ensuring that each form is tied securely to its owner.

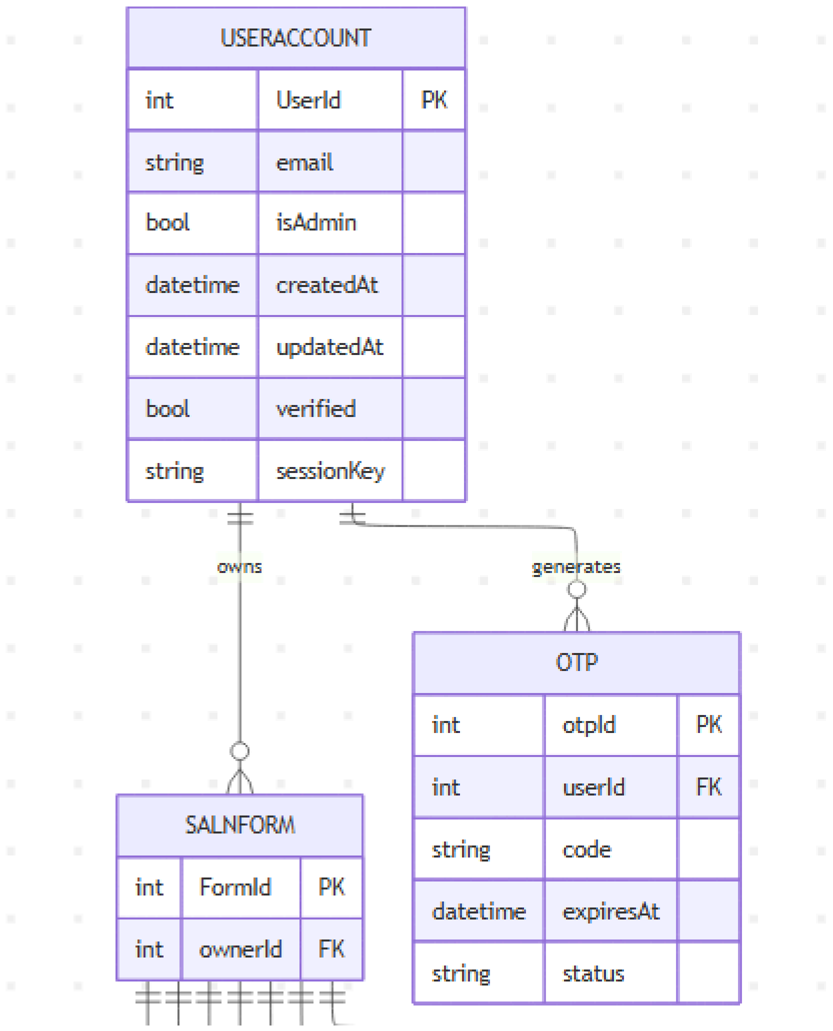


For the SALN form class, here are its main details separated within classes. The **PersonalInformation** class has a **Child** subclass since a list of **Child** is needed in the **PersonalInformation** class The abstract class **Asset** makes the implementation of **RealProperty** and **PersonalProperty** class easier (since they are related and have the same attributes). The **Metadata** class here is important for the system to know when to delete the file, and the owner of the SALN.



Under the **View** Class, we have the **LoginView** subclass, **DashboardView** subclass, and the **Modal** abstract subclass. The LoginView is where the User will start first when visiting the WebApp, and if the user has no account then he will need the RegisterView. The Dashboard view is essentially the Dashboard for the users, where they can access their account info, settings, and their SALN forms. For the Modal superclass, these are where the essential popups or fields for the SALN forms are rendered (AssetsModal, LiabiltiesModal, DeleteFormModal).

Entity-Relationship Diagrams



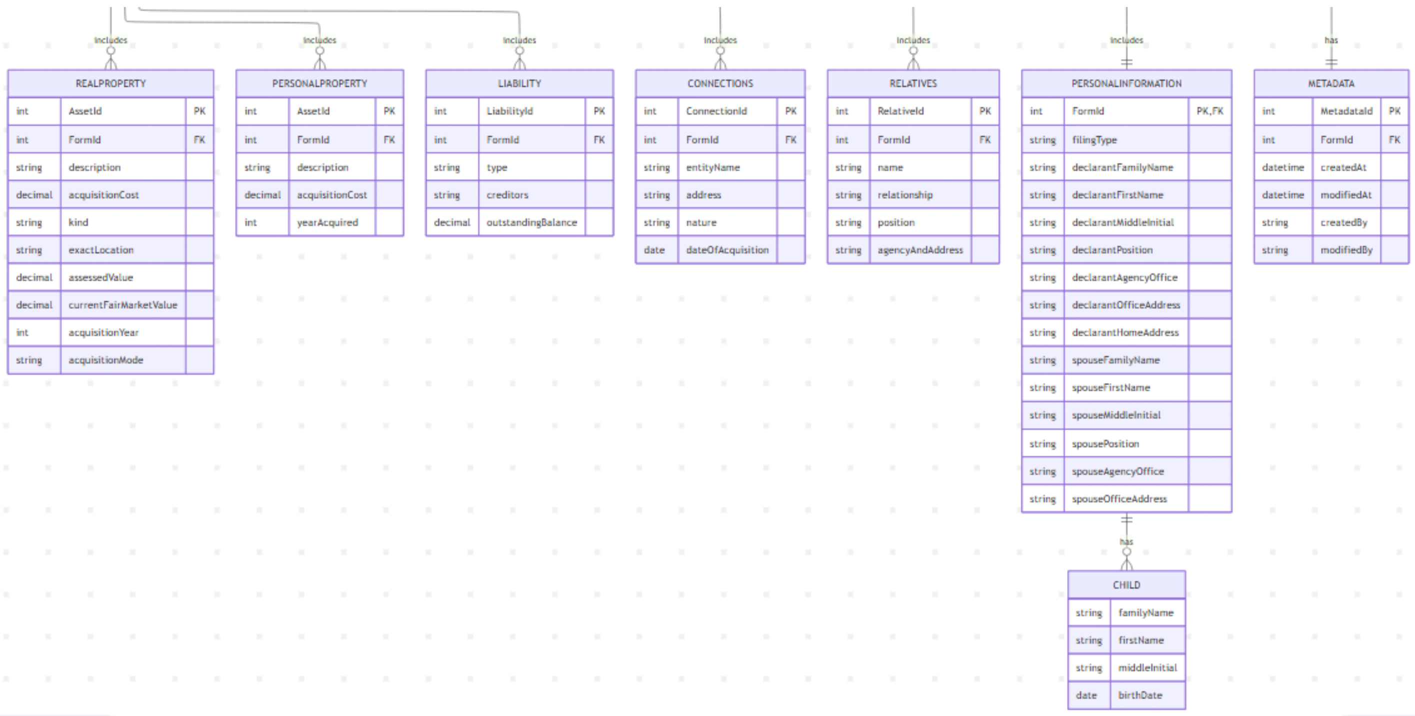
USERACCOUNT is a table that contains the information stored in each user account. This includes the following:

- UserId - an integer serving as the unique identifier for each user account. This serves as the primary key for this table.
- email - a string denoting the email address inputted by the user during registration.
- createdAt - a datetime indicating the date and time the account was created.
- updatedAt - a datetime indicating the last time the user made edits to an SALN form.
- verified - a bool indicating whether the user has verified their account with an OTP.
- sessionKey - a string indicating the session key of the user if there is an ongoing session. This attribute is set to null if there is no ongoing session.

OTP is a table that contains the OTPs that have been generated by the app. Every week, expired and used OTPs will be deleted from the database. The attributes of this table include the following:

- otpId - an integer serving as the unique identifier for each OTP. This serves as the primary key for this table.
- userId - an integer that identifies the user who needs to enter this OTP. This is a foreign key that relates the OTP to the USERACCOUNT.
- code - a string that denotes the passcode to be entered by the user. This consists of four words concatenated.
- expiresAt - a datetime indicating the date and time that the OTP expires.
- status - a string indicating the status of the OTP

SALNFORM is a table that contains the FORMid to differentiate between multiple SALNs and the OWNERid to know the author of the SALN. Referencing tables under it are its main contents (Assets, Liabilities, Connections, Relatives, Personal Information) and the metadata which stores the information needed for the SALN form (datecreated,modifiedat).



REALPROPERTY		
int	AssetId	PK
int	FormId	FK
string	description	
decimal	acquisitionCost	
string	kind	
string	exactLocation	
decimal	assessedValue	
decimal	currentFairMarketValue	
int	acquisitionYear	
string	acquisitionMode	

PERSONALPROPERTY		
int	AssetId	PK
int	FormId	FK
string	description	
decimal	acquisitionCost	
int	yearAcquired	

LIABILITY		
int	LiabilityId	PK
int	FormId	FK
string	type	
string	creditors	
decimal	outstandingBalance	

CONNECTIONS		
int	ConnectionId	PK
int	FormId	FK
string	entityName	
string	address	
string	nature	
date	dateOfAcquisition	

RELATIVES		
int	RelativeId	PK
int	FormId	FK
string	name	
string	relationship	
string	position	
string	agencyAndAddress	

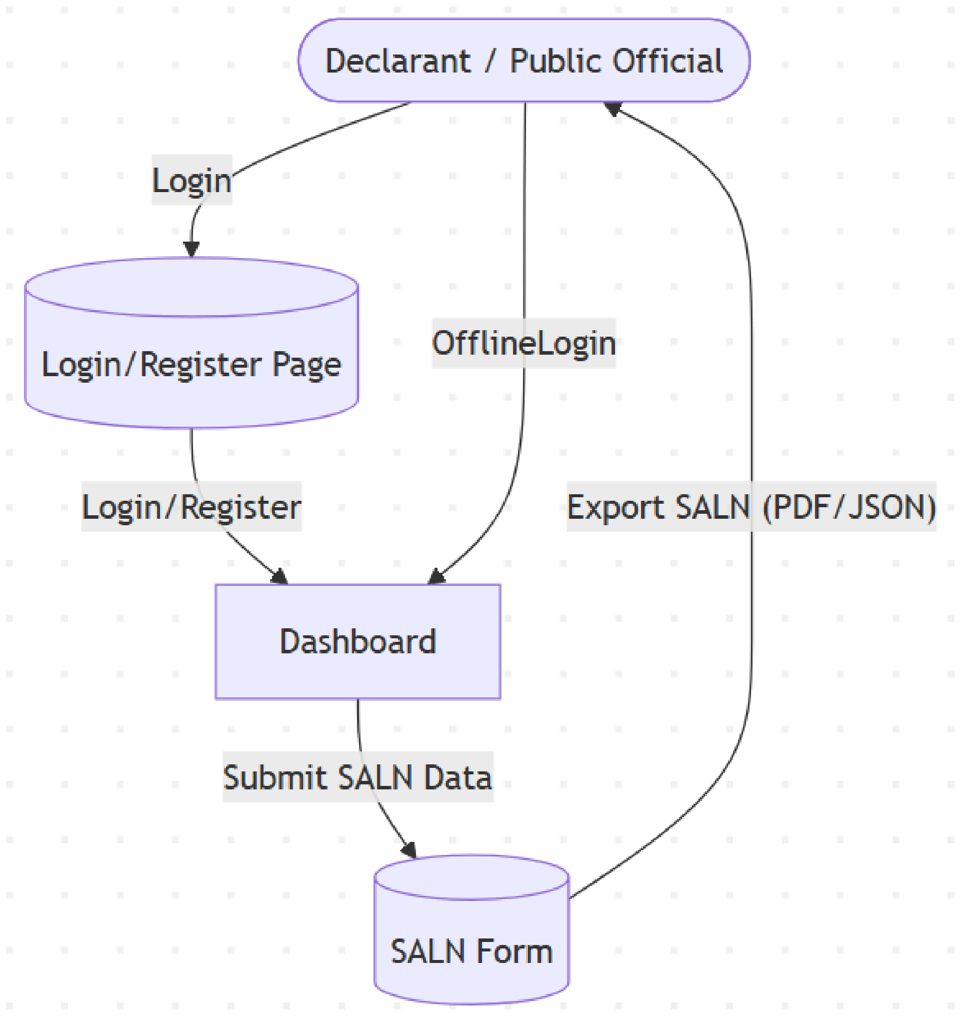
PERSONALINFORMATION		
int	FormId	PK,FK
string	filingType	
string	declarantFamilyName	
string	declarantFirstName	
string	declarantMiddleInitial	
string	declarantPosition	
string	declarantAgencyOffice	
string	declarantOfficeAddress	
string	declarantHomeAddress	
string	spouseFamilyName	
string	spouseFirstName	
string	spouseMiddleInitial	
string	spousePosition	
string	spouseAgencyOffice	
string	spouseOfficeAddress	

METADATA		
int	MetadataId	PK
int	FormId	FK
datetime	createdAt	
datetime	modifiedAt	
string	createdBy	
string	modifiedBy	

CHILD	
string	familyName
string	firstName
string	middleInitial
date	birthDate

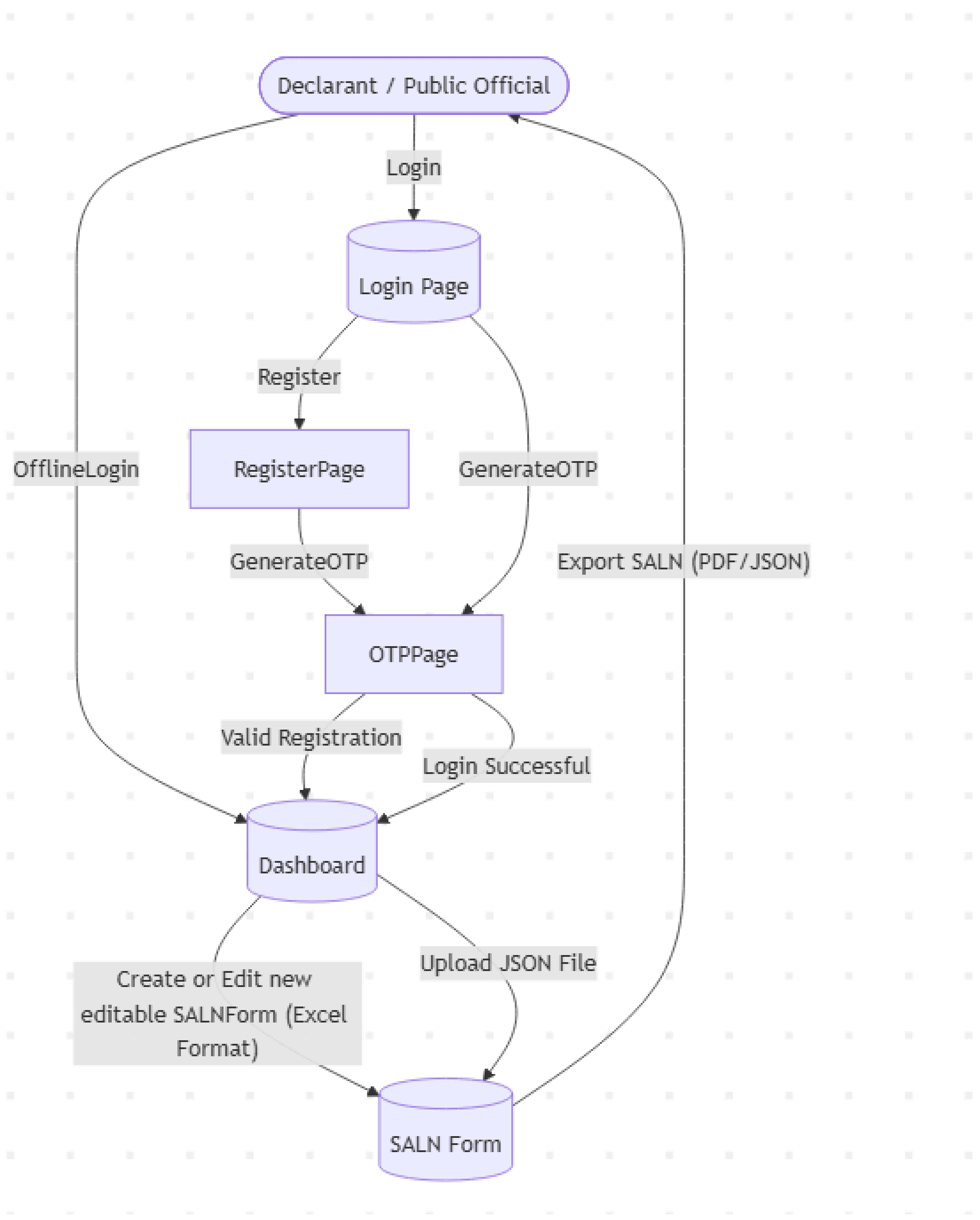
Data Flow Diagrams

LVL 0 DATA FLOW DIAGRAM



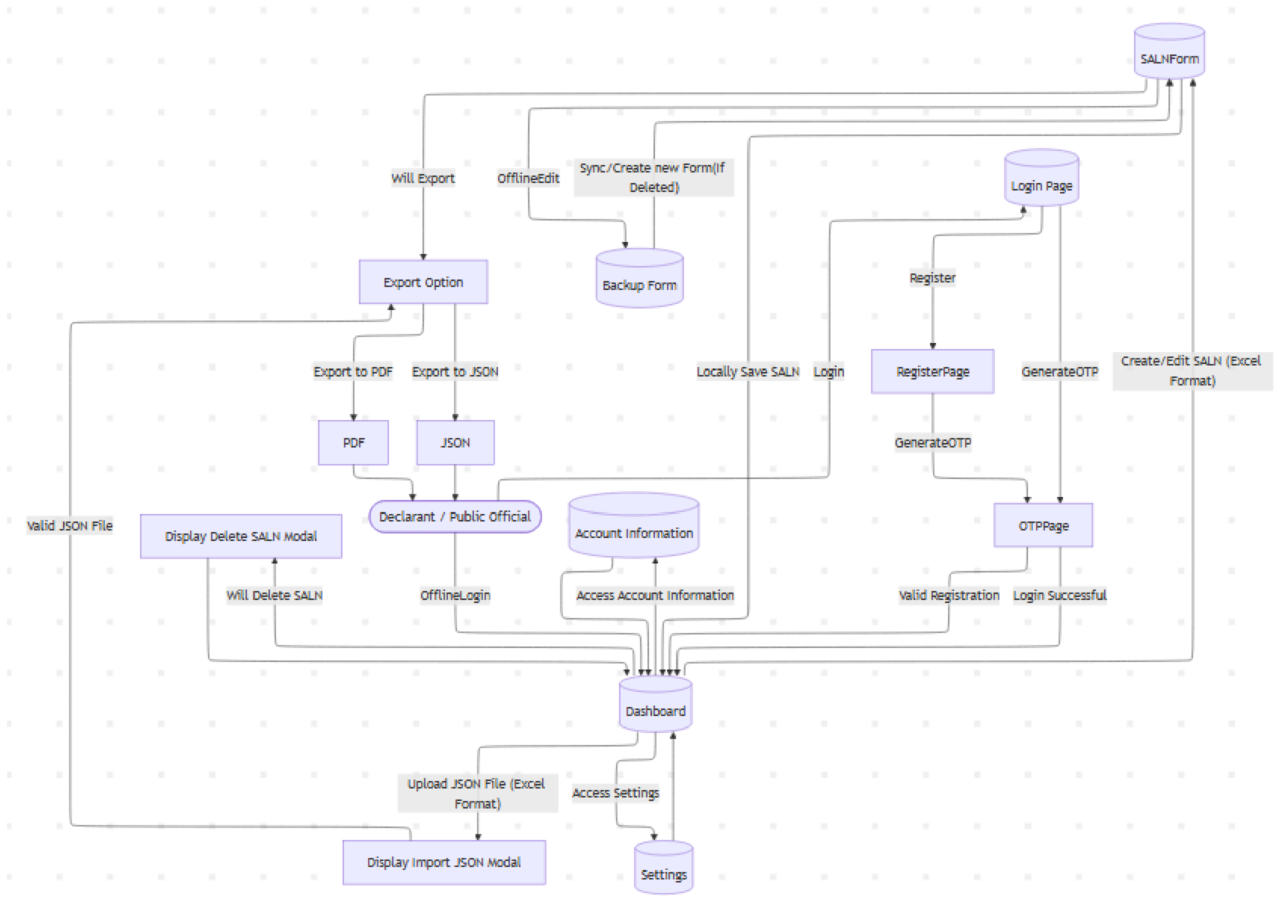
The Level 0 Data Flow Diagram exhibits the main functionality of the system. The User will just Login(Online/Offline) or Register(If the user has no account yet), Visit the dashboard to Submit/Fillup SALN Data, then it can be exported as PDF/JSON by the user.

LVL 1 DATA FLOW DIAGRAM



For the Level 1 Data Flow Diagram, the Login/Registration process is just expanded to show the step by step process of logging in the system. We can see that we need an OTP when we login to the system (Online) to validate the user. Then if the user sent the correct OTP, he/she can know access the Dashboard to edit or upload a JSON file for their SALNform.

LVL 2 DATA FLOW DIAGRAM

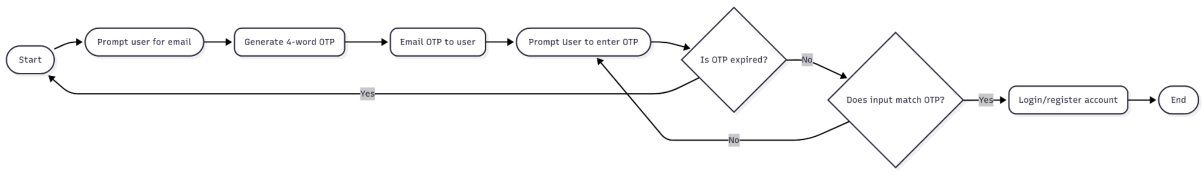


For the level 2 Diagram, we just added the other information we can get from the Dashboard, and how the system works when Creating/Editing SALNforms offline or online. We can see from the data flow diagram above that using the Dashboard, you can access your account information, settings, deleteSALNforms, create/edit SALNforms, and upload a JSONfile that contains the SALNform data. For the Offline editing of the SALN form, we implemented a policy that if the SALN data is deleted online by another device, the cached offline SALN data will be moved to a new SALN Form with a different FormId.

Component-Level Design

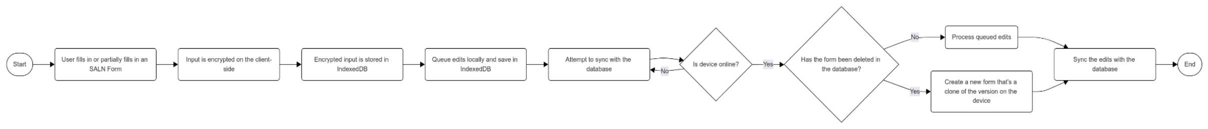
Algorithms

a. UML Activity Diagram for OTP Generation and Emailing



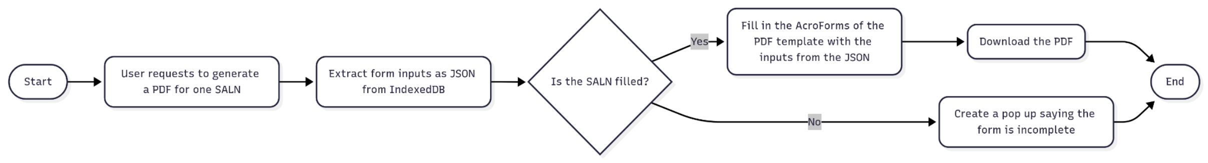
At the login page, the user will be prompted to enter an email address. Once they have done so, an OTP will be generated by the backend by doing the following: generating a random number, then using it to select four random words from the EFF's Long Wordlist. The OTP will then consist of the OTP along with a timestamp for the expiry date. An email will then be sent to the user containing the OTP using the built-in Laravel Mail API. The app will then prompt the user to enter the OTP they received. Once the user inputs, the app will check the OTP timestamp to check if it has expired or already been used. If so, it would create a pop-up, then send the user back to the login page. If the OTP has not expired, then the app would check if the user's input matches the OTP. If it does not match, the user will be prompted again. If it does, then the user will successfully be registered or logged in to their account.

b. UML Activity Diagram for CRUD


Start
User fills in or partially fills in an SALN Form
Input is encrypted on the client-side
Encrypted input is stored in IndexedDB
Queue edits locally and save in IndexedDB
Attempt to sync with the database
No
Is device online?
Yes
Has the form been deleted in the database?
No
Process queued edits
Yes
Create a new form that's a clone of the version on the device
Sync the edits with the database
End

Whenever a user makes an edit to an SALN form, the updated information gets encrypted, then stored in IndexedDB. Next, the edits are stored in a queue. The next time the device goes online, the app will attempt to synchronize with the database. On the backend, the incoming edits will be processed one by one using the queue. For each edit, if the corresponding form exists in the database, then the backend will save the edits then sync with devices logged into the account. If not, this indicates that the form was deleted through either auto-deletion or deletion from another device. A new form will then be created with the contents from the local version copied to it.

c. UML Activity Diagram for PDF Generation


Start
User requests to generate a PDF for one SALN
Extract form inputs as JSON from IndexedDB
Is the SALN filled?
Yes
Fill in the AcroForms of the PDF template with the inputs from the JSON
Download the PDF
No
Create a pop up saying the form is incomplete
End

When the user requests to generate the PDF for a form, the app first retrieves the JSON containing the form inputs from IndexedDB and checks if all the required entries of the form, other than the signature, are filled. If not, a pop-up will show up saying the form has not been completed, then the user will be redirected to the dashboard. If the form input is valid, then the values in the JSON will be filled into the inputs of the SALN PDF template stored in IndexedDB. Lastly, the PDF will be rendered, then downloaded to the user's computer.

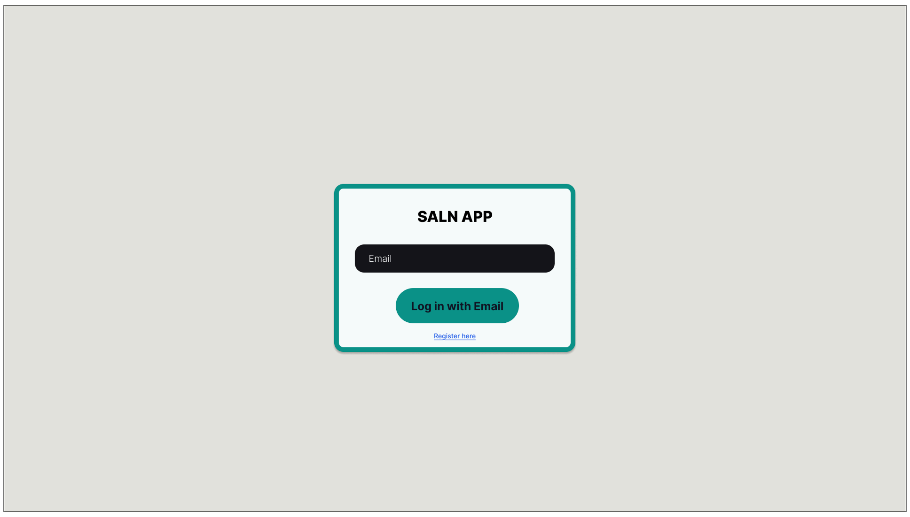
Data Structures

All of the information the user inputs in the forms will be stored in JSONs. This allows for the storage of information in the cache for offline functionality and the exchange of information with the database for CRUD.

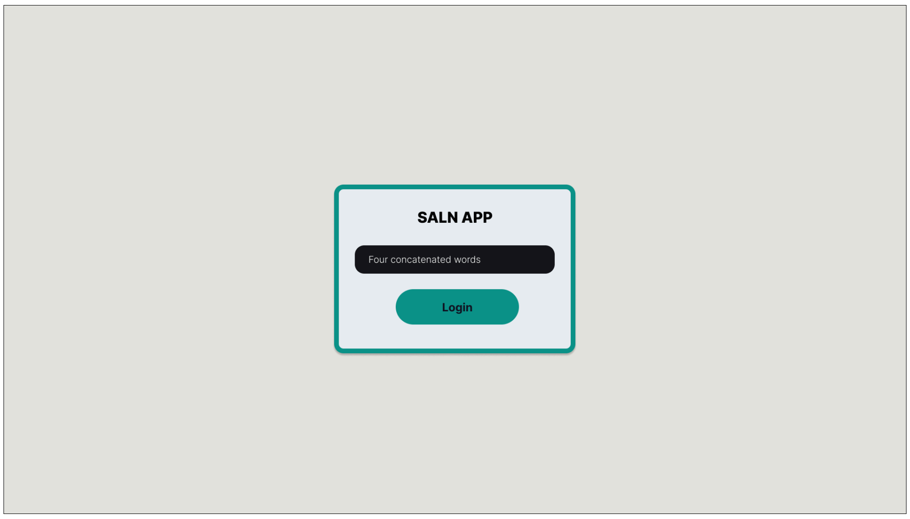
The app will make use of a queue to store and process form edits. Because queues are First In, First Out, edits will be processed in the order they are done chronologically, with the timestamp deciding which edits will be saved. As a result, conflicting edits and histories between devices logged into the same account and the database can be resolved.

Interface Elements / Usability Design

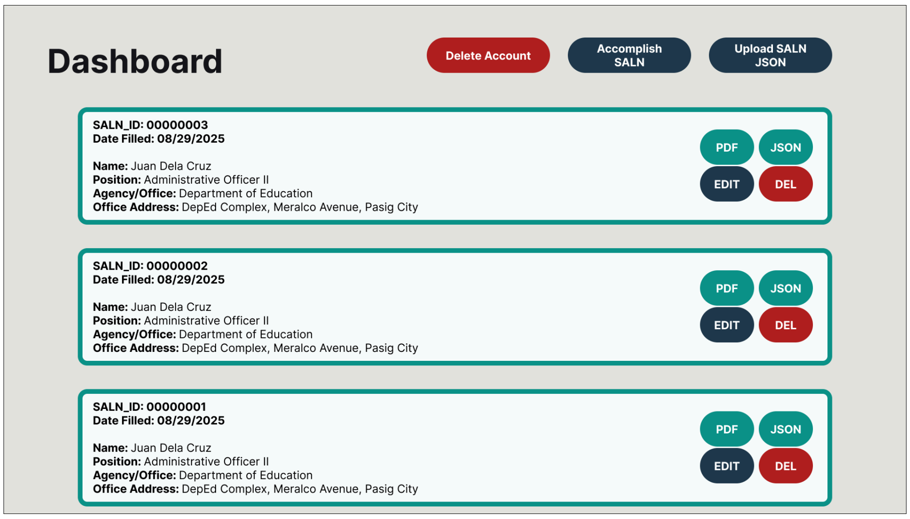
When the employee logs into the app, he will be brought to the login interface. There, he will be prompted for his email address. Additionally, if the employee does not have an account yet, he could click the register link at the bottom of the modal. The registration process will be similar to that of the logging in when it comes to the interface and necessary user input.



After entering the email, the web server will send said email four random words for OTP purposes. The employee will then be prompted to enter those four words.



Once the employee has entered the correct four words, he will be logged into the app and redirected to the dashboard page. Once on the dashboard, the SALN data of the employee will be displayed alongside with other action buttons. On the top right corner, there are buttons for general actions that could be performed. In general, an employee could choose to accomplish a new SALN form, to upload SALN data in JSON format, or to delete his account. Additionally, a list of the employee's currently available SALN forms will be displayed. Each displayed SALN form has four buttons for possible actions: to export as PDF, to export as JSON, to edit the data, and to delete the data.



SALN_ID: 00000003
Date Filled: 08/29/2025

Name: Juan Dela Cruz
Position: Administrative Officer II
Agency/Office: Department of Education
Office Address: DepEd Complex, Meralco Avenue, Pasig City

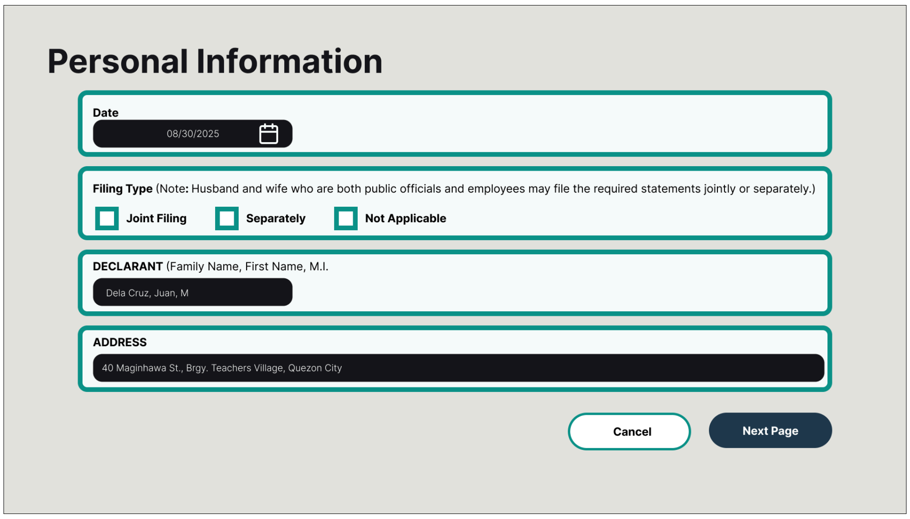
SALN_ID: 00000002
Date Filled: 08/29/2025

Name: Juan Dela Cruz
Position: Administrative Officer II
Agency/Office: Department of Education
Office Address: DepEd Complex, Meralco Avenue, Pasig City

SALN_ID: 00000001
Date Filled: 08/29/2025

Name: Juan Dela Cruz
Position: Administrative Officer II
Agency/Office: Department of Education
Office Address: DepEd Complex, Meralco Avenue, Pasig City

If the employee clicks the *Accomplish SALN* button, he will be sent to the digitalized SALN form to input SALN data. The fields being prompted mirror that of the physical SALN form. For example, here is the first page of the digital form taking in some personal information, which reflects the first few fields of the physical SALN form. Then, at the bottom of the page, there are


Personal Information

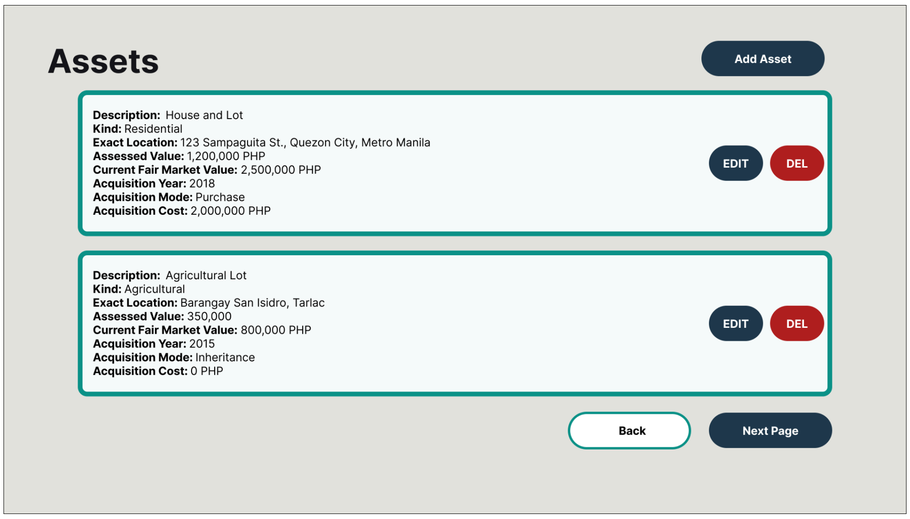
Date
08/30/2025

Filing Type (Note: Husband and wife who are both public officials and employees may file the required statements jointly or separately.)
☐ Joint Filing ☐ Separately ☐ Not Applicable

DECLARANT (Family Name, First Name, M.I.
Dela Cruz, Juan, M

ADDRESS
40 Maginhawa St., Brgy. Teachers Village, Quezon City

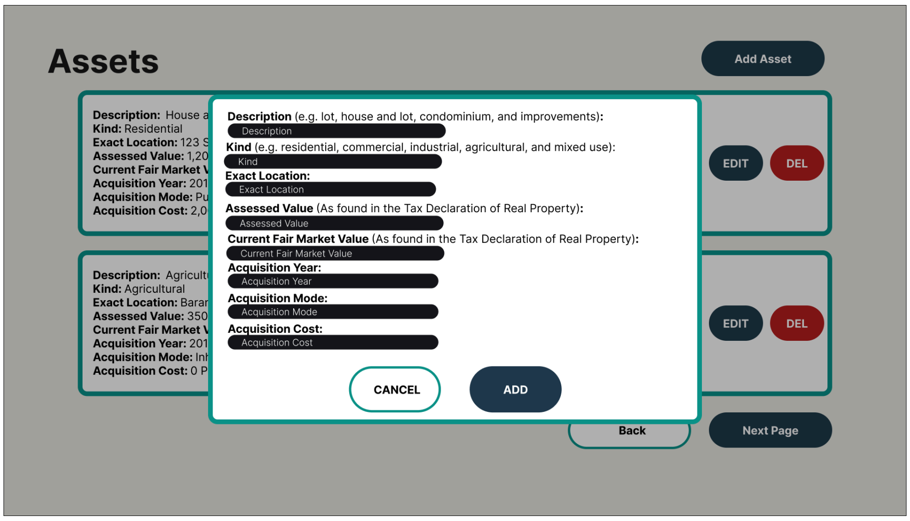
It is also worth noting that on the physical SALN form, there are tables for assets and liabilities. Instead of a table, the digitalized SALN form displays asset fields in a list of modals. All tables from the physical SALN will also share this GUI layout. On the top right, there is an action button to add another asset. For each asset, there is an option to edit the asset and then there is an option to delete the asset.


Assets

Description: House and Lot
Kind: Residential
Exact Location: 123 Sampaguita St., Quezon City, Metro Manila
Assessed Value: 1,200,000 PHP
Current Fair Market Value: 2,500,000 PHP
Acquisition Year: 2018
Acquisition Mode: Purchase
Acquisition Cost: 2,000,000 PHP

Description: Agricultural Lot
Kind: Agricultural
Exact Location: Barangay San Isidro, Tarlac
Assessed Value: 350,000
Current Fair Market Value: 800,000 PHP
Acquisition Year: 2015
Acquisition Mode: Inheritance
Acquisition Cost: 0 PHP

When adding an asset, a modal prompting for asset fields will appear at the center of the screen.

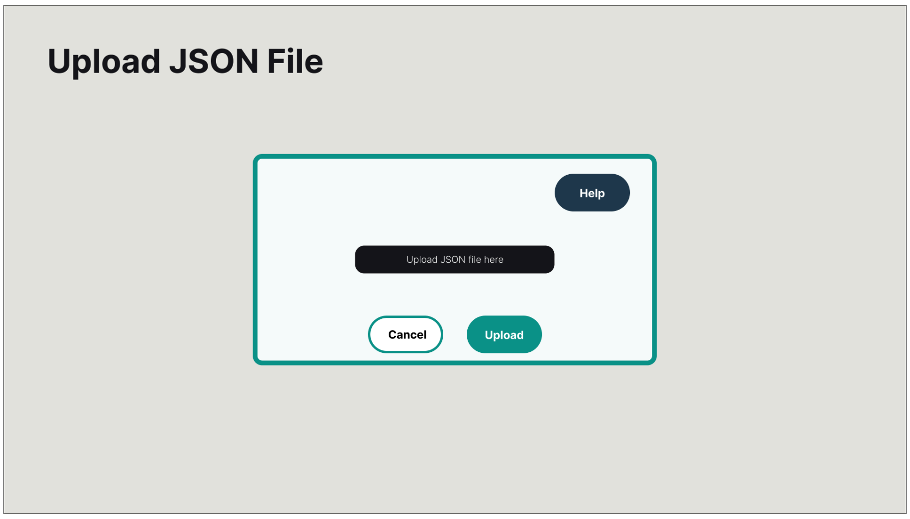

Assets

Description: House a
Kind: Residential
Exact Location: 123 S
Assessed Value: 1,20
Current Fair Market V
Acquisition Year: 201
Acquisition Mode: Pu
Acquisition Cost: 2,0

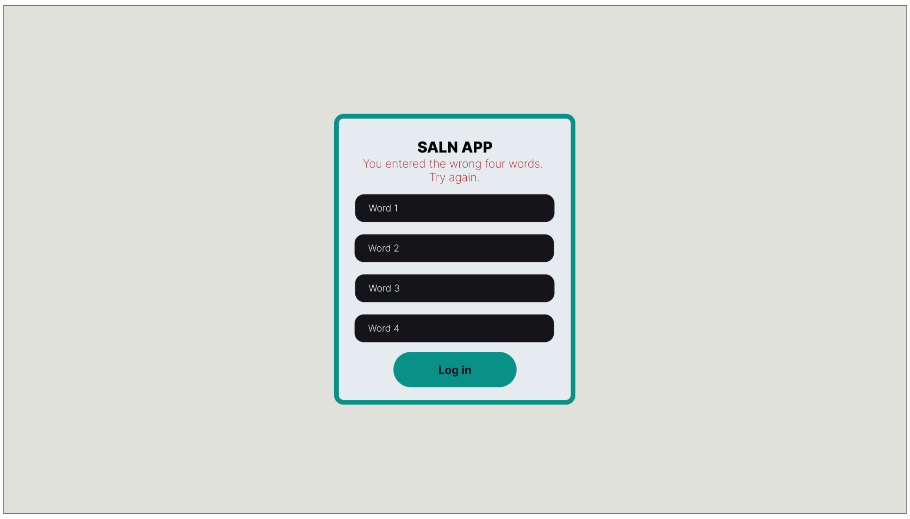
Description: Agricult
Kind: Agricultural
Exact Location: Barar
Assessed Value: 350
Current Fair Market V
Acquisition Year: 201
Acquisition Mode: Inh
Acquisition Cost: 0 P

Description (e.g. lot, house and lot, condominium, and improvements):
Kind (e.g. residential, commercial, industrial, agricultural, and mixed use):
Exact Location:
Assessed Value (As found in the Tax Declaration of Real Property):
Current Fair Market Value (As found in the Tax Declaration of Real Property):
Acquisition Year:
Acquisition Mode:
Acquisition Cost:

Going back to the dashboard, there was also a decision to upload a JSON file containing SALN form data. Clicking that navigation button will send the employee to a page prompting them to upload a JSON file. Additionally, there is a *Help* button at the top right corner of the modal. This button will bring the employee over to documentation of the JSON format for SALN form data.

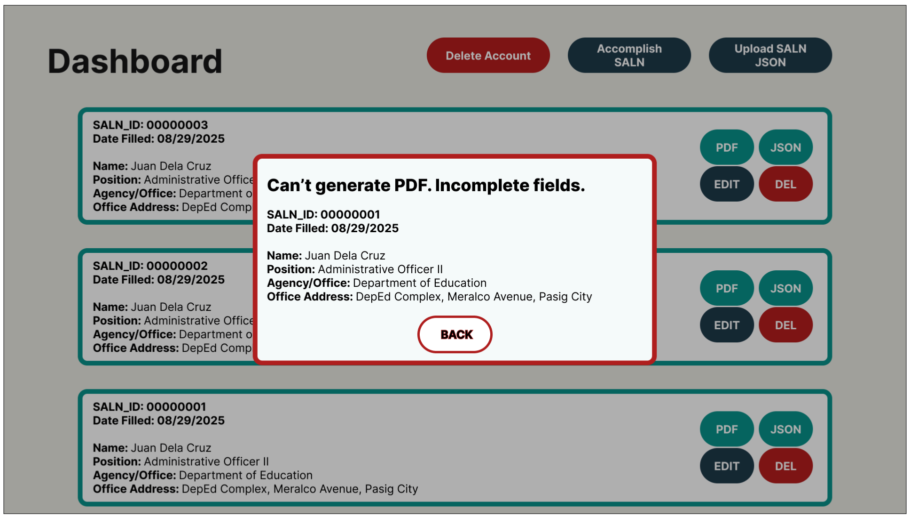


It is worth noting that the app also deals with error handling. One way would be having a text prompt show up. For example, take notice of the red text that appears once the user has entered the wrong four random numbers during a login attempt.


SALN APP

You entered the wrong four words. Try again.

Another way would be having a modal pop up. For example, a user tries to generate a PDF version of his SALN form despite said SALN form being incomplete.


Dashboard

SALN_ID: 00000003
Date Filled: 08/29/2025

Name: Juan Dela Cruz
Position: Administrative Office
Agency/Office: Department o
Office Address: DepEd Comp

Can't generate PDF. Incomplete fields.

SALN_ID: 00000001
Date Filled: 08/29/2025

Name: Juan Dela Cruz
Position: Administrative Officer II
Agency/Office: Department of Education
Office Address: DepEd Complex, Meralco Avenue, Pasig City

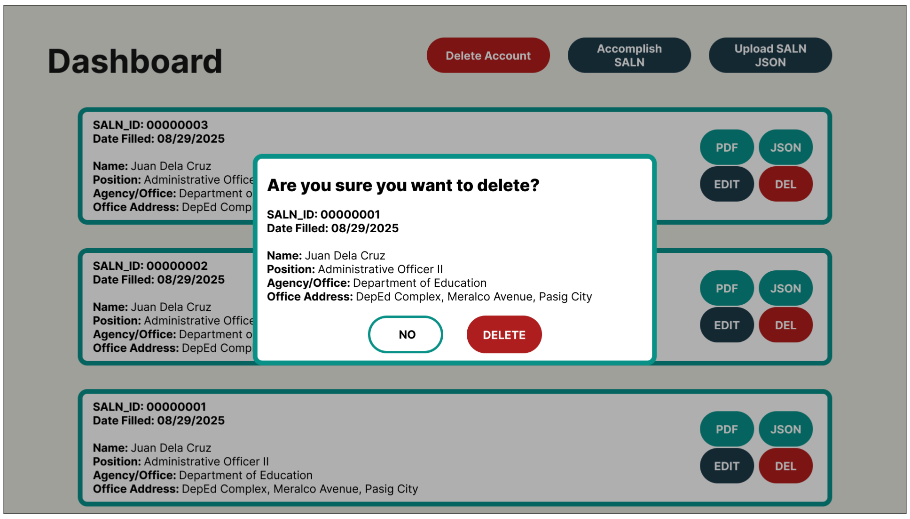
SALN_ID: 00000002
Date Filled: 08/29/2025

Name: Juan Dela Cruz
Position: Administrative Office
Agency/Office: Department o
Office Address: DepEd Comp

SALN_ID: 00000001
Date Filled: 08/29/2025

Name: Juan Dela Cruz
Position: Administrative Officer II
Agency/Office: Department of Education
Office Address: DepEd Complex, Meralco Avenue, Pasig City

The app also has user error prevention. For example, upon clicking a *Delete* button, the app will have an alert pop up asking for confirmation. For instance, when clicking the *Delete* button for a SALN form, a prompt will show up asking if the user wants to delete the SALN form along with the SALN form's distinguishing fields.



SALN_ID: 00000003
Date Filled: 08/29/2025
Name: Juan Dela Cruz
Position: Administrative Office
Agency/Office: Department o
Office Address: DepEd Comp

Are you sure you want to delete?

SALN_ID: 00000001
Date Filled: 08/29/2025
Name: Juan Dela Cruz
Position: Administrative Officer II
Agency/Office: Department of Education
Office Address: DepEd Complex, Meralco Avenue, Pasig City

SALN_ID: 00000002
Date Filled: 08/29/2025
Name: Juan Dela Cruz
Position: Administrative Office
Agency/Office: Department o
Office Address: DepEd Comp

SALN_ID: 00000001
Date Filled: 08/29/2025
Name: Juan Dela Cruz
Position: Administrative Officer II
Agency/Office: Department of Education
Office Address: DepEd Complex, Meralco Avenue, Pasig City

Additionally, since there are no complex animations in the GUI, rendering the graphics should be quick. Additionally, since the data to be used is coming from local cache, there won't be much overhead in fetching form data to be displayed compared to fetching form data from the remote database.

Interface Design - Menus and Links

Navigation mechanisms come in the form of buttons with an action function attached to them as attributes. Their layouts could be seen in the storyboard images above.

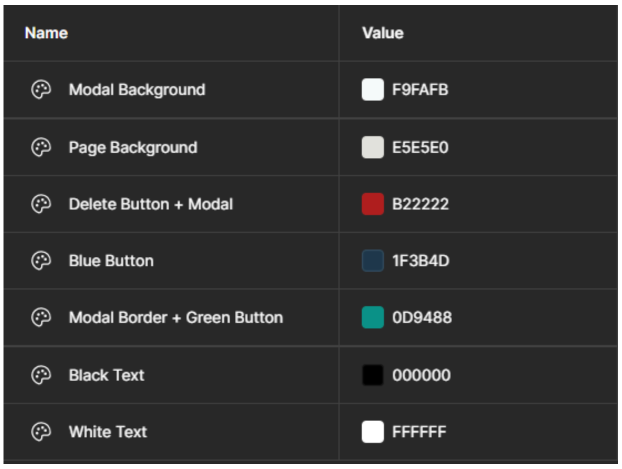
Starting off with the login interface asking for the user's email, there would be a button bringing them to the prompt for four random words. Additionally, there would also be a link sending the user to the email registration page. After successfully entering the four random words, the user would be sent to the dashboard upon clicking the *Login* button. This behavior could also be seen with the registration process. The only difference would be that there would now be a link sending the user to the login page.

In the dashboard, there are three general action buttons. For each SALN form, there are four action buttons that will affect only that SALN form.

Going into the digital SALN form, navigation buttons could be seen in the bottom right corner. The goal is to go forward or backward in the sequence of form pages. Assets and Liabilities pages have similar patterns. In general, assets would have an *Add Asset* button making a modal pop up for Asset input. For each asset, there would be two action buttons affecting only that asset.

Aesthetic Design - Layout and Graphics

This is the color scheme to be used:



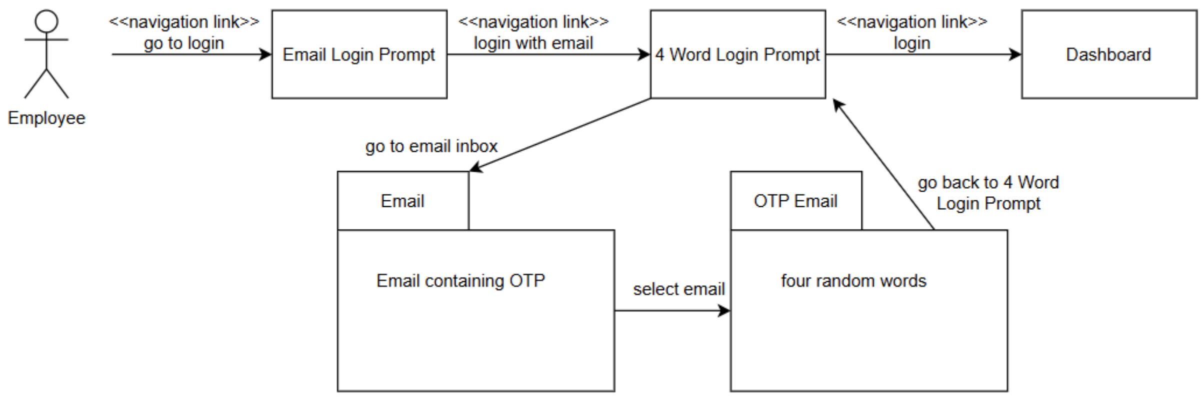
Name	Value
Modal Background	⬜ F9FAFB
Page Background	⬜ E5E5E0
Delete Button + Modal	🟥 B22222
Blue Button	🟦 1F3B4D
Modal Border + Green Button	🟩 0D9488
Black Text	⬛ 000000
White Text	⬜ FFFFFF

Height will just be flexible depending on how much content is inside these elements. For relative width sizes:

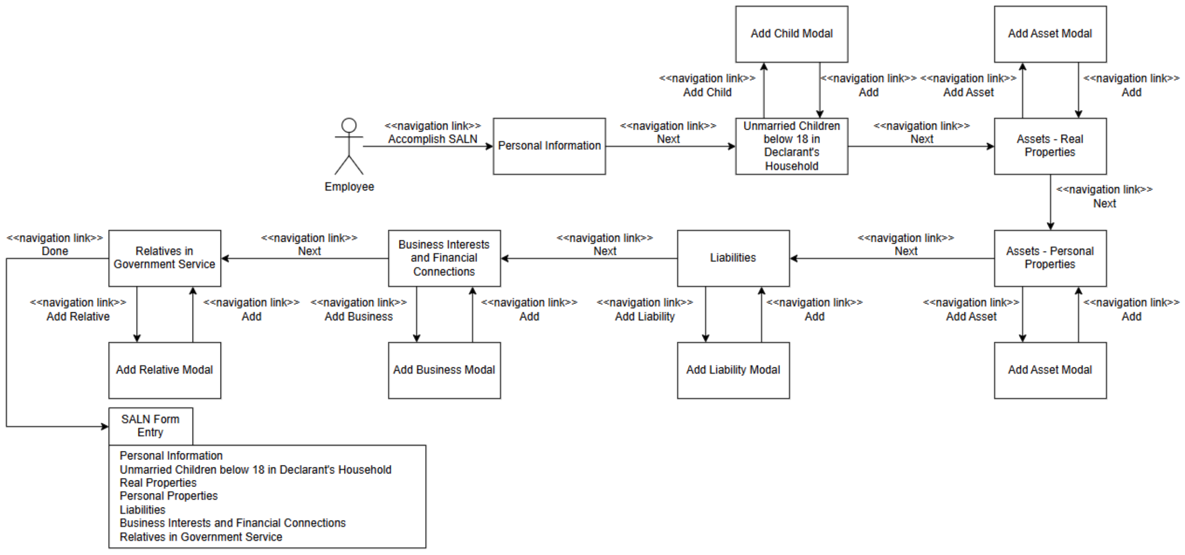
ELEMENT	SCREEN WIDTH %
Login Modal	25%
General Action Button	14%
List Modal	80%
Entity Specific Action Button	6%
General Popup Modal	45%
Asset/Liabilities Popup Modal	55%

Navigation Design

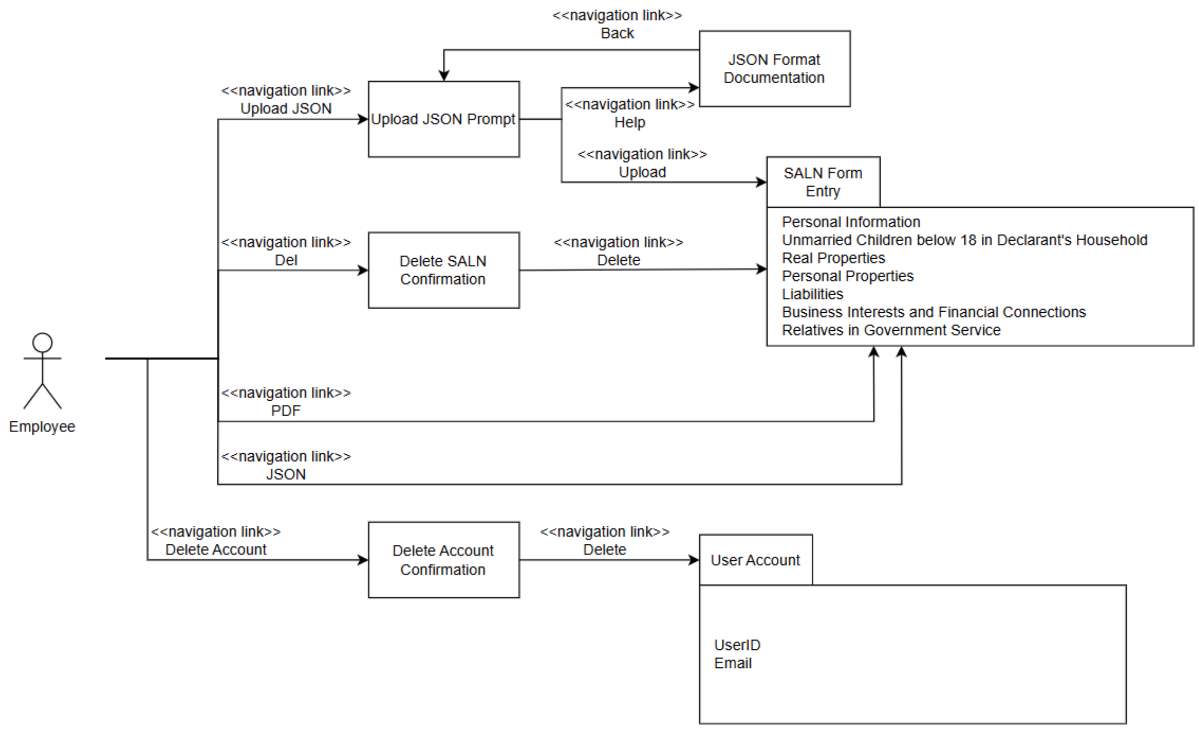
One common scenario is the login process. Registration also mirrors this navigation scheme. Upon opening the app, the user is sent to the login page prompting for the user's email. Then, the user inputs his email. Let's say it is around 20 characters long. Then he clicks the *Login With Email* button, sending him to a 4-Word Login Prompt. He then goes to his email inbox, then opens the OTP email that contains the four randomly generated words. Then he goes back to the prompt and then enters the 4-Word OTP. Let's say it is around 20 characters long. Lastly he clicks the *Login* button and gets sent to the dashboard. In this case, there are 6 mouse clicks and around 40 keystrokes, 6/46 of which are used for navigation while 40/46 are used for data entry.



Another common scenario will be accomplishing a SALN form. Consider this scenario where the employee has no spouse, no children, no family working in the government, and only one real property. We will also start analysis from the dashboard. The user clicks the *Accomplish SALN* button, which will send him to the first page of the digital SALN Form to enter personal information. Suppose that takes around 145 characters given the specifications of the scenario. Then, he will click *Next* which will send him to the page asking for unmarried children. Since he has none, he will just click *Next* and get sent to the real properties page. In this, he has one real property. So, he has to click the *Add Asset* button to put in information about that real property. Suppose this real property takes around 93 characters for all its fields. Then he would click *Add* once finished with the real property, and then *Next* to go to the next page of the SALN form. Then for the succeeding pages, since he has no entries for those, he will just click the *Next* button for them until he is *Done*. There are 238 keystrokes and there are 10 mouse clicks, 238 out of 148 are used for data entry, and 10 are used for navigation. It is also worth noting that editing a SALN form would have a similar navigation scheme to this with potentially less keystrokes since the user would be changing entries of less fields.



The next functions of the dashboard are quite straightforward. Exporting only needs one navigational click. Deleting an account or a SALN form will need two navigational clicks, one for calling the function and one for confirmation. Uploading the JSON prompt needs two navigational clicks, one going into the prompt and one confirming the upload.



<<navigation link>> Back
JSON Format Documentation
<<navigation link>> Upload JSON
Upload JSON Prompt
<<navigation link>> Help
<<navigation link>> Upload
SALN Form Entry
Personal Information
Unmarried Children below 18 in Declarant's Household
Real Properties
Personal Properties
Liabilities
Business Interests and Financial Connections
Relatives in Government Service
<<navigation link>> Del
Delete SALN Confirmation
<<navigation link>> Delete
Employee
<<navigation link>> PDF
<<navigation link>> JSON
<<navigation link>> Delete Account
Delete Account Confirmation
<<navigation link>> Delete
User Account
UserID
Email