# SALN Digitalization App

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

LAMig

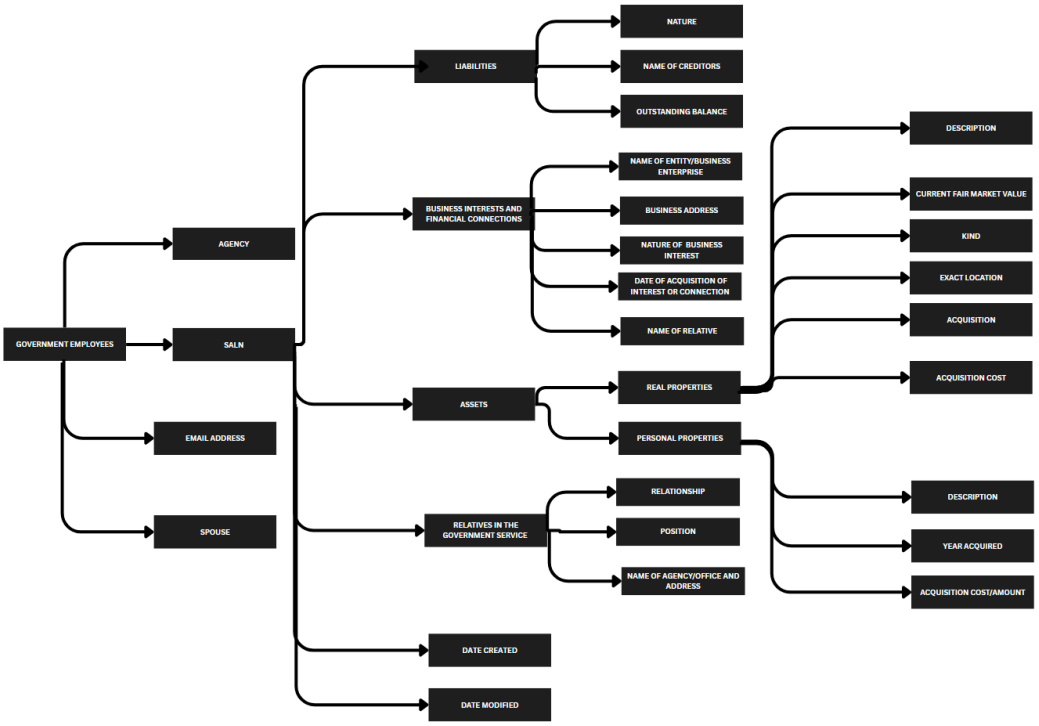
Client: Associate Prof. Rommel Feria

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.

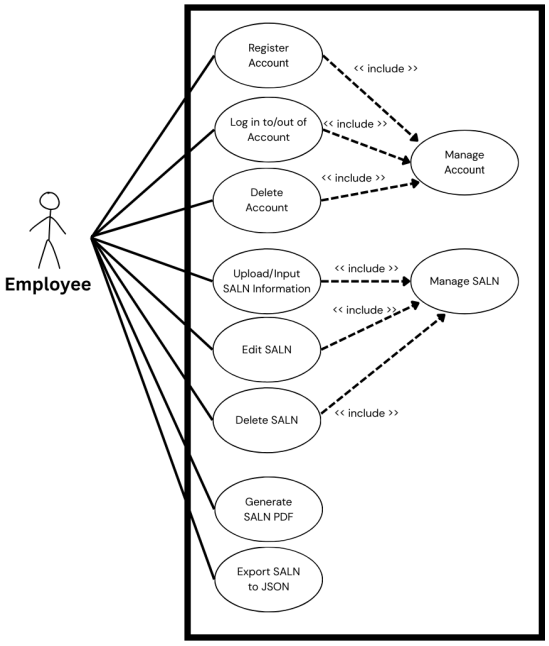
## Interaction Model

*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.

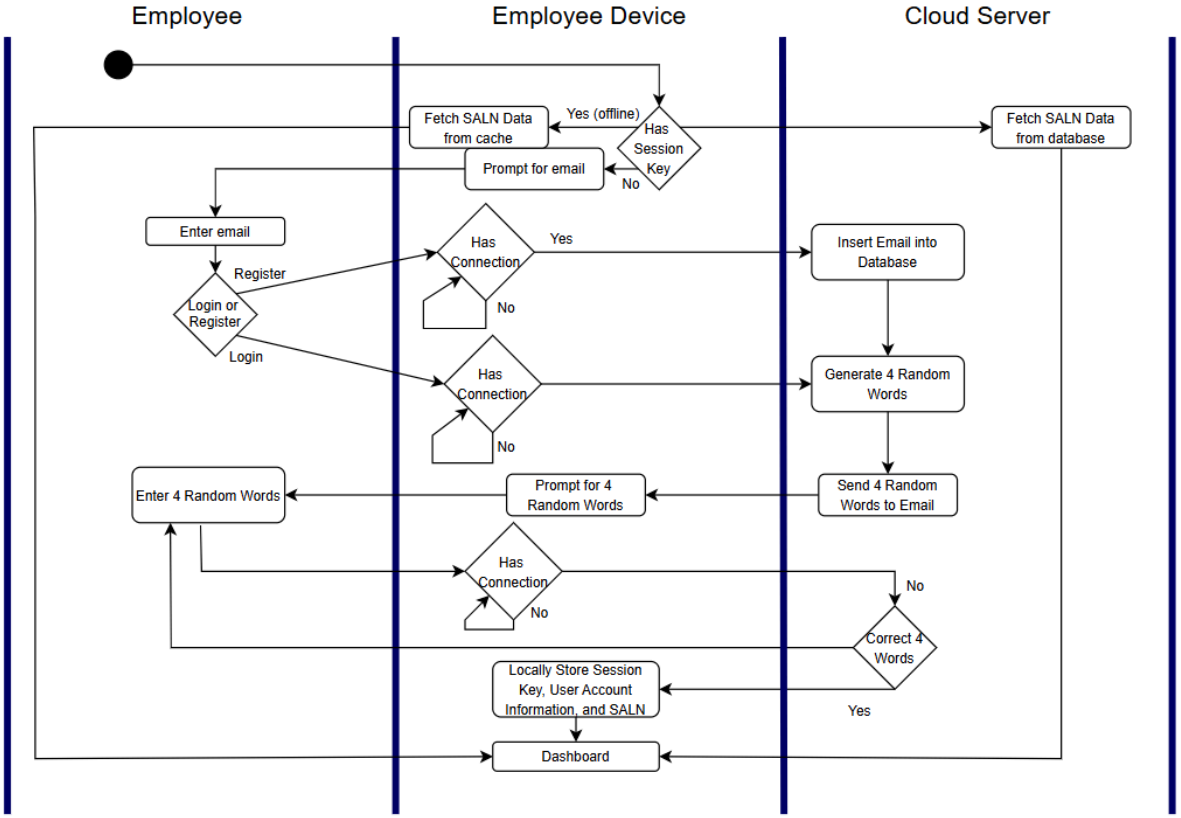
The use cases can be divided into account management, SALN management, and SALN exportation.

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*

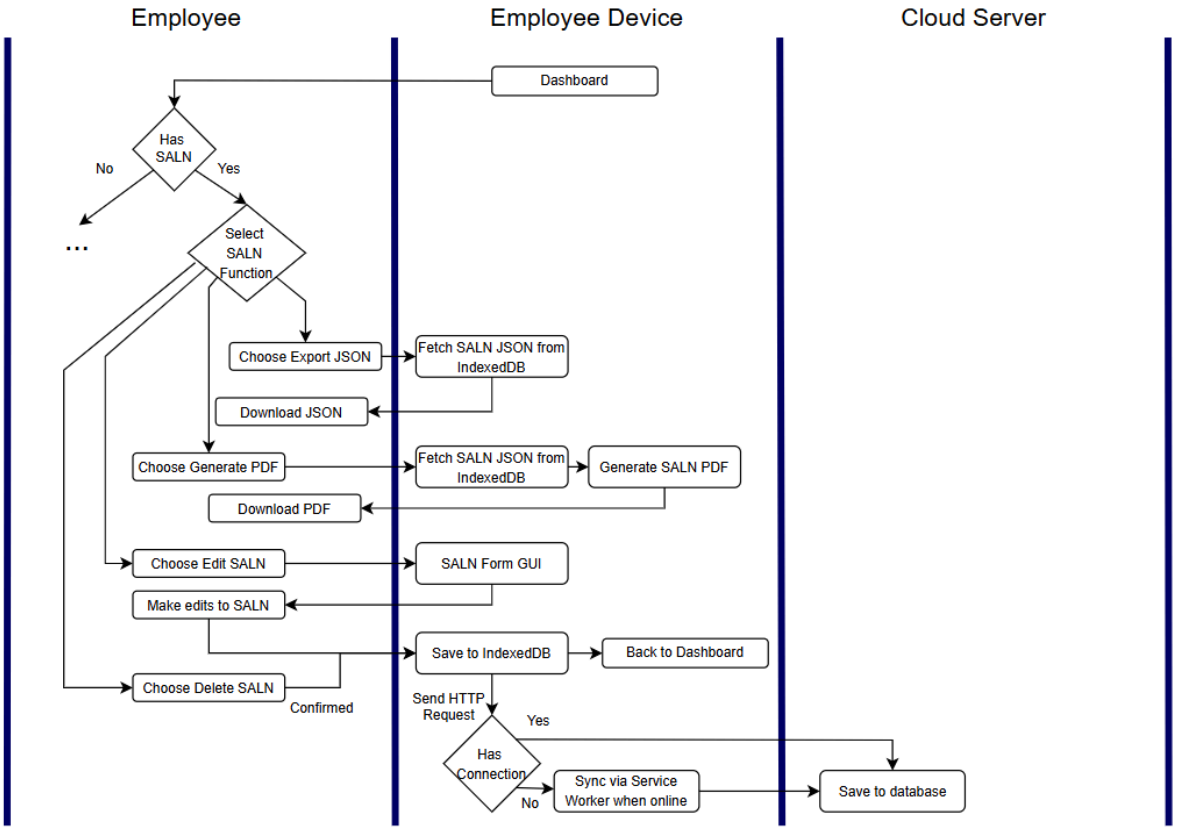



The first part of the swimlane diagram of the whole system is the login. The SALN app first checks if the user has a session key stored in IndexedDB. If so, he will immediately be logged in and be sent to the dashboard. If he was automatically logged in offline, he will use the latest SALN data that 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 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 would select if he plans 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.

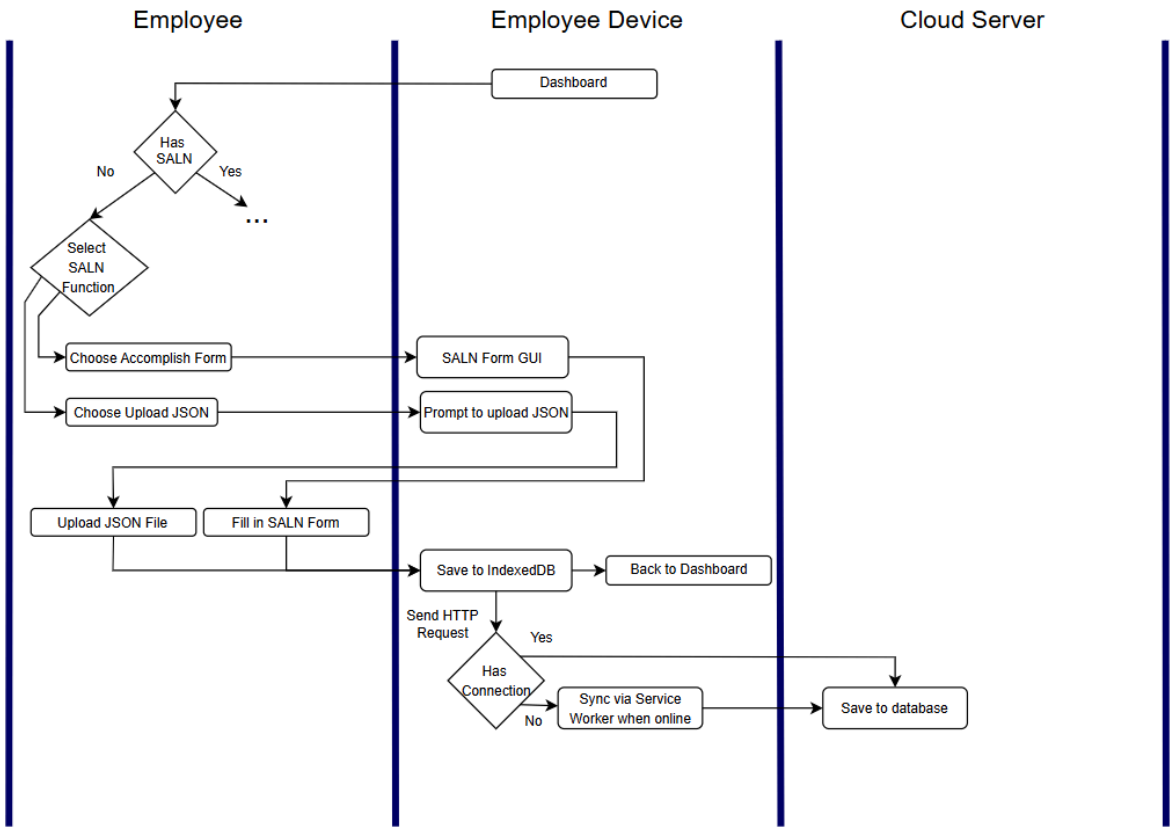



The second swimlane diagram shows use cases in which we are exporting SALN data from the app. If the user already has a SALN form, he has the option to export the SALN form data as a JSON file, the option to generate and download a PDF file of the SALN form data, the option to edit his SALN form data, and the option to delete his SALN form data.

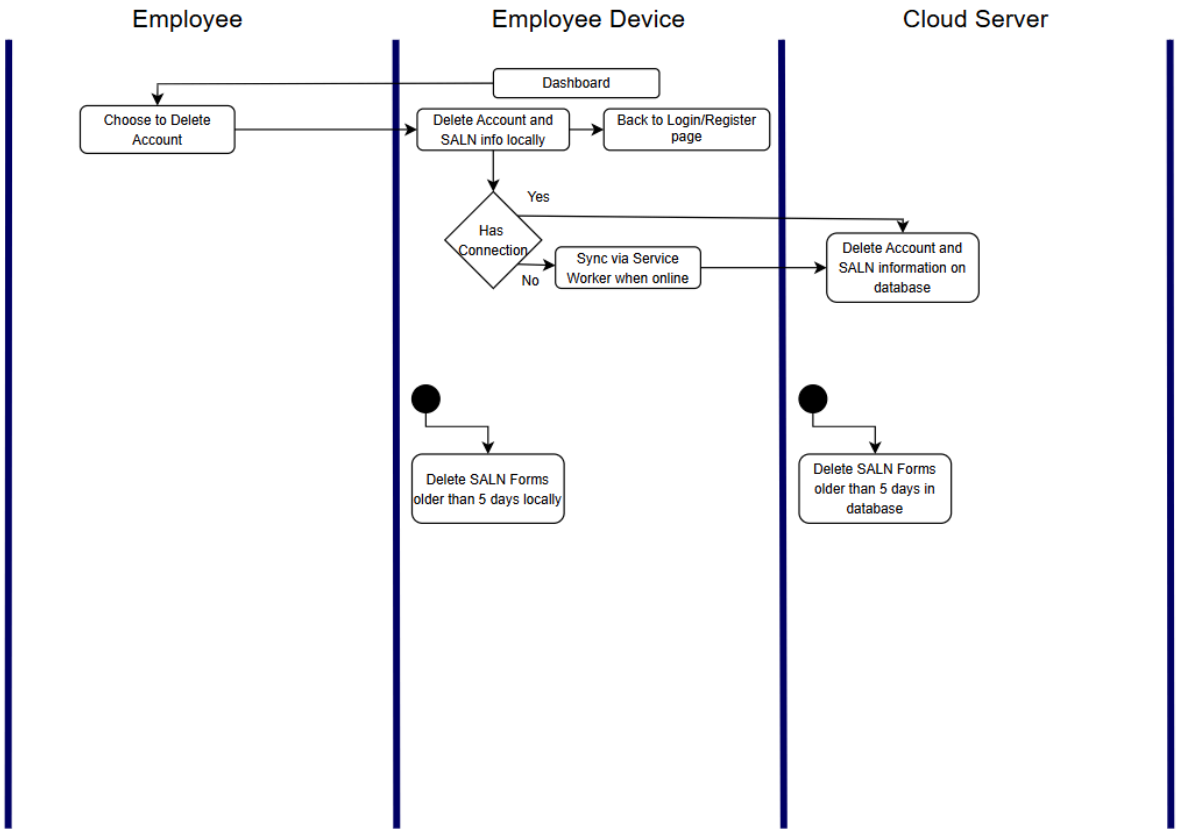
Upon looking at the Export JSON and Generate PDF options, fetching the SALN data is done from IndexedDB. This means that exporting the user's SALN data could be done completely offline.

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.



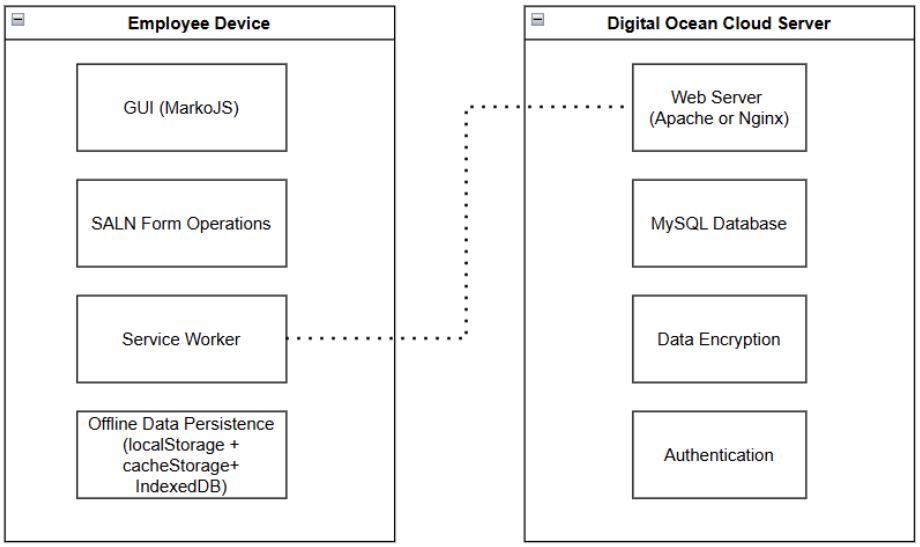

The third swimlane diagram contains SALN functions about uploading new SALN data. If the user doesn't have a filled SALN form yet, the dashboard will give him the options to accomplish the form via the app, or to upload SALN form data in JSON format. Upon clicking either option, the app will send the 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.




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




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 or Nginx 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 by Laravel's built-in encryption and decryption services. 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

Describe the architectural styles used in your web app (Section 13.7). Show how your design addresses the technical attributes of the "quality requirement tree" (see Figure 13.1). Document the architectural decisions made for your project. For more details, see sample Architectural Decisions artefact and chapter 9 of our textbook.

### Content (Data) Design

*Class Diagrams*

Show all classes (see Section 6.5.1 as a guide for identifying the right classes) and their associations and dependencies (Section 6.5.5). Indicate visibilities and data types of attributes (Section 6.5.2) and operations (Section 6.5.3). Use Figure A1.2 or Figure 13.3 as a guide.

*Entity-Relationship Diagrams*

Define the data objects, files, tables, or databases that are processed within the system. Indicate attributes and relationships. Use Figure 6.8 as a guide. Include the description of the file format and/or database schema (format of database tables)

*Data Flow Diagrams*

Define the data objects and transformations within the system. Provide DFD levels 0-2. Use Figures 7.1-7.3 as a guide.

### Component-Level Design

Specify the system components – define the data structures of all local data objects and algorithmic/procedural detail of all processing within a component (use the sub-questions below as a guide). Include interfaces (see Figure 8.5) and dependencies (e.g., sharing, flow, constrained - see Section 9.5.2). Will you be using any COTS components? (See chapter 10). Refer to Chapter 8 and Chapter 10 as you discuss design concepts that you have implemented.

*Algorithms*

Describe complex functional/processing details of the web app through a UML activity diagram. For example, when computing a motion trajectory for an animate figure in a game, you may use some numerical or computer-graphics algorithms. Will the path coordinates be pre-computed and stored in a look-up table or will they be computed using a spline interpolation algorithm? Or, when assessing stock market movements, you may be using statistical algorithms.

*Data Structures*

Does your system use any complex data structures, such as arrays, linked lists, hash tables, or trees? Specify your criteria for deciding what data structure to use, e.g., performance vs. flexibility.

**Interface Elements / Usability Design**

If your system prints some forms or generates periodic reports, this is also considered part of the user interface and the format of forms/reports must be specified in this section. See section 11.5.2. The guide questions in section 11.3.2 and 11.3.3 might be useful. Show how your interface design addresses the issues enumerated in Section 11.4.3.

*Interface Design - Menus and Links*

Describe/provide the layout and form of navigation mechanisms.

*Aesthetic Design - Layout and Graphics*

Describe/provide the template for the layout and graphic design of the web app (i.e., color schemes, styles, etc.). Note the guidelines given in Section 13.5.

*Navigation Design*

See guide questions in Section 7.5.8 to design the mechanics of navigation in your web app. Select several typical usage scenarios and, as you walk through the flow of events, count and report the number of mouse clicks and/or keystrokes that are needed to accomplish the task. What fraction of these goes to user-interface navigation vs. clerical data entry? Create navigation semantic units to describe your user-friendly navigation design (see Figure 13.9).