Active Reading Assistant
SRS/SDS/Project Plan

Spring Term 2024 - CS 422 Anthony Hornof

Ethan Hyde (EH)
John Hooft Toomey (JHT)
John O'Donnell (JO)
Ben Bushey (BB)

Table of Contents

1. SDS Revision History **3**
2. System Overview **3**
3. Software Architecture **4**
3.1 User Side Modules: 4
3.2 Server Infrastructure Modules: 4
3.3 Module Interaction Description 4
3.4 Architecture Design Rationale 7
4. Software Modules **7**
4.1 Interactive Learning Hub Module (ILH Module) 7
4.1.1 Role and Primary Function 7
4.1.2 Interface Specification 7
4.1.3 Static and Dynamic Models 8
4.1.4 Design Rationale 9
4.1.5 Alternative Designs 9
4.2 Note Taking Module 10
4.2.1 Role and Primary Function 10
4.2.2 Interface Specification 10
4.2.3 Static and Dynamic Models 10
4.2.4 Design Rationale 11
4.2.5 Alternative Designs 12
4.3 PDF Selector Module 12
4.3.1 Role and Primary Function 12
4.3.2 Interface Specification 12
4.3.3 Static and Dynamic Models 12
4.3.4 Design Rationale 13
4.3.5 Alternative Designs 13
4.4 PDF Viewer Module 14
4.4.1 Role and Primary Function 14
4.4.2 Interface Specification 14

4.4.3 Static and Dynamic Models....14
4.4.4 Design Rationale.... 15
4.4.5 Alternative Designs.... 15
4.5 User Account Login Module....15
4.5.1 Role and Primary Function.... 15
4.5.2 Interface Specification....16
4.5.3 Static and Dynamic Models.... 16
4.5.4 Design Rationale....17
4.5.5 Alternative Designs.... 17
4.6 ARA Note Database Module....17
4.6.1 Role and Primary Function.... 17
4.6.2 Interface Specification....17
4.6.3 Static and Dynamic Models.... 17
4.6.4 Design Rationale....18
4.6.5 Alternative Designs.... 19
4.7 Note Database API Module.... 19
4.7.1. Role and Primary Function.... 19
4.7.2. Interface Specification....19
4.7.3. Static and Dynamic Models.... 20
4.7.4 Design Rationale....22
4.7.5 Alternative Designs....22
5. Dynamic Models of Operational Scenarios (Use Cases)....23
5. Project Plan.... 24
5.1. Task Assignments and Management Plan....24
5.1.1. Initial Decisions....24
5.1.2. Communication.... 24
5.1.3. Monitoring and Reporting.... 25
5.2. Build Plan....25
5.2.1. Build Plan Sequence of Steps....25
5.2.2 - Build Plan Rationale.... 26
5.3 - Meeting Timeline....26
1. Initial meeting (4/11/24)....26
2. Meeting 2 (4/15/24)....26
3. Meeting 3 (4/17/24)....27
4. Meeting 4 (4/22/24)....27
5. Meeting 5 (4/24/24)....27
6. Meeting 6 (4/28/24)....27
7. Final meeting (4/29/24)....27
5.4. Project Plan Revision History....28
5.4.1 - Initial Project Plan....28
5.4.2 - Second Project Plan....30

6. Software Requirements Specification..... 31
1. SRS Revision History..... 31
2. The Concept of Operations (ConOps)..... 31
2.1. Current System or Situation..... 32
2.2. Operational Features of the Proposed System..... 32
2.3. User Classes..... 33
2.4. Modes of Operation..... 33
2.5. Operational Scenarios (Also Known as “Use Cases”)..... 33
3. Specific Requirements..... 34
SQ3R Assistance (“Scaffolding”)..... 34
Login..... 34
Target Platform..... 34
Data Storage..... 35
18. References..... 36
19. Acknowledgements..... 36

1. SDS Revision History

Date	Author	Description
04-29-24	BB	Added role and primary function sections under section 4
04-29-24	BB	Added Interface Specification sections under section 4
04-29-24	JHT	Added Static and Dynamic models for Software Modules.
04-28-24	JHT	Completed System Overview and Software Architecture sections.
04-28-24	JHT	Updated SDS for Final Submission.
04-12-24	JHT	Removed repetitive Server Logic Component.
04-12-24	JHT	Improved descriptiveness in Server Infrastructure Components.
04-12-24	EH	Updated TOC and provided SRS modifications note.
04-12-24	ALL	Created initial document.

2. System Overview

The ARA software architecture is organized into two main categories: User Side Modules and Server Infrastructure Modules. Users begin by logging in through the User Account Login Module, which connects to the ARA Note Database via the MongoDB Server Connection Component. After selecting a PDF, the Interactive Learning Hub (ILH) Module manages the interface, utilizing the PDF Viewer Module for display and the Note-Taking Module for notes. The ILH User Interaction Component enables user control and navigation within the system.

Throughout, the Note Database API Module ensures seamless interaction with the database. This architecture facilitates efficient user engagement and data management within the ARA software.

3. Software Architecture

3.1 User Side Modules:

- Interactive Learning Hub Module:
    - ILH Display Component: Responsible for presenting the Active-Reading Assistant (ARA) interface and enabling user interaction with Tkinter.
        - ILH User Interaction Component: A set of buttons and menus that allows the user to navigate and control their ARA experience. Lets the user navigate the PDF, Logout, Save, Hide PDF, and navigate back to PDF selection.
    - Note-Taking Module: A set of text input boxes that manages the creation and display of hierarchical notes.
        - Chapter Title Text Box Component: Area for users to enter Chapter Title.
        - Section Headings and Notes Text Box Component: Organized under chapter titles for easy navigation, use has the ability to create new section notes for chapter. Can be expanded or collapsed.
    - Prompt System Component: Offers SQ3R method guidance, toggleable visibility.
    - PDF Viewer Module: Renders the PDF selected by the user.
- PDF Selector Module: Allows the user to select what PDF they want to load.
- User Account Login Module: Allows user login to desired account.

3.2 Server Infrastructure Modules:

- ARA Note Database Module:
    - MongoDB Server Connection Component: Establishes connection to the MongoDB server and handles connection errors so ARA can still be used without ARA Note Database connection.
    - ARA Note Data Management Component: Manages the storage, and retrieval of user notes through the creation of a specialized Data Structure designed to store user notes data, and user session information.
- Note Database API Module:
    - Note Services Component: Handles note-related interactions such as Save, Retrieve, and Delete notes.

3.3 Module Interaction Description

In the ARA software, the modules are grouped into two main categories, Server Infrastructure and User Side Modules. When a user initiates the program, they engage with the User Account Login Module to access their account. This module interfaces with the MongoDB Server Connection Component to establish a connection to the ARA Note Database. When the user selects a non-admin account to login to, users interact with the PDF Selector Module to choose a PDF for loading. The PDF Selector Module passes the user's account name and PDF selection to the Interactive Learning Hub (ILH) Module, which utilizes the Note Database API Module to retrieve any previously stored notes for the PDF. The Note Database Module uses the ARA Note Data Management Component to deliver the data in a structured format usable by the ILH.

The ILH creates an interactive window for the user. The PDF Viewer Module allows the ILH Module to display the correct highlighted PDF, while the Note-Taking Module displays the notes associated with the PDF. The ILH User Interaction Component is a subsystem of the ILH Display Component, and is a set of buttons and menus that allow the user to navigate through the ARA. This component encompasses various functionalities, including navigating through the PDF, logging out, saving notes, hiding the PDF, and returning to PDF selection. Furthermore, it interfaces with the ARA Note Database Module, Note Database API Module, PDF Selector Module, and User Account Login Module to enable seamless interaction between these components. Thus, the system's architecture ensures cohesive interaction among its modules, facilitating efficient user engagement and data management within the ARA software.

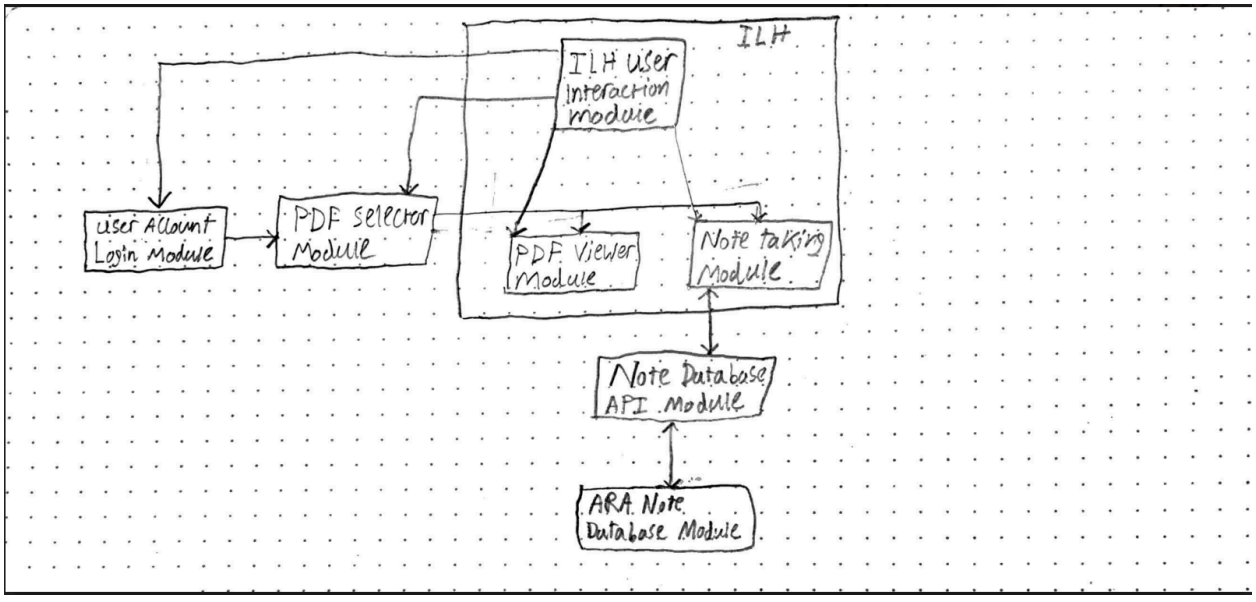

Figure 3.3.1: ARA system architecture model at the Module level.

Previous designs:

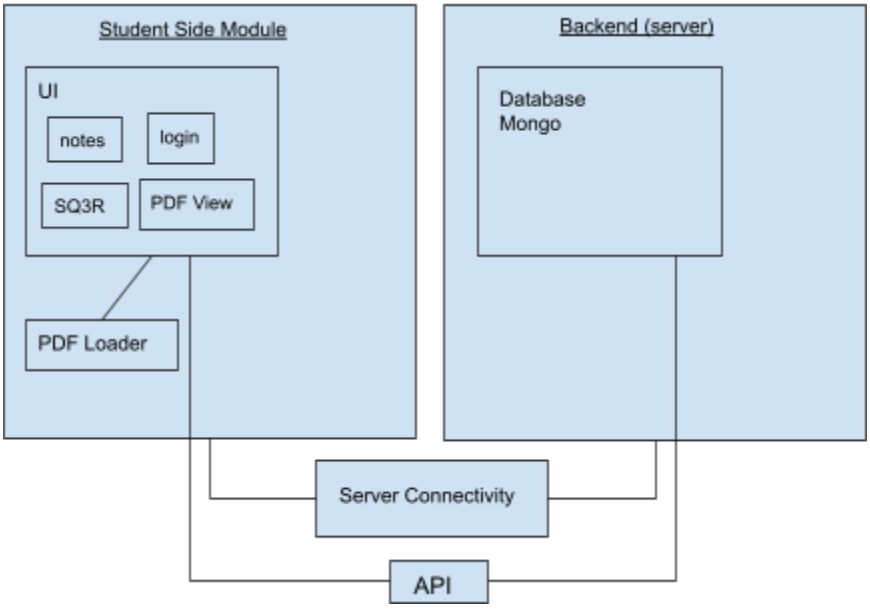

Figure 3.3.2: First draft of our System Model.

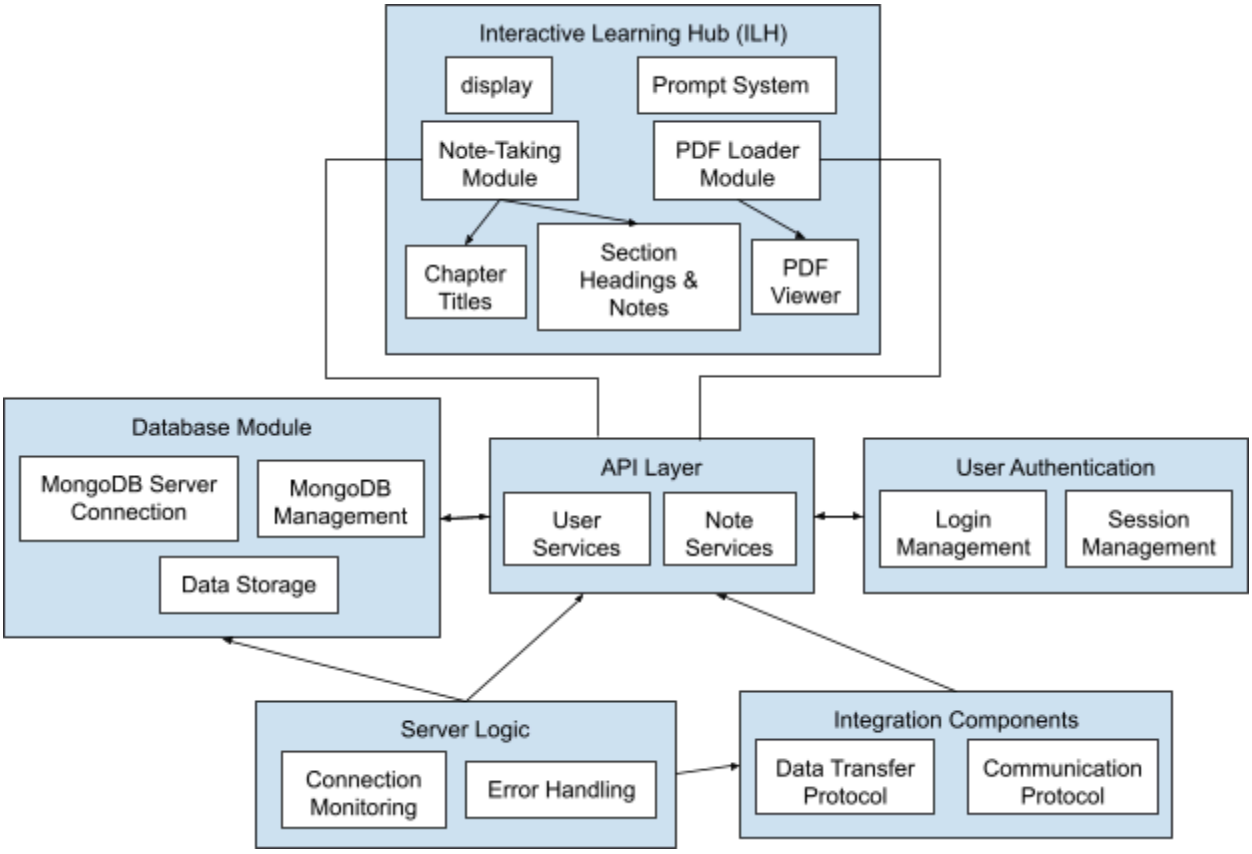

Figure 3.3.3: Second draft of our System Model.

3.4 Architecture Design Rationale

The architectural design of the Active-Reading Assistant (ARA) software was meticulously crafted to provide users with a seamless and intuitive learning experience while ensuring robustness and flexibility in handling data interactions. The decision to employ a modular architecture, categorized into User Side Modules and Server Infrastructure Modules, stems from the need to streamline functionality and optimize system performance.

We also chose to use a system containing modules with nested components, computationally independent of one another but working together to complete the software as a whole. By thoughtful designing the components functionality and grouping them together to form elegant modules, we were able to design a system that functions efficiently and its structure is easy to understand. You can see this emphasis on efficient design in the progression of our System Model, where the final version (Figure 3.3.1) looks smaller and less interconnected than the second draft version (Figure 3.3.3). The final version of the System Model conveys the general structure of the software modules without complicating the model with every module's sub-component.

4. Software Modules

4.1 Interactive Learning Hub Module (ILH Module)

4.1.1 Role and Primary Function

The ILH Module is the main user interface of the educational application, designed to control interactive learning. Its primary purpose is to offer logical integration of note-taking, viewing PDFs, and instructional prompts into a single interactive setting. The module offers study-related PDF documents, lets users annotate and organize notes connected to document parts, and uses the SQ3R approach to guide users through the course material. By processing user actions, from navigating documents to saving notes, the ILH Module acts as the interactive backbone of the learning experience, connecting the user's Tkinter window UI interactions with User Notes database operations to ensure continuity and personalization of the learning process.

4.1.2 Interface Specification

- Database Interface: Utilizes the Database class to collect user notes.
- Interacts with the MongoDB User Notes Database for data retrieval and storage operations.
- Authentication Interface: links note-taking activities to the authenticated user making it so there is user-specific data.

- File System Interface: Works with the operating system's file handling for the temporary creation and deletion of files necessary in PDF rendering, through the use of os library.

4.1.3 Static and Dynamic Models

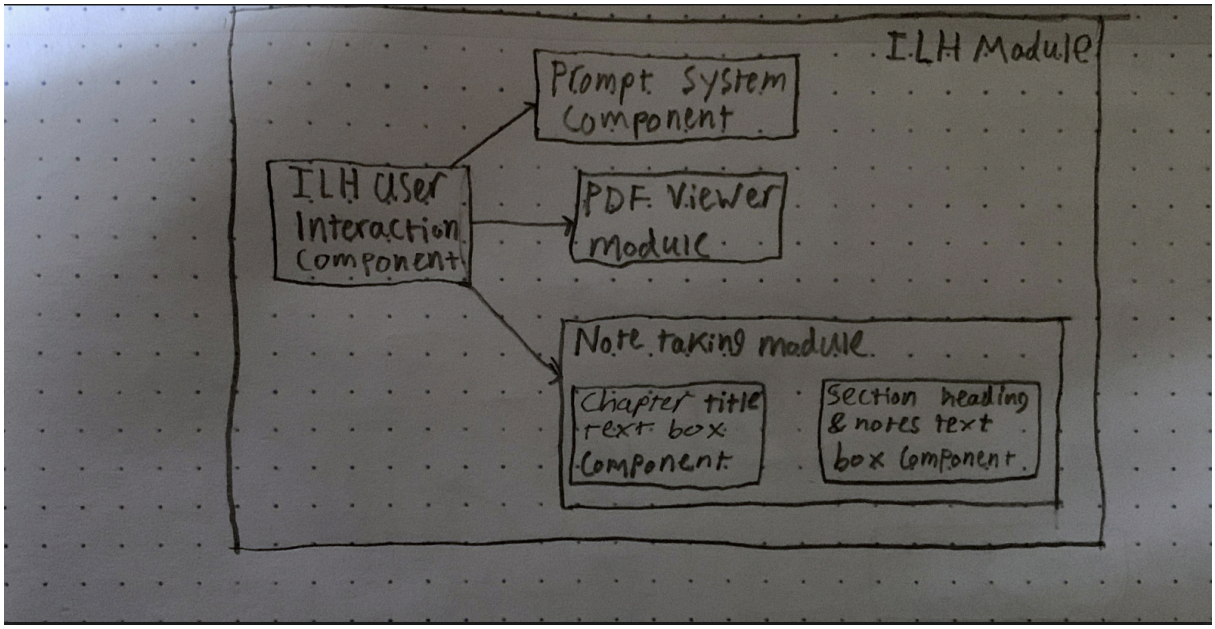

Figure 4.1.3.1: Static Model of Interactive Learning Hub Module.

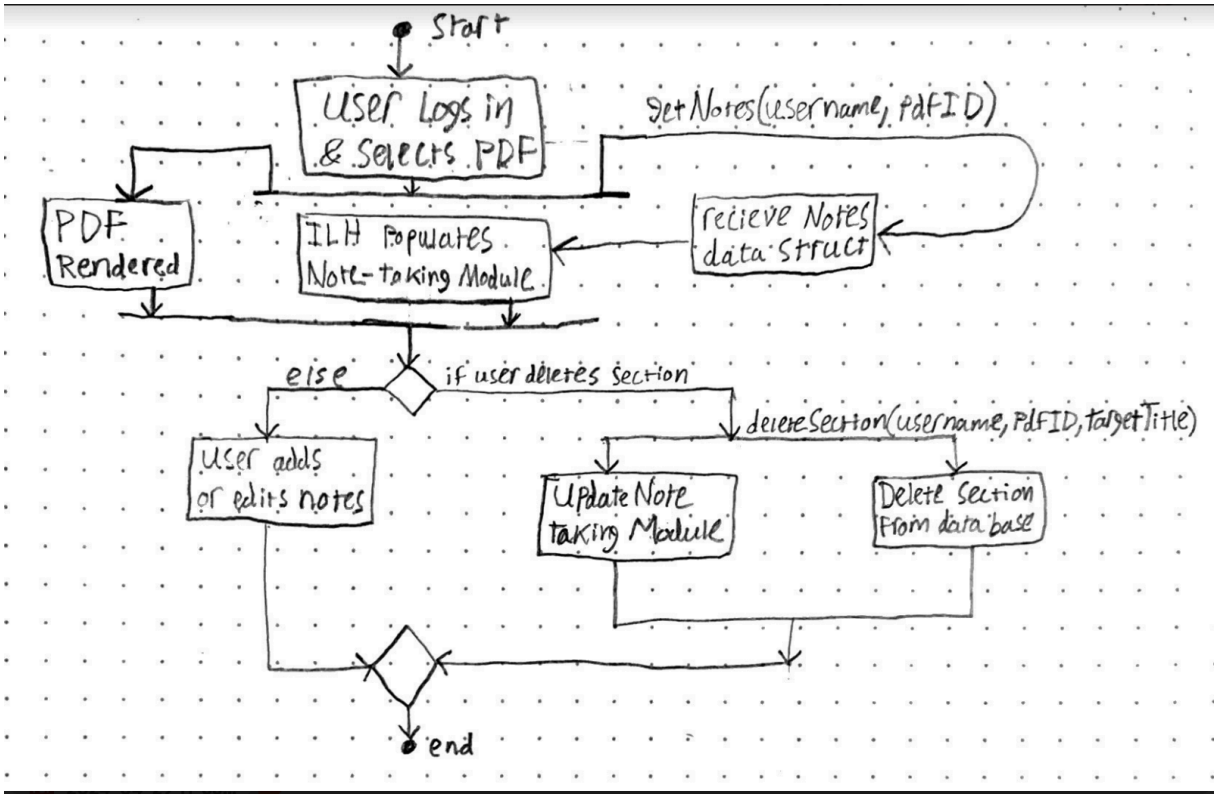

Figure 4.1.3.2: Dynamic Model of Interactive Learning Hub after User Login / PDF Selection.

4.1.4 Design Rationale

The ILH was designed to be user friendly and modular. The ILH interface module was designed to run with or without a stable connection to the database. This allows the user to make changes locally that are not saved within the MongoDB database. The ILH user interface was broken down into smaller, more cohesive modules that make the software easier to test, debug and use. The Note-Taking module, PDF display module, and Prompt system modules were all created separately and displayed in specific locations within the ILH user interface to make the program more user-friendly.

4.1.5 Alternative Designs

Figure 4.1.5.1 below shows the initial design for the ILH module. Since then we decided to adjust the PDF related modules to make them more accurately represent their function. We kept the PDF Viewer module in the ILH as it is one of its essential functions, but we renamed the PDF loader module to the PDF Selector Module and moved it outside of the ILH. We did this as we felt that the PDF Selector Module achieved a separate goal from the ILH and should be represented as such.

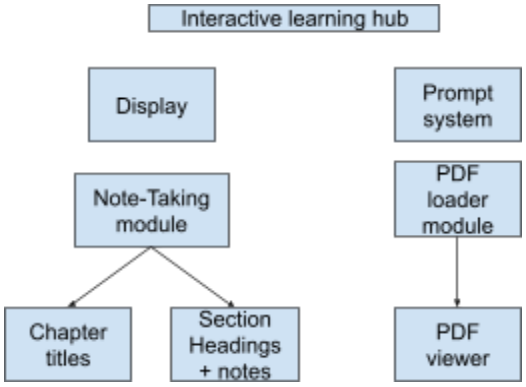

Figure 4.1.5.1: First draft of ILH model.

4.2 Note Taking Module

4.2.1 Role and Primary Function

The Note Taking Module is a submodule within the Interactive learning hub that provides a system where the user can create and organize personal notes. This is done by retrieving user input from the textboxes that are displayed in ILH. The notes that are created are then stored in relation to the user's section title(s) and chapter title. This module helps support users by giving them options to annotate and reflect on their material in a more organizational way when viewing different PDFs.

4.2.2 Interface Specification

- Interaction with PDF Viewer: Allows users to take and manage their notes for the PDF document of their choice.
- Uses PDF identification for save/load methods.
- Database Operations: Directly interfaces with the Database class, using methods: updateUserNotes() and deleteSection(). It allows the notes entered by the user to be stored in the MongoDB database and retrieved or updated.
- User Interface Functions: Uses TKinter entry and text widgets within the NoteSection class for input fields.
- Buttons “Add Section” or “Delete” are used to help organize/edit user notes:
- Session Autosave: user notes are saved upon exit, logout, pdf selection, or window closer through event handlers that call the save_notes method.
- Prompt System Interaction: When a user adds a section or writes notes, the prompt system will call the ‘Recite’ prompt to help guide the user

4.2.3 Static and Dynamic Models

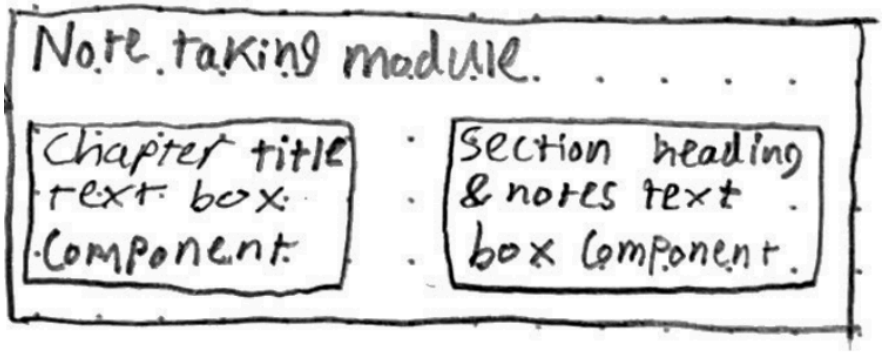

Figure 4.2.3.1: Static model of Note Taking Module showing its sub-components.

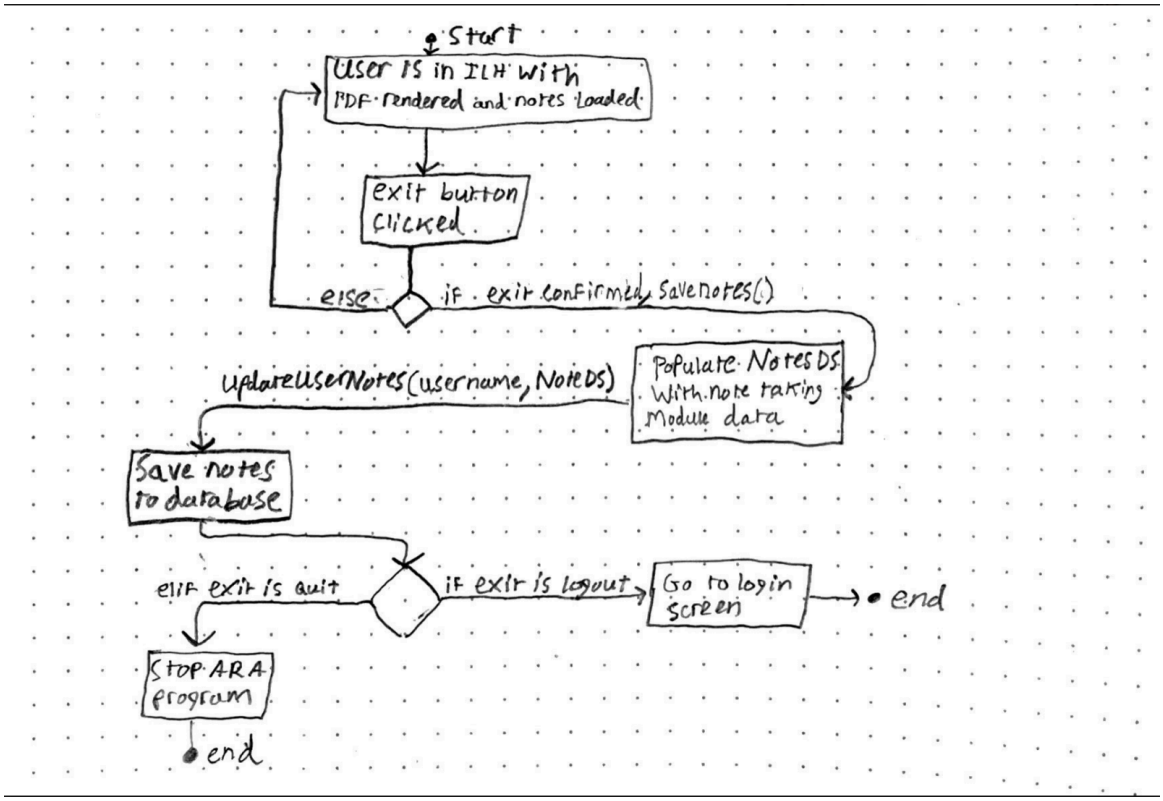

Figure 4.2.3.2: Dynamic Model of User exiting program and Note Taking Module automatically saving notes using Note Database API Module.

4.2.4 Design Rationale

We chose to use this design to handle user inputs in a way that formats the notes in a way that is easy to read, and resizable so the user can only display sections of their choosing. It promotes organization of their notes and would help organize their thoughts within the reference materials. This also helps to support the SQ3R active learning practices.

4.2.5 Alternative Designs

Another design that we considered was having one big text box, and using markup characters to separate the sections from the title and body of the notes. We decided to not pursue this because of the complexity added for searching strings of characters.

4.3 PDF Selector Module

4.3.1 Role and Primary Function

The PDF Selector Module provides the functionality for users to select and load PDF documents into the ILH to study. The primary function of this module is to present the available PDFs to the user, handle the selection process, and ensure the chosen document is correctly loaded and ready to be interacted with the PDF Viewer Module.

4.3.2 Interface Specification

- User Account Login Module: The user selects from four predefined options (User1, User2, User3, Admin) which then calls the PDF Selector Module excluding: 'Admin'
- PDF Selection and Loading: a pop-up window is created that prompts the user with a list of buttons assigned to unique PDFs using a TKinter-based selection window.
- Transition: Once the user selects a PDF with these buttons, the module triggers the loading of the document by giving the file path to the PDF Viewer Module.
- Transition: the username and selected PDF ID are passed to the Interactive Learning Hub Module, linking the selected document with the user's session.

4.3.3 Static and Dynamic Models

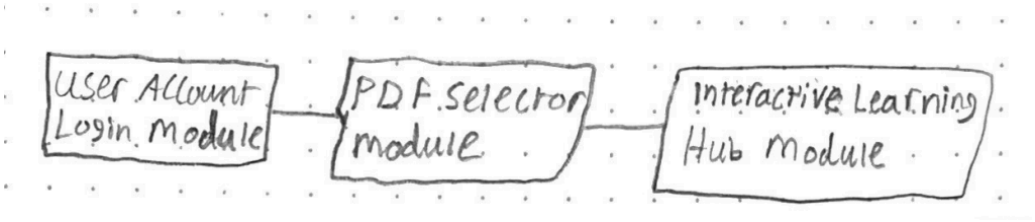

Figure 4.3.3.1: Static model of PDF Viewer Module showing its connections to other modules.

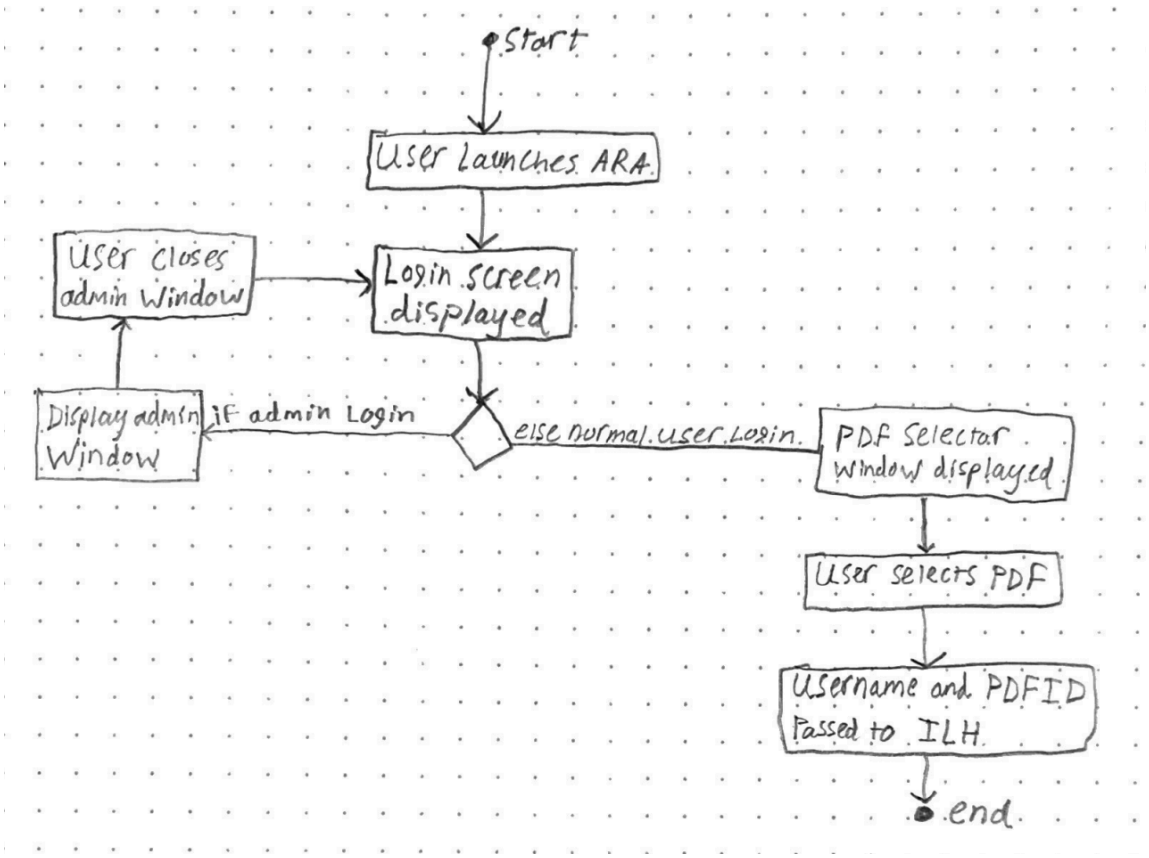

Figure 4.3.3.2: Dynamic model of flow from program start to PDF Selector Module, to Interactive Learning Hub Module

4.3.4 Design Rationale

The PDF selector module was designed to be as simple as possible to assure that we reduced coupling and made it user friendly. The module was designed to have 3 preloaded PDFs stored in the resources directory that can be chosen with only 1 click. Each PDF has a unique PDF id that is sent to the database along with the username when the PDF is selected.

4.3.5 Alternative Designs

Another design that we considered was a grid view format, displaying the thumbnails for each PDF inside of the PDF selection popup. Because of the challenges we had to face with TKinter's ability to display images, we decided to not pursue this. It was more of a streamlined approach to just display buttons with the chapter titles for the user.

4.4 PDF Viewer Module

4.4.1 Role and Primary Function

The PDF Viewer module in the ARA application is the primary interface for displaying the PDF documents that users interact with. The main function is to properly render and display a PDF onto the application's canvas. Users are then allowed to navigate through the pages and complete the SQ3R techniques in the ARA that help with their studying or review process

4.4.2 Interface Specification

- Canvas: Utilizes a TKinter canvas widget where the PDFs get displayed
- Navigation Buttons: Buttons for "Next Page" and "Previous Page" are used to help navigate through the PDF, and display the appropriate page to the ILH.
- PDF Rendering: The PyMuPDF library (fitz) is used to load and render the PDF's pages onto the canvas. This involves storing it to a temporary file and generating a pixmap.
- Scalability: The PDF is sized to the proper dimensions in order to be fit in the boundaries of the canvas.

4.4.3 Static and Dynamic Models

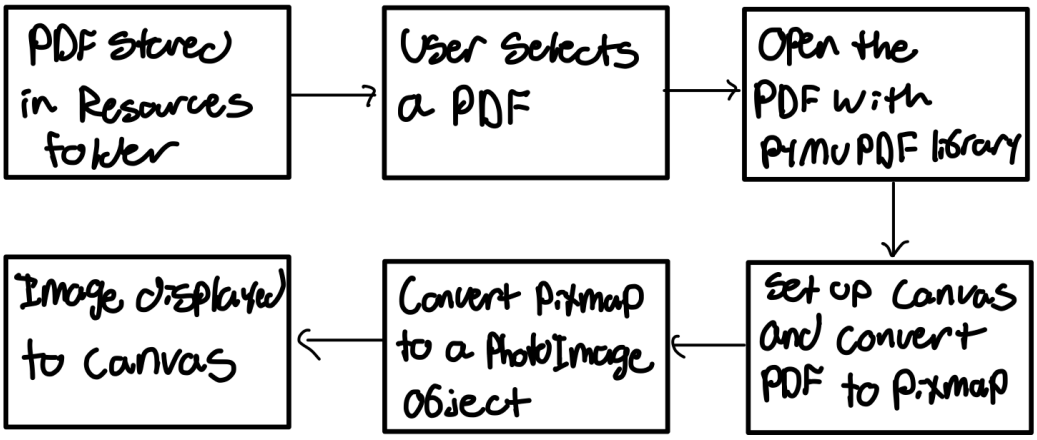

Figure 4.4.3.1 - Static model of our PDF viewing module. Shows the order of operations of how a PDF gets displayed into the ILH.

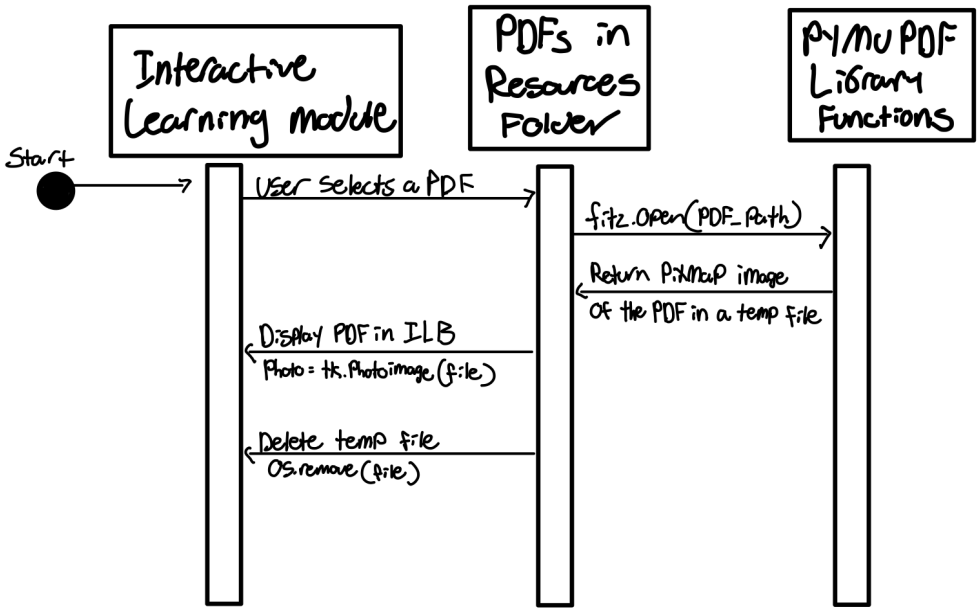

Figure 4.4.3.2 - Dynamic model of our PDF viewing module. Demonstrates the behavior of the program for loading a PDF into the ILH.

4.4.4 Design Rationale

The PDF Viewer was designed with the user in mind. By using the PyMuPDF library, we were able to render the PDFs using a library that is regularly updated, open source, and known for it's fast and reliable performance with processing PDF files. We chose to use a canvas for the PDF display to use TKinter's ability to create a user interface that is simple and reliable.

4.4.5 Alternative Designs

Other designs we considered was a separate PDF viewing window that wasn't directly part of the ARA interface. We decided to not pursue this idea because it would make the user have to deal with managing more windows than necessary.

4.5 User Account Login Module

4.5.1 Role and Primary Function

The User Account Login Module is used to provide security and personalization aspects to the Interactive Learning Hub Module. Its role is to set up the stage for the main application window by determining different users and the administrator through a login. To do that, this module gives the user different login options to select from, ensuring that users are presented with an interface that matches their work, and retains context/ changes made throughout the session.

4.5.2 Interface Specification

- Authentication Interface: Employs a set of predefined usernames (User1, User2, User3, Admin) presented through TKinter buttons on the login screen. Users initiate their session by selecting the corresponding button, bypassing the need for password entry.
- Role-Based Control: Differentiates functionality based on the selected username: the 'Admin' button routes to an administrative interface, while 'User1', ‘User 2’, and 'User3' proceed directly to the PDF Selector Module.
- Calls PDF Selector Module: The module signals the PDF Selector Module to display available PDFs to the user, passing along the username as an identification parameter for the session

4.5.3 Static and Dynamic Models

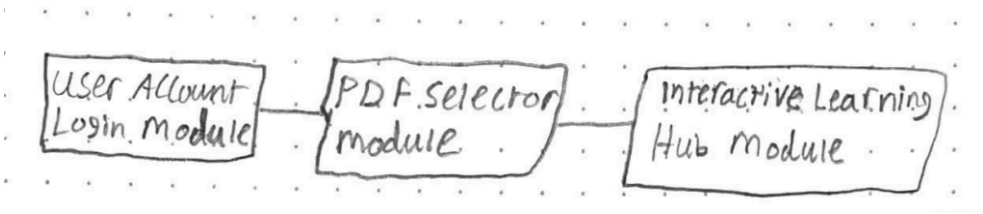

Figure 4.5.3.1: Static model of User Account Login Module showing its connections to other modules.

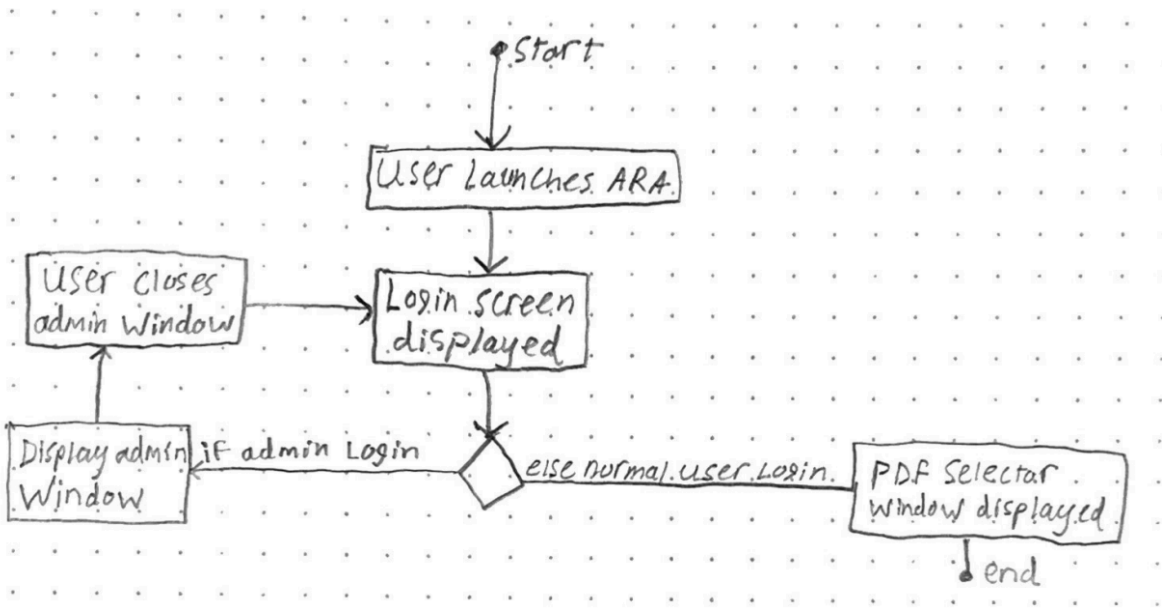

Figure 4.5.3.2: Dynamic model of flow from User Account Login Module to PDF Selector Module.

4.5.4 Design Rationale

The User Account Login module was created according to the updated software requirements specifications. The module is meant to be simple and user-friendly, allowing a “student” user to login to any of 3 predefined users with only 1 click. We decided to separate this module into 2 more simple modules for student users and admins.

4.5.5 Alternative Designs

Our original prototype included functionality to allow users to create accounts and save usernames and passwords. We strongly favored this approach before the software requirements specification was updated and required us to make the login module more user friendly, allowing the user to login with only 1 click. Having the user create accounts makes this software more secure and allows student users to create more than 3 accounts.

4.6 ARA Note Database Module

4.6.1 Role and Primary Function

The ARA Note Database Module is responsible for managing the storage, retrieval, update, and deletion of user notes within the application. Its primary function is to interface with the MongoDB database to make sure that the user notes are securely stored and that the integrity and consistency of note data are maintained across user sessions.

4.6.2 Interface Specification

- MongoDB Server Connection Component: Establishes and maintains the connection to the MongoDB server, using credentials and connection strings to ensure secure access to the database.
- ARA Note Data Management Component: CRUD operations for notes, stores user notes that include fields such as pdfID, chapterTitle, and sections.
- addSection(title, notes): method for inserting new note sections into the database.
- Note Database API Module: Offers an API for the ILH to use database operations. This includes functions to create new notes, updating existing ones, deleting, and fetching all notes for a given user and PDF.

4.6.3 Static and Dynamic Models

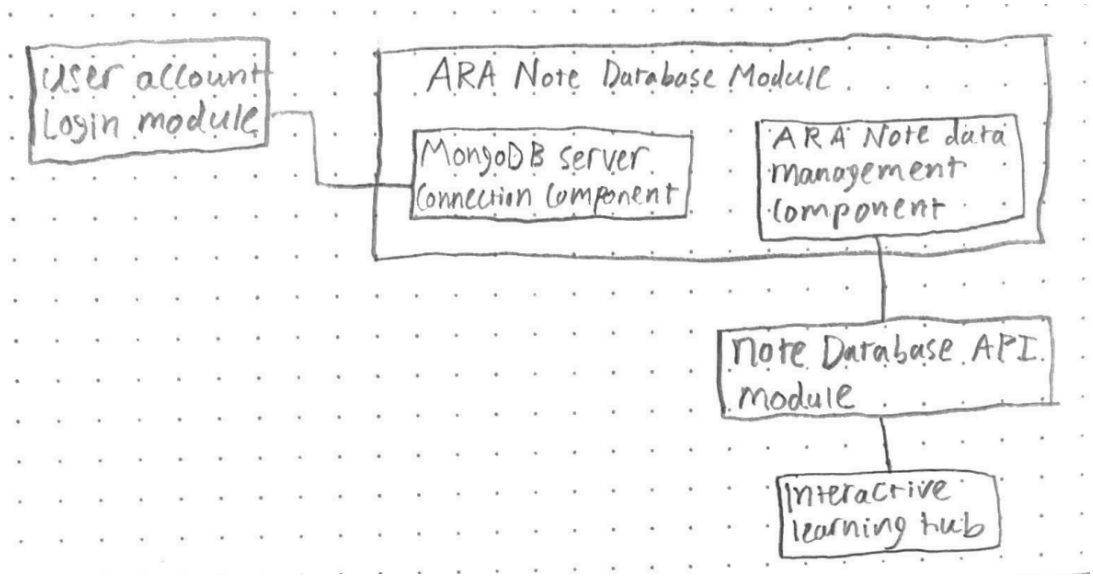

Figure 4.6.3.1: Static model of the ARA Note Database module, showing its sub-components and connections to other modules.

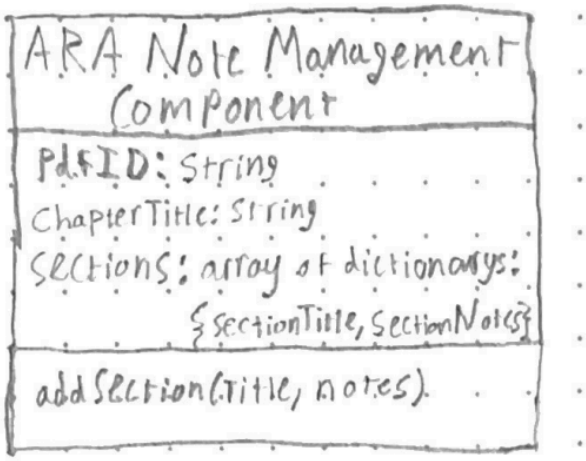

Figure 4.6.3.2: Static class model of ARA Note Management Component, a sub-component of the ARA Note Database Module.

4.6.4 Design Rationale

We chose to use an architecture that was modular, because it was easier to integrate into other modules. It ensured that the user data was accurately stored and easy to retrieve. The system is also very scalibile. If we were to allow for more users to use our system, it would be very easy to give them a section in the database, as well as make modifications or upgrades to the database to handle more traffic from additional users.

4.6.5 Alternative Designs

The other option we were looking at pursuing was an SQL database by using C++ instead of Python. The reason we chose to use MongoDB was simply because of complexity. It was easier for us to build a robust database using MongoDB over SQL, as SQL seemed to require a skillset that we weren't prepared to use.

4.7 Note Database API Module

4.7.1. Role and Primary Function

The Note Database API Module serves as the primary communication path between the Tkinter Visual UI and the User Notes MongoDB database. Its primary role is to use a set of APIs that process database operations. This module facilitates automatic saving, retrieval, and management of notes.

4.7.2. Interface Specification

- Save Operation: On user exit, calls the saveUserNotes method, gathers all of the current user's notes, and updates the database accordingly.
- Retrieval Operation: When application starts or when a user selects a PDF, the module's getNotes method is called to fetch any existing notes for the PDF, using the username and PDF ID to query the database.
- Update and Deletion Operations: any edits to the notes trigger the module's updateNote method, which sends updated notes to the database. deleteSectioncis called to remove specific database data
- Logout and Quit: If the user logs out or quits, the saveUserNotes method is called so that all current notes are saved before termination, so no data is lost

4.7.3. Static and Dynamic Models

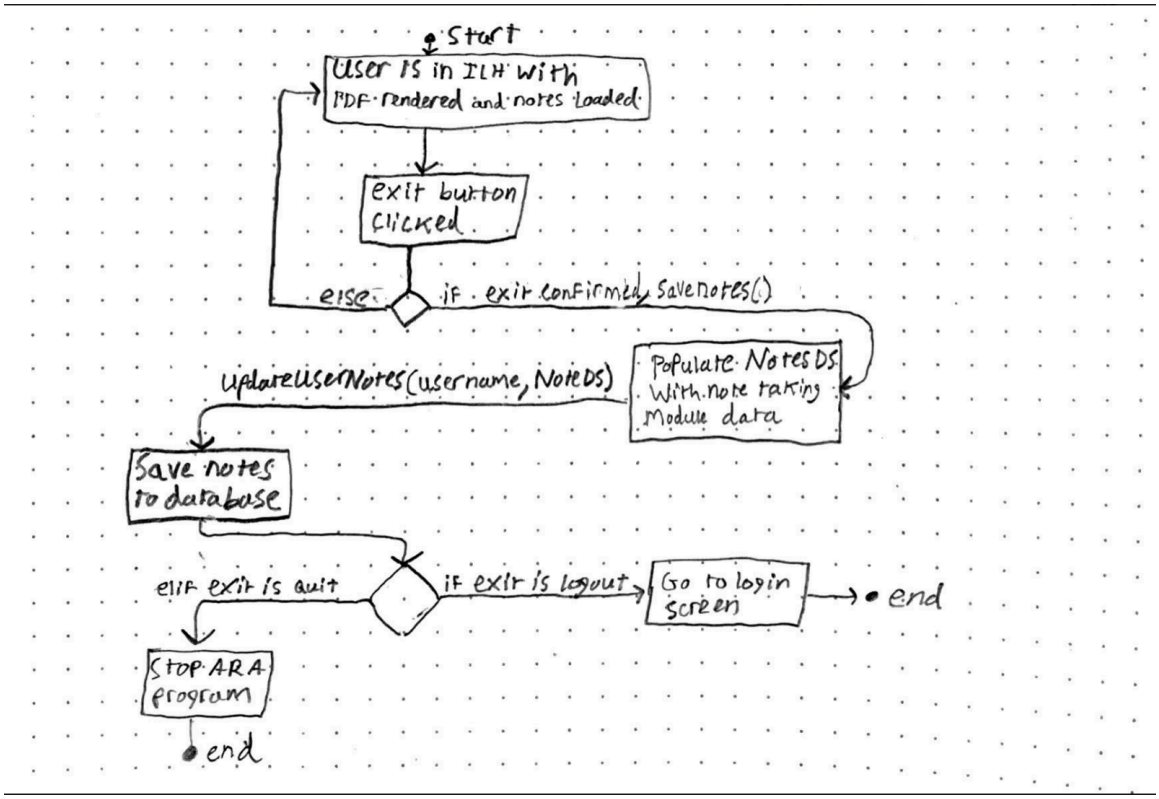

Figure 4.7.3.1: Dynamic model of Note Database API Module being used to <u>automatically</u> save notes when program is exited.

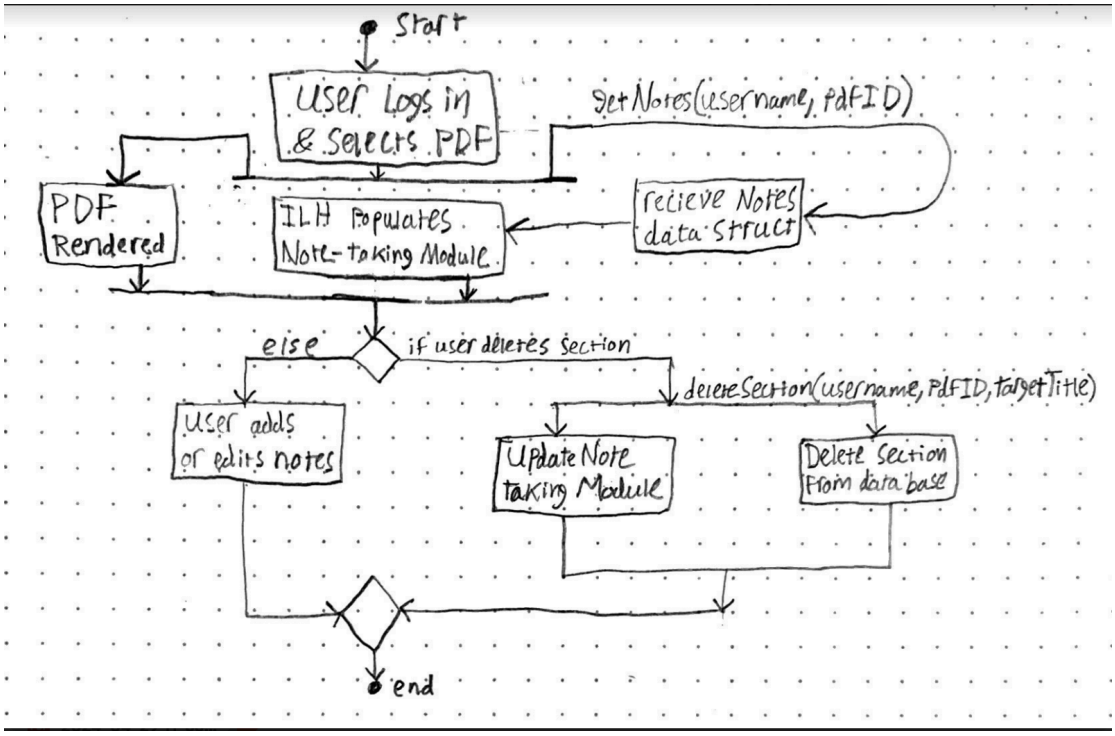

Figure 4.7.3.2: Dynamic model of Note Database API Module being used to delete a section.

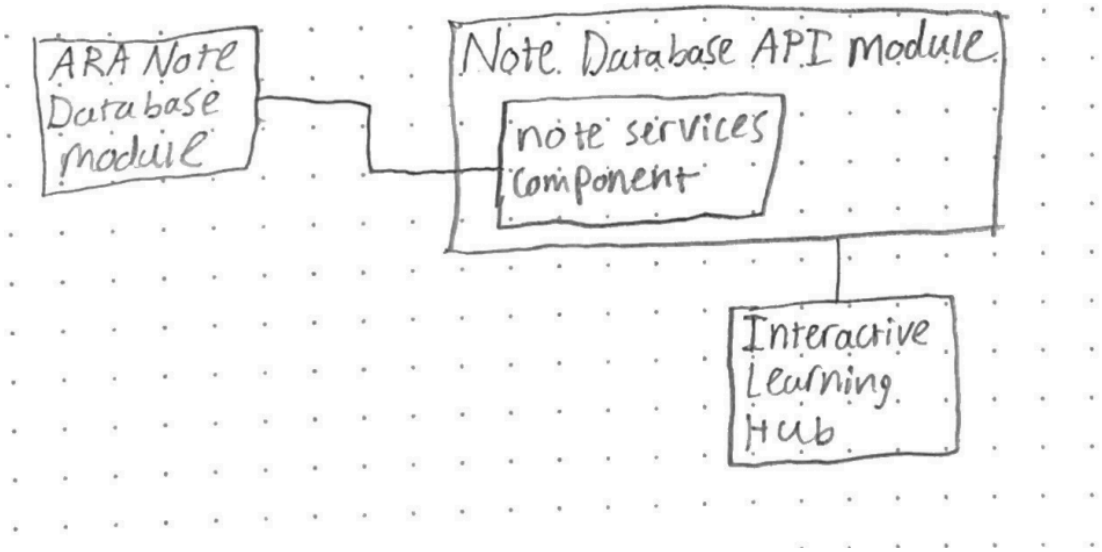

Figure 4.7.3.x: Static model of Note Database API Module showing its sub-components and connections to other modules.

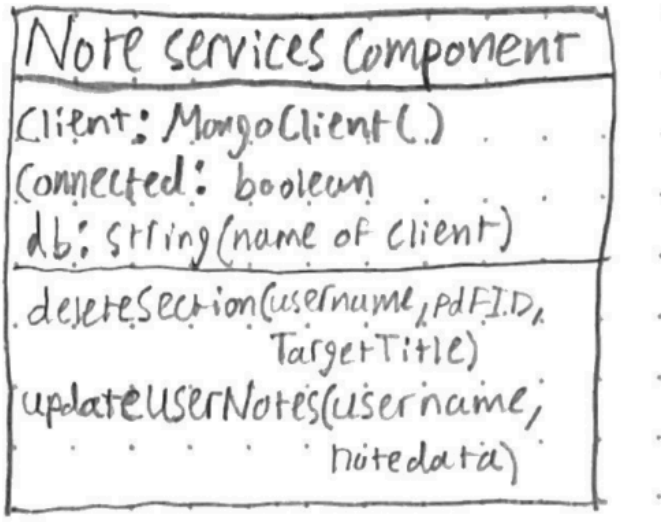

Figure 4.7.3.x: Static class model of Note Services, a sub-component of the Note Database API module.

4.7.4 Design Rationale

We chose to have an API module just to abstract the complexity of the database interactions. It allows us to have a simpler interface for when we need to query from the database. It also is able to improve the performance of the transactions, adding to the user experience.

4.7.5 Alternative Designs

This was the only design we considered for the database. From our initial meetings, we planned on creating API functions that could be reused in order to communicate with the database.

5. Dynamic Models of Operational Scenarios (Use Cases)

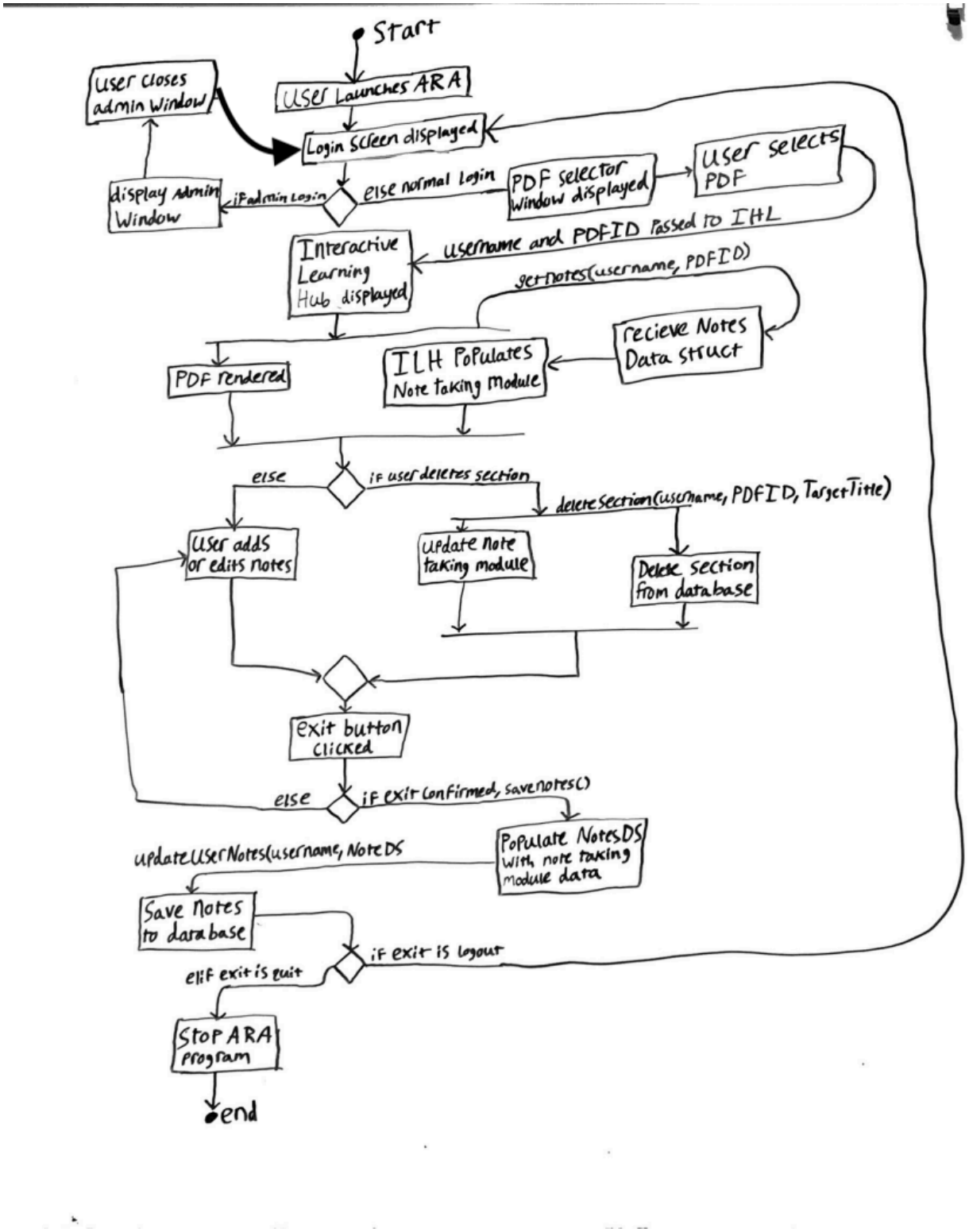

Figure 5.1: Dynamic flow state model of User navigating through the ARA Software, from Login to Exit.

5. Project Plan

5.1. Task Assignments and Management Plan

Date Assigned	Assignee	Task
2-20-2024	Ethan Hyde	Project Manager/Support dev, MongoDB server developer
2-20-2024	John Hooft	API function creation/fullstack dev
2-20-2024	John O'Donnel	TKinter Interactive Learning Module Designer
2-20-2024	Ben Bushey	PDF loading/Tkinter ILH button implementations

Our group was run collaboratively using a structure that contains defined roles for each person by utilizing the SRS document, holding regular meetings, and communicating regularly via a Discord server. Everybody had equal input on the design choices, and decisions were agreed upon during our group meetings. Group members were expected to bring up issues without judgment, and a final decision would be made by the project manager based off a majority vote.

5.1.1. Initial Decisions

The initial decisions in our group were made by using brainstorming and collaboration techniques to ensure that we would gather everyone's input. Even though most initial design choices are likely to change as the project progresses, it was important that our first diagrams would give everyone a clear idea on what they should work on, and how that would complement what other members were working on. What we found worked best for us was an in person meeting in the science library, and drawing static diagrams on the whiteboard consisting of the SRS requirements, then creating modules for each of those requirements.

5.1.2. Communication

Our group was able to communicate regularly via a Discord server that we all had access to. All of the documents such as the SRS, SDS, Project plan, Gantt Chart, etc were shared in the Discord "Resources" channel, and updated regularly. We help both in person and online meetings

multiple times per week to see how each person's parts have been going, and confirm what the next steps are going to be, making changes as we see fit.

5.1.3. Monitoring and Reporting

Progress tracking and reporting will be done in group meetings, but also done regularly via Discord. Each member is held responsible for following through on their tasks, and reporting any issues in the event that they may arise. Utilization of the Gantt chart will be used to visualize these changes, and updated if a task took longer/shorter than expected. This continuous monitoring allows for adjustments to be made while still meeting our milestones. We tried to emphasize that "The only thing worse than bad news, is bad news late."

5.2. Build Plan

5.2.1. Build Plan Sequence of Steps

The *Active Reading Assistan*t (ARA) consists of 7 Modules: Interactive Learning Module, Note Taking, PDF Selector, PDF Viewer, User Account Login, ARA Note Database, and Note Database API Module. The focus in the initial week of development was drawing out the contents of these modules, and creating a map for how they will connect to build the entire system.

After the first week, we had created a basic prototype for the ARA software. This was a command line interface that we could use to test the API functions, database setup and connectivity, and overall flow the diagrams we created in our meetings.

Once we had the prototyped developed and working, we were able to create the ILH interface and add functionality to the buttons by using the methods created in our prototype. These included: saving notes, selecting pdfs, show/hide notes, show/hide pdf, show/hide SQ3R prompts, select user, etc. By testing these functions thoroughly, we were left with a robust model entering the final week of development. During this final week, we were able to focus on refining our existing methods and work on the usability aspect of our software.

The figure below (Figure 5.2.1) is of our final Gantt chart, showcasing the order of our developmental steps. It shows who did what, when they did it, how long it took them, and the order of our general tasks as well.

CS 422 GANTT CHART

PROJECT TITLE	ARA Software, CS 422
PROJECT MANAGERS	Ethan H (EH), John HT (JHT), John O (JO), Ben B (BB).

TASK ID	TASK TITLE	TASK OWNER	START DATE	DUE DATE	DURATION	PCT OF TASK COMPLETE
1	MongoDB Server Setup	EH, JHT, JO, BB	4/12/2024	4/16/2024	5	100%
1.1	API functions - save notes, delete section, update notes, retrieve notes	JHT	4/13/2024	4/17/2024	9	100%
1.2	TKinter UI Prototype	JO, EH	4/15/2024	4/19/2024	8	100%
1.3	MongoDB Data Mangement Component (DS)	JHT	4/15/2024	4/18/2024	8	100%
1.4	PDF Loader - loads selected PDF to ARA	JO, EH	4/18/2024	4/22/2024	5	100%
1.5	PDF Viewer - scrollable PDF viewing module	JO, EH	4/18/2024	4/22/2024	5	100%
1.6	Note Heirachy Text Boxs	BB	4/21/2024	4/25/2024	5	100%
1.7	Login Management Component	JO, EH	4/22/2024	4/24/2024	5	100%
1.8	Interactive Learning Hub Navigation Buttons	JO, BB	4/24/2024	4/27/2024	4	100%
1.9	Admin / Base User functionality	BB	4/24/2024	4/27/2024	4	100%
1.1	SQ3R Guiding Prompts	JO, BB	4/24/2024	4/27/2024	4	100%
1.11	ARA Session Management Component	BB, JHT	4/25/2024	4/28/2024	4	100%
1.12	SRS/SDS document updating	EH, JHT, JO, BB	4/27/2024	4/29/2024	3	90%
1.13	Comment tkinter and mongo py files	JO	4/26/2024	4/28/2024	3	100%
	Update project plan, more descriptive	Eh, JHT	4/27/2024	4/27/2024	1	100%

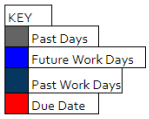

Figure 5.2.1

5.2.2 - Build Plan Rationale

The build plan was designed to establish a robust foundation before adding features to our complex modules. By structuring the development process in this way, the team was able to focus on the functionality of each component before moving onto the next one, or adding smaller features to each component. This minimizes the impact on potential setbacks that would have come up due to developing with the entire software in mind.

Risk management was another factor that we wanted to consider. Regular meetings and communication gave us an opportunity to allow for early detection of any issues that may come up, and squash bugs that would inhibit the team's ability to build upon a component's functionality. Risk reduction strategies included code review, regular testing of edge cases, and maintaining a flexible task schedule to accommodate for unexpected challenges.

5.3 - Meeting Timeline

1. **Initial meeting (4/11/24)**
    a. Attendees: EH, JHT, BB, JO
    b. Finalize project scope and deliverables, assign roles, establish communication procedures and set up future meetings.
    c. Decisions made: Agreed on our project's major milestones represented in the Gantt Chart, scheduled weekly meetings
2. **Meeting 2 (4/15/24)**
    a. Attendees: EH, JHT, BB, JO

    b. Reviewed progress on the MongoDB server setup, JHT presented API function prototypes
    c. Decisions made: MongoDB setup confirmed by EH, API development underway
3. **Meeting 3 (4/17/24)**
    a. Attendees: EH, JHT, BB, JO
    b. Checked progress on API development and ILH prototype
    c. Decisions made: ILH prototype adjustments suggested by BB and JO
    d. API integration progress noted, minor delays expected
4. **Meeting 4 (4/22/24)**
    a. Attendees EH, JHT, BB, JO
    b. Assessed progress on PDF loader and viewing functionalities, started development of login screen and exit screen methods
    c. Decisions made: PDF loading and viewing progress approved, start development of admin and regular users
5. **Meeting 5 (4/24/24)**
    a. Attendees EH, JHT, BB, JO
    b. Updates on ILH navigational buttons and user functionality
    c. Decisions made: Enhancements to ILH approved. These included where to place SQ3R prompts, how to guide the user through the software. Delay in prompts accounted for
6. **Meeting 6 (4/28/24)**
    a. Attendees EH, JHT, JO
    b. Final review of all pending tasks. Check progress on project documentation
    c. Decisions made: Final push for any incomplete tasks/bugs. Documentation responsibilities given out and checked on.
7. **Final meeting (4/29/24)**
    a. Attendees EH, JHT, BB, JO
    b. Final project review and closure of remaining tasks. Continue with project documentation for remaining modules
    c. Decisions: Support and maintenance plan established, completion of documentation

5.4. Project Plan Revision History

Before our final project plan, we went through two different drafts for our project plan, which are in the following 2 sections, 5.4.1 and 5.4.2. We updated them according to the project plan specifications linked on the CS 422 website to ensure that all requirements would be met

5.4.1 - Initial Project Plan

Below is our first draft project plan which was completed on 4/7/2024. We would then build upon this project plan which to be included in our initial submission for the Active Reading Assistant.

Project Plan

Ethan Hyde, John Hooft Toomey, John O'Donnel, Ben Bushey

Language - Python
Database framework - MongoDB
GUI - Tkinter

Management Plan:

- Team Organization:
    - Tkinter User Interface Dev: John O
    - MongoDB Server Dev: Ben
    - API / Fullstack Dev: John H
    - Project Manager / Support Dev: Ethan H
- Our team meets twice a week:
    - Every Monday at 5:15pm over Discord.
    - Every Wednesday at 10:00pm in the Science Library.
- Team decision making is done collaboratively, when making project impactful decisions team members will consult their peers to pick the best option.

Monitoring and reporting:

- Our team utilizes a trello kanban board to keep track of what everyone is working on, has completed, and needs to do.
- Most likely will be used to organize who does what software module.

Build Plan: (will be derived from Gannt chart developed in class Tuesday Apr 9th)

A rationale for the build plan:

- Critical Path of Development:
    - MongoDB Server setup.

- API Layer functions (fetch data, push additional data, add new data, etc).
- Tkniter prototype UI to test API layer functions.
- In parallel:
    - PDF Loader / Viewer Module.
    - SQ3R guiding prompts.
    - Note Hierarchy Text boxes.
    - MongoDB Management Component.
        - Data Structure to store user info, used for both sending and receiving data from DB.
- User Login Module
- Session Management Component.
- Server Logic Module for Connection Management and Error Handling.

5.4.2 - Second Project Plan

Below is our second project plan that was included in our initial submission on 4/12/2024.

Active Reading Assistant Project Plan

Ethan Hyde, John Hooft Toomey, John O'Donnel, Ben Bushey

Management Plan:

Team Organization:

Project Manager / Support Dev: Ethan H
Tkinter User Interface Dev: John O
MongoDB Server Dev: Ben
API / Fullstack Dev: John H

Team Meeting Schedule:

Every *Monday*: 5:15pm over Discord.
Every *Wednesday*: 10:00am in the Science Library.

Monitoring and Reporting:

To monitor and report progress, our team utilizes a Trello Kanban board along with a Gantt Chart. The Kanban board tracks individual tasks, completed work, and upcoming assignments. It will also serve to organize the distribution of software modules among team members. The Gantt chart will help visualize the roadmap and scheduled milestones we plan to achieve. Team decision-making is collaborative, with members consulting their peers when making impactful decisions.

Work Breakdown Schedule and Project Schedule:

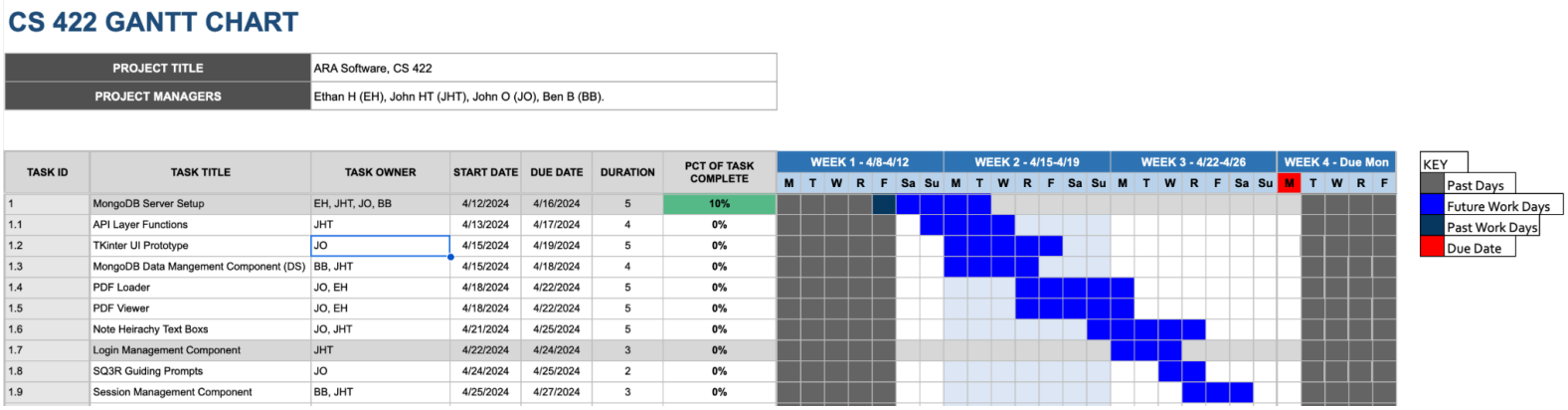

Figure 1: Gantt Chart with project milestones.

Rationale for the Build Plan:

The system is divided into specific parts to ensure a systematic and efficient development process. The critical path of development includes essential steps such as setting up the MongoDB server, developing API layer functions, creating a Tkinter prototype UI to test these functions, and then continuously improving and implementing the requirements listed in the SRS. Parallel development of other modules such as the PDF Loader / Viewer and the SQ3R Guiding Prompts aims to maximize efficiency. The chosen steps are prioritized based on their significance to the overall functionality of the system and their interdependencies.

6. Software Requirements Specification

1. SRS Revision History

Date	Author	Description
4-28-2024	JO	Updated document for Final Submission
4-11-2024	EH	Updated document for Initial Submission
4-4-2024	JHT	Created the initial document using the template from https://classes.cs.uoregon.edu/24S/cs422/Templates.html

2. The Concept of Operations (ConOps)

This document is the Software Requirements Specification (SRS) for the Active-Reading Assistant (ARA). ARA is a computer program that is designed to support the SQ3R reading method for PDF documents.

ARA is a software tool that guides students through the use of the SQ3R active reading technique. SQ3R stands for Survey, Question, Read, Recite, Review. SQ3R provides students (and other readers) with a structured approach for reading textbooks and other technical material. Learning such material is best accomplished by first skimming the high-level structure of a chapter (or other section of text), and then generating questions, and then reading the material primarily to answer the initial questions (McKeachie & Svinicki, 2013).

The goal of SQ3R is to motivate students to engage in *active learning* by (a) generating questions, (b) moving through the text with the goal of answering those questions, and then (c) quizzing yourself on what you learned. The goal is to encourage and support “deep processing” of the text (McKeachie & Svinicki, 2013, p.30).

In the first pass through a chapter, the *survey*, the student should skip over a lot of text, such as to only (a) read the headings and topic sentences and (b) study the figures and diagrams. It is expected that students will eventually read the entire chapter, but only after first constructing a mental understanding of the structure of the chapter, and how its major pieces fit together (ibid.).

The basic concept of the ARA system is that it will provide a structured, hierarchical, note-taking facility that guides students through the use of SQ3R active reading technique, by providing
(a) appropriate prompts for the next step in following the SQ3R, (b) a structured organization of text fields in which the user can type their notes, and (c) an easy means of quizzing yourself on what you read.

ARA should store the user's notes on a server that can be accessed from anywhere with full internet access. This permits the user to use multiple different computers to use the SQ3R method to read a textbook, and always add to their previous notes rather than starting fresh.

ARA should provide, on the server, sample PDFs of textbook chapters, with all of the important *survey* content highlighted, such as headings, topic sentences, and diagrams. This should help new SQ3R users to understand how much of the chapter they should be reading during the *survey* phase.

2.1. Current System or Situation

There are substantial digital-document paper-based resources available to assist students in the use of SQ3R. For example: Virginia Tech (2024), McKeachie & Svinicki (2013), Robinson (1946), and University of Oregon (2024). A brief internet search suggests that there are few if any pieces of software created specifically for helping students learn and use SQ3R.

2.2. Operational Features of the Proposed System

The key operational features of ARA include: (a) guiding the user through the use of SQ3R, (b) the display of "starter" PDF files that visually highlight what should be looked-at during the *survey* phase, (c) a means to capture hierarchical notes in which the user can do SQ3R tasks (such as writing questions and answers), (d) supporting self-quiz, and (e) storing all of the data on a server.

The system will specifically follow the SQ3R technique that is described by many university academic academic guidance centers, including here at the University of Oregon (2024), such as at https://engage.uoregon.edu/learning-resources/.

2.3. User Classes

There are two user classes:

1. A "student" who is attempting to use SQ3R, with the assistance of ARA, to engage in active learning while reading. The "student" could be anyone, not necessarily someone in school, but this SRS will refer to this user as the student.

2. A system administrator who sets up the server. The administrator's duties will be to set up the MySQL (or other) server that the ARA system will use. This would be a moderately-skilled computer user familiar with installing software, editing text files, and using the Unix command prompt. The system should provide instructions for this user on how to set up and test the server.

2.4. Modes of Operation

The system has one primary mode of operation, in which the server is running, and a student runs ARA which gains client access to the server.

2.5. Operational Scenarios (Also Known as "Use Cases")

"Students" can use SQ3R with the assistance of ARA to engage in *active learning* by (a) generating questions, (b) moving through the text with the goal of answering those questions, and then (c) quizzing yourself on what you learned. The goal is to encourage and support "deep processing" of the text (McKeachie & Svinicki, 2013, p.30).

"Admin" users can use this software to set up the MongoDB database that is used by the ARA system.

3. Specific Requirements

The basic functionality must include the following:

Externally-Visible User Interactions

1. A visual field for taking notes.
2. Automatic storage on the server.
3. The system should support a hierarchy of notes such as: Chapter Title (single line, entered by user)
   Section Heading (single line of text, entered by user)
   Notes for that section (scrolling text, with some kind of separation between lines.)
4. The system should support the hiding of notes so that users can quiz themselves on what they wrote down. This could be accomplished, for example, with a single mouse-click (such as, on a button) to hide the notes, and a single mouse-click to unhide notes
5. Pre-loaded PDFs that have all of the important *survey* content highlighted, such as headings, topic sentences, diagrams. (At least three chapters from Sommerville should be included.)
6. Pre-loaded complete notes from one entire Sommerville chapter, to provide sample data to see how the system can work.

SQ3R Assistance ("Scaffolding")

7. The program should guide the user through the use of SQ3R with prompts such as: "SURVEY: Glance over the headings in the chapter to see the few big points."
   4a. "Prompts" should guide the user through the use of SQ3R. Prompts should be non-modal: They should not block any text the user types, should not require the user to read them or provide any input, and should not be interactive. For example, the prompts could appear as headings above the text fields.
   4b. The user should be able to turn the prompts on or off, making the prompts either present or absent.

Login

8. There should be *NO* need to set up a username and password for this initial version of the software. Any login information should be built in to the software.
   [Correction made on 4-16-2024.]

Target Platform

9. The software should run on a laptop or desktop machine. The program should be designed for use with a real keyboard, not a smartphone. The system should run on macOS 12.

Data Storage

10. Data should be stored on a server using either mysql or mongo.
11. The initial version of the system can have the server and ARA client running on the same machine. The ideal final version of the system should permit the administrator to set up the server on a remote machine, and have ARA access that server using the Internet.
12. The system should save all notes that the user enters, and should never delete user data without a warning.
13. The system should provide a warning, at startup, if the server connection cannot be established, and data will not get saved. (The ideal system would save the changes locally until the server connection could be established, and then save the changes. But this will not be a requirement with this system.
14. Some Aspects of the System Can be "Mocked Up". The system should initially have three predefined users, with no passwords necessary. It should be very easy to select which user you are logging on as, such as with a single click, or selecting usernames from a list. (Yes, this would of course create security problems if the system were to be deployed on the Internet, but these seems like a "solved problem" that would be easy to implement at a later time.)

Build-Related Constraints

15. System Document File Formats

All system-related and system-development-related documents that are intended for human reading must be in either plain text or PDF. For example, Microsoft Word, Microsoft Excel, or markdown language documents must be converted into plain text or PDF.

16. Programming Constraints

The following language may be used:

- Python 3 along with the Python Standard Library https://docs.python.org/3/library/index.html, but no other imports except for *mysql* or *pymongo* unless written permission is requested and granted. (Please send brief email requests explaining what libraries you would like to use, provide a link to the library website, and explain why you would like to use them.)
- The GUI framework tkinter is part of the Python Standard Library.
- Python code must run in 3.12.

17. Installation Constraints

- Instructions must be provided for how to compile/run the code and install the system on a machine running macOS.
- There can be at most 20 commands required to set up the server, compile the code, and run

the program.

- An experienced computer programmer should not require more than 20 minutes working alone with the submitted materials to set up the server, and compile and run the code.

All code, except for approved library imports, should be submitted. Please do not direct the person installing the software to a location to download the code.

- No server connections may be required for either installing or running the software, except for setting up mysql on ix (or ix-dev), and you must provide instructions on how to set this up. The instructions should be as simple as the MongoDB and MySQL guidance at https://systems.cs.uoregon.edu/wiki/index.php?n=Help.Tools (along with whatever files are needed to create your tables and such). [Added on 4-19-2024.]
- No virtual environments may be used.
- No gaming engines such as Unity may be used

18. References

McKeachie, W., & Svinicki, M. (2013). McKeachie's teaching tips. Cengage Learning.

Robinson, F. P. (1946). Effective study.

University of Oregon Tutoring and Academic Engagement Center. (2024). Reading Strategy: SQ3R. Accessed from this website: https://engage.uoregon.edu/learning-resources/. Direct link: https://engage.uoregon.edu/files/2014/09/SQ3R_Worksheet.pdf. Accessed 4-4-2024.

Virginia Tech Cook Counseling Center. (2024). SQ3R: Reading/Study System. https://ucc.vt.edu/academic_support/study_skills_information/sq3r_reading-study_system.html. Accessed 4-4-2024.

Michigan State University. (2022). Reading a Textbook Effectively. https://natsci.msu.edu/students/current-students/student-success-resources/academic-success/habits-to-develop-outside-of-class/study-strategies/reading-a-textbook-effectively/. Accessed 4-4-2024.

19. Acknowledgements

This SRS builds on the template from https://classes.cs.uoregon.edu/24S/cs422/Templates.html