Florida International University
School of Computing and Information Sciences

Software Engineering Focus

Final Deliverable

Agenda Management System
Version: 1.0

Team: Adrian Rodriguez, Carlos Vera, Cristian Cepeda, Julian Popiloff, Oscar Padilla

Product Owners: Jay de la Campa, Margaret Brisbane

Mentor: Masoud Sadjadi

Instructor: Masoud Sadjadi

The MIT License (MIT)

Abstract

This document presents the information necessary to gain a good understanding of the background, necessity, technology used, efforts put into the first version of the Agenda Management System Application. Currently, many departments at Miami-Dade county don't have a system in place to manage and track the modification of agenda items (matters or items) before they get sent to the agenda coordinator's office. This causes many accountability issues when trying to track down who edited a matter last, or who has the most up to date copy. Agenda matters are transferred between peers either through email, word documents with comment annotations, fax, and even hard copies.

Our software aims to help facilitate this process by providing a portal where a user within a department can submit, edit, view, and route matters to other users of the application, so that a specific matter can have a history of who modified it and when. The software would be a one stop shop that streamlines the process of agenda management through help of user accounts, dynamic PDF creation, audit trails, and more.

Table of Contents

INTRODUCTION

Current System

The current system of the application supports the creation, editing, and viewing of items, as well as the secure authentication of users and authorization through use of JWT. The application also supports the naive and advanced searching of existing items within the application. Currently, the application is built on top of the NodeJS platform, utilizing JavaScript, React, Express, and Amazon Web Services' S3 and RDS services. This current system is the first version of the product. There is no uniform platform in place that the Miami-Dade IT Department uses to facilitate agenda management. The current system is catered to the Miami-Dade IT Department, and future versions should seek to expand the type of items that the application supports by meeting with other departments within Miami-Dade County.

Purpose of the New System

The purpose of the new system (which is the current system that we developed over this semester) is to streamline the agenda management process within a department by providing a web application portal. The application has defined roles for a user of the application which defines the actions a user can take within the application. A user of the application can be an Administrator, a Manager, and a User. An Administrator can create user accounts, and modify all items, as well as create. A Manager can create items, and modify all items. A User can create items and modify items only that are assigned to themselves or items that they created. Once a user is logged in, they can perform any action they are allowed to. When it comes to items, the meat of the application is dedicated to the modification and editing of items. Viewing an item shows all the information of an item, including the history of that specific item.

USER STORIES

The following section provides the detailed user stories that were implemented in this iteration of the project. These user stories served as the basis for the implementation of the project's features. This section also shows the user stories that are to be considered for future development.

Implemented User Stories

- **Title:** Documentation Tools
- **User Story Number:** AGDA-42
- **User Story:** As a user, I would like to provide a documentation tools page, so that I can describe and refer to all development tools.
- **Acceptance Criteria:**

- React JS
- Redux
- Material UI or Bootstrap
- AWS
- S3 - Documentation
- RDS - Relational DB
- EC2 - Server on the Cloud
- Elastic Beanstalk

- **Title:** Item Feed – Back end
- **User Story Number:** AGDA-72
- **User Story:** As a user, I would like to see a feed of items, so that I can be up to date on items that were modified while I was not on the application.
- **Acceptance Criteria**:
  - Must support:
  - Items that are assigned to the logged in user are displayed in a table on the user's home page
  - Items are sort-able via columns
  - Items per page can be changed
- Use Results page to display the data, where column are generated dynamically from query results
- Note: Consider relevance, time sensitivity, and recently modified.

- **Title:** Email and Password Fields
- **User Story Number:** AGDA-75
- **User Story:** As a User, I would like to have a login form section so that I can input my credentials and it can send the credentials to a server and be authenticated.
- **Acceptance Criteria:**
  - Text field so that actor can input their email
  - Text field so that actor can input their password
  - Submit button to send Actors credential to server for authentication
  - Forgot link that redirects to another link

- **Title:** Current User Propagated Throughout Application
- **User Story Number:** AGDA-81
- **User Story:** As a user I would like to stay logged in across pages, so that I can access and modify the correct items without having to log in again.
- **Acceptance Criteria:**
  - A JSON web token is generated for a user when they log in.
  - A JSON web token is present in the user's session as they use the application.
  - The JSON web token is checked when a user tries to interact with the application and request data.

- **Title:** Basic Info Fields for Profile
- **User Story Number:** AGDA-82

- **User Story:** As a User, I would like to have access to a profile page so that I can edit my personal information.
- **Acceptance Criteria:**
    - Profile page must display personal name, position, email, display name, and phone number.
    - Actor can edit display name, and phone number.

- **Title:** Statistics Link
- **User Story Number:** AGDA-83
- **User Story:** As a User, I would like to navigate to a statistical page so that I can visually see statistics that may be relevant to me.
- **Acceptance Criteria:**
    - Create a button to link to this page for future semesters to finish.
    - Use Chart.js or D3.js to create statistical graphs that represent some data based on the number of items you may have assigned to you.
- **Title:** Navigation Bar
- **User Story Number:** AGDA-84
- **User Story:** As a user, I want a navigation bar to access different parts of the application, so that I can quickly jump to them from whichever page I am on.
- **Acceptance Criteria:**
    - Global navigation bar for traversing parts of the application.
    - Need a search bar and an advanced filter to go along with it.
    - Profile page would be accessed via a button with their face.

- **Title:** Search for an item by ID
- **User Story Number:** AGDA-86
- **User Story:** As a user, I would like to search for an item by ID, so that I can easily find an item when I know its ID.
- **Acceptance Criteria:**
    - Searching for an item by an input ID returns the correct
    - **Note:** Id in this context is the name of a matter/item

- **Title:** Search for an item by title
- **User Story Number:** AGDA-88
- **User Story:** As a user, I would like to search for an item by title, so that I can find an item if I know it's title, or a portion of it.
- **Acceptance Criteria:**
    - Input string returns all relevant items that contain that string.

- **Title:** Search for an item by matter subtype
- **User Story Number:** AGDA-89
- **User Story:** As a user, I would like to search for an item based off who requested it, so that I can find an item that a certain user had requested.
- **Acceptance Criteria:**
    - Searching for an item with a specific requester returns the specific item if it exists.

- **Title:** Search for an item by active assignee
- **User Story Number:** AGDA-90
- **User Story:** As a user, I would like to search for an item that has been assigned to someone, so that I can see the item at its current state.
- **Acceptance Criteria:**
    - Searching for an item with an input user assignee returns the appropriate items

- **Title:** Search for an item by time to expiration
- **User Story Number:** AGDA-91
- **User Story:** As a user, I would like to search for an item by providing a time range, so that I can view items that fall within that time range.
- **Acceptance Criteria:**
    - Providing an expiration date shows all items that fall before the expiration date.

- **Title:** Search for an item by contract number
- **User Story Number:** AGDA-93
- **User Story:** As a user, I would like to search for an item based on the procurement value, so that I can see an item that has the specified number I searched for.
- **Acceptance Criteria:**
    - Inputting an integer value to the advanced search and submitting the query returns the items that have that value.

- **Title:** Template - Front-end
- **User Story Number:** AGDA-94
- **User Story:** As a User I would like the application to populate the correct fields of a specific item type so that I can fill in the correct information in for an item subtype.
- **Acceptance Criteria:**
    - A specific subtype has its own specific fields for the body, or sections.
    - The fields are stored in the database and are populated upon type and subtype selection.

- **Title:** Set Visibility Based on Users Permissions
- **User Story Number:** AGDA-99
- **User Story:** As a User, I would like user level access to pages so that I can view and access the pages that correspond to me only.
- **Acceptance Criteria:**
    - Determine different levels of user permissions and implement a system for the admin to edit those permissions.
    - Check user role and assign routes to each user so that they can later access them

- **Title:** View Item History Trail with Downloadable Versions
- **User Story Number:** AGDA-101

- **User Story:** As a user I would like to be able to view an item's history and download it's different versions so that I can keep track of who changed an item and have access to past drafts.
- **Acceptance Criteria:**
    - Have version controlled AWS S3 bucket accessible
    - Develop a GET API route and controller function in the backend using the AWS SDK to retrieve/download the selected version of a PDF attached to an item.
    - Keep track of versions and author of last edit attached to a document/matter on the History table
    - Display the history attached to a document/matter in order of latest on the view item page

- **Title:** Display the fields of an item
- **User Story Number:** AGDA-102
- **User Story:** As a User I would like to view the fields that an item has so that I can view the information an item has within the application.
- **Acceptance Criteria:**
    - Viewing an item shows that items information
    - Viewing an item shows that items history

- **Title:** Permission Check
- **User Story Number:** AGDA-103
- **User Story:** As a User I would like items to only be editable by those allows so that users don't mistakenly edit items not relevant to them.
- **Acceptance Criteria:**
    - A user with the wrong role cannot edit an item
    - A user can only edit an item that they created or is assigned to them
    - An 'access restricted' page is shown if a user tries to edit an item they aren't allowed to edit

- **Title:** Advanced search page that offers filtering
- **User Story Number:** AGDA-126
- **User Story:** As a user, I would like to be brought to a page where I can filter general searches so that I can be more specific in the item I want to see.don't mistakenly edit items not relevant to them.
- **Acceptance Criteria:**
    - There exists a form with more filtering options that can be submitted (include Title as a filter at least)
    - When the form is submitted, a results page is shown. The form is still accessible from this results page
    - **Note:** Brought to this page after a regular search, includes results and filtering opportunities.

- **Title:** Item Feed – Front end

- **User Story Number:** AGDA-127
- **User Story:** As a user, I would like to see a feed of items, so that I can be up to date on items that were modified while I was not on the application.
- **Acceptance Criteria:**
    - Items that are assigned to the logged in user are displayed in a table on the user's home page
    - Items are sort-able via columns
    - Items per page can be changed
    - Use Results page to display the data, where column are generated dynamically from query results
    - **Note**: Consider relevance, time sensitivity, and recently modified.

- **Title:** Choose Type and Subtype for Item Creation - Backend
- **User Story Number:** AGDA-129
- **User Story:** As a developer I would like to have access to the types and subtypes of items in the database so that a chosen subtype's pre-defined template can be utilized on the front-end to prepare for item creation.
- **Acceptance Criteria:**
    - Develop a GET API route and controller function to query the sections of the chosen type and subtype from the 'Type' table on the database holding the pre-defined item templates.

- **Title:** Creating an Item by Submitting the Fields - Backend
- **User Story Number:** AGDA-130
- **User Story:** As a developer I would like to store the fields submitted from a created item on the database so that I can upload a generated PDF to the AWS S3 bucket for version control and easy download, and keep track of an item's history.
- **Acceptance Criteria:**
    - Develop a POST API route and controller function to insert a new item to the 'Matter' table.
    - Use the PDFkit JS library to convert the item's data to a PDF object.
    - Develop a POST API route and controller function using the AWS SDK to upload the PDF object to an AWS S3 bucket with version control.
    - Develop a POST API route and controller function to insert the S3 version id and relevant item data to the 'History' table.

- **Title:** Choose type and subtype
- **User Story Number:** AGDA-131
- **User Story:** As a User I would like to select a type and subtype of an item when I’m creating it so that I can have the correct fields populate when I’m creating a new item.
- **Acceptance Criteria:**
    - All available types are populated and able to be selected
    - Selecting a type populates the subtype selector with the correct options
    - Selecting a subtype is allowed

    - Submission of a type and subtype leads to a separate page containing that type and subtypes information

- **Title:** Creating an item page
- **User Story Number:** AGDA-132
- **User Story:** As a User I would like to have an item I create be saved in the application so that it can be viewed and edited later by myself and other users of the application.
- **Acceptance Criteria:**
    - A user can edit the fields of an item
    - A user can submit the item successfully

- **Title:** Create a PDF or Document from Our Item Objects
- **User Story Number:** AGDA-133
- **User Story:** As a User, I would like the application to auto generate a pdf once I submit a new item so that I can download it and send it to someone for review or approval.
- **Acceptance Criteria:**
    - Use users input to create and auto generate a PDF from the create item tab.
    - Store the PDF on a backend service so I can later retrieve it and view it.
    - Have the PDF be formatted so it can display the information properly.

- **Title:** Define Procurement Elements from Glossary
- **User Story Number:** AGDA-134
- **User Story:** As a developer I would like to define the elements that make up all the types of procurement so that predefined item type and subtype sections can be stored and accessed from the database.
- **Acceptance Criteria:**
    - Procurement documents and the sections that define them are analyzed.
    - Use PostgreSQL to insert all the possible type/subtype match-ups for items into the AWS RDS database.
    - Each match-up row will include an attribute of array type defining the template sections.

- **Title:** Edit an item
- **User Story Number:** AGDA-139
- **User Story:** As a User I would like to be able to select an item that exists to edit so that I can keep its information up to date and accurate.
- **Acceptance Criteria:**
    - Ability for user to edit fields of an item by user input.
    - Correct item information appears when editing.
    - Submission button to submit edits.

- **Title:** Edit/Update an Item - Backend
- **User Story Number:** AGDA-140

- **User Story:** As a developer I would like to be able to update the fields of an item on the database so that I can keep an item's information up-to-date with an accurate track of it's history of changes.
- **Acceptance Criteria:**
    - Develop a PUT API route and controller function to update an item on the 'Matter' table.
    - Use the PDFkit JS library to convert the updated item's data to a PDF object.
    - Connect the POST API route and controller function using the AWS SDK to upload the PDF object to an AWS S3 bucket with version control.

- **Title:** Search for last user to edit file (mtlastuser)
- **User Story Number:** AGDA-141
- **User Story:** As a user I would like to be able to refine the filters of the search so that I can specify the last user to edit the item.
- **Acceptance Criteria:**
    - There is a new case-insensitive filter in the advanced search form that for "Last Editted By" or similar
    - When submitting the form, I should be redirected to the results page showing the correct results

- **Title:** Search Results have hyperlinks on item names for viewing item details
- **User Story Number:** AGDA-148
- **User Story:** As a user I would like to see hyperlinks on the search results so that I can see more details of the items
- **Acceptance Criteria:**
    - The rows under Matter Title column are hyperlinks
    - Clicking on the rows redirects me to the view item page for that row

<u>Pending User Stories</u>

- **Title:** Forgot password logic
- **User Story Number:** AGDA-74
- **User Story:** As a user, I would like to click a link that would send a notification to the system administrator with a password reset, so that I can regain access to my account if I forget my password or if my account is compromised.
- **Acceptance Criteria:**
    - A notification is sent to the admin to send a reset link for the requester's account
- **Title:** Inbox of Items
- **User Story Number:** AGDA-76
- **User Story:** As a user, I want an inbox that holds all my items so that I can easily access all the items that were routed to me.
- **Acceptance Criteria:**
    - Show an inbox of all the items related to a user.
- **Title:** Login attempts

- **User Story Number:** AGDA-77
- **User Story:** As an administrator, I want to restrict the amount of times a user can attempt to log in so that brute force techniques cannot be used to access the system.
- **Acceptance Criteria:**
    - Restricting user login if they have N (tbd) number of failed attempts
    - Send a notification to the system administrator after this limit has been reached
- **Title:** Notifications
- **User Story Number:** AGDA-78
- **User Story:** As a user, I want notifications so that I can be alarmed when I have been assigned an item or it's been approved.
- **Acceptance Criteria:**
    - Functionality to send email on an individual directive upon assignment to Deputy Mayor to include staff (Assistant, to be defined), to include: Legistar item number, Requester. Meeting date, Assigned to, Due date, Agenda item number, Agenda information, Directive Status and directive verbiage.
    - Approval tracking sends emails when the item is assigned to an approver.
    - Navbar notifications on the web application
    - Some sort of simple messaging via email
- **Title:** Item Action Shortcuts
- **User Story Number:** AGDA-80
- **User Story:** As a user I would like item action shortcuts easily accessible so that I can quickly view items based on priority or relevancy, or create an item.
- **Acceptance Criteria:**
    - View Items, Priority, Relevant, Create Item buttons with routes

- **Title:** Define MOU
- **User Story Number:** AGDA-95
- **User Story:** As a user, I would like to create a MOU item, so that I can create a MOU item.
- **Acceptance Criteria:**
    - The database has the info for an MOU item
    - The ability to create an MOU item is capable within the application

- **Title:** Define Lease Agreement
- **User Story Number:** AGDA-96
- **User Story:** As a user, I would like to create a Lease Agreement item, so that I can create a Lease Agreement item.
- **Acceptance Criteria:**
    - Lease agreement is present in the database
    - Lease agreement item can be generated on the application

- **Title:** Define a new custom Template
- **User Story Number:** AGDA-97
- **User Story:** As a admin, I would like to create a custom template to be added to the list of available templates, so that more templates are available for use, if needed.
- **Acceptance Criteria:**

    - Create template page allows custom fields to be configured.
- **Title:** Save the state of an Item
- **User Story Number:** AGDA-98
- **User Story:** As a user, I would like to save the state as my item as either in progress or revision version, so that my progress is saved if i have to quickly edit the item
- **Acceptance Criteria:**
    - Cache user input cyclically

- **Title:** View the history of how an item has changed
- **User Story Number:** AGDA-100
- **User Story:** As a user, I would like items to have a version history displayed, so that I can view when/how an item was modified.
- **Acceptance Criteria:**
    - Display history of item on sidebar of item's view page.

- **Title:** Send notification to Watchers
- **User Story Number:** AGDA-104
- **User Story:** As a user, I would like to be notified if an item I am watching is modified, so that I can stay informed and keep up with changes.
- **Acceptance Criteria:**
    - Notification system should notify users through their inbox of any edits to their followed items.

- **Title:** Send notification to Owners
- **User Story Number:** AGDA-105
- **User Story:** As a user, I would like to be notified if the item I own is modified, so that I can stay informed and keep up with changes.
- **Acceptance Criteria:**
    - Notification system should notify users through their inbox of any edits to their created items.

- **Title:** Handing the item off, re-routing
- **User Story Number:** AGDA-108
- **User Story:** As a user, I would like to be able to re-route and hand off an item from the edit item page, so that an item can continue through its process manually.
- **Acceptance Criteria:**
    - Add area in template for an item where user can change who an item will be sent to, or add additional people, and send it out.

- **Title:** Handing the item off, re-routing
- **User Story Number:** AGDA-109
- **User Story:** As a user, I would like to be able to re-route and hand off an item from the view page, so that an item can continue through its process manually.
- **Acceptance Criteria:**
    - Add area in template for an item where user can change who an item will be sent to, or add additional people, and send it out.

- **Title:** Handing the item off, re-routing
- **User Story Number:** AGDA-110
- **User Story:** As a user, I want to hand off an item to another user in the system, so they can perform any changes to the item as necessary.
- **Acceptance Criteria:**
    - Entering a user's name to route to ensures that the item routes to that person.
    - If that person's status is away, then it gets routed to the secondary person.
    - If the secondary person is away, then the item can't be routed.

- **Title:** Display rate of activity
- **User Story Number:** AGDA-111
- **User Story:** As a user I would like to see How frequently an item has been modified so that I can see its rate of activity.
- **Acceptance Criteria:**
    - Number of times modified is displayed on the item
- **Title:** Statistics about items
- **User Story Number:** AGDA-112
- **User Story:** As a user I would like to see the statistics about a specific item so I can get more information about it.
- **Acceptance Criteria:**
    - Viewing an item also shows the stats of the item

- **Title:** Show up to date remaining balance
- **User Story Number:** AGDA-113
- **User Story:** As a user I would like to see the remaining balance of funds in a department so I know if I can make a procurement item with a cost amount.
- **Acceptance Criteria:**
    - Cost of department is shown in the application

- **Title:** Configurable workflow and queue of work items to prevent assignment of work to unavailable status users by having an item move to the next available person
- **User Story Number:** AGDA-115
- **User Story:** As a user I would like to have an item assigned to me be routed to a different user if I am unavailable so that an item can be worked on if I am out of office.
- **Acceptance Criteria:**
    - A user that gets an item assigned to them has the item go to the next available user if they are unavailable

- **Title:** Statistical/summary-based reporting of all Directives by requester by status and Directive type for a time-frame
- **User Story Number:** AGDA-116
- **User Story:** As a user I would like to see a report based on time frame for more detailed information.
- **Acceptance Criteria:**
    - Report generated based on input parameters

- **Title:** Rejecting an item routed to you
- **User Story Number:** AGDA-120
- **User Story:** As a user I would like to reject an item routed to me so that I can not be held accountable for an item that is not meant for me.
- **Acceptance Criteria:**
    - An item assigned to a user can be routed back to the user that assigned it to them
    - The user that denied it has no entry in the history table

- **Title:** Display the number of times a user has logged in the application
- **User Story Number:** AGDA-128
- **User Story:** As a user I would like to see the number of times a user has logged in the application so I can see how often they are using the app.
- **Acceptance Criteria:**
    - Viewing a user's profile shows how many times they have logged into the application.

- **Title:** Validate user input for creating items
- **User Story Number:** AGDA-136
- **User Story:** As a user I would like user input to be validated when creating an item so that values are consistent throughout documents.
- **Acceptance Criteria:**
    - Assignee number and display name should be visible
    - Due date should be correct format
- **Title:** Profile Picture upload
- **User Story Number:** AGDA-144
- **User Story:** As a user I would like to be able to upload a custom profile picture so that other users can easily identify me as a person.
- **Acceptance Criteria:**
    - Uploading of images backed onto S3.
    - FE interface to allow upload selection.

- **Title:** Accurate Format for Generated PDFs
- **User Story Number:** AGDA-145
- **User Story:** As a user I would like PDFs to look more like the memorandums we use within the department for greater accuracy
- **Acceptance Criteria:**
    - The PDFs have the MDC logo
    - The pdfs have the correct table format and structure as a regular memorandum

PROJECT PLAN

This section describes the planning that went into the realization of this project. This project incorporated the agile development techniques and as such required the sprints to be planned. These sprint plans are detailed in the section. This section also describes the components, both software and hardware, chosen for this project.

Hardware and Software Resources

The following is a list of all hardware and software resources that were used in this project:

Hardware	Software	
● AWS Servers ● ThinkPad x230 ● MacBook Pro 2013 ● MacBook Pro 2015 ● MacBook Pro 2017	● Jira 7.11.0 ● Confluence 6.10.0 ● Bitbucket 5.12.0 ● Git 2.19 ● Hipchat 2.4 ● Slack ● Google Hangouts 1.2 ● Gmail ● MacOS ● Windows 10 ● Linux Ubuntu 16.04	● React 16.5.0 ● React Router 4.3.1 ● Axios 0.18.0 ● Bootstrap 4.1.3 ● JSS 9.8.7 ● JQuery 3.3.1 ● NodeJS 11.2 ● Express 4.16.3 ● PostgreSQL 10 ● AWS Services ○ RDS ○ S3 ● AWS SDK 2.342.0 ● PDFkit 0.8.3 ● PG 7.5.0 ● Bcrypt 3.0.2

Sprint Plans

Sprint 1:

For this sprint, the team did a lot of set up work, in addition to a lot of requirements gathering. The project being a 1.0 meant that there was no immediate development work to be done. Figuring out what the product owner's wanted was key for the first few weeks. We held 10 daily scrum meetings, as well our planning, review, and retrospective meetings. Below are the assigned tasks that were related to getting the project started. We had 10 work days that ranged from August 31, 2018, and September 13, 2018.

Assigned Tasks and User Stories:

Type: ID	**Title**
Task: AGDA-7	Define key terms
Task: AGDA-55	Map out user interactions via preliminary User Story Mapping

Task: AGDA-8	Attend meeting with Arleene
Task: AGDA-10	Identify possible file types
Task: AGDA-15	Utilize the blueprint
Task: AGDA-25	Find a sufficient diagram tool
Task: AGDA-38	Define front end library
Task: AGDA-43	Source code version control
Task: AGDA-47	Look at LaserFiche
Task: AGDA-6	Research and analyze uploaded flowcharts
Task: AGDA-9	Gather information about departments by meeting with them
Task: AGDA-11	Analyze previous matter example that passed
Task: AGDA-14	Research possible design patterns to apply

Sprint 2:

The second sprint of Version 1.0 of Agenda Management System consisted of six tasks and one story. The table below can be used as a reference to the tasks and story we as a team set out to accomplish in a two week sprint that started on September 16 and ended on September 27. Since this was version 1.0 of the application our product owners where still getting us up to speed with legislative vocabulary and some of the workflows of how matter items are created and where some of these items might end up. In summary we kept on researching and analyzing information provided my MDC ITD and kept discussing the potential technology tools we would be using for the development of the application.

Assigned Tasks and User Stories

Type: ID	**Title**
Task: AGDA-6	Research and analyze uploaded flowcharts
Task: AGDA-9	Gather information about departments by meeting with them
Task: AGDA-11	Analyze previous matter example that passed
Task: AGDA-41	Cloud based services
Story: AGDA-42	Document tools
Task: AGDA-45	Testing framework

Task: AGDA-55	Map out user interactions via preliminary User Story Mapping

Sprint 3:

This Sprint was focused on the architecture and design of our project. Therefore the majority of work items were tasks, with a few user stories as development had begun. Below are the assigned tasks and user stories for Sprint 3. There were 10 work days this Sprint, from September 30th, 2018 to October 11, 2018.

Assigned Tasks and User Stories:

Type: ID	**Title**
Task: AGDA-6	Research and analyze uploaded flowcharts
Task: AGDA-9	Gather information about departments by meeting with them
Task: AGDA-14	Research possible design patterns to apply
Task: AGDA-27	API route definitions
Task: AGDA-28	Connect to data store
Task: AGDA-32	Wireframes for content
Task: AGDA-33	Define theme/palette
Task: AGDA-35	Mockup on design tool
Task: AGDA-39	Data store and model
Task: AGDA-46	Pick a UI framework
Task: AGDA-11	Analyze previous matter example that passed
Task: AGDA-26	User privilege + tokens
Task: AGDA-36	Analyze current legislative portal for design conventions
Task: AGDA-37	Define reusable components
Task: AGDA-40	Current user propagated throughout application
Task: AGDA-44	Continuous Integration / Continuous

	Deployment
User Story: AGDA-72	Matter Version Control
User Story: AGDA-81	Item feed
User Story: AGDA-82	Basic info fields
User Story: AGDA-84	Navigation Bar
User Story: AGDA-94	Define procurement template
Task: AGDA-119	Web route definitions

Sprint 4:

This Sprint was mostly focused on the front-end development of the project. Therefore, the majority of work items consisted of user stories, with a few tasks that were based on refining back-end elements. Below are the assigned tasks and user stories for Sprint 4. There were 10 work days for Sprint 4, ranging from October 14, 2018 to October 30, 2018.

Assigned Tasks and User Stories:

Type: ID	**Title**
Task: AGDA-31	Document-based storage linkage
Task: AGDA-37	Define reusable components
Task: AGDA-44	Matter version control
User Story: AGDA-75	Email and password fields
User Story: AGDA-82	Basic Info Fields for Profile
Task: AGDA-11	Analyze previous matter example that passed
Task: AGDA-26	User privilege + tokens
Task: AGDA-36	Analyze current legislative portal for design conventions
Task: AGDA-40	Continuous integration / Continuous deployment
User Story: AGDA-72	Item Feed - Back end
User Story: AGDA-81	Current user propagated throughout the application
User Story: AGDA-84	Navigation Bar

User Story: AGDA-85	Role Definition at a department level
User Story: AGDA-94	Template - Front end
User Story: AGDA-95	Define MOU
User Story: AGDA-96	Define Lease Agreement
User Story: AGDA-102	Display the fields of Item
Task: AGDA-119	Web route definitions

Sprint 5

This sprint focused on search capabilities, and viewing items, the personal item feed, and a few other functionalities and security checks. There were 10 work days this Sprint, from October 30th, 2018 to November 11, 2018.

Assigned Tasks and User Stories:

Type: ID	**Title**
User Story: AGDA-72	Item Feed - Back end
User Story: AGDA-83	Statistics Link
User Story: AGDA-85	Role Definition at a department level
User Story: AGDA-88	Search for an item by title
User Story: AGDA-89	Search for an item by matter subtype
User Story: AGDA-91	Search for an item by time to expiration
User Story: AGDA-93	Search for an item by contract number
User Story: AGDA-99	Set visibility based on users Permissions
User Story: AGDA-101	View Item History Trail with Downloadable Versions
User Story: AGDA-102	Display the fields of Item
User Story: AGDA-103	Permission Check
Task: AGDA-119	Web route definitions
User Story: AGDA-126	Advanced search page that offers filtering - Front end
User Story: AGDA-127	Item Feed - Front end

User Story: AGDA-133	Create a pdf or document from our item objects
Bug: AGDA-135	Search Item by active assignee works on one chrome version
User Story: AGDA-139	Edit an Item Page - Front End
User Story: AGDA-140	Edit/Update an Item - Backend
User Story: AGDA-141	Search for last user to edit file (mtlastuser)
Task: AGDA-142	Add upper and lower case sensitivity for advanced search
User User Story: AGDA-72	Item Feed - Back end

Sprint 6:

This sprint focused on search capabilities and version control for items. The majority of the work is represented in user stories and there were otherwise few bugs and tasks. There were 10 days of work in the sprint, from November 11 to November 25.

Assigned Tasks and User Stories:

Type: ID	**Title**
User Story: AGDA-72	Item Feed - Back end
User Story: AGDA-83	Statistics Link
User Story: AGDA-85	Role Definition at a department level
User Story: AGDA-88	Search for an item by title
User Story: AGDA-89	Search for an item by matter subtype
User Story: AGDA-91	Search for an item by time to expiration
User Story: AGDA-93	Search for an item by contract number
User Story: AGDA-99	Set visibility based on users Permissions
User Story: AGDA-101	View Item History Trail with Downloadable Versions
User Story: AGDA-102	Display the fields of Item
User Story: AGDA-103	Permission Check
Task: AGDA-119	Web route definitions
User Story: AGDA-126	Advanced search page that offers filtering - Front end

User Story: AGDA-127	Item Feed - Front end
User Story: AGDA-133	Create a pdf or document from our item objects
Bug: AGDA-135	Search Item by active assignee works on one chrome version
User Story: AGDA-139	Edit an Item Page - Front End
User Story: AGDA-140	Edit/Update an Item - Backend
User Story: AGDA-141	Search for last user to edit file (mtlastuser)
Task: AGDA-142	Add upper and lower case sensitivity for advanced search
User User Story: AGDA-72	Item Feed - Back end

SYSTEM DESIGN

This section contains information on the design decisions that went into this project. The architecture patterns are outlined and explained. The entire system is shown in a package diagram and the subsystems are explained. Finally, the design patterns used in the project are discussed.

Architectural Patterns

Two patterns that were applied when architecting the application were event-driven (EDA) and model-view-controller (MVC). An initial iteration of this application uses simple EDA to drive visual changes within the application. Based on the users interactions with the application different pages may be displayed differently. Eventually our EDA should be converted to use the redux framework where all changes to state will be in a unary directional flow with a central store to manage all the application state. With actions that will detect a state change and reducers that will take care of dispatching new state changes.

As for our MVC approach, we decided to separate our user interface code from any logical code. This was done to decouple components in our application. This allowed for high cohesion and low coupling of our application so that we could reuse our components across the application.

System and Subsystem Decomposition

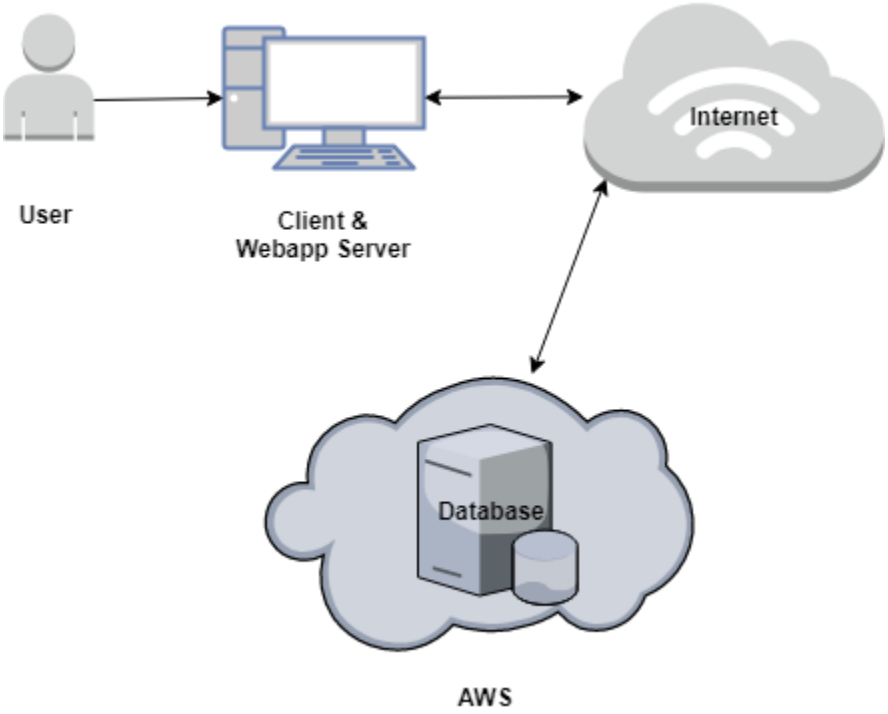
The system and subsystem decomposition diagram can be referenced to get a clearer picture as to how the application is structured. In essence, the application is a 2-for-1 full stack application that utilizes a backend server as well as a front end compilation through use of Express.js and React JS, respectively. The server harbors all the API routes and endpoints that are linked up to the front end via Axios function calls. The API calls enable a user to retrieve information from the database, if they are logged in.

The application is structured as follows: The server portion of the application lies in the root of the project. Everything is served out from server.js. The server.js file makes reference to the routes directory, which defines the endpoints of the API and their respective controller functions, which lie in the controllers directory. Our application makes use of yarn as the package manager for all the dependencies. The package manager reads out of the package.json file that outlines the metadata and dependencies of our application. We have 2 configuration files that lie in the root of the project as well, which are used to provide the credentials to the S3 bucket and the RDS instance.

The entire front end lies within the client directory. Similarly to the root of the project, this directory contains a package.json that contains all the front end dependencies, which we manage with yarn. The components for react all live within the src directory. The main entry point of the application's front end is App.js, which in turn renders the appropriate page based on the user's browser location. All the page components are within the pages directory, global components are within the components directory, the routes that a role has access to are defined in the roles directory, and the assets (such as css or images) are within the assets directory.

To further illustrate how the components interact with each other, refer to the appendix for the front end diagrams.

Deployment Diagram



Design Patterns

Similar to an MVC design pattern, our project's design separates views and behaviors contained in the client folder. The client directory contains many component folders where both a view and a container component are located. The view components, named in the style of 'NameV.js', are responsible for the user interface, receiving props with data that are used for rendering the view. The container components, named in the style of 'NameC.js', are responsible for fetching data from the backend, dispatching actions that make changes to application state, and other logic that then is rendered with the view component.

The server directory is the root directory, therefore containing the client folder. It contains two main directories including a routes folder and a controllers folder. The routes folder is for defining the API endpoints, where a route such as '/item' would be contained within the item.js file. The controllers folder defines the logic that handles the requests and responses from a route. For instance, there's the 'itemController.js' file which contains controller functions such as 'item_view_get()' which makes an API GET request from route '/item/:id' to the database when called. The resulting data, if any, is fetched from the front end controller components.

SYSTEM VALIDATION

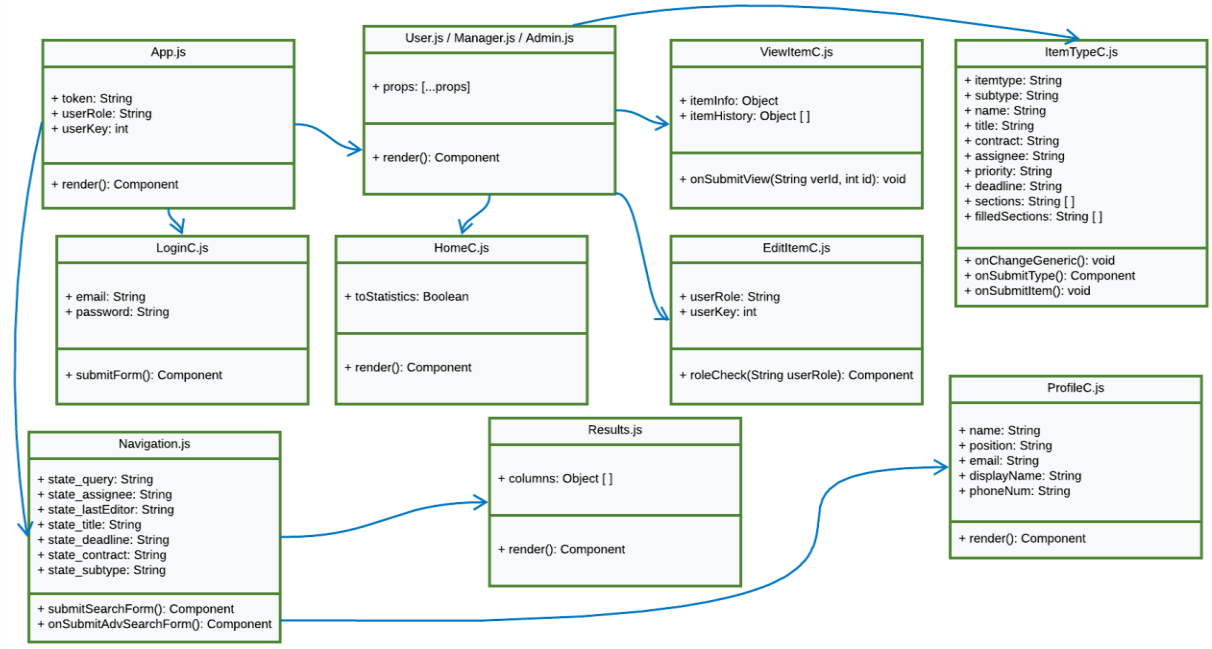
In order to validate the system, we did unit testing on the user stories each developer worked on. The unit testing consisted of sunny day/rainy day testing. Sunny day is where we expect the best possible situation in the user interaction with the component. The developers tested the components and recorded the actual results with the expected results. The same goes with rainy day testing, where the worst possible outcome expected happens if a user interacts with the component. Actual results were recorded against expected results.

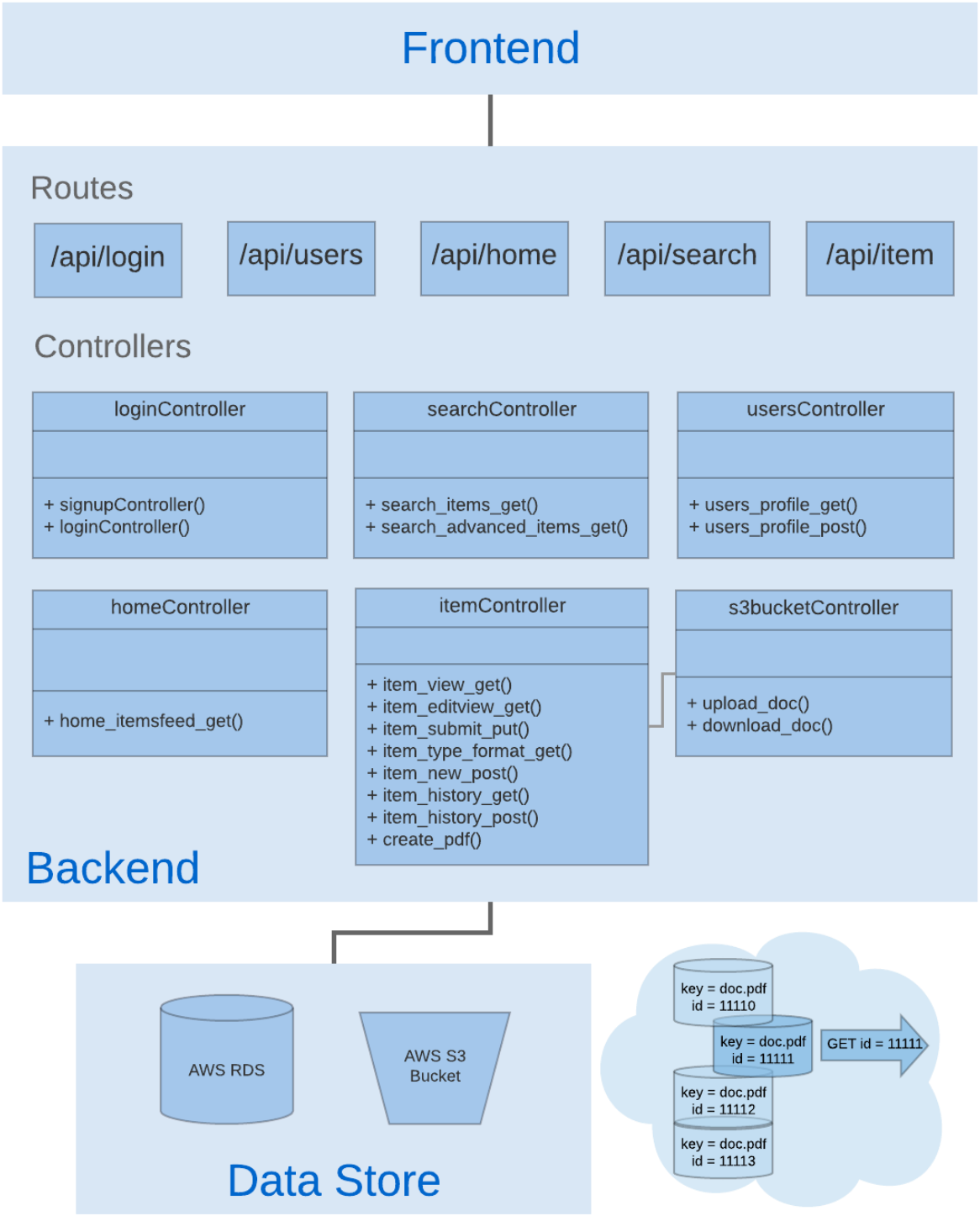
GLOSSARY

APPENDIX

Appendix A - UML Diagrams

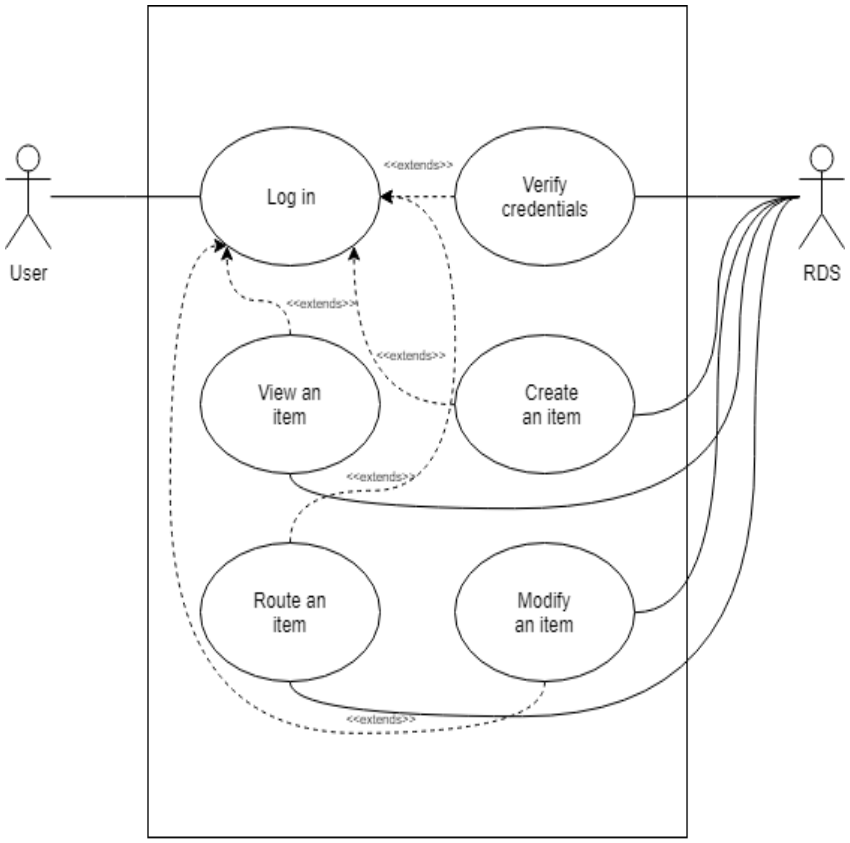
I. System and Class Diagrams


App.js
+ token: String
+ userRole: String
+ userKey: int
+ render(): Component
User.js / Manager.js / Admin.js
+ props: [...props]
+ render(): Component
ViewItemC.js
+ itemInfo: Object
+ itemHistory: Object [ ]
+ onSubmitView(String verId, int id): void
ItemTypeC.js
+ itemtype: String
+ subtype: String
+ name: String
+ title: String
+ contract: String
+ assignee: String
+ priority: String
+ deadline: String
+ sections: String [ ]
+ filledSections: String [ ]
+ onChangeGeneric(): void
+ onSubmitType(): Component
+ onSubmitItem(): void
LoginC.js
+ email: String
+ password: String
+ submitForm(): Component
HomeC.js
+ toStatistics: Boolean
+ render(): Component
EditItemC.js
+ userRole: String
+ userKey: int
+ roleCheck(String userRole): Component
ProfileC.js
+ name: String
+ position: String
+ email: String
+ displayName: String
+ phoneNum: String
+ render(): Component
Navigation.js
+ state_query: String
+ state_assignee: String
+ state_lastEditor: String
+ state_title: String
+ state_deadline: String
+ state_contract: String
+ state_subtype: String
+ submitSearchForm(): Component
+ onSubmitAdvSearchForm(): Component
Results.js
+ columns: Object [ ]
+ render(): Component

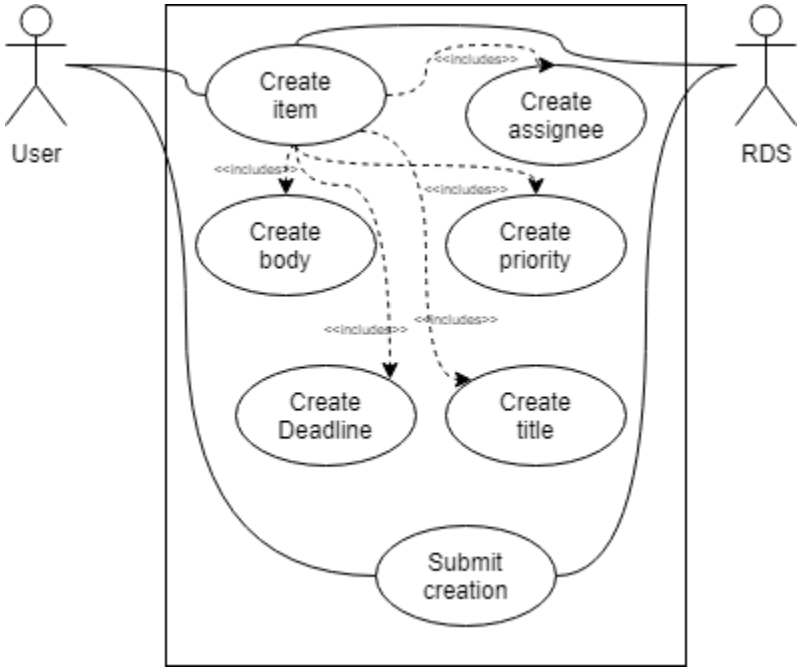

Frontend
Routes
/api/login
/api/users
/api/home
/api/search
/api/item
Controllers
loginController
+ signupController()
+ loginController()
searchController
+ search_items_get()
+ search_advanced_items_get()
usersController
+ users_profile_get()
+ users_profile_post()
homeController
+ home_itemsfeed_get()
itemController
+ item_view_get()
+ item_editview_get()
+ item_submit_put()
+ item_type_format_get()
+ item_new_post()
+ item_history_get()
+ item_history_post()
+ create_pdf()
s3bucketController
+ upload_doc()
+ download_doc()
Backend
AWS RDS
AWS S3 Bucket
Data Store
key = doc.pdf
id = 11110
key = doc.pdf
id = 11111
GET id = 11111
key = doc.pdf
id = 11112
key = doc.pdf
id = 11113

II. Use Case story Diagrams

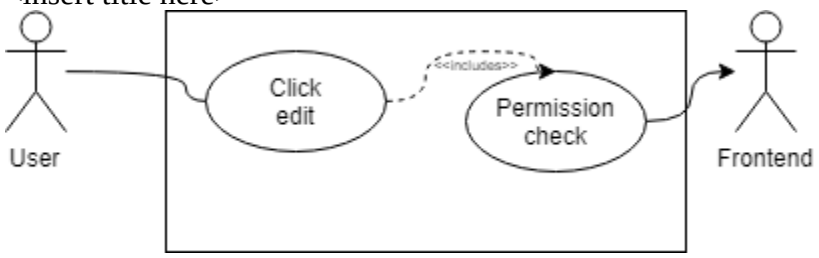
<insert title here>



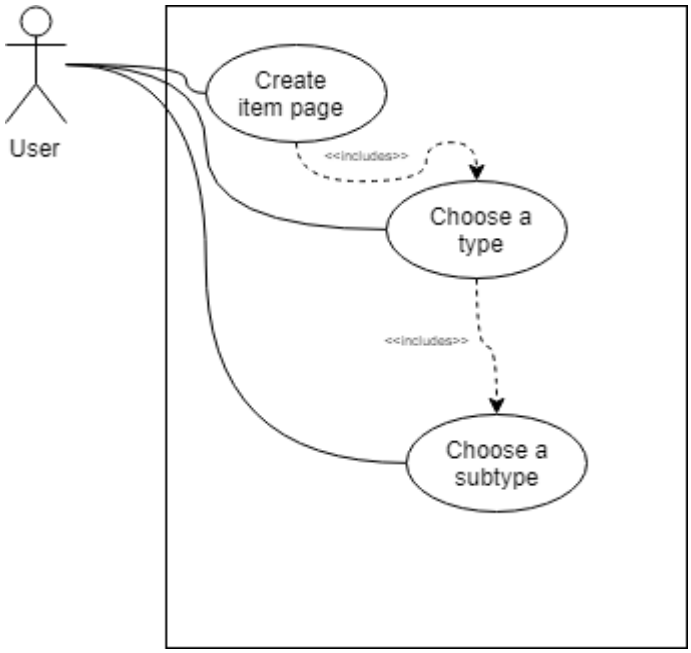
<insert title here>



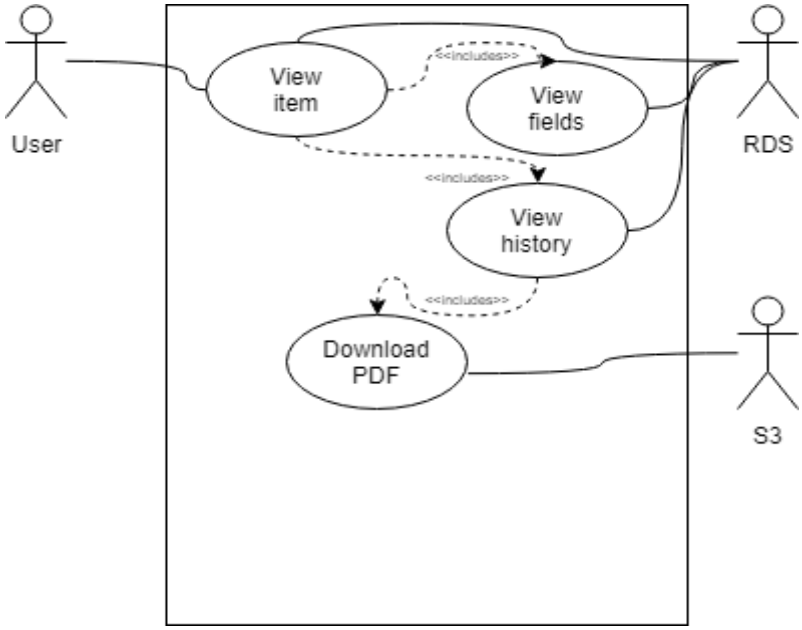
<insert title here>



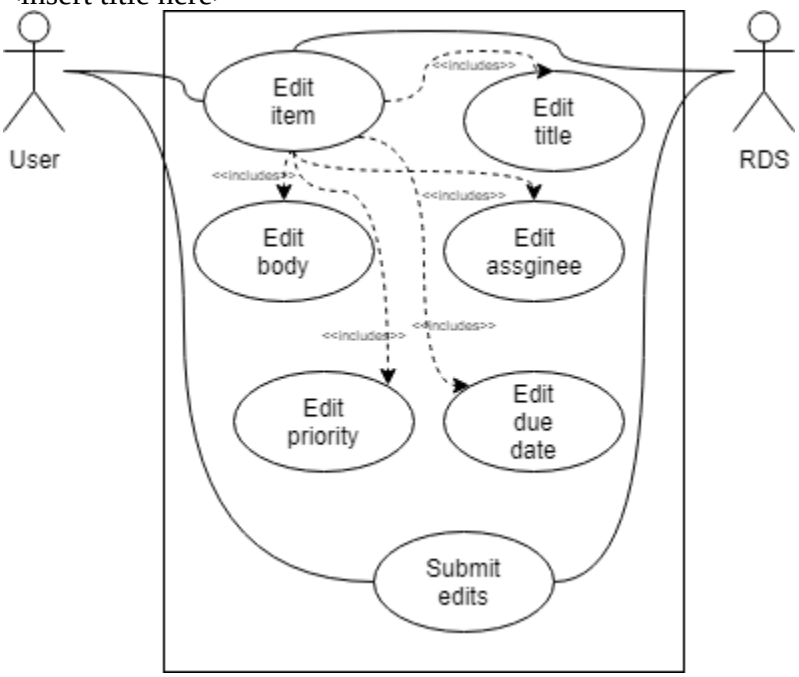
<insert title here>



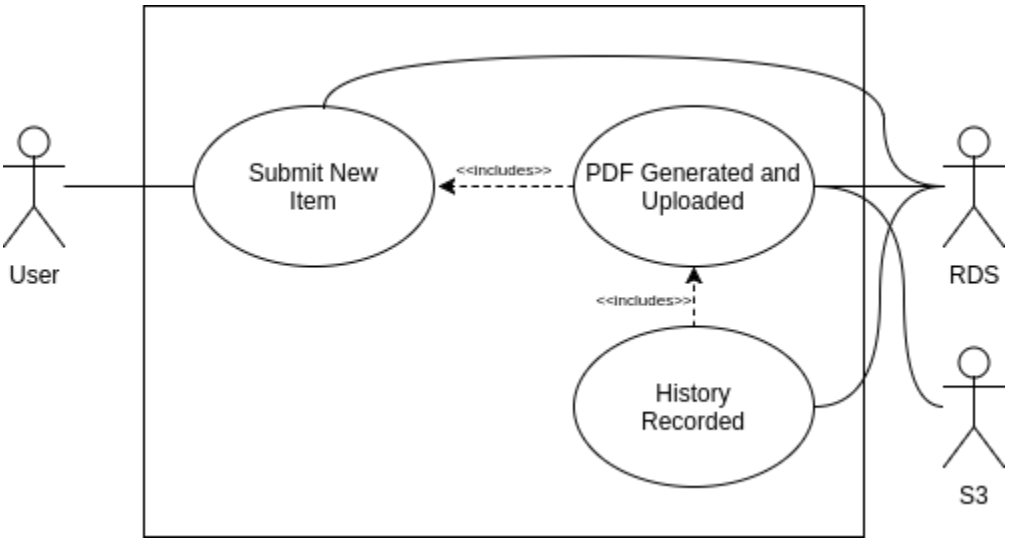
<insert title here>



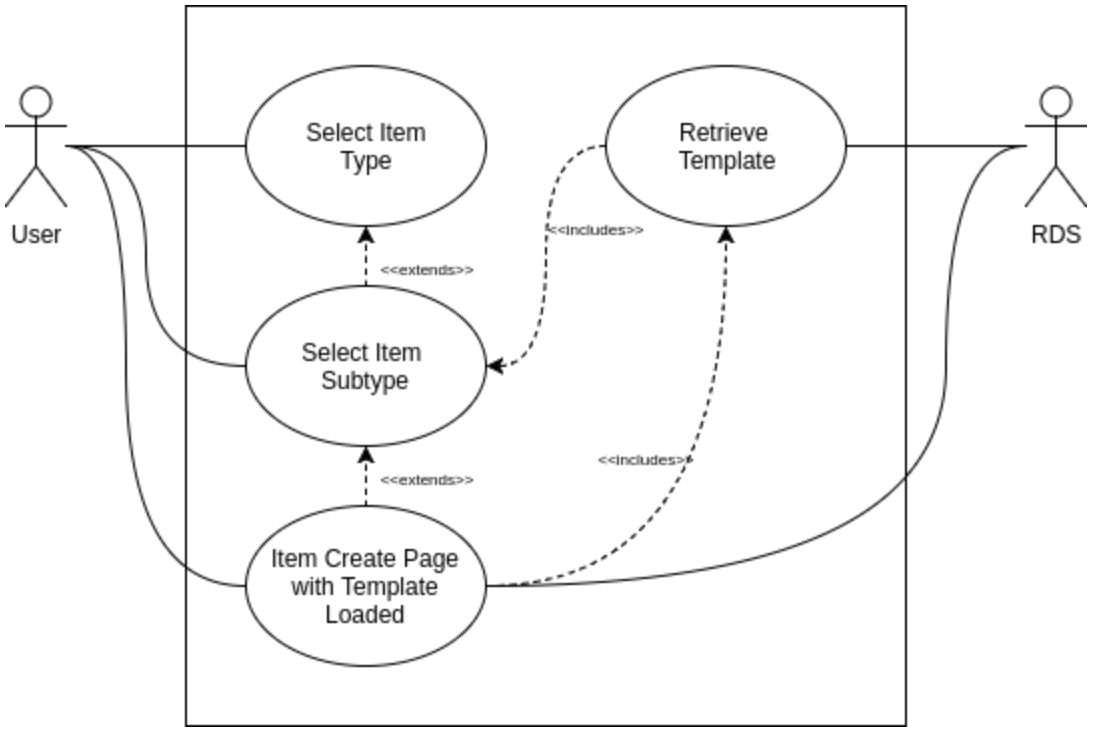
<insert title here>



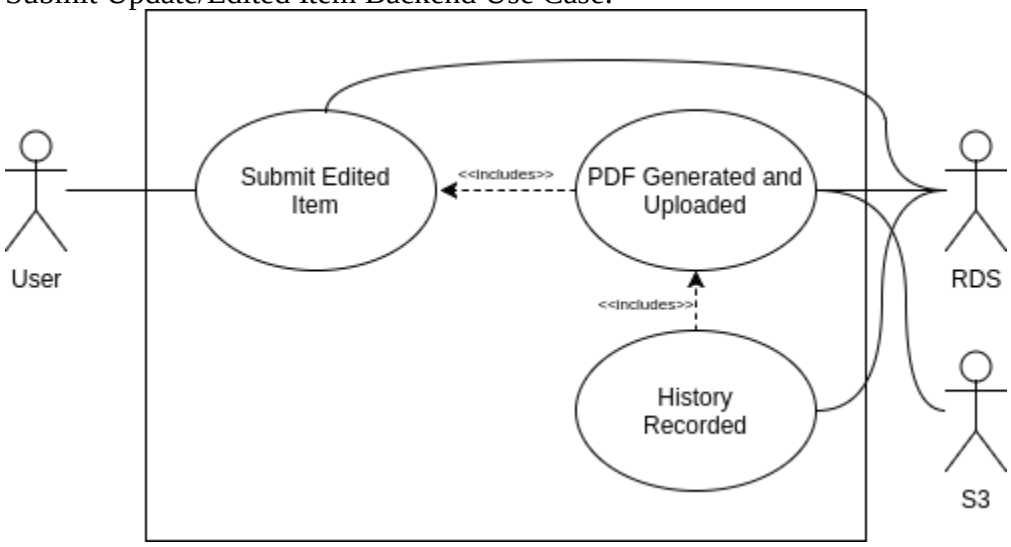
Submit New Item Backend Use Case:



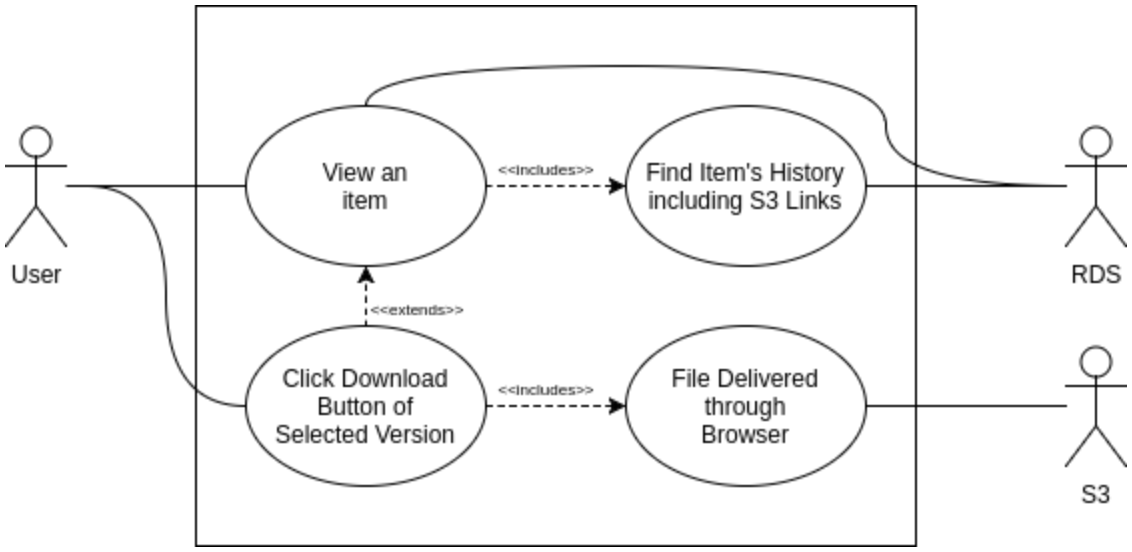
Choose Type and Subtype Backend Use Case:



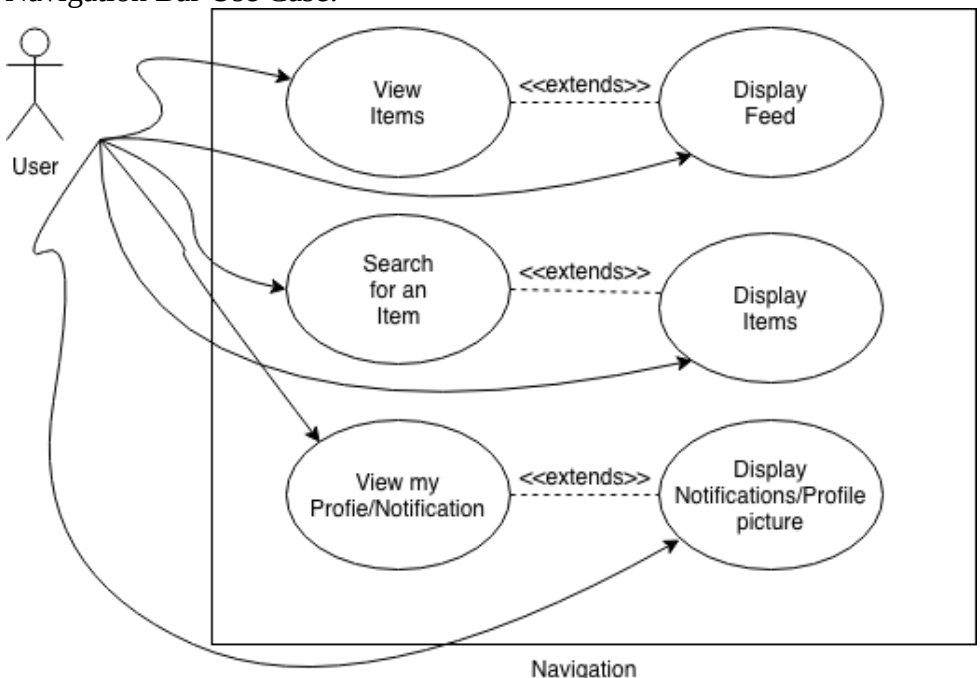
Submit Update/Edited Item Backend Use Case:



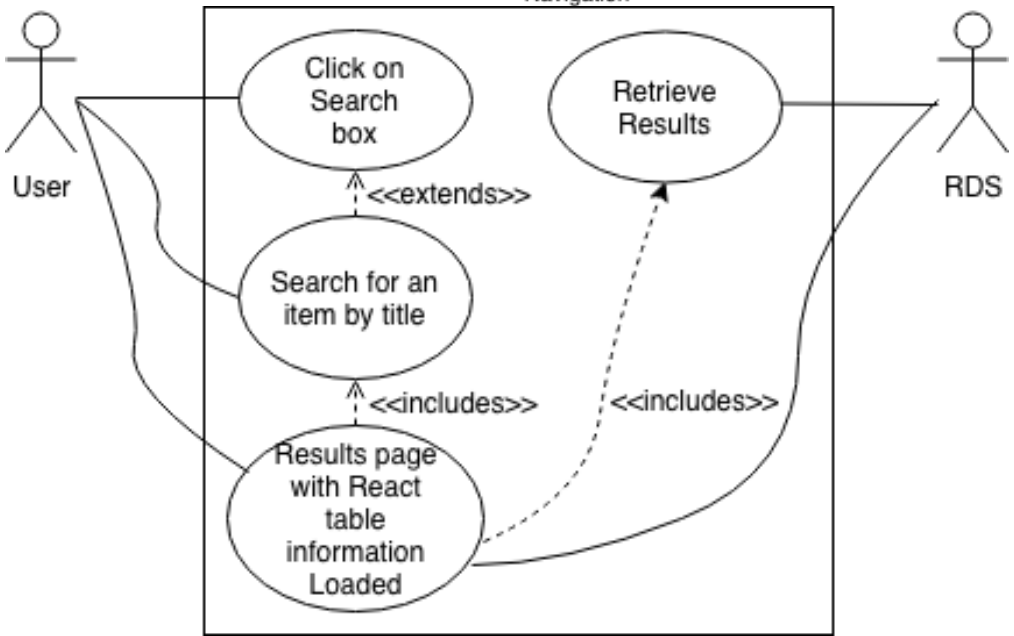
View an Item's History and Download Document Version Use Case:



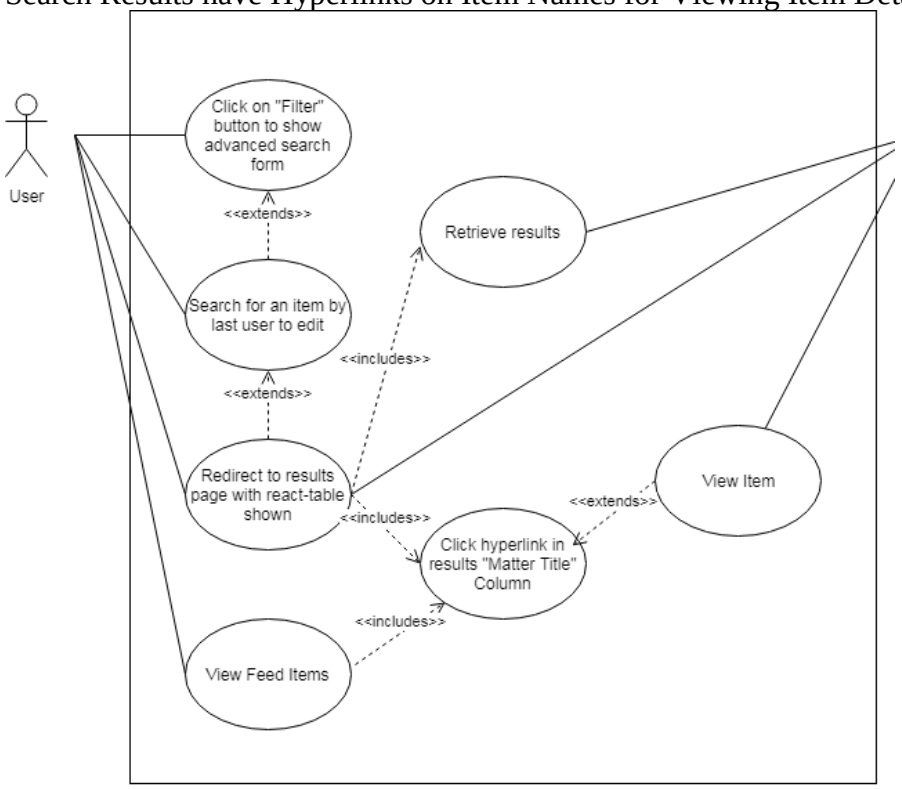
Navigation Bar Use Case:



Search Agenda Item Use Case:

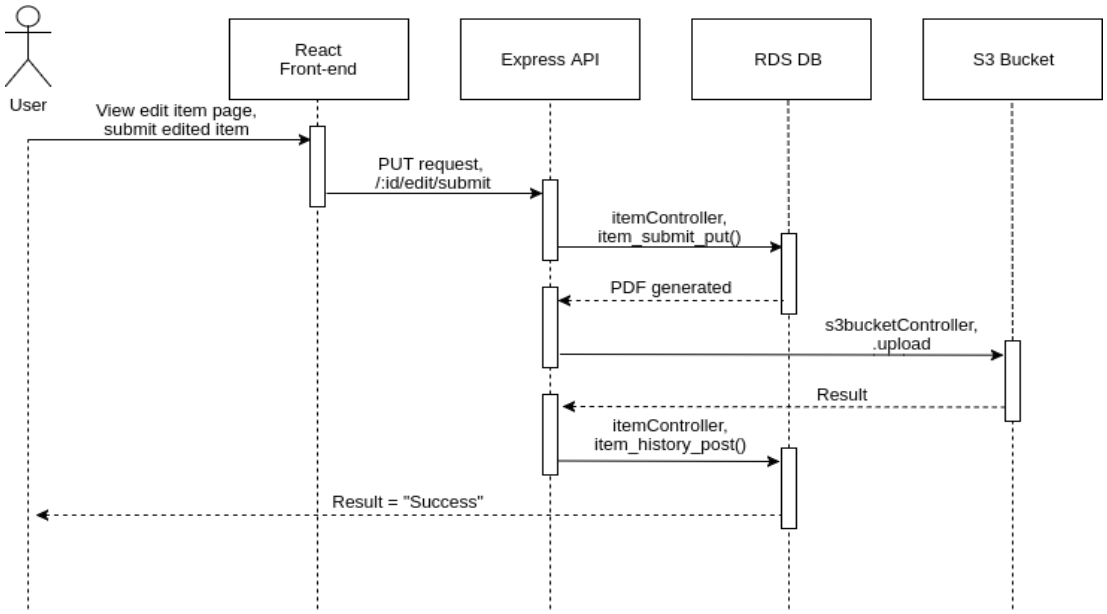


Search Results have Hyperlinks on Item Names for Viewing Item Details Use Case

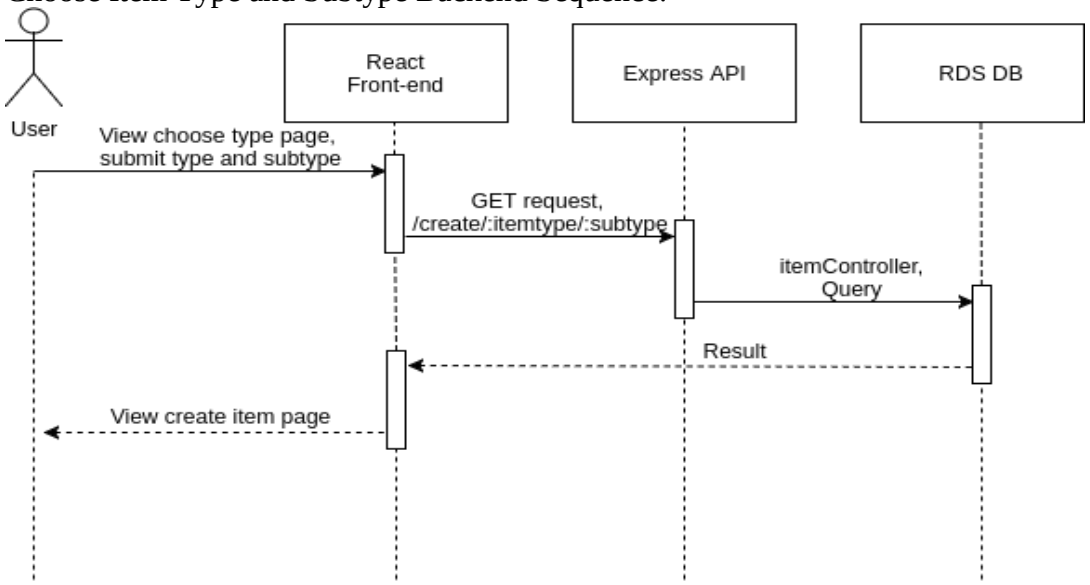


III. Sequence Diagrams

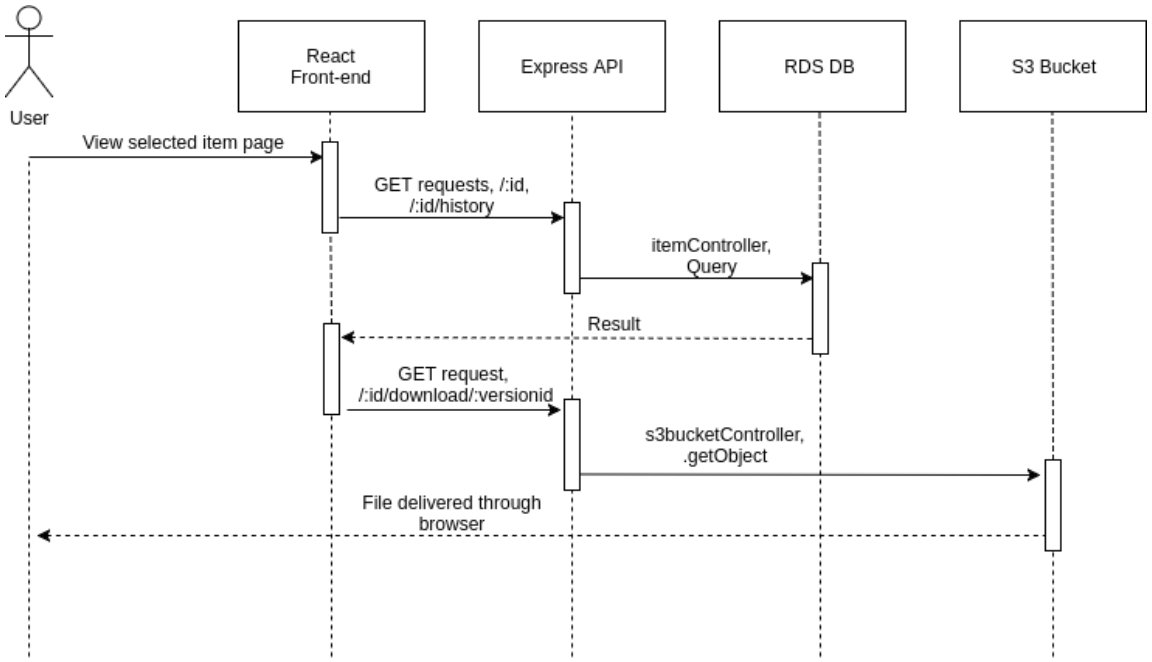
Edit/Update an Item Backend Sequence:



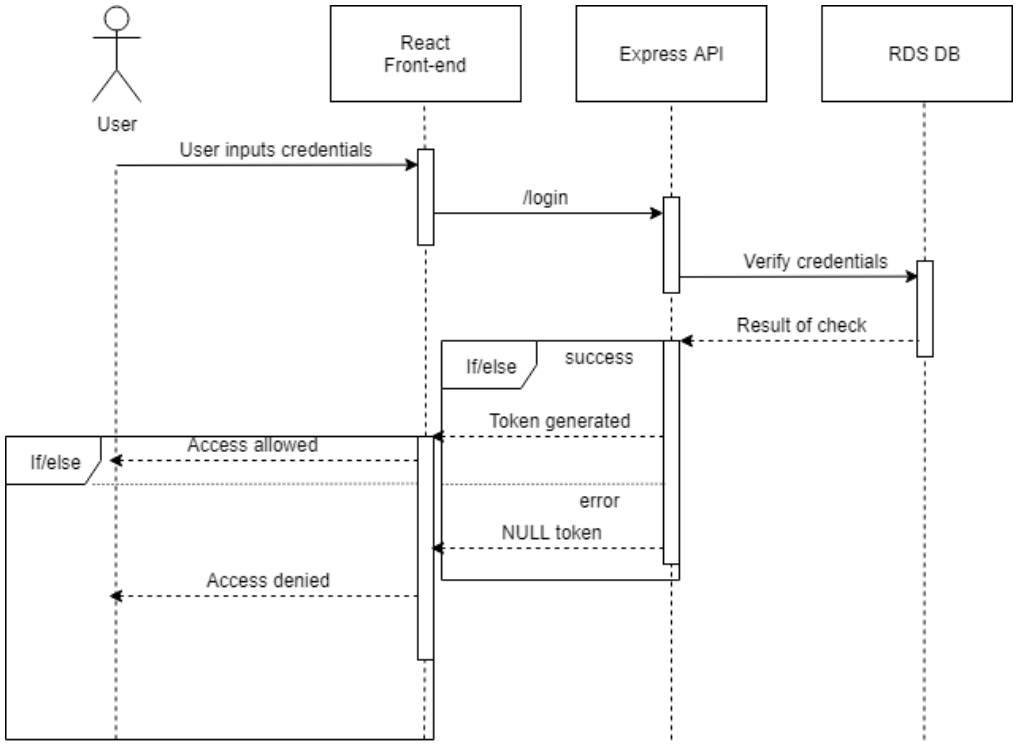
Choose Item Type and Subtype Backend Sequence:



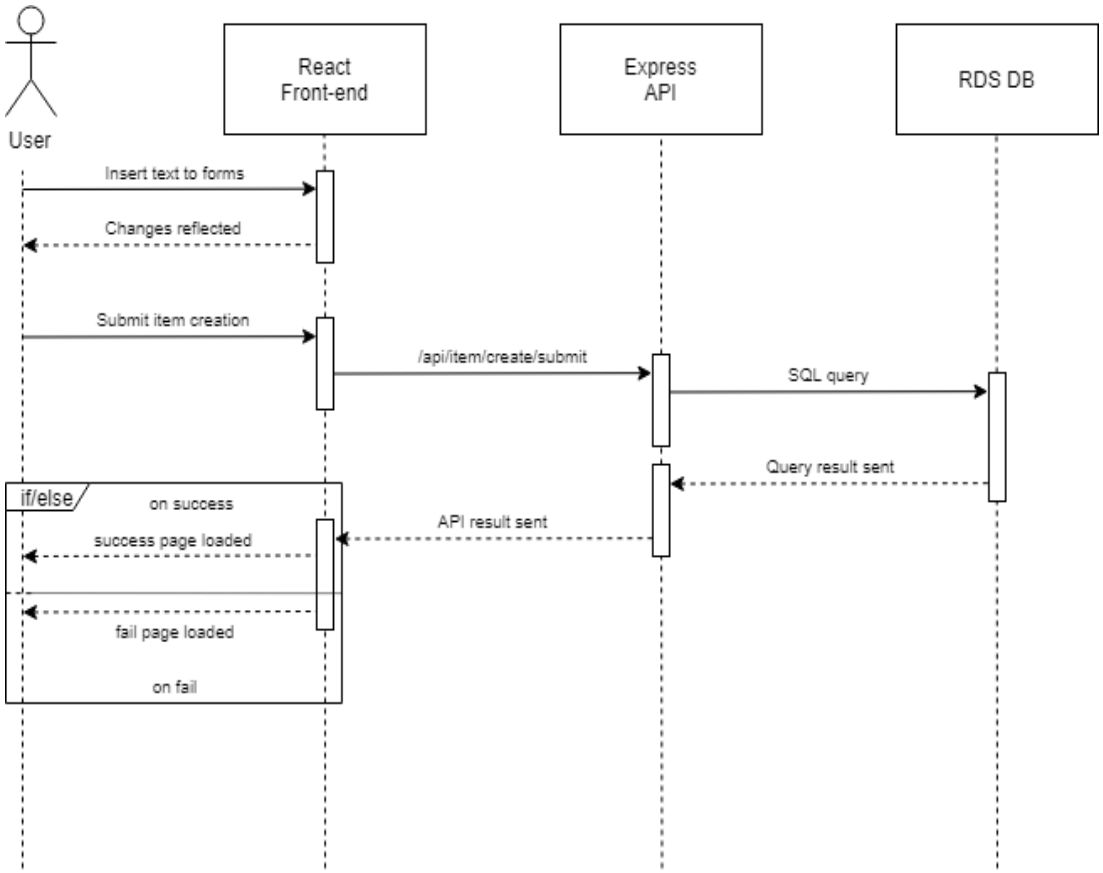
View an Item Backend Sequence:



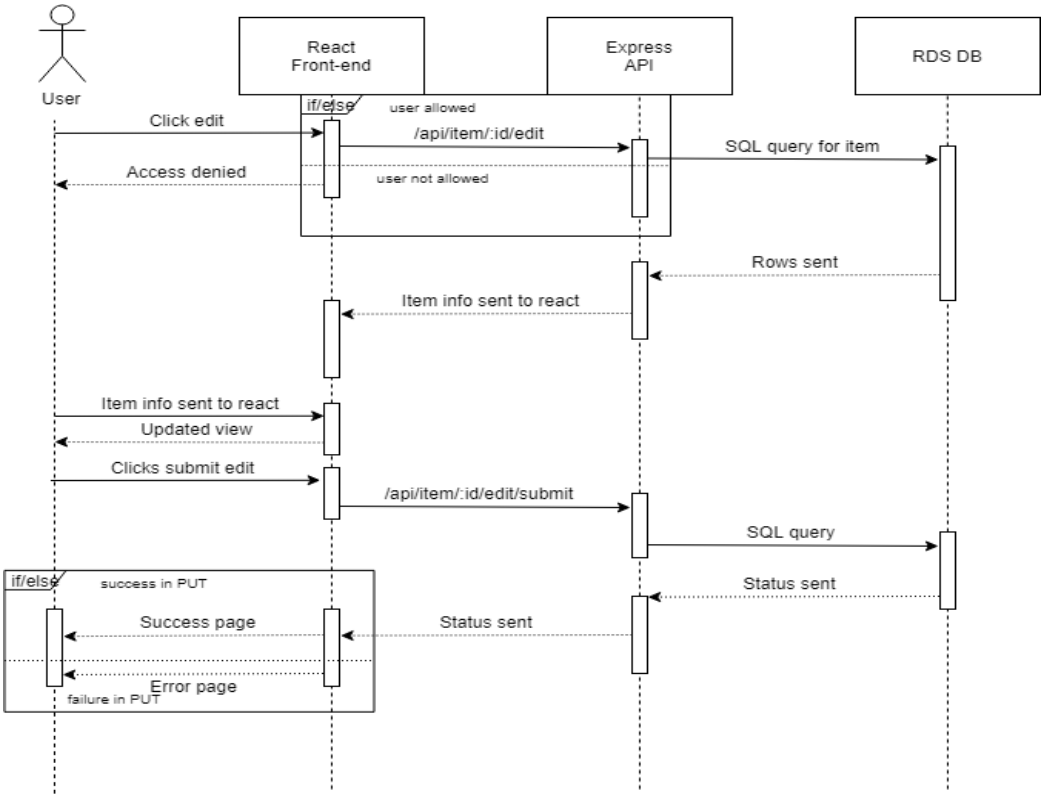
<insert title here>



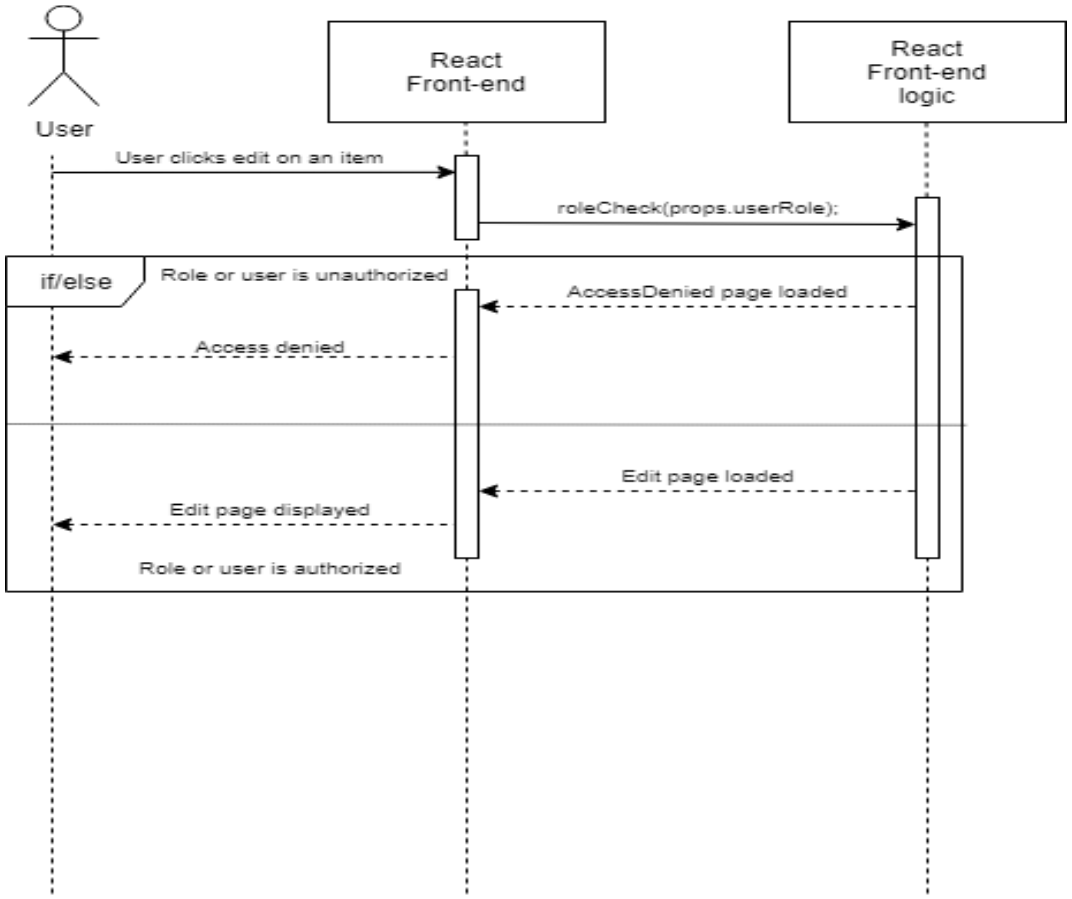
<insert title here



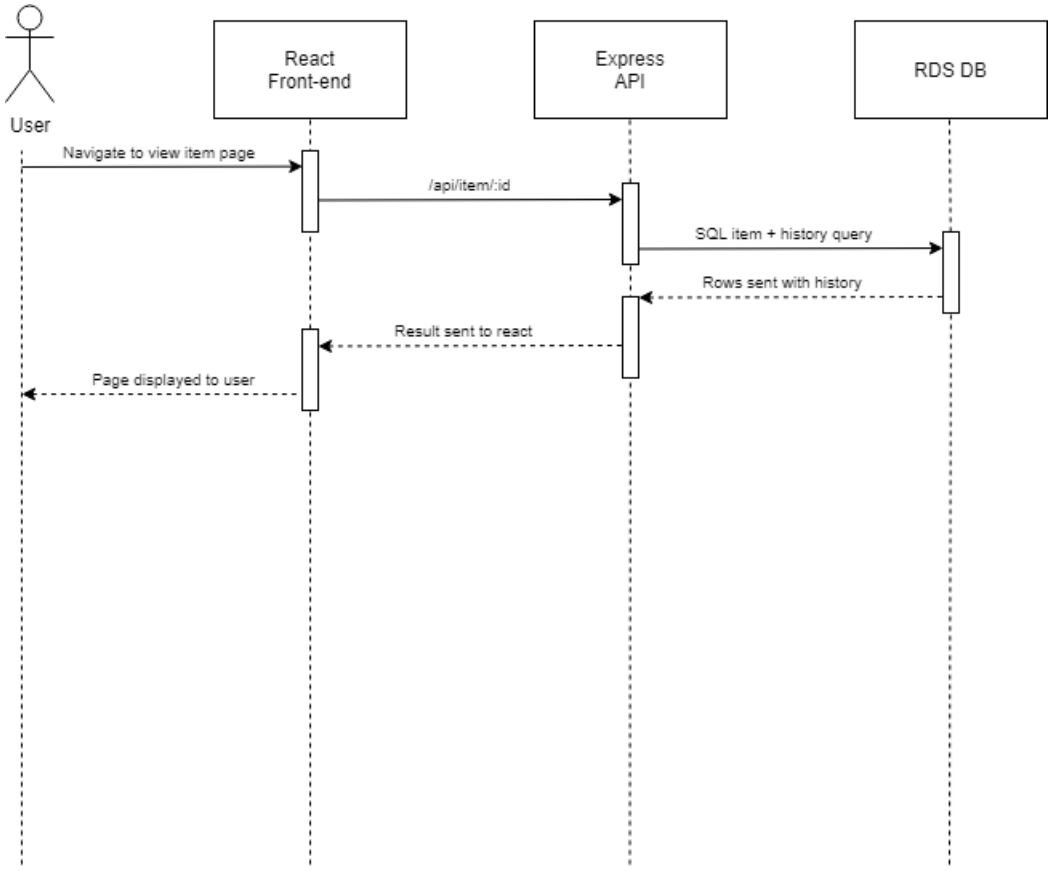
<insert title here>



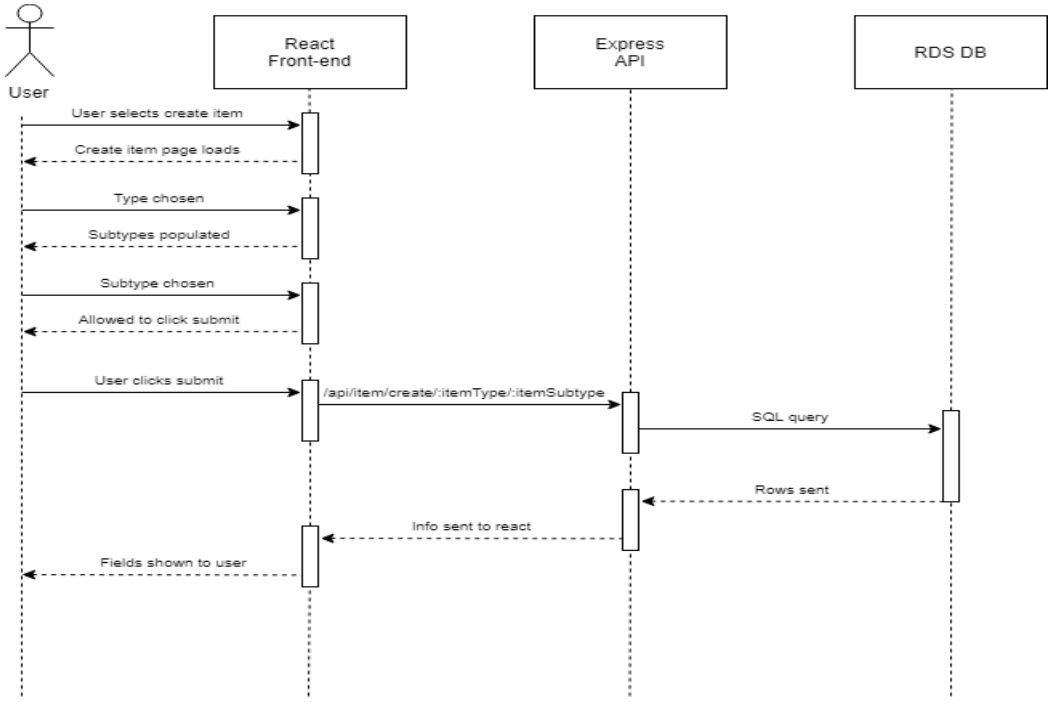
<insert title here>



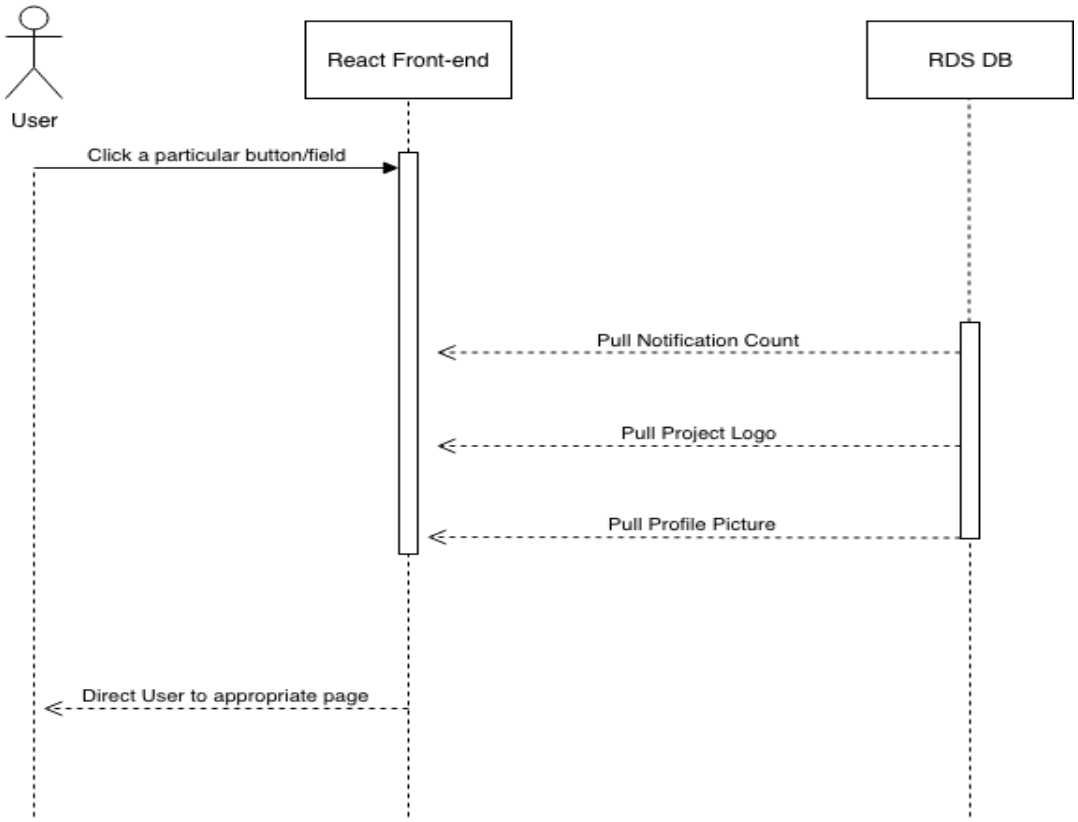
<insert title here>



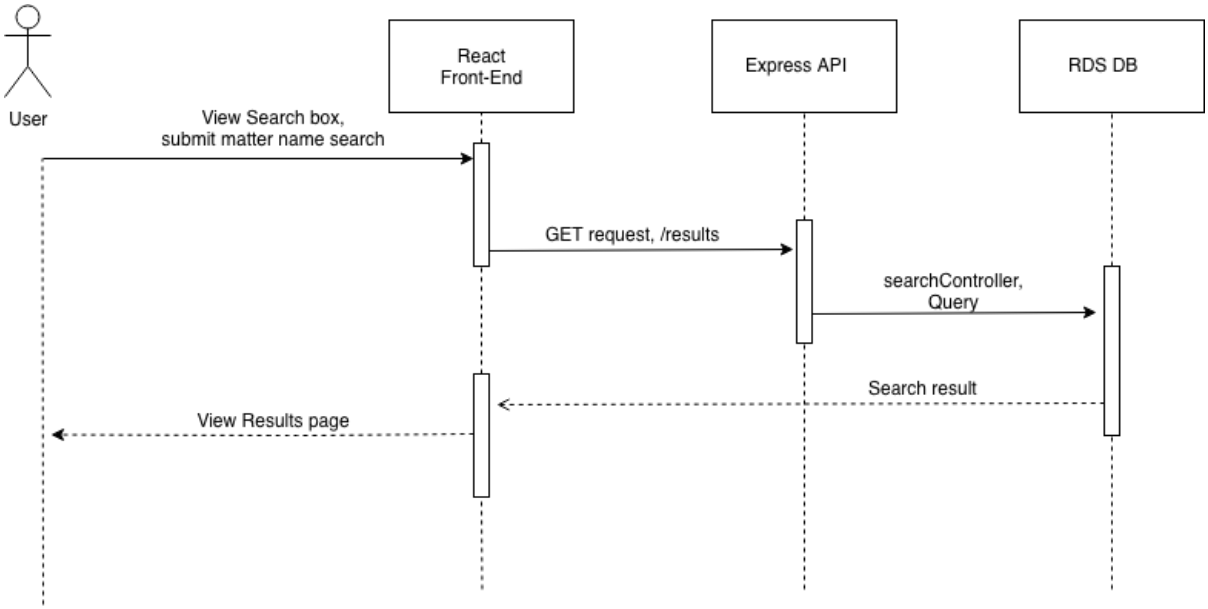
<insert title here>



Navigation Bar Sequence Diagram:

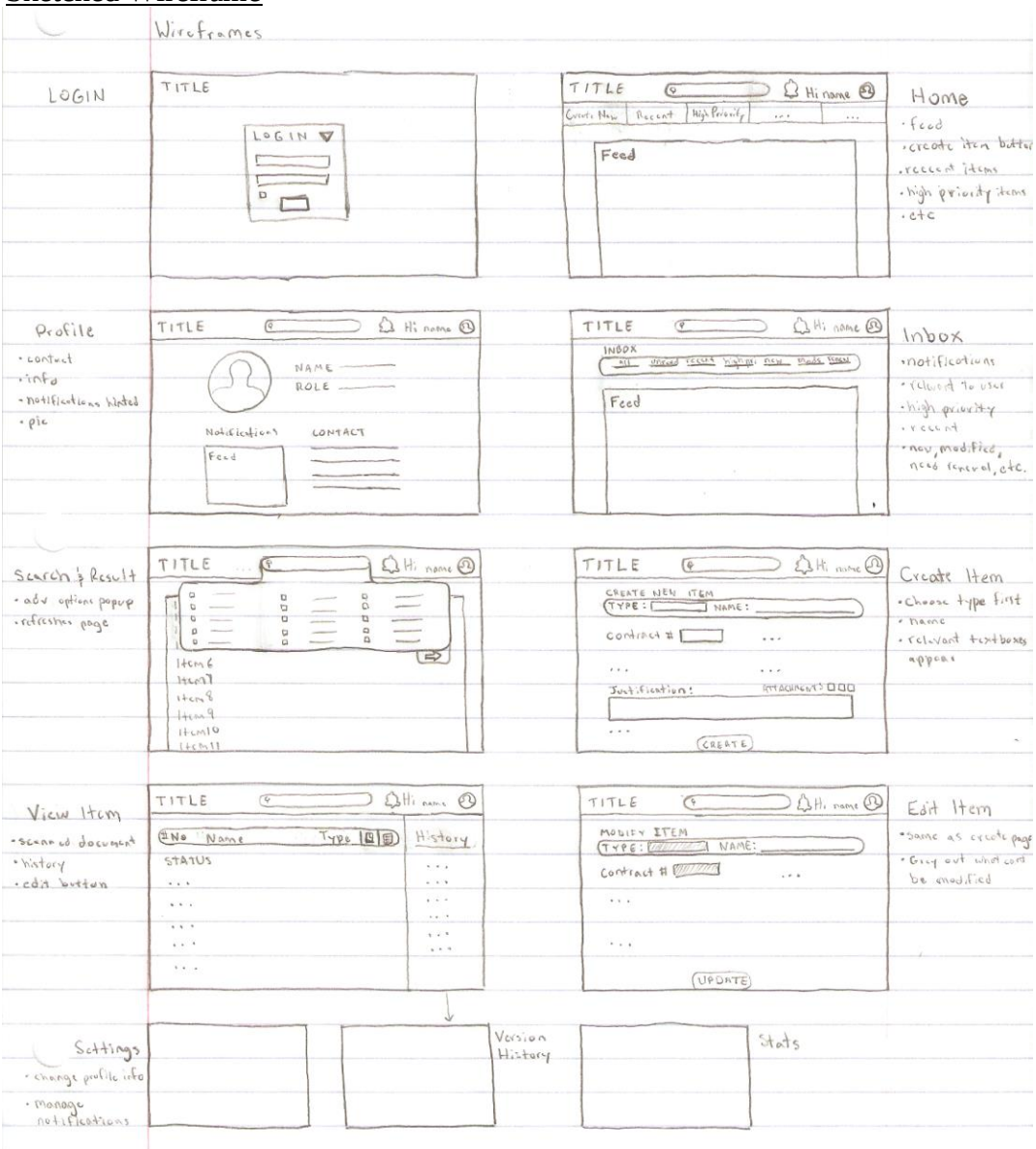


Searching for an Item Sequence Diagram:

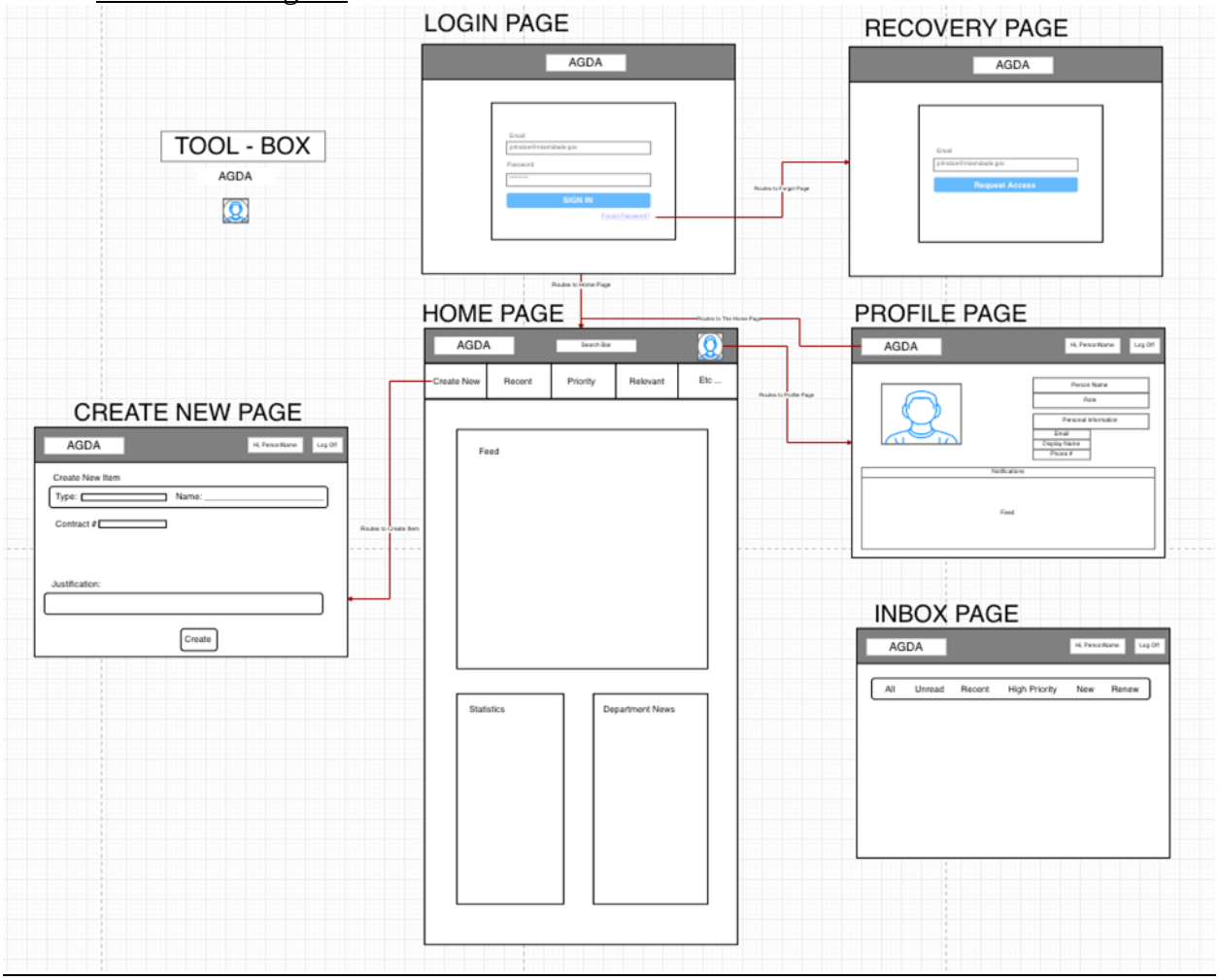


Appendix B - User Interface Design

I. Sketched Wireframe



II. Wireframe Diagram



Appendix C - Sprint Review Reports

Sprint 1:

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: All.

Since no explicit development was done, as per the instructions of the product team, no show and tell was done for features.

Instead, our current understanding of their system was showcased, and we received feedback for that. All tasks for sprint 1 were accepted, the ones that were not completed were moved to sprint 2.

Synopsis of meeting:

Arleene and Margaret went over our assignment, which at this point is to focus on the legislative process within the department in the county. We have tentative meetings with the Information Technology Department, the Water & Sewer Department, and Aviation.

Arleene went over a ton of legislative item types that are created within the Information Technology Department, and helped us refine our existing flowchart that was created from research and analyze uploaded flowcharts.

In addition to this, the development team gained insights on the contents of an item, and how it's reflected when it becomes a memorandum.

The tasks that were moved to sprint 2 are the following:

- - Research and analyze uploaded flowcharts
    - We decided that we needed to understand the flow more clearly, so we decided to keep this in progress.
- - Gather information about departments by meeting with them
    - Gathering information about specific items within different departments was the clear direction the team had given us in regards to our next sprint.
- - Analyze previous matter example that passed
    - Analyzing a previous matter example that passed is necessary to understand how an item transforms as it is created and refined up until it gets sent to the Mayor's office.
- - Research possible design patterns to apply
    - Finding the appropriate design patterns or conventions we would be using for the software solution is still in progress.

<u>Sprint 2:</u>

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: All.

It was at this point, after Jay got back from vacation, and we were all able to have a meeting together, that we were on the same page. We could begin development starting sprint 3.

- Task, Cloud based services: AGDA-41

- - Cloud based services

- Task, Document tools: AGDA-42
    - - Document tools
- Task, Testing framework: AGDA-45
    - - Testing framework
- Task, Map out user interactions via preliminary User Story Mapping: AGDA-55
    - - Map out user interactions via preliminary User Story Mapping

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Sprint Planning meeting.

No tasks were rejected and moved back into the backlog.

Sprint 3:

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: .

- Current user propagated throughout the application

Instead, our current understanding of their system was showcased, and we received feedback for that.

Static web app shown to Jay Alvarez de la Campa and further discussion about more functionalities to implement. Jay asked to include stories for tracking user activity.

Work completed in Sprint 3:

- - Research and analyze uploaded flowcharts
- - Gather information about departments by meeting with them
- - Research possible design patterns to apply
- - Connect to data store
- - Wireframes for content
- - Define theme/palette
- - Site map/hierarchy

- - Mockup on design tool
- - Data store
- - Pick a UI Framework

The tasks that were moved to sprint 4 are the following:

- - Analyze previous matter example that passed
- - User privilege + tokens
- - Analyze current legislative portal for design conventions
- - Define reusable components
- - Continuous integration / Continuous deployment
- - Matter version control
- - Item Feed - Back end
- - Current user propagated throughout the application
- - Basic Info Fields for Profile
- - Navigation Bar
- - Template - Front end
- - Web route definitions

Sprint 4:

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners:

- Task, Document-based storage linkage: AGDA-31
    - - Document-based storage linkage
- Task, Define reusable components: AGDA-37
    - - Define reusable components
- User Story, Email and password fields: AGDA-75

- - Email and password fields

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Sprint Planning meeting.

- User Story, Basic Info Fields: AGDA-82
    - Basic Info Fields for Profile
- User Story, Navigation bar: AGDA-84
    - - Navigation Bar
- User Story, Item feed: AGDA-72
    - - Item Feed - Back end
- User Story, Current user propagated throughout the application: AGDA-81
    - - Current user propagated throughout the application
- Task, User privilege + token: AGDA-26
    - - User privilege + tokens
- Task, Web route definitions: AGDA-119
    - - Web route definitions
- Task, Continuous integration / continuous deployment: AGDA-40
    - - Continuous integration / Continuous deployment
- Task, Analyze previous matter example that passed: AGDA-11
    - - Analyze previous matter example that passed

Sprint 5:

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: All.

- User Story, Current user propagated throughout the application: AGDA-81

    - - Current user propagated throughout the application

- User Story, Navigation bar: AGDA-84
    - - Navigation Bar
- User Story, Creating an item - choosing type - backend: AGDA-129
    - - Choose Type and Subtype for Item Creation - Backend
- User Story, Creating an item - submitting the fields - backend: AGDA-130
    - - Creating an Item by Submitting the Fields - Backend
- User Story, Creating an item - choosing a type - front end: AGDA-131
    - - Choose Type and Subtype for Item Creation - Frontend
- User Story, Creating an item - submitting the fields - front end: AGDA-132
    - - Creating an Item Page with Submission Fields - Frontend
- User Story, Define procurement elements from glossary: AGDA-134
    - - Define Procurement Elements from Glossary
- User Story, Template - Front end: AGDA-94
    - - Template - Front end
- User Story, Search for an item by ID: AGDA-86
    - - Search for an item by ID
- User Story, Item Feed - Back End: AGDA-72
    - - Item Feed - Back end
- User Story, Search for an item by active assignee: AGDA-90
    - - Search for an item by active assignee
- User Story, Search for an item by title: AGDA-88
    - - Search for an item by title

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Sprint Planning meeting.

- User Story, Role definition at a department level: AGDA-85
    - - Role Definition at a department level
- User Story, Create a pdf or document from our item objects: AGDA-133
    - - Create a pdf or document from our item objects
- User Story, Item Feed - Front end: AGDA-127
    - - Item Feed - Front end
- User Story, Display the fields of Item: AGDA-102
    - - Display the fields of Item
- User Story, Search for an item by money value: AGDA-93
    - - Search for an item by contract number
- User Story, Search for an item by recents: AGDA-92
    - - Search for an item by recents ı
- User Story, Search for an item by time to expiration: AGDA-91
    - - Search for an item by time to expiration
- User Story, Search for an item by requester: AGDA-89
    - - Search for an item by matter subtype

Sprint 6:

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: All.

- User Story - Item Feed - Back end
- User Story - Statistics Link
- User Story - Role Definition at a department level

- User Story - Search for an item by title
- User Story - Search for an item by matter subtype
- User Story - Search for an item by time to expiration
- User Story - Search for an item by contract number
- User Story - Set visibility based on users Permissions
- User Story - View Item History Trail with Downloadable Versions
- User Story - Display the fields of Item
- User Story - Permission Check
- User Story - Web route definitions
- User Story - Advanced search page that offers filtering - Front end
- User Story - Create a pdf or document from our item objects
- User Story - Search Item by active assignee works on one chrome version
- User Story - Edit an Item Page - Front End
- User Story - Edit/Update an Item - Backend
- User Story - Search for last user to edit file (mtlastuser)
- User Story - Add upper and lower case sensitivity for advanced search

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Sprint Planning meeting.

- User Story - Item Feed - Front end
- User Story - Validate user input for creating items

Appendix D

User Manuals

With the current version of the application, all users have the same functionality.

A user logs into the application with valid credentials. Upon logging into the application, a user is greeted with the home screen. A navigation bar at the top of the page allows a user to view their profile, make searches, and create items. The home screen shows a feed of items relevant to a user.

Logging in
Start the application. Enter valid credentials and click submit. If valid, you will be taken to the homepage.

Searching for an item
Click the textbox at the top of the navigation bar. Enter an item name and press enter to see results of items that match the name.
A user can also click filter to search on more parameters. A modal pops up with more text boxes. After filling these out, click search. All relevant results appear.

Viewing an item
An item can be viewed by clicking the item name in the feed or results page of a search.

Editing an item
To edit an item, navigate to its view item page. Click the edit button to edit the item. Only a user with Manager or Admin privileges can edit all items. A user with regular permissions can only edit items that they themselves have created, or items they are assigned.
After editing an item by modifying the text boxes, click submit. A page showing the result of the edit will be shown.

Viewing the history of an item
Navigate to the view item page. The history is shown on the right side of the page. Click the download button to download a PDF of that item at that point in time.

Creating an item
Click the create item button on the navigation bar. Select a type and subtype, and click submit. A form pops up with all the fields to fill out. Ensure that you have a proper format for the date. Ensure that the assignee is an integer. Click submit to create the item. A result page will be shown that says the status of the users actions (success/fail).

Viewing user profile
Click the arrow on the navbar next to the image, and click profile. Here a user can edit their information.

Logging out
Click the arrow on the navbar next to the image, and click logout. You will be taken to the login page.

Installation/Maintenance Document

- Prerequisites Are that

- You need GIT installed on your computer
  - https://git-scm.com/
- You need to install yarn or npm
  - https://yarnpkg.com/en/
  - https://www.npmjs.com/
- As a Side Note
  - We choose to use yarn.
  - You should also note that you shouldn't use yarn and npm at the same time so never have a (package-lock.json) and (yarn.lock) file at the same time.
  - The following steps where installed on a Macbook Pro using Visual Studio Code. But the steps are as generic as possible that you should be able to do the same on any operating system.

- STEP 1
  - Open up Visual Studio Code and used the integrated command line to access the file where you will like to clone the project into.
    - TYPE → cd Desktop
- STEP 2
  - Visit the bitbucket project's repository and copy the HTTP cloneable link to download the file.
    - NOTE → This link might change with time
- STEP 3
  - Use Visual Studio Code integrated command line to clone the project into your desired directory
    - TYPE → git clone https://sp-jira.cis.fiu.edu:7443/scm/agda/agenda-management.git
- STEP 4
  - Now that the project has been cloned, go ahead and open it as well in Visual Studio Code to view the file structure.
  - Inside the directory named Website/ you'll have to add two files named
    - database-info.json
    - s3bucket-info.json
  - Example and explanation of these files will be found inside of the Database folder which will be inside of the Code directory.
- STEP 5
  - Now that the project has been clone and the two json files have been added to the directory named Website/ we will need to access that directory in the integrated command line
    - TYPE → cd agenda-management/Code/Website/
- STEP 6
  - Once you're inside the Website directory of the project you'll need to install the backend node dependencies.
    - TYPE → yarn install
  - Once the backend node dependencies have been installed, you'll need to access the frontend code and install those dependencies as well

        - TYPE → cd client
        - TYPE → yarn install
- STEP 7
    - Once the frontend and backend node_modules have been installed, you'll need to go back up a directory and start the application locally so that you can view it.
        - TYPE → yarn dev

For the Maintenance portion of this document please refer to the video portion of Maintenance, the user story videos to get a more in-depth explanation of the features we installed, or the README file.

Shortcomings and Wish List Document

Both wishlist and shortcomings of the Agenda Management System are largely embodied in the documentation as JIRA user stories and tasks.

Broadly speaking, the functionalities that we would have liked to add but could not allocate the time for are the following: The logic for a "forgot password" scenario, A notifications/inbox system for users to see messages and high-importance items, viewing diffs between the versions of items, better-formatted PDFs for saved item versions, automated re-routing (aka re-assignment) of items, and a dashboard for showing several statistics (such as rate of users' activites and other statistics about items).

The shortcomings of our system include: The only item/matter types being handled are procurements, currently there is no general-user role (in other words, a user that can only access but not change data), client-side validation for form field inputs, autocomplete handling for formfields, and migrating server-level SQL queries onto stored procedures on the database server.

Database and Document Storage Document

Quick overview of the AWS services we used or planned to use:

- We highly recommend following any AWS basics tutorial if no experience. ACloudGuru's is popular, but most others are enough to get started.
- Create AWS Free Tier account using credit card credentials.
- We recommend AWS Elastic Beanstalk (Includes EC2, RDS, S3, and more) if you expect to host the server on the cloud (EC2). Elastic Beanstalk would automate the setup of many of the AWS services.
- Descriptions of the most relevant services.
    - DATABASE: AWS RDS for PostgreSQL database.
        - Most recent version.
        - t2.micro instance class is cheapest and sufficient.
        - Will need username, password, db name, and db endpoint for connection with server.
        - Make publically accessible for simplified setup.
    - DOCUMENT STORAGE: AWS S3 bucket for storing documents (PDFs) of items on the cloud.

        - Select S3 Standard. Keeps objects durable, stable, and immediately available.
        - Enable version controlling, through AWS S3 selected bucket properties.
        - Will need an Access Key and Security Key generated through AWS IAM
        - Make publically accessible for simplified setup.
    - AWS IAM (Identity and Access Management) for permissions and access among team members and AWS resources.
        - Create IAM user accounts and passwords with rights to access AWS services like S3 and RDS for all developers.
        - You don't want to login as root account frequently.
        - Activate multi-factor authentication (with Google Authenticator on phone) on the root account for more security.
        - Save root login information somewhere secure.
        - Create account alias so that console login is simple. e.g. fiu-agda.signin.aws.amazon.com/console
        - Make sure every user is aware of their region. They may not be able to locate a RDS DB if on wrong region for instance.
        - Generate Access Key ID and Secret Access Key for at least 1 user to use for S3 bucket access.
    - AWS CloudWatch to monitor and set alerts for AWS resources to avoid unexpected charges.
        - AWS requires credit card information to signup therefore you should set email billing alerts.
        - Review up-to-date Free Trial account limitations.
    - AWS Route 53 can be used to purchase domain names once you're ready to deploy.
    - AWS EC2 (Elastic Compute Cloud) servers on the cloud using virtual machines.

Once AWS Services (RDS & S3 specifically) are prepared:

- CREATE TABLES ON DATABASE:
    - Open PGADMIN (or similar software), connect to RDS DB.
        - Hostname is the DB Endpoint found on AWS RDS database view.
    - Use provided CREATE TABLE sql from 'agda-db-1.0.sql' to create the tables using the query tool.

- FOR ACCESS TO RDS DATABASE AND S3 BEFORE INSERTING DATA:
    - Modify 'database-info.json' in project folder with accurate information.

```
{
        "dbUser":"username",
        "dbHost":"endpoint.here.us-east-2.rds.amazonaws.com",
        "dbDatabase":"dbname",
        "dbPassword":"password",
        "dbPort":5432
```

```
}
```

- Modify 's3bucket-info.json' in project folder with the bucket name, IAM Access Key, and Secret Access Key

```
{
    "bucketName":"fiu-agda-bucket-name",
    "aKey":"accesskey",
    "secretKey":"secretaccesskey"
}
```

- IMPORTANT: Add both files to .gitignore as to keep that information secure.
- In the future, this information should be put... <where to put db info>

- ADD DATA TO DATABASE:
  - For TYPE and USERS table, use provided INSERT sql from 'insert.sql'
    - Open PGADMIN (or similar software), connect to RDS DB.
    - TYPE table values provided by MDC ITD.
    - Insert Users <Cristian>
  - For MATTER and HISTORY table, use POSTMAN to insert Matters
    - Reason: So entire process of generating pdf, uploading to s3, and history recorded is done
    - Open POSTMAN, import provided 'AGDA-POSTMAN.postman_collection.json' file
    - 'POST SUBMIT NEW ITEM' = POST for http://localhost:4000/api/item/create/submit to create item
    - Body, use raw JSON
    - Provided example JSON on 'insert.sql'
    - Another option is to just run the application and create/edit items.

- Database diagrams are available on Database/diagrams, including the schema and attribute domain.
- The database schema was based on the provided ITD schema, modified to the needs of this application.
- The database schema diagram was created with app.quickdatabasediagrams.com

REFERENCES

https://yarnpkg.com/ - Package manager. Runs on top of npm.
https://nodejs.org/en/ - Platform where all this runs on.
https://expressjs.com/ - Server framework. Definition of API and middleware.
https://www.npmjs.com/package/bcrypt - Used to encrypt and decrypt passwords.
https://node-postgres.com/ - Used to interface with the Postgresql database.
https://nodemon.io/ - Process manager to host the application.
https://www.npmjs.com/package/concurrently - Used to run processes in parallel.
http://pdfkit.org/docs/getting_started.html - Library to dynamically create PDFs.
https://babeljs.io/ - Transpiler to allow the use of ES6 syntax.

https://reactjs.org/ - Front end components, great for reusability and readability.
https://reacttraining.com/ - Lets the application become a single page application through use of front end routing.
https://www.npmjs.com/package/axios - HTTP Request library. Used to contact endpoints that our Express app defines.
https://getbootstrap.com/ - CSS framework.
https://webpack.js.org/ - Bundler.