**Technical Proposal**

**Information Management System**



**ATHEEB INTERGRAPH SAUDI CO.**

2/F Energy Center, Doha Road, P.O. Box 795, Al-Khobar 31952, Saudi Arabia

**Contents**

[Document History 3](#_Toc183417320)

[Definition/Acronyms 4](#_Toc183417321)

[1. Executive Summary 5](#_Toc183417322)

[1.1 Introduction 5](#_Toc183417323)

[1.2 About Atheeb Intergraph 5](#_Toc183417324)

[2. Goals and Objectives 6](#_Toc183417325)

[2.1 Purpose of the document 6](#_Toc183417326)

[2.2 Objectives 6](#_Toc183417327)

[3. System / Solution Overview 6](#_Toc183417328)

[3.1 Project Scope 7](#_Toc183417329)

[3.2 In-Scope Features 7](#_Toc183417330)

[3.3 Out-of-Scope Features 8](#_Toc183417331)

[4. Dashboard Management 8](#_Toc183417332)

[4.1 Approach and Methodology 8](#_Toc183417333)

[4.1.1 Approach 8](#_Toc183417334)

[4.1.2 Methodology 8](#_Toc183417335)

[4.2 Security Requirements 9](#_Toc183417336)

[4.2 System Architecture 9](#_Toc183417337)

[4.3 Data Model Definition 10](#_Toc183417338)

[4.4 System Integration 12](#_Toc183417339)

[4.5 Supported Protocols 12](#_Toc183417340)

# Document History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 30-August-2024 | 1 | Initial version | AISC |
|  |  |  |  |

Definition/Acronyms

|  |  |
| --- | --- |
| AISC | Atheeb Intergraph Saudi Co. |
| COMPANY | Saudi Aramco |
| AI | Artificial Intelligence |
| ATS | Automated Tracking System |
| CCTV | Closed Circuit Television |
| EMS | Environment Management System |
| HIP | Hazard Identification Plan |
| LoRa | Long Range Radio |
| PTW | Permit to Work |
| PWAS | Proximity Warning & Alarm System |
| RFID | Radio Frequency Identification |
| CR | Change Request – (Customized Functionality Changes) |
| SCC | Safety Command Centre |

# Executive Summary

## Introduction

Atheeb Intergraph Saudi Co. (hereunder referred as “AISC”) is pleased to submit this technical Proposal to for **AI Based Smart Solutions for Safety Monitoring and Enhancements**

## About Atheeb Intergraph

|  |  |
| --- | --- |
|  | **Atheeb Intergraph is part of Atheeb Holding Company.** Atheeb Holding is committed to the Kingdom of Saudi Arabia Vision 2030. |



**Atheeb Intergraph has a team of 250+ people in the Middle East, with offices in Riyadh, Khobar, Jeddah and Abu Dhabi.** The combined experience of the technical team is in excess of 1000 man-years -- all related to supporting projects in the Middle East. This experienced services capability allows Intergraph to take on a wide range of support and consulting activities for our clients in the Middle East.

The team in Middle East provides first line support for the complete range of Intergraph solutions. Where the local knowledge is not adequate, we are backed by the European regional support group, which ensures very rapid turnaround of problems by having a group to turn to within the Middle East time zone. Problems that are still unresolved at the regional level are then escalated to the global support organization in Huntsville, Alabama, U.S.A

Our large presence in the Middle East has allowed us to move beyond a technology supplier relationship, to establish a true partnership with this industry over the last few years. We have many hundreds of trained users, but in order to meet an increasing demand we have established local training program to better service our customers and to ensure that the growing demand for trained users can be met.

# Goals and Objectives

Large organizations face increasing challenges in managing growing volumes of information. In such a situation, there is a high possibility of inefficient handling of documents, inadequate search capabilities, and lack of centralized storage, which can lead to lost productivity, data redundancy, and potential compliance risks. To address these issues, there is a need for a robust Information Management System (IMS) that centralizes data storage, streamlines access, and enhances collaboration while maintaining security and compliance.

By implementing the IMS, the project seeks to create a unified platform that enhances productivity and supports informed decision-making through better information access and management.

## 2.1 Purpose of the document

This Functional Design Specification (FDS) outlines the design, features, and functionality of the proposed IMS. It aims to:

* Define the system's capabilities to meet the identified business needs.
* Provide a clear framework for development, testing, and implementation.
* Ensure alignment between stakeholders' expectations and the delivered solution.

By implementing the IMS, the project seeks to create a unified platform that enhances productivity and supports informed decision-making through better information access and management.

Included in this document will be the detailed functional requirements including use cases, system inputs and outputs, process flows, diagrams, and mock-ups.

## 2.2 Objectives

* Create a single source for organizational information.
* Minimize time spent searching for and organizing data.
* Ensure secure access to sensitive information.
* Automate routine processes, thereby improving operational efficiency.

# System / Solution Overview

The Information Management System (IMS) is a centralized software platform designed to store, organize, retrieve, and manage organizational data and documents efficiently. The system offers robust features for document management, user access control, advanced search capabilities, and analytics. It is intended to streamline information workflows, enhance collaboration, and ensure secure and compliant data handling.

## 3.1 Project Scope

The IMS software will consist of the following key modules:

**3.1.1** **Workflow Management System**: This will streamline business processes, task assignments, and approvals.

**3.1.2** **Document Management System**: This will allow secure document storage, sharing, and revision / version control.

**3.1.3** **Dashboard Interface**: This dashboard will handle user queries, extract required data from databases via RESTful APIs, and display data visualizations as requested.

## 3.2 In-Scope Features

The IMS will include the following functionalities:

**3.2.1 Document Storage and Retrieval**

* Text Search (google search)
* Quick Search - Based on Name & Title
* Advance Search
* Content search
* Content Extraction
* Bulk Load Documents
* Create Documents with Properties & Files
* AI-based Search Feature (eg: using ChatGPT-like prompts)

**3.2.2 Document Management**

* Tag Categorization
* Document Categorization

**3.2.3 Workflow Management**

* Role-based Creation / Approval /
* Reject
* Graphical Viewing of Workflow
* Reports & Alerts

**3.2.4 Revision Control**

* Tracking Document Changes and History
* Ability to revert to previous revisions

**3.2.5 Access Control and Permissions**

* Role-based access control
* User authentication

**3.2.6 Dashboard**

* Workflow-related Graphical Report and Analytics
* User authentication

**3.2.7 Integration**

* Integration with other systems

## 3.3 Out-of-Scope Features

The following are outside the scope of this document and the initial system release:

* Integration with third-party tools (e.g., CRM, ERP systems) beyond basic API support.
* Extensive offline capabilities or support for standalone desktop versions.
* Hardware procurement or IT infrastructure setup for the deployment environment.

# Dashboard Management

## 4.1 Approach and Methodology

This section outlines the systematic approach and methodology to be adopted for the successful development, implementation, and maintenance of the IMS.

## 4.1.1 Approach

**4.1.1.1 User-Centric Design**: Prioritize usability through prototypes/wireframes.

**4.1.1.2 Modular Development**: Use a modular architecture for easy scalability and integration with other systems.

**4.1.1.3 Integration-First Strategy**: Ensure seamless connectivity with third-party systems, live data sources, RESTful APIs, and SQL databases.

## 4.1.2 Methodology

**4.1.2.1 Requirement Analysis:** Gather functional and non-functional requirements of the end-users.

**4.1.2.2 Agile Development:** Develop the system iteratively, using sprints to incrementally add features.

**4.1.2.3 Testing and Validation**: Perform usability testing to ensure the visualizations are accurate and easy to interpret. In addition, test integration with data sources to validate real-time and historical data accuracy.

**4.1.2.4 Integration and Interoperability**: Ensure seamless data exchange and synchronization among the three system modules (Document Management, Workflow Management, and Dashboard System).

**4.1.2.5 Continuous Feedback Loop**: Engage users regularly for feedback and usability enhancements and iterate rapidly to address evolving business needs.

**4.1.2.6 Training and Change Management**: Provide training sessions and documentation for end-users.

## 4.2 Security Requirements

The Dashboard Management module must ensure robust security measures to protect sensitive data. As the system handles integration with multiple data sources, user credentials, and real-time analytics, it is critical to implement comprehensive security protocols. These requirements aim to safeguard data during transmission, prevent unauthorized access, and provide an audit trail for accountability, ensuring compliance with industry standards and best practices for secure information management.

The Security Requirements for the Dashboard Management System will include:

* Role-based access control to restrict dashboard views based on user roles.
* Use secure authentication protocols.

## 4.2 System Architecture

The Dashboard Management module is designed with a scalable architecture to ensure seamless functionality, high performance, and ease of integration. The architecture includes the following key components:

**4.2.1 Frontend:**

* Developed using AngularJS, providing a dynamic and responsive user interface.
* Enables customizable dashboards and interactive data visualizations for end users.

**4.2.2 Backend:**

* Built on Python Django, offering a secure and efficient server-side framework.
* Manages business logic, user authentication, and interaction with the database and integration layer.

**4.2.3 Database:**

* Utilizes PostgreSQL as the primary database.
* Ensures reliable and scalable storage for structured data, including user information, widget configurations, and historical data.

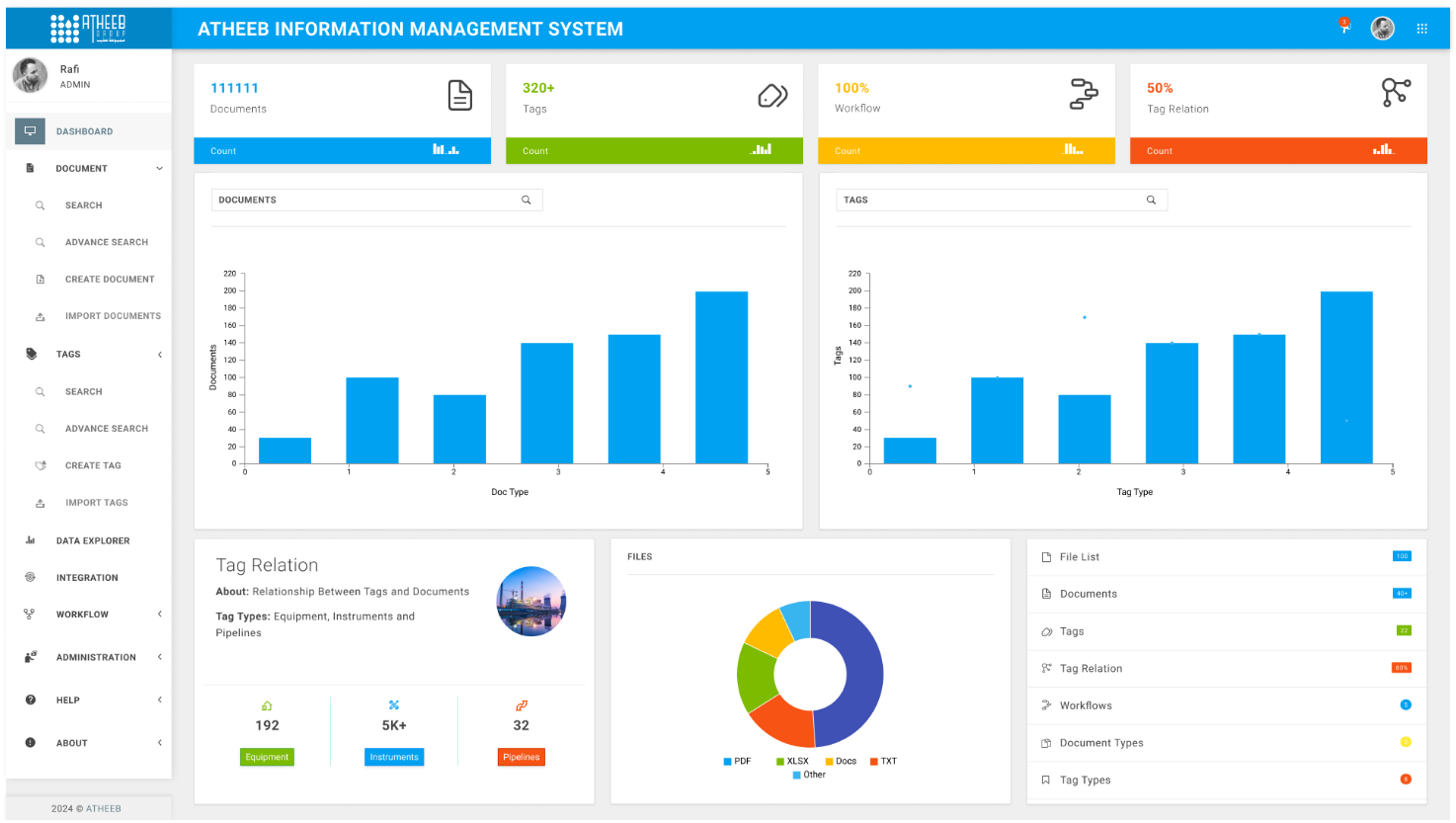
**4.2.4 Integration Layer:**

* Employs RESTful APIs to interface with external systems:
* Third-party systems for enterprise data integration.
* SQL Databases for on-demand data extraction and reporting.

This architecture ensures that the dashboard module is modular, scalable, and capable of providing a seamless user experience while integrating with complex external systems.

## 4.3 User Interface

The User Interface for the Dashboard Management System is designed to prioritize usability and aesthetics while providing functionalities for data visualization and user interaction. It allows for diverse user roles, from administrators to viewers, with a clean and responsive design.



**4.3.1 Key Features of the User Interface**

**4.3.1.1 Customizable Dashboards**:

* + Users can create and configure multiple dashboards.
  + Color-coded widgets for quick recognition of data categories.

[Screenshot of a customized dashboard.]

**4.3.1.2 Widget / Chart Library**:

* + Provides a library of pre-designed widgets such as bar charts, line graphs, pie charts, donut charts, and tables.
  + Users can configure widgets on their dashboards by selecting from the library.

[Screenshot of dropdown of different chart types]

**4.3.1.3 Navigation**:

* + A collapsible sidebar menu allows quick access to:
    - Dashboards
    - Data sources
    - Settings
  + Breadcrumb navigation for tracking user location within the application.

[Screenshot of left sidebar with all options un-collapsed]

**4.3.1.4 Role-Based Views**:

* + Custom views based on user roles:
    - **Admin View**: Access to all dashboards, widgets, and user management features.
    - **Viewer View**: Read-only access to dashboards and widgets.

**4.3.1.5 Interactivity**:

* + Hover-over tooltips provide context for data points in widgets.

[screenshot: widget with a tooltip appearing on hover, showing additional details.]

**4.3.1.6 Notifications**:

* + Real-time notifications displayed under a ‘Notifications’ icon on the header for:
    - Dashboard updates
    - System alerts

[screenshot of a notification message when the notification icon is hovered over or clicked]

**4.3.2 UI Components**

1. **Login Page**:
   * Clean and simple interface with a username/password field and optional two-factor authentication input.
   * "Forgot Password" link for recovery.
2. **Dashboard Page**:
   * Displays a grid-based layout with customizable widgets.
   * Top navigation bar with quick actions (e.g., add a widget, refresh data, export dashboard).
   * Sidebar for accessing saved dashboards and settings.
   * Options to export reports in formats like PDF, JPG/ PNG
3. **Widget Configuration Panel**:
   * Admin can configure each widget’s:
     + Data source
     + Axis Labels
     + Title
     + Filters
     + Visualization type
   * Other users can only configure each widget’s visualization type
   * Live preview of widget settings before saving.
4. **Settings Page**:
   * Allows admin to:
     + Manage profile settings
     + Configure data sources
     + Set refresh intervals for dashboards
     + Adjust notification preferences

**4.3.3 Dashboard Navigation Menus and Submenus**

The dashboard includes the following menus for enhanced functionality:

1. **Document Management Menu**:
   * Displays the documents database, allowing users to manage and organize files.
   * **Submenus**:
     + **Search**: Simple keyword-based search for documents.
     + **Advanced Search**: Search documents based on specific attributes like tags, date, category, or a combination of attributes. Documents can also be searched using using wildcards like ‘\*’.
     + **Create Document**: Allows users to upload, revise, and categorize new documents.
     + **Import Documents**: Facilitates batch importing of multiple documents using Excel Sheets.
2. **Assets Menu**:
   * Displays information about different assets.
   * **Submenus**:
     + **Search**: Quick search for assets.
     + **Advanced Search**: Search documents based on specific asset/tag attributes or a combination of asset/tag attributes.
     + **Create Tag**: Allows creation of new tags or asset entries.
     + **Import Tags**: Supports importing tags in bulk from Excel sheet.
3. **Integration**: Allows admin to set up integrations with third-party systems.

* Allows admin to:
* connect to APIs exposed by third-party systems
* connect to and extract information from database using SQL queries

1. **Workflow**:
   * Helps manage the workflow system by allowing admin to create, modify, and track workflows.
   * Further details about this will be discussed in Section \_\_\_\_
2. **Administration**:
   * Manages system-wide settings, user roles, and permissions.
3. **Help**:
   * Provides access to FAQs, user guides, and tutorials for troubleshooting and system navigation.
4. **About**:
   * Displays information about the system version, developers, and contact details.

**4.3.4 Chart Types**

The dashboard provides various charts to visualize data dynamically. Each chart type serves a specific purpose for data representation:

[sreenshots of each type of chart]

1. **Horizontal Bar Chart**:
   * Displays data comparisons across categories.
   * Ideal for showing performance metrics or data segmented by category.
2. **Vertical Bar Chart**:
   * Highlights changes in data over time or between groups.
   * Commonly used for time-series data.
3. **Pie Chart**:
   * Represents proportions or percentages.
   * Useful for visualizing how different components contribute to a whole.
4. **Donut Chart**:
   * Similar to a pie chart but with a hollow center, often used to display summary statistics in the middle.
   * Ideal for comparing segments while also highlighting the total.
5. **Line Chart**:
   * Shows trends and patterns over time.
   * Common for tracking progress or fluctuations, or growth.

### **Aggregation Widgets**

In addition to charts, the dashboard includes customizable widgets to display aggregated data, for example, count of the number of entries or items in a dataset (e.g., number of completed tasks, number of files).

## 4.4 Data Model Definition

The data model for the Dashboard Management System is designed to ensure efficient organization, retrieval, and manipulation of data required for visualization, reporting, and user interactions. The key entities, attributes, and relationships are detailed below:

**4.4.1 Entities**

**4.4.1.1 User Attributes:**

* **UserID:** Unique identifier for each user.
* **Username**: Login name for the user.
* **Password**: Encrypted password for authentication.
* **Role**: User role (e.g., Admin, Manager, Viewer) defining access permissions.
* **Email**: Contact email for notifications and password recovery.
* **Preferences**: User-specific preferences such as default dashboard layout.
* **CreatedAt**: Timestamp for account creation.
* **LastLogin**: Timestamp for the user's last login.
* **Status**: Whether the user is currently an active one or deactivated
* **Deactivation Date**: If status if user is ‘deactivated’

**4.4.1.2 Dashboard Attributes:**

* **DashboardID**: Unique identifier for each dashboard.
* **Name**: User-defined name for the dashboard.
* **OwnerUserID**: The user who created/owns the dashboard.
* **SharedWith**: List of users or groups with access to the dashboard.
* **Widgets**: References to widgets associated with the dashboard.
* **CreatedAt**: Timestamp for when the dashboard was created.
* **UpdatedAt**: Timestamp for the last modification.

**4.4.1.3 Widget Attributes:**

* **WidgetID**: Unique identifier for each widget.
* **Type**: Type of visualization (e.g., Bar Chart, Line Graph, Table).
* **DataSourceID**: Reference to the data source used by the widget.
* **Configuration**: JSON or schema detailing settings such as filters, thresholds, and axes labels.
* **Position**: Coordinates or layout details for placement on the dashboard.
* **LastUpdated**: Timestamp for the last data update.

**4.4.1.4 DataSource Attributes:**

* **DataSourceID**: Unique identifier for the data source.
* **Name**: Descriptive name for the data source.
* **Type**: Source type (e.g., third-party systems, SQL, RESTful API).
* **ConnectionDetails**: Configuration details for connecting to the source.
* **Status**: Current status of the connection (e.g., Active, Inactive).
* **LastSynced**: Timestamp for the last synchronization with the data source.

**4.4.1.5 ActivityLog Attributes:**

* **LogID**: Unique identifier for each log entry.
* **UserID**: Reference to the user performing the action.
* **Action**: Description of the action performed (e.g., "Added Widget", "Updated Dashboard").
* **Timestamp**: Timestamp of the action.
* **Details**: Additional context or metadata for the action.

**4.4.1.6 DebugLog Attributes**

Each day a new log file ( a basic TXT file) will be created. There has to be a timestamp whenever a new file is created. There will be a limit on the size of the file. If file size exceeds the limit, the system will create a new file and add it to to the DebugLog):

* **logID**: Unique identifier for each log file created – file should be named based on date, month, year.
* **ErrorMessage**: Error message generated by the system.
* **Timestamp**: Timestamp of the action
* **Details**: Additional context or metadata for the action.

**4.4.2 Relationships**

**4.4.2.1 User ↔ Dashboard:**

* **One-to-Many**: A single user can create multiple dashboards, but each dashboard is owned by one user.
* **Many-to-Many (SharedWith):** Dashboards can be shared with multiple users or groups.

**4.4.2.2 Dashboard ↔ Widget:**

* **One-to-Many:** A dashboard can contain multiple widgets.

**4.4.2.3 Widget ↔ DataSource**

* **One-to-One**: Each widget connects to a single data source to retrieve its data.

**4.4.2.4 User ↔ ActivityLog:**

* **One-to-Many**: Each user generates multiple activity logs.

## 4.5 System Integration

The Dashboard Management System is designed to integrate seamlessly with a variety of external systems to enable real-time data access, processing, and visualization. This integration ensures that the dashboard acts as a centralized platform, consolidating information from diverse sources for efficient decision-making. Below is a detailed breakdown of the system integration framework.

**4.5.1 Data Sources**

* **Third-party Systems**: Integrate with third-party systems to fetch enterprise data.
* **SQL Queries**: Retrieve data using custom SQL queries from databases like MySQL, PostgreSQL, or Oracle.
* **APIs**: Leverage RESTful APIs for integration with third-party applications and platforms.

**4.5.2 Integration Middleware**

The middleware layer acts as a bridge between the dashboard and external systems. It handles data transformation, error handling, and request/response routing. The system will use Python-based middleware using Django Rest Framework (DRF).

Python Django ORM is used to communicate with the database and get the required data. Rest APIs will be used to get information in the form of JSON data to the front end, Angular is used to filter out the data to be displayed. All CRUD operations are also done through the REST APIs.

**4.5.3 Data Refresh Mechanism**

Widgets can be configured for periodic data refreshes or triggered manually by the admin.

## Supported Protocols

**4.6.1 API Integration:**

* RESTful APIs with JSON responses for lightweight and widely supported communication.
* Error handling for stable performance. Log files will be generated for error handling and major functionalities.

**4.6.2 Database Integration:**

* SQL queries executed directly against PostgreSQL or other supported databases.
* Secure connections using SSL/TLS.

**4.6.3 Third-party System Integration:**

* Third-party system-specific connectors or APIs will be used for data extraction.

## Widget User Experience: Sequence of Steps

This section outlines the sequence of steps a user will take to interact with and manage widgets on the Dashboard Management System. The steps cover initial setup, data display, customization, and third-party integration processes.

**4.7.1 Adding a Chart to a Widget on the Dashboard**

1. Accessing the Dashboard:
   * The user logs in and navigates to their personalized dashboard.
   * The default dashboard view is displayed with existing widgets (if any).
2. Selecting a Widget:
   * A library of available widgets (e.g., bar charts, line charts, aggregation widgets) is available as a dropdown menu on each widget.
   * The user selects a widget type (e.g., horizontal bar chart) from the library.
   * A preview of the widget is shown, including placeholder data and visualization options.
3. Choosing a Data Source:
   * The user is prompted to select a data source from available integrations:
     + SQL Databases
     + RESTful APIs
     + Third-party Systems
   * The system displays a dropdown menu or search bar to locate the desired data source.
   * Once the user selects a source, they are prompted to enter a query or command to fetch the required data from the database
4. Configuring Widget Settings:
   * The user sets parameters for the widget, such as:
     + Axes
     + Labels
   * A real-time preview updates as settings are adjusted.

**4.7.3 Viewing and Interacting with Data**

4.7.3.1 Hovering for Details:

* + Hovering over a data point on the widget (e.g., a bar in a bar chart) displays a tooltip with additional information (e.g., exact value, percentage).

4.7.3.2 Exporting Widget Data:

* + The user clicks the export icon on a widget to download data in formats like CSV, Excel, or PDF.

**4.7.4 Managing Aggregation Widgets**

1. Configuring an Aggregation Widget:
   * The user selects a widget type like “Sum” or “Count” from the widget library.
2. Choosing Metrics:
   * The user selects a specific metric (e.g., total revenue, number of tasks).
   * Filters (e.g., by date range or category) can be applied to refine the aggregation.
3. Displaying Data:
   * The widget prominently displays the calculated value, such as: “Average Tasks Completed: 150”

# 5. Document Management

## 5.1 Approach and Methodology

This section outlines the approach and methodology for the development, implementation, and maintenance of the Document Management System (DMS), ensuring a secure and user-friendly solution for document storage, retrieval, and organization.

## 5.1.1 Approach

**5.1.1.1** **Centralized Repository Design**: Create a unified platform for storing and managing documents.

**5.1.1.2 Version control:** Enable access to version-controlled files and metadata.

## 5.1.2 Methodology

5.1.2.1 **Requirement Analysis**: Identify document workflows, access requirements, and metadata needs. Start with basic features (upload, retrieve, and search documents) and progressively add advanced functionalities like versioning and tagging.

5.1.2.2 **Data Migration**: Execute migration of existing documents to the new system with minimal downtime.

5.1.2.3 **Testing and Validation**: Perform usability testing for ease of document navigation and retrieval.

5.1.2.4 **Continuous Feedback Loop**: Engage users regularly for feedback to refine search functionality and workflow enhancements.

## 5.2 Security Requirements

The Document Management System (DMS) must ensure robust security protocols to protect organizational and user data. Security requirements include:

* Role-based access control to restrict document visibility and editing based on user roles (e.g., Admin, Manager, Viewer).
* Maintain a detailed log of document activity, including uploads, edits, and access events.

## 5.3 System Architecture

The DMS architecture is designed to support secure and efficient document management.

**5.3.1 Frontend:**

* Built using AngularJS to provide a responsive and intuitive user interface.
* Enables file uploads and navigation of folders.

**5.3.2 Backend:**

* Developed with Python Django, providing secure file handling and metadata management.
* Manages user authentication and permissions, and integration with document meta data.

**5.3.3 Database:**

* PostgreSQL serves as the primary database for storing metadata, user roles, and document references.
* Document files are stored in a scalable object storage system such as Amazon S3

**5.3.4 Integration Layer:**

* RESTful APIs allow integration with document storage system for document storage and retrieval.
* Batch upload and metadata import supported via APIs or Excel sheets.

## 5.4 User Interface

**5.4.1 Key Features of the User Interface**

#### 5.4.1.1 Document Library:

* + Centralized repository for all documents.
  + Searchable by metadata or custom filters.

[Screenshot: Document library with a search bar and metadata columns.]

#### 5.4.1.2 Document Upload:

* + Allows users to upload files directly to folders using the interface.
  + Users can also upload documents in bulk

[Screenshot: File upload into a folder using the interface.]

**5.4.1.3 Advanced Search:**

* + Enables detailed filtering by attributes like upload date, file type, and keywords.
  + Supports wildcard searches for flexible querying.

[Screenshot: Advanced search menu with multiple filter options.]

**5.4.1.4 Version Control:**

* + Maintains version history for documents.
  + Users can view, revert, or compare document versions.

[Screenshot: Version history panel with timestamps and user activity.]

**5.4.1.5 Role-Based Views:**

* + Admins can manage user roles and permissions.

**5.4.1.6 Admin Controls:**

Admins have the authorization to do the following:

* + create classes and properties
  + set user permissions for document access

[Screenshot: Admin view with role management options.]

**5.4.2 UI Components**

1. **Document Search:**
   * Quick search of documents using keywords
   * Filtering documents by type, date of creation, etc.

[Screenshot: Quick search page]

1. **Folder Structure:**
   * Hierarchical tree view of files and folder structure.
   * Allows admins to create, rename, or delete folders.

[Screenshot: Folder structure with nested subfolders.]

1. **Metadata Editor:**
   * Enables users to add or update metadata like title, author, etc for each document.

[Screenshot: Metadata editor panel with editable fields.]

1. **Document Viewer:**
   * Built-in viewer for previewing documents without downloading.
   * Supports PDF, Word, Excel, and image files.

[Screenshot: Document preview window.]

1. **Work Breakdown Structure Management:**
   * Provides a tree view of files and documents for hierarchical exploration.
     + Allows admin to explore parts of the database interactively.
     + Allows admin to create new folders and upload new documents within specific folders
2. **Revision Management**
   * The interface has two options for revision management: Revise and Update
   * Every time a data item related to a document is revised, a copy of its record is created with its revision number incremented by 1. This makes it easy to track revisions and revert to older versions, if needed.

[this will be demonstrated with a screenshot here]

1. **Content Search**
   * Users can search documents by content.
   * The interface will have a text input field to accept the search excerpt from the user and all documents that contain the given excerpt will be displayed in list view. From this view, the user can view documents as needed.
2. **Content Extraction**
   * Users can retrieve specific information or data from within a given document for further analysis or use.----need to understand this a little more

## 5.5 Data Model Definition

The data model for the Document Management System is designed to ensure efficient organization, retrieval, and manipulation of documents and meta-data. The key entities, attributes, and relationships are detailed below:

**5.5.1 Entities**

**5.5.1.1 User Attributes:**

* UserID: Unique identifier for the user.
* Role: Permissions assigned to the user (e.g., Admin, Viewer).

**5.5.1.2 Admin Class Attributes:**

For each class created, a new entry will be added to the Admin Class table

[show image of the table below]

* class\_id: Unique identifier for each class
* class\_name: Descriptive name for the class
* revisable: Boolean value to specify if the class can be updated once created
* creation\_date: Timestamp for when the class was created
* created\_by: User who created the class
* last\_updated\_date: Timestamp for when the class was last updated
* updated\_by: User who last updated the class
* terminated: A Boolean value to specify if the class has been deleted or not
* termination\_date: Timestamp for when the class was deleted (if terminated=True)

**5.5.1.3 Property Class Attributes:**

For each class created, the admin can customize its properties. For each property of each class a new entry will be added to the Property Class table with all details of the particular property

[show image of the table below]

* prop\_id: Unique identifier for each class property
* class\_id: ID of the class that the property belongs to
* prop\_name : Descriptive name for the property
* display\_name: Name of the field as displayed in the input form
* prop\_data\_type: Data type of the property
* prop\_display\_control: Control to be displayed depending on the data type mentioned in the prop\_data\_type attribute
* prop\_display\_control\_enabled: Specifies if the property’s control will be displayed in the form or not.
* grouping\_section: Specifies the group that the property field will be displayed as part of in the input form
* display\_size: Specifies the size of the input control in the form
* display\_order: Specifies the order of the control in the form
* validation: Specifies any validation to be performed for the particular control
* null\_validation
* read\_only: Specifies if the property is editable or not, once created
* mandatory: Specifies if the value of the specific property is manadatory or not.
* creation\_date: Timestamp for when the property was created
* created\_by: User who created the property
* last\_updated\_date: Timestamp for when the property was last updated
* updated\_by: User who last updated the property
* terminated: A Boolean value to specify if the property has been deleted or not
* termination\_date: Timestamp for when the property was deleted (if terminated=True)

**5.5.1.4 Folders Class Attributes:**

All files uploaded to the vault can be organized by the admin in a hierarchical folder structure. To keep track of the folders and subfolders, a Folders class will be created, that will have the following properties:

* folder\_id: Unique identifier for each folder
* folder\_name: Descriptive name for the folder
* parent\_folder\_id: Id of the parent folder that this folder is under
* creation\_date: Timestamp for when the folder was created
* created\_by: User who created the folder
* last\_updated\_date: Timestamp for when the folder was last updated
* updated\_by: User who last updated the folder
* terminated: A Boolean value to specify if the folder has been deleted or not
* termination\_date: Timestamp for when the folder was deleted (if terminated=True)

**5.5.1.5 Files Class Attributes:**

To keep track of the files, a separate Files class will be created, that will have the following properties:

* file\_id: Unique identifier for each file
* file\_name: Descriptive name for the file
* parent\_folder\_id: Id of the parent folder that this folder is under
* creation\_date: Timestamp for when the folder was created
* created\_by: User who created the file
* last\_updated\_date: Timestamp for when the file was last updated
* updated\_by: User who last updated the file
* terminated: A Boolean value to specify if the file has been deleted or not
* termination\_date: Timestamp for when the file was deleted (if terminated=True)

**5.5.1.6 Document Class Attributes:**

This class will be created dynamically and its properties can be customized as needed by the admin. The properties will considered as the columns for the newly created document table.

# Asset Management

## 6.1 Approach and Methodology

This section outlines the approach and methodology for the development, implementation, and maintenance of the Asset Management System, ensuring a secure and user-friendly solution for storage, retrieval, and organization of tags/files relating to assets of an organization.

## 6.1.1 Approach

**6.1.1.1** **Centralized Repository Design**: Create a unified platform for storing and managing assets and documents related to assets.

**6.1.1.2 Version control:** Enable access to version-controlled files and metadata.

## 6.1.2 Methodology

6.1.2.1 **Requirement Analysis**: Start with basic features (upload, retrieve, and search assets and progressively add advanced functionalities like versioning and tagging.

6.1.2.2 **Data Migration**: Execute migration of existing assets to the new system with minimal downtime.

6.1.2.3 **Testing and Validation**: Perform usability testing for ease of navigation and asset information retrieval.

## 6.2 Security Requirements

The Asset Management System must ensure robust security protocols to protect organizational and user data. Security requirements include:

* Role-based access control to restrict asset and related document visibility and editing based on user roles (e.g., Admin, Manager, Viewer).
* Maintain a detailed log of document activity, including uploads, edits, and access events.

## 6.3 System Architecture

The DMS architecture is designed to support secure and efficient document management.

**6.3.1 Frontend:**

* Built using AngularJS to provide a responsive and intuitive user interface.
* Enables file uploads and navigation of folders.

**6.3.2 Backend:**

* Developed with Python Django, providing secure file handling and metadata management.
* Manages user authentication and permissions, and integration with asset meta data.

**6.3.3 Database:**

* PostgreSQL serves as the primary database for storing metadata, user roles, and document references.
* Document files are stored in a scalable object storage system such as Amazon S3

**6.3.4 Integration Layer:**

* RESTful APIs allow integration with asset database for storage and retrieval of data related to assets.

## 6.4 User Interface

**6.4.1 Key Features of the User Interface**

#### 6.4.1.1 Asset Library:

* + Centralized repository for all assets.
  + Searchable by metadata or custom filters.

[Screenshot: Document library with a search bar and metadata columns.]

#### 6.4.1.2 Asset Upload:

* + Allows users to upload asset data as well as files related to assets.
  + Users can also upload documents in bulk

[Screenshot: File upload into a folder using the interface.]

**6.4.1.3 Advanced Search:**

* + Enables detailed filtering by attributes like upload date, and keywords.
  + Supports wildcard searches for flexible querying.

[Screenshot: Advanced search menu with multiple filter options.]

**6.4.1.4 Version Control:**

* + Maintains version history for documents.
  + Users can view, revert, or compare document versions.

[Screenshot: Version history panel with timestamps and user activity.]

**6.4.1.5 Role-Based Views:**

* + Admins can manage user roles and permissions.

[Screenshot: Admin view with role management options.]

**6.4.2 UI Components**

**6.4.2.1 Asset Search:**

* + Quick search of assets using keywords
  + Filtering assets by type, date of creation, etc.

[Screenshot: Quick search page]

**6.4.2.2 Metadata Editor:**

* + Built-in viewer for previewing documents without downloading.
  + User can view a list of documents related to a given asset, and can then choose to view any of these documents in the document viewer.

**6.4.2.3 Relationships Management**

* + Admins can specify relationships between documents and assets, to make it easier to find all documents related to a given asset, as well as all assets related to a given document

## 6.5 Data Model Definition

The data model for the Document Management System is designed to ensure efficient organization, retrieval, and manipulation of documents and meta-data. The key entities, attributes, and relationships are detailed below:

**6.5.1 Entities**

**6.5.1.1 Asset Class Attributes:**

This class will be created dynamically and its properties can be customized as needed by the admin. The properties will considered as the columns for the newly created Asset table.

[Asset table image will be shown here]

**6.5.1.2 Relationship Class Attributes:**

To keep track of the relationships between assets and documents, a Relationship class will be created, that will have the following properties:

* relationID: Unique identifier for the relationship.
* description: Description of how the asset and document are related
* assetID: ID of the related asset (from the Asset table)
* docID: ID of the related document (from the Document table)