Information Technology (IT) Assets Metadata Repository System Documentation

CS2815: Small Enterprise Team Project

Team 36 - Group report

By: Thomas Gould, Owain Feasey, Isaac George, Manav Sharma, Luke Akehurst, Sean Khoo, Janit Sudeesh

Contents

Statement of Relative Contribution

	Design, Planning and Coordination	Coding and Testing	Other	Signature
Luke	14	15	14	[illegible]
Manav	14	15	14	[illegible]
Isaac	16	15	16	I.George
Tom	14	15	14	[illegible]
Janit	14	15	14	J Sudeesh
Owain	14	10	14	O.Feasey
Sean	14	15	14	[illegible]
Dmani*	0	0	0	

*Not present throughout.

Technical Documentation

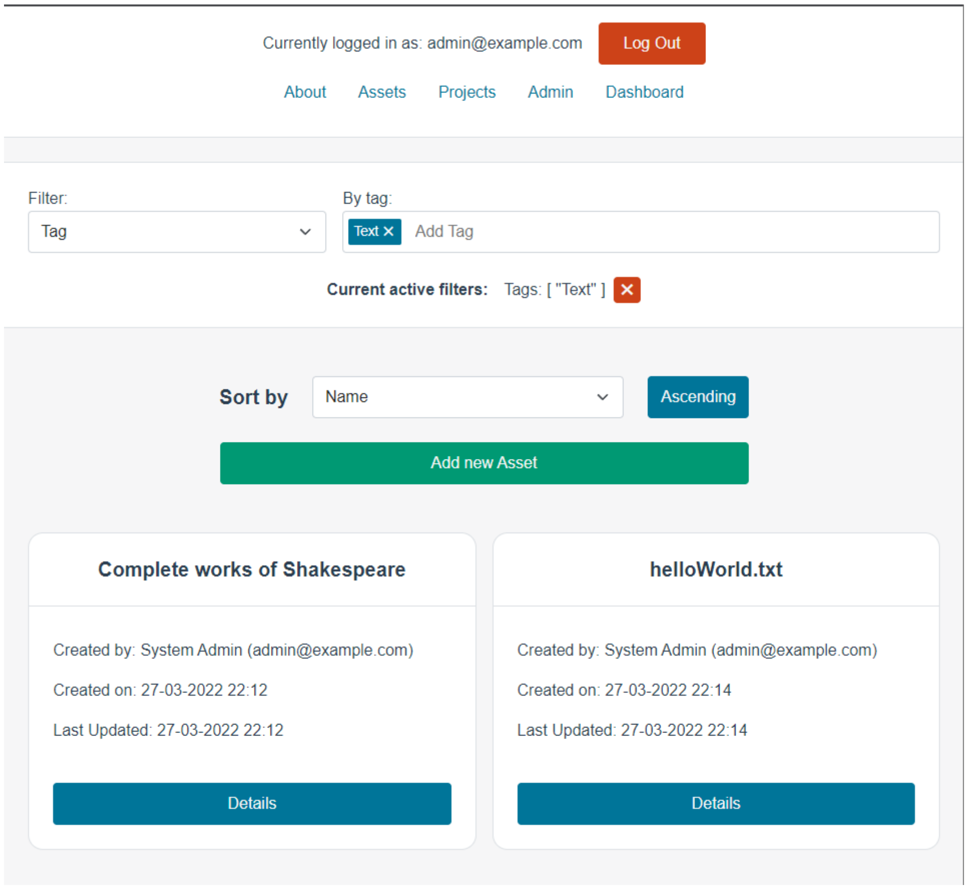
High-level Description

Frontend Overview

The frontend was built using JS and the Vue3 framework. We used bootstrap for styling. Additionally, tag and toast packages were used as helpful UI tools. The main pages are stored in the views folder, with the components to be loaded throughout the program stored in the components folder.

Frontend Views

The main asset viewing page is shown below. It uses a card-based layout to enforce a clear visual hierarchy for asset information. Initially, a table was used, though it becomes cluttered and difficult to view as more columns are added. There can also be multiple active filters at once, to narrow down the search if there are many assets in the system.



Complete works of Shakespeare

Created by: System Admin (admin@example.com)

Created on: 27-03-2022 22:12

Last Updated: 27-03-2022 22:12

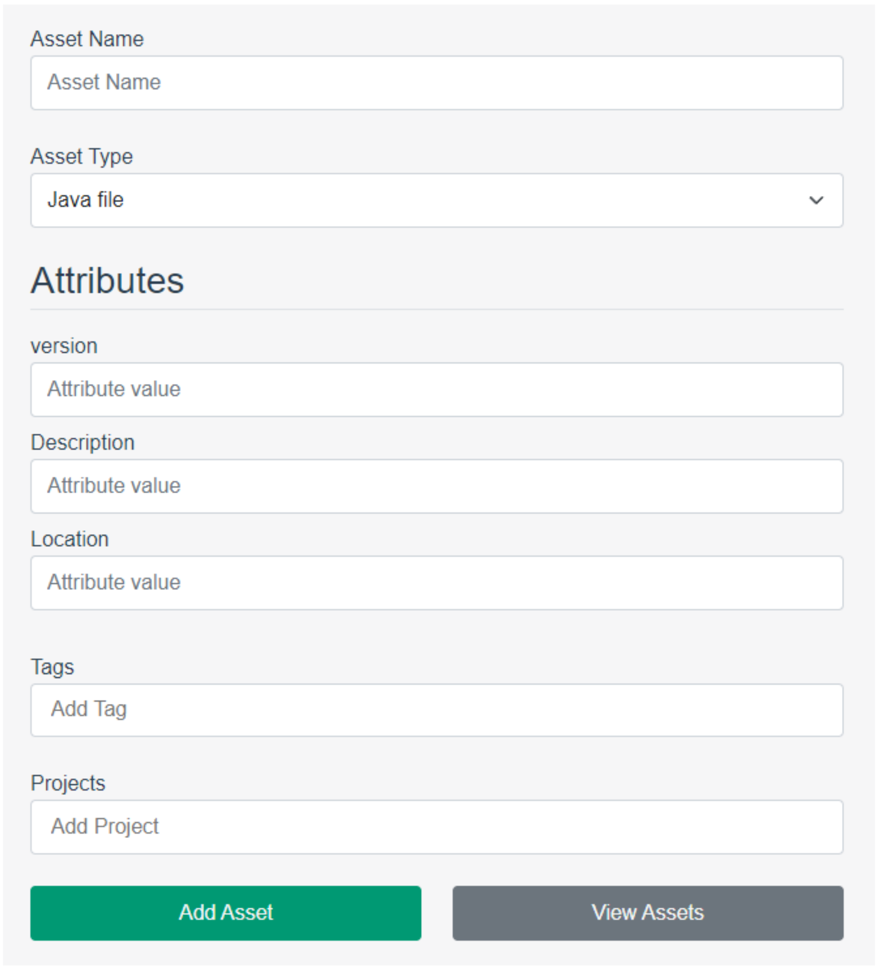
helloWorld.txt

Created by: System Admin (admin@example.com)

Created on: 27-03-2022 22:14

Last Updated: 27-03-2022 22:14

Upon clicking the “Add new Asset” button, the cards and filters are replaced with a simple form. The attributes within the form change dynamically, depending on the asset type selected.


Asset Name

Asset Name

Asset Type

Java file

Attributes

version

Attribute value

Description

Attribute value

Location

Attribute value

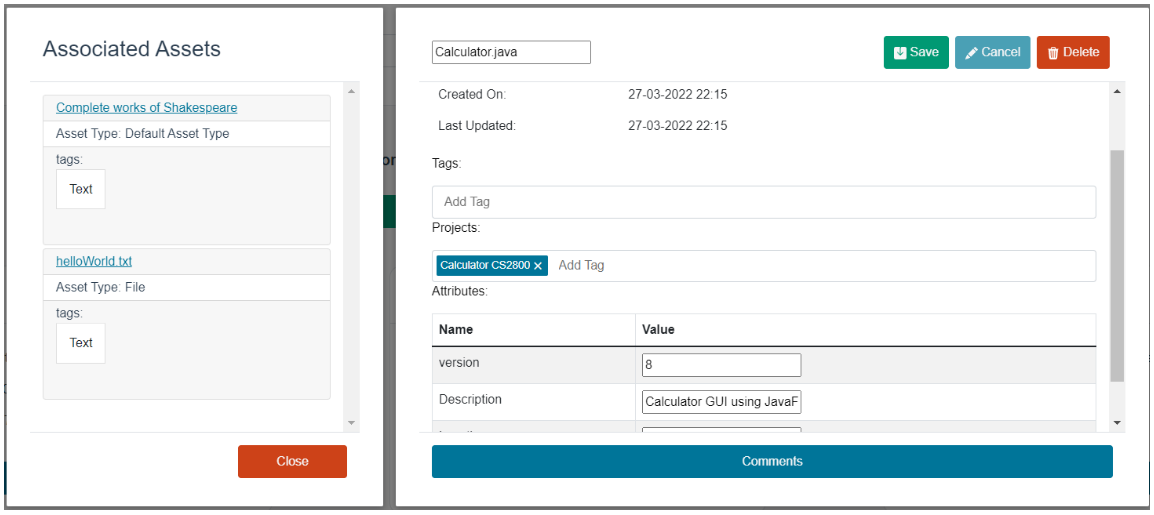
Tags

Add Tag

Projects

Add Project

Modals are used when clicking on an asset to display more details about them. Here, many of the fields and parts of an asset can be edited and updated if the user has permission. The Associated Assets modal displays assets that are related to the current asset via shared tags, projects, or a shared asset type. Clicking on an associated asset brings up the modal for their information.


Associated Assets

Complete works of Shakespeare
Asset Type: Default Asset Type
tags:
Text

helloWorld.txt
Asset Type: File
tags:
Text

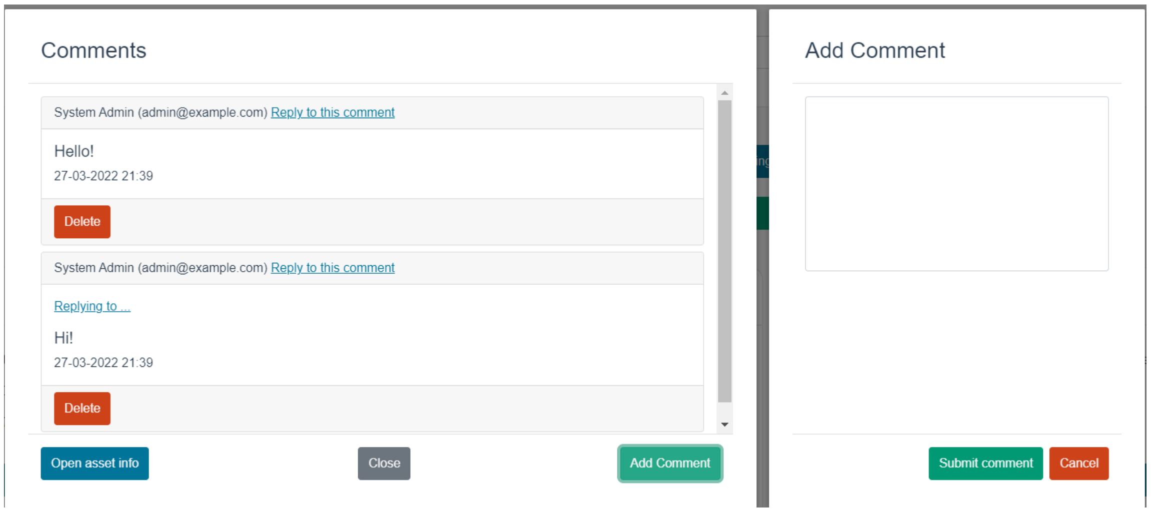
Calculator.java

Created On: 27-03-2022 22:15
Last Updated: 27-03-2022 22:15

Tags:
Add Tag
Projects:
Calculator CS2800 × Add Tag
Attributes:

Name	Value
version	8
Description	Calculator GUI using JavaF

Comments are displayed in a similar modal, showing any comments anyone has made for that specific asset. They can be replied to or deleted, though deletion can only apply if the comment was made by the current user. Deleting a comment will also delete all replies to that comment.

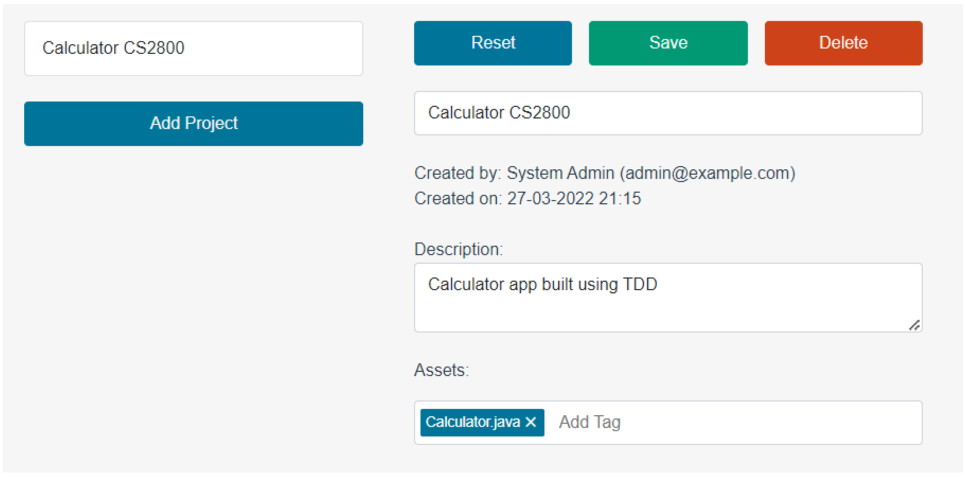

Comments

System Admin (admin@example.com) Reply to this comment
Hello!
27-03-2022 21:39

System Admin (admin@example.com) Reply to this comment
Replying to ...
Hi!
27-03-2022 21:39

Add Comment

The projects tab allows for CRUD (create, read, update, and delete) operations to be performed on projects. The assets included in a project can be updated using the tag style box. Basic information about each project is also shown, such as who the creator is.



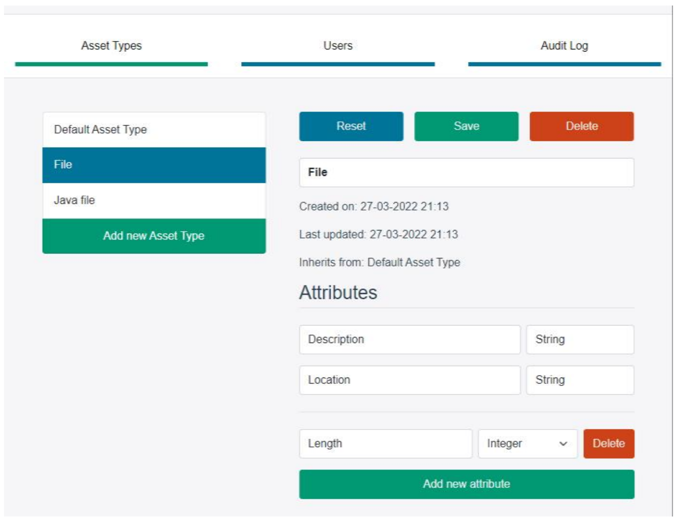
Calculator CS2800

Created by: System Admin (admin@example.com)
Created on: 27-03-2022 21:15

Description:
Calculator app built using TDD

Assets:
Calculator.java ✕ Add Tag

From the admin page, asset types can be viewed. On the left is a list of all asset types currently in the system. Once an asset type is selected, its information is displayed on the right. This page allows for full CRUD operations on asset types, only updating the database when the "Save" button is pressed. Asset types also inherit from one another. In every instance of the system, the "Default Asset Type" is present. This acts as a base for types to inherit from and will be the oldest relative of all types. Child asset types will obtain the attributes of their parents and all asset type they inherit from. The inheritance will form a tree structure with the "Default Asset Type" as the root.



File

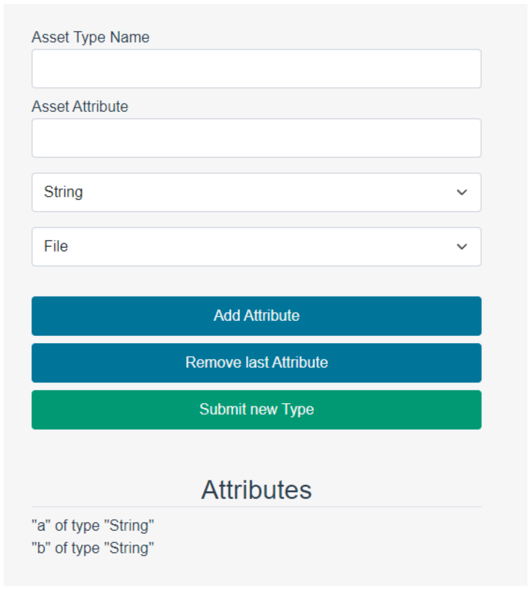
Created on: 27-03-2022 21:13

Last updated: 27-03-2022 21:13

Inherits from: Default Asset Type

Attributes

Description	String
Location	String
Length	Integer

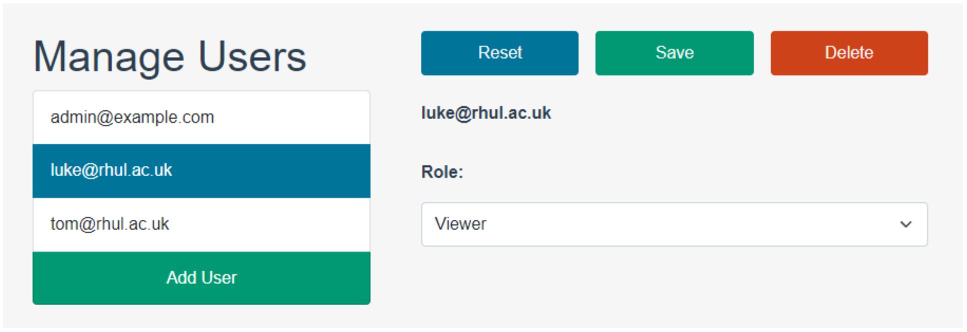


Attributes

"a" of type "String"
"b" of type "String"

When creating a new asset type, a simple form is used. A list of attributes can be formed below, showing the name of the attribute and the type. If the asset type selected as the parent contains any attributes that are the same as the one in the list, the duplicates from the list are removed.

In the middle tab, all users can be viewed, and CRUD operations can be performed. User roles can be seen and edited on this tab.



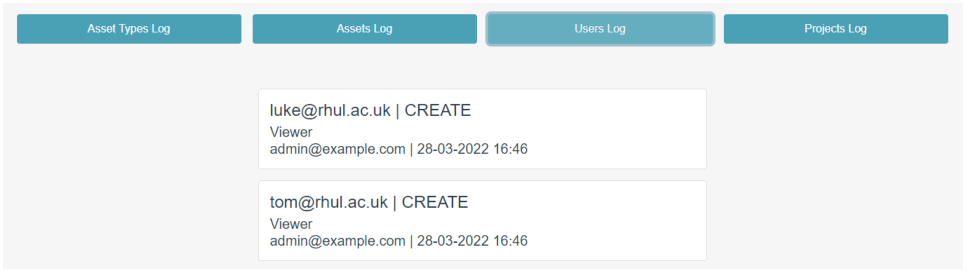
Manage Users

admin@example.com
luke@rhul.ac.uk
tom@rhul.ac.uk

luke@rhul.ac.uk

Role: Viewer

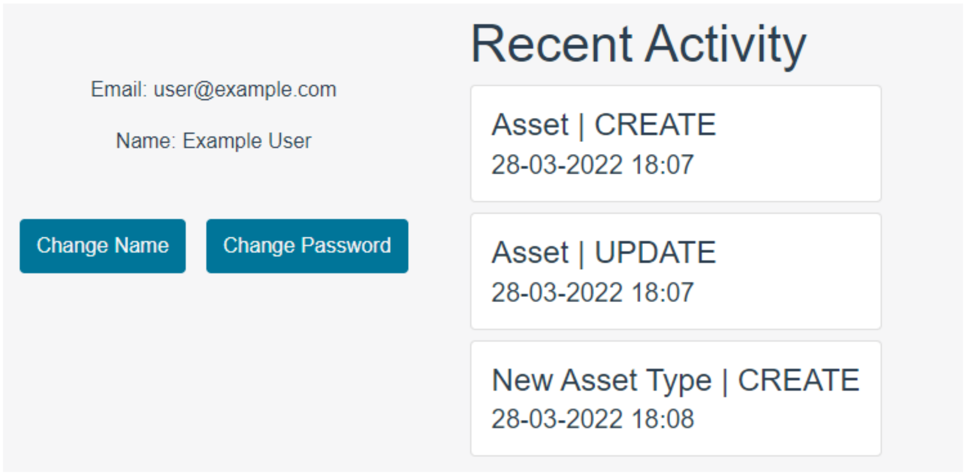
In the right tab, all changes to the database can be seen as records. Four different logs are available: asset types, assets, users, and projects.



luke@rhul.ac.uk | CREATE
Viewer
admin@example.com | 28-03-2022 16:46

tom@rhul.ac.uk | CREATE
Viewer
admin@example.com | 28-03-2022 16:46

The dashboard tab is for viewing your account. All recent activity of the current user is listed on the right, and on the left are buttons that will take you to pages to change your name and password.



Email: user@example.com

Name: Example User

Recent Activity

Asset | CREATE
28-03-2022 18:07

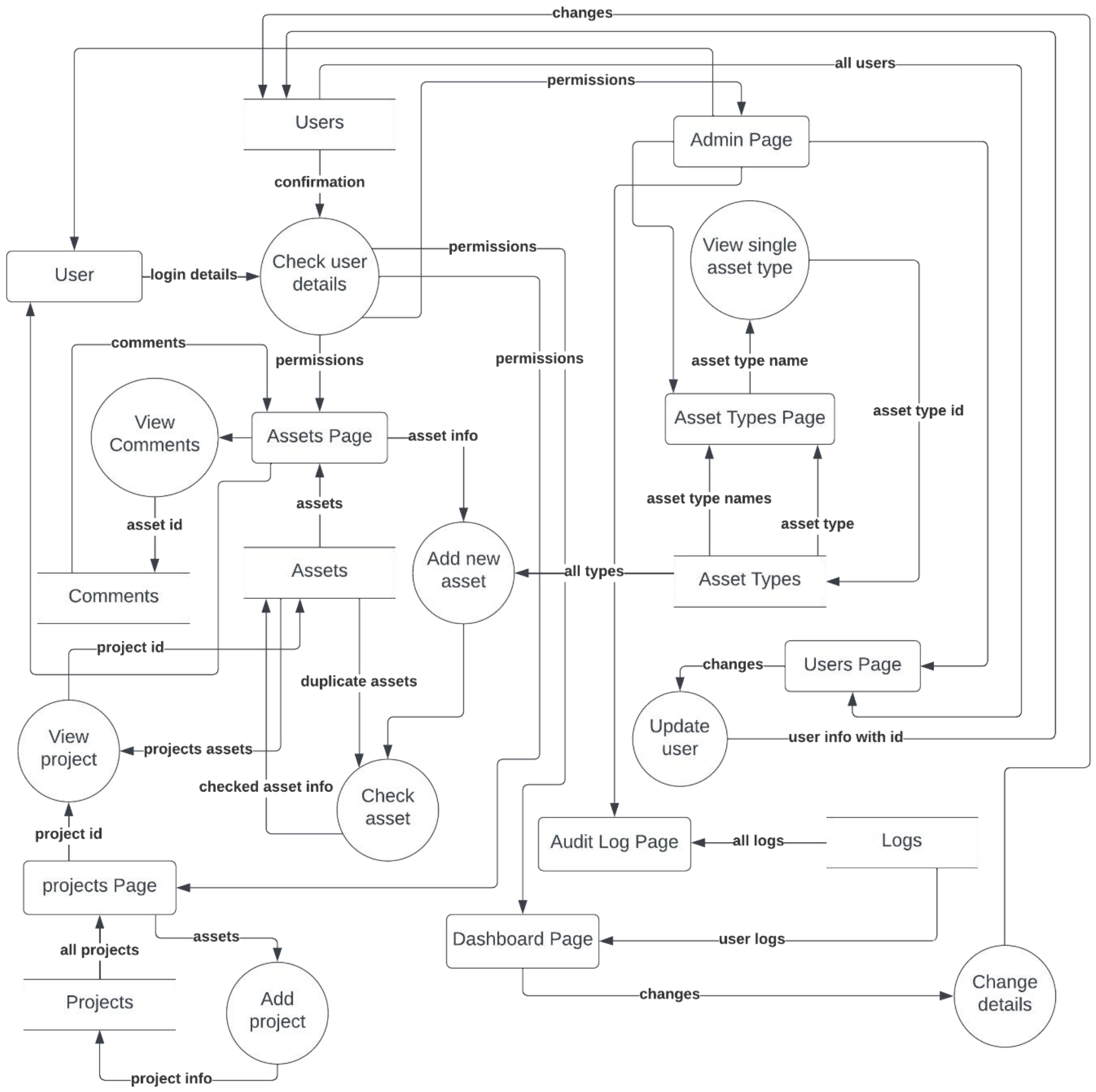
Asset | UPDATE
28-03-2022 18:07

New Asset Type | CREATE
28-03-2022 18:08

Backend Overview

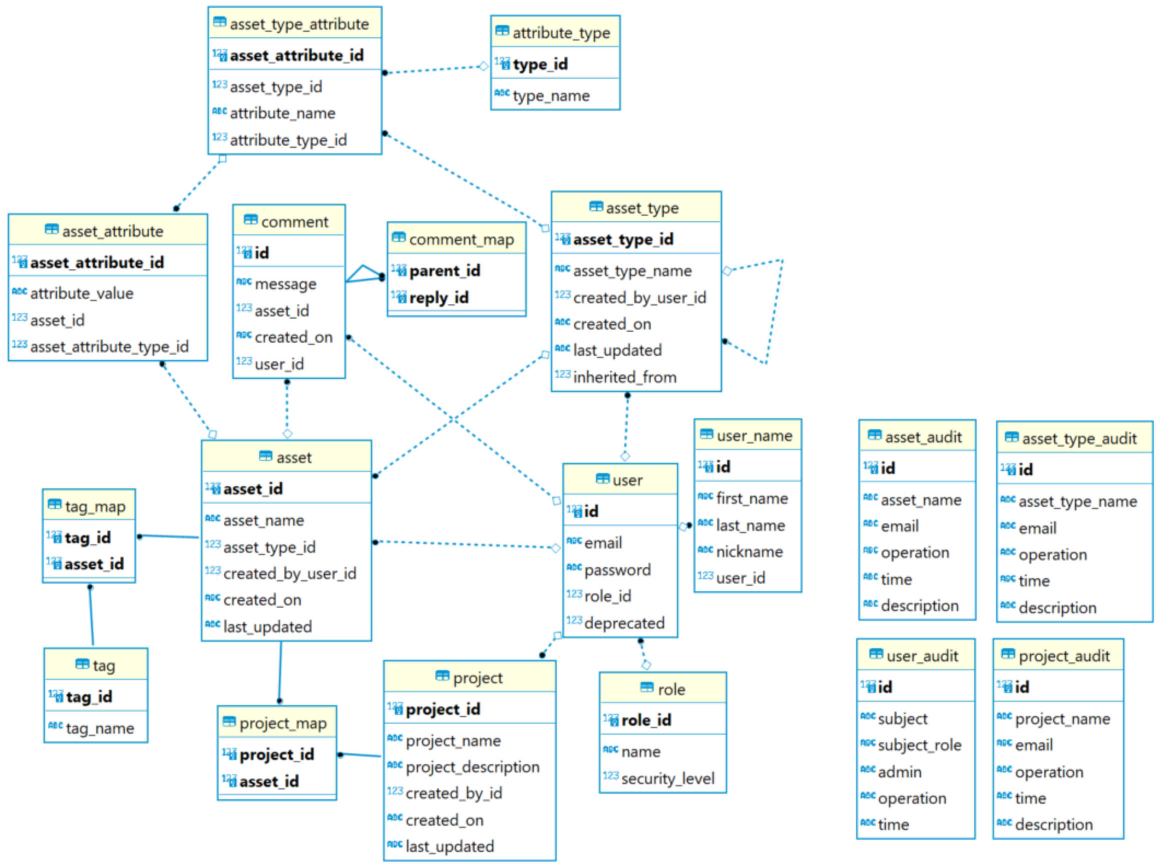
The two main technologies used for the backend are the Flask web framework, used as a web application programming interface (API), and SQL alchemy, used as an object relational mapping (ORM) tool, all written in Python. The backend is separated into several distinct sections defined by blueprints in Flask, which organises the API endpoints into logical groups. Most of the specific information about these blueprints and their organisation is explained in the "Description of Packages" section.

Dataflow Diagram



This diagram shows the flow of data through the application, from the user's login to all the CRUD operations present in the system.

Entity Relationship Diagram



The main data structure is broken into four tables: asset_types, assets, asset_type_attribute and asset_attribute.

An asset type has a name, created_by, created_on and inherit_from. Asset types refer to themselves when they want to inherit attributes from other asset types. Examples for an asset type name could be “Python file” or “Java file”.

An asset type attribute is an attribute of an asset type. They have a name, asset type id and attribute type id. The asset type id is the id of the asset type it belongs to. An asset type attribute name could be author or description. The attribute type id refers to the datatypes that can be given for the attribute values of that attribute type. For example, an attribute type of “Lines”, would have a type of “Integer”.

An asset is a version of an asset type. They have a name, asset type id, created_by and created_on. The asset type id refers to what asset type it is. If an assets’ “asset type” is Python file, then the assets name could be “helloworld.py” or “driver.py”

An asset attribute is the value of an attribute of an asset. They have a value, asset id and asset attribute type id. The asset id refers to which asset they belong to, and the asset attribute id refers to which asset type attribute the value represents. If an asset attribute type is “description”, then the asset attribute value would be a description written by the user.

The user and user_name tables refer to the users in the application. The users table holds all information needed throughout the application for access, while the user_name table holds other less important details. The table relates to many other tables via the id of the user who has created different objects in the application. Each user has a role represented by the role table, where the security level determines what pages and routes within the application are accessible.

The tag table is used to associate different assets with each other. Each asset can have many tags, and each tag can be used with many different assets.

The project table is used to store projects which relate to assets. Each project has a name, description, creator, and dates. The project_map table is used to relate projects and assets in a many-to-many relationship. Each project can have multiple assets, and each asset can have many projects.

The comment table is used to store comments made on assets. They contain a message, along with a creation date and ids for the author of the comment, as well as the asset it is applied to. The comment_map table is used to relate reply comments to the parent comment that they are replying to.

The audit tables store all create, update, and delete functions performed on assets, asset types, projects, and users. They aren’t related to the tables they audit, as they need to hold records if the objects are removed from the database.

Description of Packages

Backend

- app.py: The entry point for the backend, which loads in all external dependencies and runs the contents of jobs.
- test: These are the tests that are performed on the endpoints in api/route files.
- jobs: This module comprises of common tasks that need to be run when the backend is started, namely loading default values into the database.
- api/models: This directory comprises all the SQL Alchemy database models, which translate to the DB table structure.
- api/schemas: This directory contains all the marshmallow schemas used throughout the application, which use the models stored in api/models. In this context, a schema defines the fields that should be returned from a query, and any table joins.
- api/utils: These files contain code that needs to be shared across multiple API endpoints, but don't classify or associate with a specific route.
    - [asset | asset type | project | user] audit: These files contain the functions to store audit logs for the respective module.
    - Validation: This file contains the decorator (code that will run before each endpoint it's assigned to) used to validate the parameters that are passed to a given API endpoint.
- api/route: Each file contains a Python Flask blueprint, which is a way to separate routes into their own files.
    - Asset Types: Endpoint "/asset/types", this blueprint contains all operations that can be performed on Asset Types. It depends on the corresponding files inside api/model and api/schema.
    - Assets: Endpoint "/asset", this blueprint contains all operations that can be performed on assets. To communicate with the database, it depends on models and schemas corresponding to Asset Types and Assets.
    - Audit: Endpoint "/audit", this blueprint facilitates retrieving audit logs and can only be accessed by users with security levels of 100 (i.e., administrators). These routes depend on the schemas and models corresponding to audit.
    - Auth: Endpoint "/auth", this blueprint is concerned with any operations that a user can perform with regards to authentication, namely register and login. It should be noted that only users with security level 100 (i.e., administrators) can register new users. It makes use of the corresponding models and schemas.
    - Comments: Endpoint "/comment", this blueprint contains operations that can be performed on comments for assets. Users can only delete their own comments, and there is no functionality provided for modification. This blueprint depends on Asset and comments models and schemas.

    - Project: Endpoint “/project”, this blueprint contains operations that can be performed on projects (a way of relating assets). It depends on the models and schemas relating to assets and projects.
    - Role: Endpoint “/role,” this blueprint facilitates getting all roles and contains a decorator that checks that the authenticated user is authorised to perform the given operation. This file depends on the models and schemas relating to roles.
    - Tag: Endpoint “/tag,” the sole purpose of this blueprint is retrieving all currently existing tags in the system. Tags are created and assigned to assets from within route/assets files. Blueprint only depends on its corresponding schema.
    - User: Endpoint “/user,” this blueprint is concerned with operations that can be performed on users, with exception to authentication. Only the current user can change their password or display name. Only administrators change user roles (including their own) and delete users. This blueprint only depends on its corresponding models and schemas.

Frontend

Only the contents of the “src” directory are of significance, all other files are boilerplate required to make Vue.js work correctly. Hence, when listing the paths to files below, “/src” will be emitted from the start for readability. The only noteworthy file is vue.config.js, required to proxy request to the backend.

- api/index.js: Handles all interaction with the backend, contains reference to specific endpoints, so they don’t need to be repeated around the application. This file handles authentication by intercepting requests and responses to and from the backend.
- components: In this context, components are reusable modules that can be embedded in other pages of components around the application.
    - AddAsset: Manages the creation of new assets.
    - AddComment: A modal form used to publish new comments on assets, accompanied by the comments modal.
    - AddProject: A form for creating new projects.
    - AssetInfo: A modal used to display information about current assets. Allows assets to be updated and deleted. Additionally links to other components, such as associated assets and comments.
    - AssociatedAssets: This modal is only shown when the asset info modal is open. It displays a list of assets associated to the current asset, which is ordered by association, where the assets with the strongest association are at the top.
    - AuditLog: This component displays all current audit (change) logs for various aspects of the application (namely: Asset Types, Assets, Users, Projects). Only administrators can access overall audit logs, hence this component’s presence in the admin dashboard.
    - Comments: Displays comments for a specific asset, and links to the AddComment component to allow users to add comments.

    - CreateAssetType: Linked to from within the asset types view in the admin dashboard, this component allows administrators to create new asset types.
    - ManageProjects: Only available to administrators, this page lists details about projects and their assets, and links to AddProjects.
    - ManageUsers: Administrator page for viewing information about system users and enables changing their roles.
    - Register: Facilitates enrolling new users into the system and can only be used by administrators.
    - ViewAssetTypes: This component, only present in the admin dashboard, displays detailed information about asset types. Allows admins to create, edit and delete asset types.
- router/index.js: This file maps each view to the URL they are accessed from and defines security parameters of each of them, such as security level.
- store/index.js: This is the Vuex global storage for the application, which is used to hold information that is required across pages. For example, the logged in user and what security level they have.
- utils/index.js: Holds frequently used functions, which are then imported in whichever file they are required, preventing code repetition.
- Views: In this context, a view is a page that a user can navigate to either via a URL or through the navigation bar.
    - About: Contains a brief description of the IT Asset Metadata Repository System.
    - AdminDashboard: As the name suggests, only accessible to Administrators, and facilitates all the operations unique to them that they can perform.
    - Login: Entry point for the application, used to authenticate users.
    - UserDashboard: Display information about the specific user logged in, such as their name. Provides the functionality for users to change their display name and password.
    - ViewAssets: The main page of the application, which displays all current assets in the system.
- App.vue: The page to which all other views and components are mounted and defines functionality that is shared across all pages, such as the navigation bar.
- main.js: Basic Vue.js setup. This file bears no significance to this application than it would any of Vue.js project and wasn't edited over the course of development.

User Stories

Viewers:

- As a viewer, I want to securely login/logout from the program, so my information is kept secure.

- As a viewer, I want to view all the assets at once, and be able to view detailed information about any specific asset. This should show the name of the asset, as well as its attributes and any relevant tags.
- As a viewer, I want to sort the database, so I can see the data in a preferable format.
- As a viewer, I want to search and filter the database in various ways, such as name of asset, date created, asset type, etc. so I can quickly search for database entries and filter the display to show relevant assets to me.
- As a viewer, I should be able to search the database and app intuitively, so I can have a good user experience. When creating a new asset, I would like to navigate there via the view assets page.
- As a viewer, I want the application to present the most relevant related assets, so I can have a better user experience.
- As a viewer, I would like to have a page where I can see my details and change my password and username.
- As a viewer, I want to view who created what asset, and, if it has been updated at any point, the time of the latest change

Normal Users:

- As a user, I want to be able to add, edit/update and remove assets to the system.
- As a user, I want to be able to associate assets using tags when viewing and creating them. Then, filter by individual tags, to find all assets that share any tags the asset has, or any assets that have all the same tags.
- As a user, I want my changes to the assets recorded over time, so I can recall the changes I have applied.
- As a user, I want to add and edit projects that can be associated with assets.
- As a user, I want to comment on assets and reply to comments on each asset, so I can discuss assets on the system.
- As a user, I want certain asset types to have certain default attributes.

Admins:

- As an admin, I want to create and delete asset types, so I can create custom assets, as well as remove any unused asset types.
- As an admin, I want to update asset types, such as adding or removing attributes, as well as being able to rename the asset type and attribute names within it.
- As an admin, I want to enforce roles assigned to users so I can restrict access to pages.
- As an admin, I want asset type attributes to have types, so I can add validation to inputs.
- As an admin, I want a page that shows me all the actions I can do as an admin, so I can easily use the system and swap between functions.
- As an admin, I want to assign and change the roles of users, so that I can give/restrict access to features and pages. Higher roles should inherit all the abilities of lower roles.
- As an admin, I want to have an audit log to show any activity and changes made to assets, asset types, users, and projects.
- As an admin, I want asset types to have default fields to inherit from, so I can quickly create and group asset types. As a viewer, I would like to have a page where I can see my details and change my password and username.

Movie Clip

https://www.youtube.com/watch?v=jQzSnulzX80