TECHNICAL DOCUMENT

The docker directory provided contains all the files required to build the database and the environment required to run this project. Use an application like Docker to deploy the environment in a container. Once Docker is installed, run docker compose up at your command line. The relevant images will be pulled down (php-apache, mariadb, phpmyadmin), then the containers will be started.

Contents Description

html/ - contains cw2 dir and index.php which will have an entry point into the project

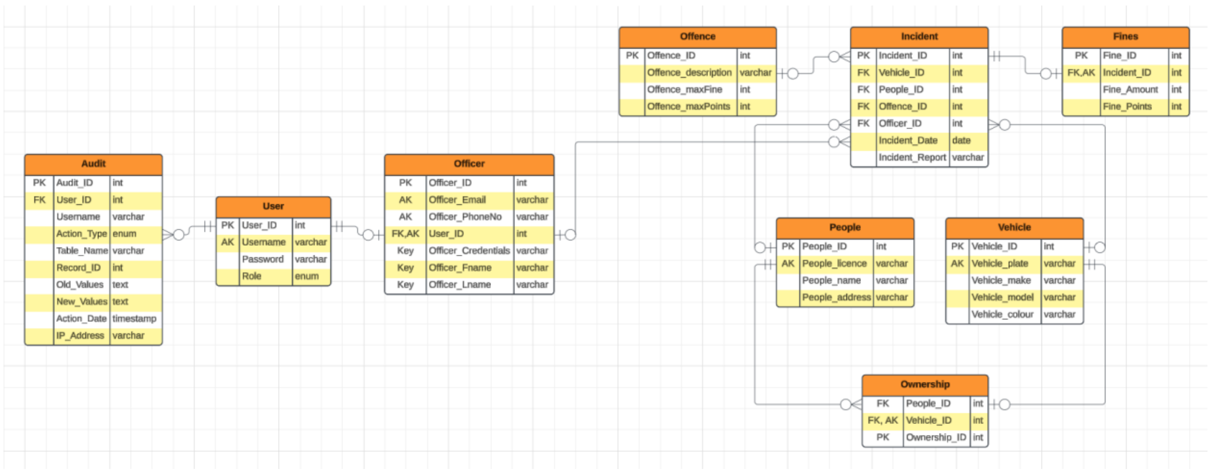
html/cw2/ - Contains all php, js and css files that are part of this project

mariadb/ - SQL files that are used to build the database when Docker starts; this includes the database for this project

mariadb-data/ - this is where the mariadb Docker container's database stores its files

php-apache/ - contains the instructions to build the php-apache image

Entity Relationship Diagram of the database



Made using lucidchart

The relational database for this project has 9 relations. The relationships and the cardinality are explained by crow's foot notation. I shall provide a brief explanation as well.

Abbreviations

PK – Primary Key

FK – Foreign Key

AK – Alternate Key (Not Null and Unique)

Tables

Audit: Each audit record **must** have **one** user ID associated with it.

User: Each user **may** have **many** audit records. Each user **may** have **one** associated officer record.

Officer: Each officer **must** have **one** associated user ID. Each officer **may** have **many** incident reports.

Offence: Each offence **may** be associated to **one** incident.

Fines: Each fine **must** be associated with **one** incident.

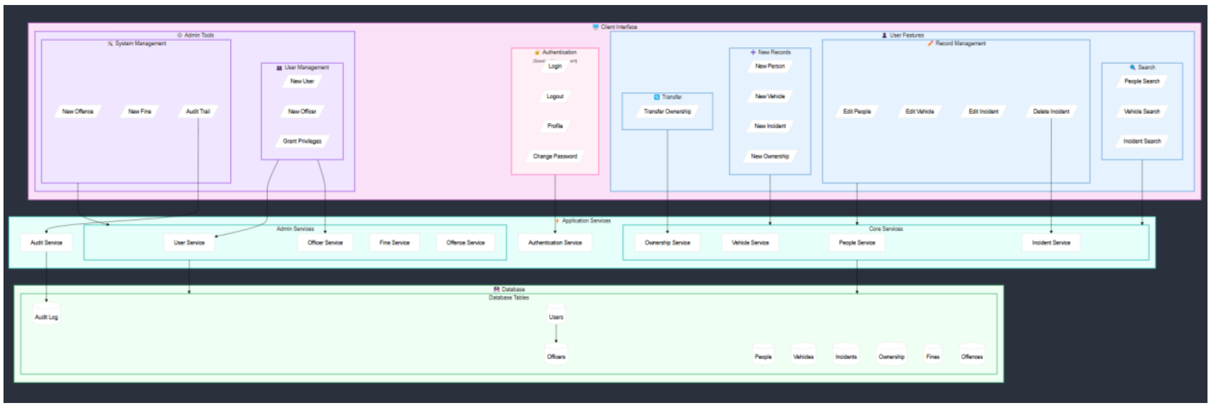
People: Each person **may** have **many** incidents reported about them. Each person **may** own **many** vehicles.

Vehicle: Each vehicle **may** have **many** incidents reported about it. Each vehicle **may** have **one** owner.

Incident: Each incident **may** have **one** associated offence. Each incident **may** have **one** associated fine. Each incident **may** have **one** associated person. Each incident **may** have **one** associated vehicle. Each incident **may** have **one** reporting officer.

Ownership: Each ownership **must** have **one** owner and **one** vehicle.

Architecture

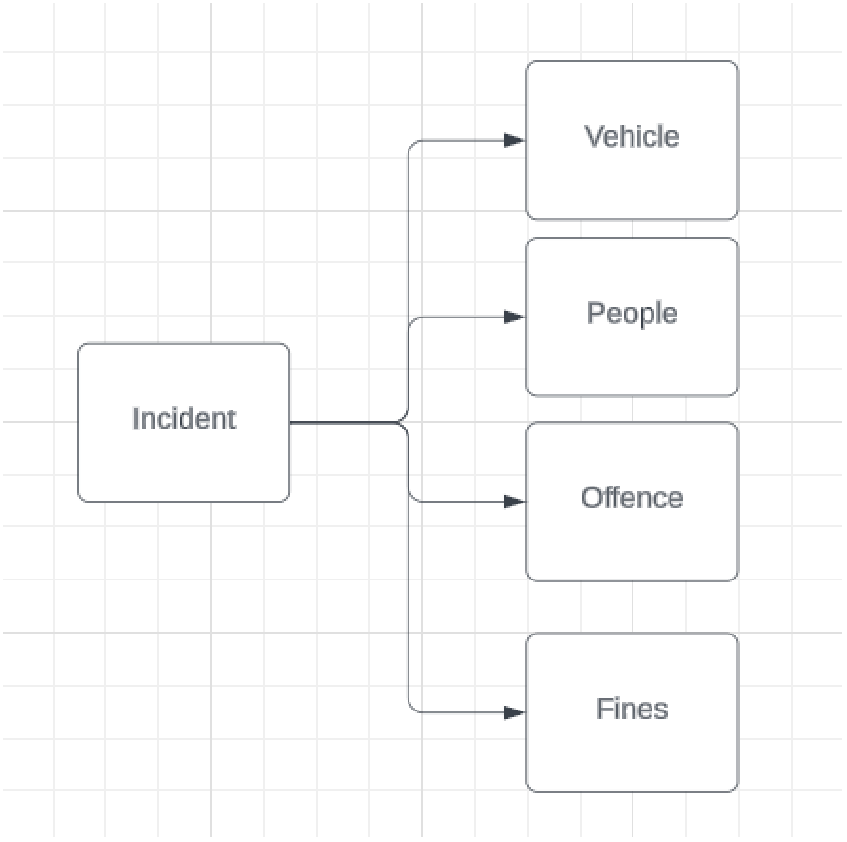


Code Generator + mermaid.live

Web Interface Files: All contents of the html/cw2 directory of the docker folder

At the login page, if the user logs in successfully, the user_id, username and officer_id (if it exists for the user) is stored as session variables to be used in other webpages across the sites.

The dashboard displays the username the credentials, first name, last name of the officer in the first container (div class container). The session officer ID is used to prepare a query that finds the attributes of the current officer in session. This page also displays the most recent 5 incidents reported and quick stats.



Data flow for Recent incidents in dashboard. Made using Lucidchart

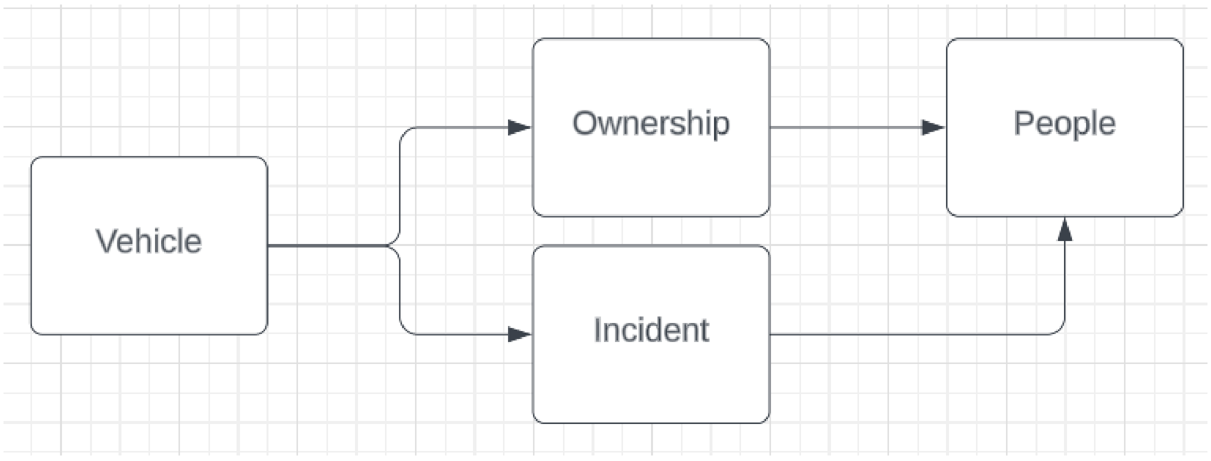
Incident table is joined with these tables by the foreign keys (the respective IDs) and ordered by date descending, limit by 5. The quick stats section counts the incidents, sums fines and points from the fines table, and sums number of rows with null values for incident join fines.

The profile section uses a query to find the officer information using the session stored officer ID and displays the information on the page.

Change Password uses a query to update the password attribute in the User table using the session user_id variable.

People search only queries the people table.

Vehicle search finds related information from multiple tables.



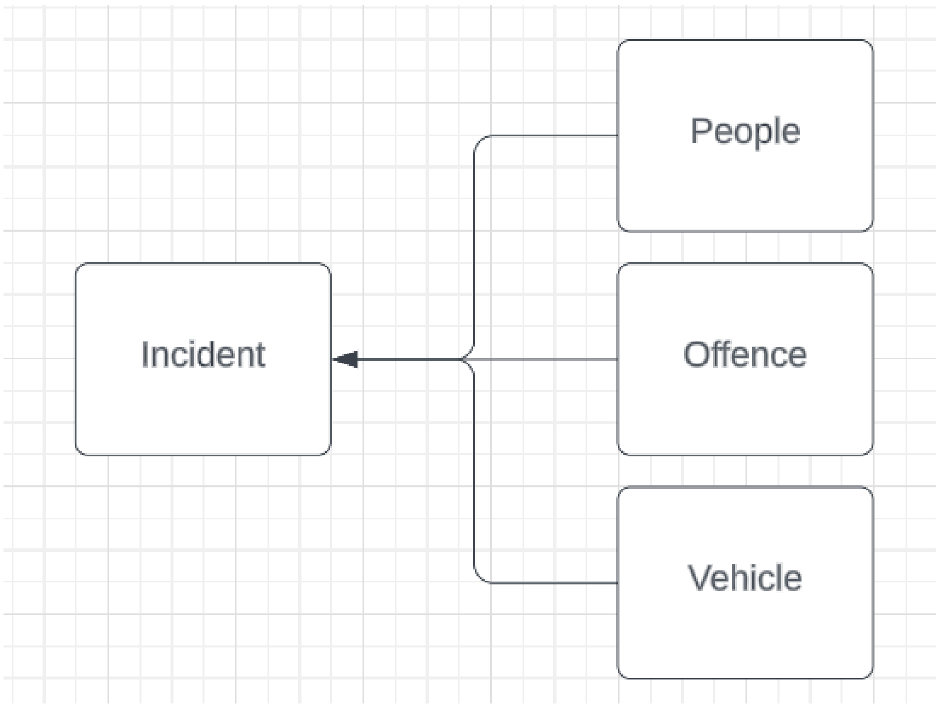
Data flow for vehicle search. Made using LucidChart

Vehicle joins on ownership joins on people; vehicle joins on incident joins on people. This ensures that we get the owner's name, and the name of the person involved in the incident.

The add and edit feature for People and Vehicle lets you add a new entity or update the value of any attribute of the respective entity. All vehicle attributes are mandatory, and name and licence are mandatory for people. This design decision was adopted so that each person has a unique identifier (licence number). There will be cases when the person in the report does not have a licence, the officer is urged to fill in the name as a placeholder, since the licence details can be edited later. There are consistent checks throughout the system to check for people with the same licence number to avoid duplication of the same licence number.

The data flow for search incident follows the same principle as the recent incidents section of the dashboard page. It collects data from five joined tables and shows it as a record in the search results.

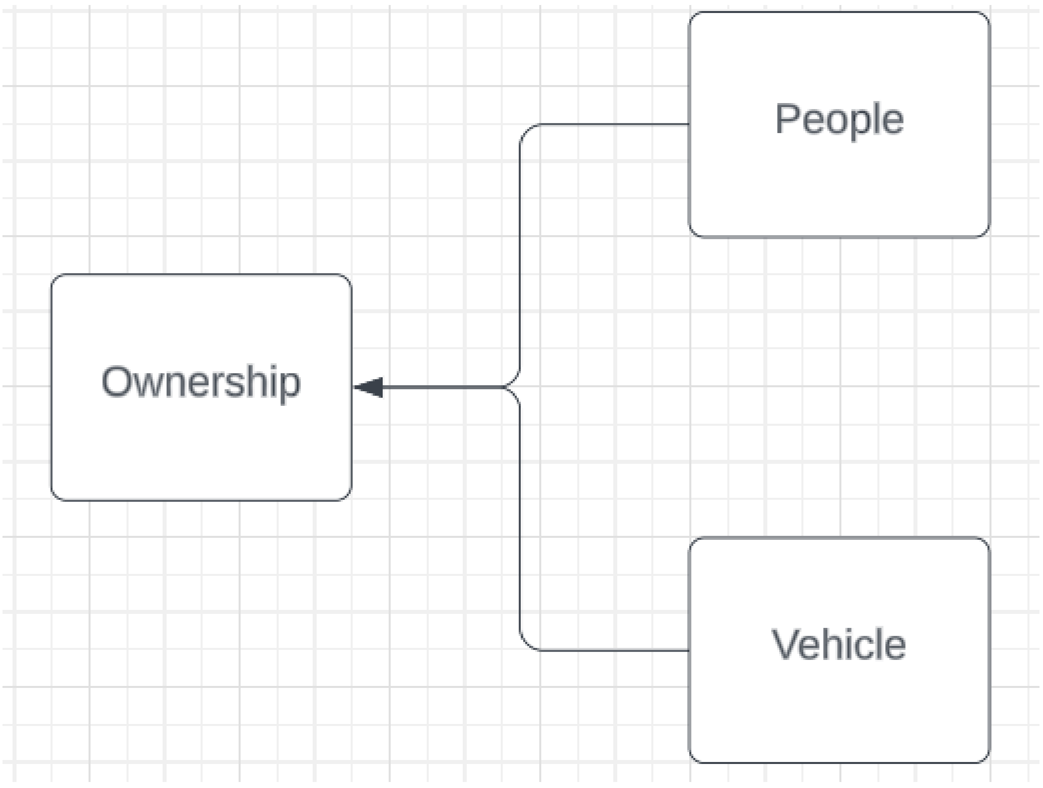
New incident and Edit incident fetches primary keys from different tables to add or update a record. Since most attributes of the incident table are foreign keys, the respective IDs of the inputs provided must be fetched from the respective tables before inserting or updating the record. Vehicle details check against the plate number in the vehicle table, People details check against the licence number in the people table, offence check against the description in the offence table, officer id is fetched from the session variable.



Data flow for new/edit incident. Made using LucidChart

The only mandatory fields for incident creation are the date and report. This decision was

Ownership table handles the vehicle, and people details the same way new incident and edit incident does. It is worth noting that all three of these pages uses AJAX requests to lookup the input for the unique identifiers (plate number and licence number) when the respective field loses focus and searches whether the entered value corresponds to any entity currently in the system. A JavaScript script auto fill the related fields if they are and makes those fields read-only. Otherwise, it lets the user enter the details by setting the field state to editable.

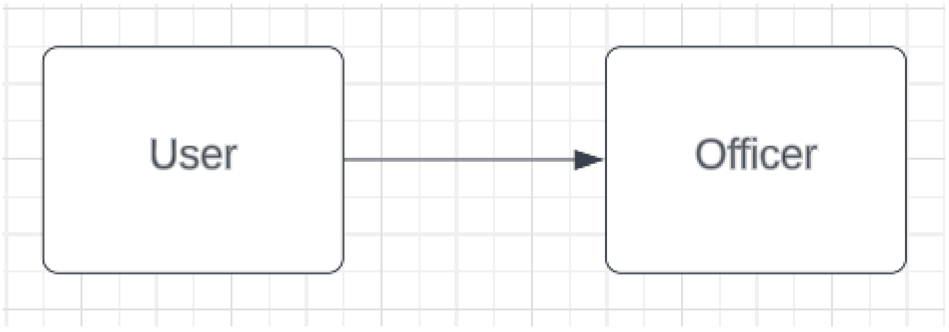


Data flow for new ownership. Made using LucidChart

Since ownership, new incident and edit incident must handle different actions on the same page (inserting and updating), different actions are abstracted into php functions to make the control flow much organised.

For all the edit pages, the entity ID has been passed on as parameters in the URL and fetched using GET. The initial approach was to use the search input, but when there are multiple results and you decide to edit another entity, this approach will fail.

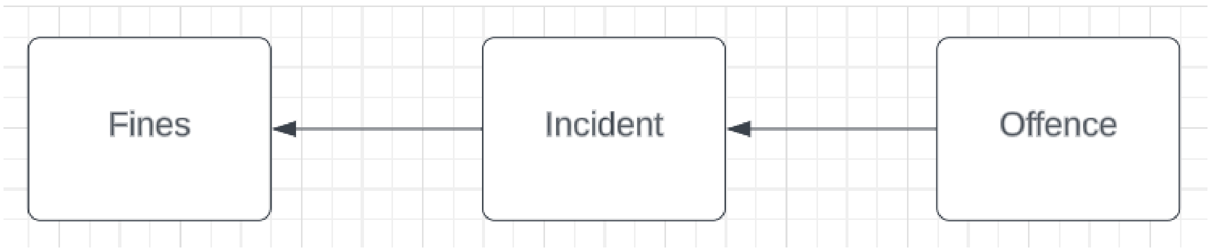
Moving onto the admin tools, the new user adds a record to either just the user table or the user and officer table depending on whether the user is an officer.



Data flow for new officers. Made using LucidChart

New Offences adds a new offence to the offence table.

New Fines creates a fine for an incident but needs to know the maximum value and maximum points it can set for the incident based on the offence that is associated with the incident. A JavaScript script then sets that as the limits for those fields.



Data flow for new fines. Made using LucidChart

The Audit Trail also uses a functional programming approach since there are so many things to keep track of. The backend only must handle queries to a single table which is the audit. The filter function on this webpage uses concatenation assignment operator to change the queries based on the filters that are applied. The audit-functions are stored in another separate file and is called in each file that requires any kind of logging. This file also contains another function that gets the old values before the action proceeds at the backend and is logged against the new values in the audit table.

Each webpage gets a dedicated css file in case there needs to be any changes to a specific page Every css file has @media styling for a responsive design and to be used on mobile devices.

Contact any of the current admins for any related information.