COMP1004 Coursework

Overview

For this coursework we will use the same database as the one used for the SQL quiz. You will build a front end using HTML/CSS/JavaScript that connects to the database and allows users to query and update the database.

Specification

Front end

The front end must be implemented using HTML, CSS, and JavaScript. No library or framework is allowed such as Bootstrap (CSS) or React (JavaScript). No other language is allowed, either, such as PHP or Python.

The front end code must be stored in a **GitHub repository**. GitHub is similar to the GitLab you used for other modules. Please refer to the online guide if you are not familiar with GitHub.

The front end must be live online using the **GitHub Pages** and can communicate with the back end database hosted on Supabase (more details below). Please refer to this page if you are not familiar with GitHub Pages. The live version on GitHub Pages will be used for marking.

Back end

The back end database must use Supabase, which provides an online PostgreSQL database (relational).

Access to the database must be through the **REST API** generated by the Supabase. Connecting directly to the database or creating your own REST API is not allowed.

Access to Supabase must use its JavaScript client. No other library is allowed.

Database

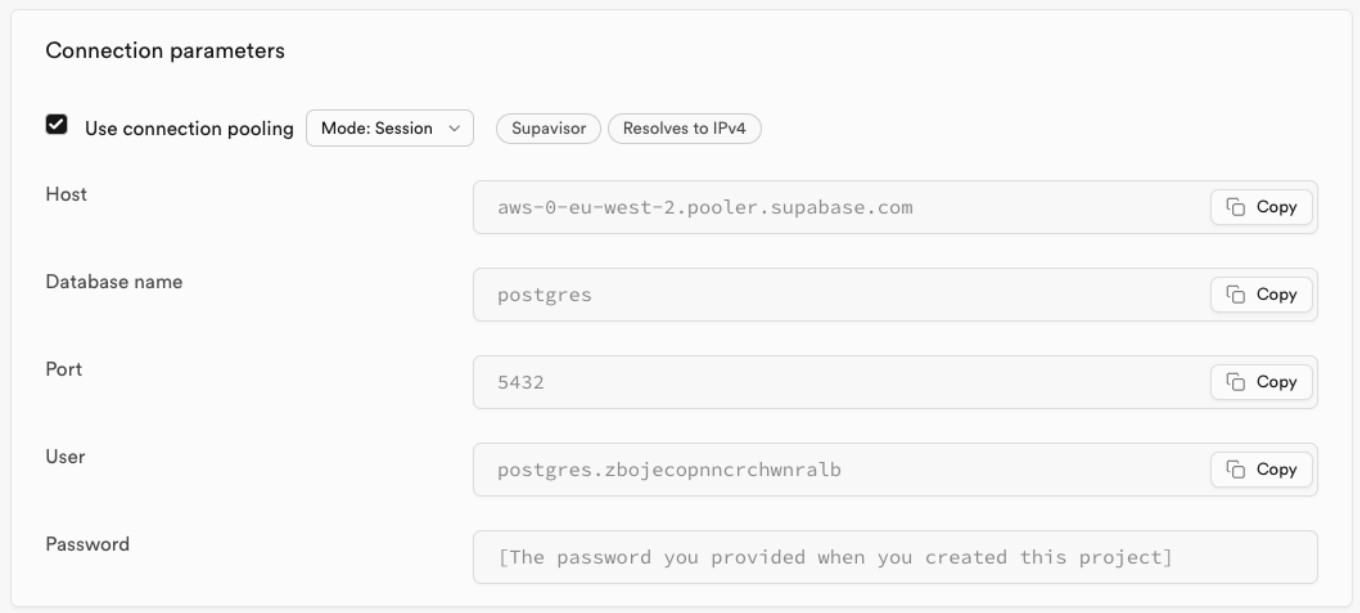
This coursework will use a simplified version of the database used for the module quiz with only two tables: `People` and `Vehicle`. The files are in the attached zip file in SQLite (`.db`) and CSV format.

The database tables and records must be kept as they are. You need to restore the database to its original state when submitting the work if changes are made during development. Any change would cause error during marking and lead to loss of marks.

There are a few ways to import the data into Supabase (more details on this page):

- You can always create the tables and enter the data manually, which might take less time than the options below.
    - There are only two tables ('people' and 'vehicles') and not many records in each.
    - Please make sure the data is entered correctly, otherwise may cause error for marking.
- Using the CSV import available on the Supabase web interface (details here).
    - The CSV files for each table are in the attached zip file.
    - You need to create the table in Supabase before importing the CSV file.

- Using pgloader for import the SQLite file (.db). This is mentioned on the same page.
    - This works best on unix-like systems such as Linux.
    - For MacOS, it can be installed using Homebrew.
    - For Windows, it can be installed under the Linux Subsystem.
    - You can find the connection settings under 'setting' >> 'database' in Supabase (below is the screenshot of an example).



Submission

Only **one zip file** will be accepted for submission.

The zip file should include all the files for the front end, i.e., a copy of your GitHub repository. There is no need to include any files for the back end or database.

The zip file should also include a **markdown** (**preferred**) or **txt** file that has (**no Word or PDF file**):

1. The URL to the GitHub Page where the front end is hosted.
    - This is the version that will be used for marking, so please make sure it is working.
    - Please do not make any further changes after the submission, otherwise it will be considered as late submission (using the last commit date/time).
2. A description of the additional work for HTML, CSS, JavaScript, and/or database.
    - Please see the marking rubrics below for the details about additional work (to achieve full mark for each criteria).
    - Please describe **separately for each aspect**, i.e., HTML/CSS/JavaScript/database, what the additional work is, and where they are (file name and line number).
    - Please say so in this document if you don't attempt any of these.
    - Only the work described will be considered for mark.

Marking

Marking library

- Some of the marking criteria will be evaluated with a library such as playwright.

- There will be specific requirements to ensure this, such as the text of the link to a search page has to be 'People search'.
- Falling to follow these requirements will lead to error and loss of marks. For example, if there is no link with text 'People search', the library will regard this as a link is missing, even there is a working link called 'Search - People'.
- These requirements and a sample test file will be released later. The sample file will not include all the tests that will be used for marking. You are welcome to create your own tests based on the coursework criteria (see below).
- Please use services such as cron-job.org to stop your database from pausing (Supabase pauses a database after 7 days of inactivity). Please see the lecture on Supabase for details (week 9). It will not be possible to mark the database part of the coursework if Supabase is paused.

Rubric

The total coursework mark is 25. Meeting all the 'criteria' (below the rubrics table) with get you 80% for each aspect (html/csss/js/db)/. To achieve full mark, you may need to go beyond what is covered in the lecture. Please see the last row of the rubric table for details.

The marking is based on the end result, e.g., whether a required feature is available and working as prescribed. You are free to choose how it is implemented so long as it meets all the requirements, e.g., not using an external library.

Marks	**HTML (5)**	**CSS (5)**	**JavaScript (10)**	**Database (5)**
0	No HTML code is provided.	No CSS code is provided.	No JavaScript code is provided.	No database is provided.
1 (2 for JavaScript)	Some HTML code is provided but there are many errors.	Some CSS code is provided but there are many errors.	Some JavaScript code is provided but there are many errors.	A database is provided but there are many errors.
2 (4 for JavaScript)	Some HTML code is provided and some of the HTML criteria are met (see criteria below).	Some CSS code is provided and some of the CSS criteria are met (see criteria below).	Some JavaScript code is provided and some of the JavaScript criteria are met (see criteria below).	A database is provided but there are a few errors.
3 (6 for JavaScript)	HTML code provided meets most of the HTML criteria (see criteria below).	CSS code provided meets most of the CSS criteria (see criteria below).	JavaScript code provided meets most of the JavaScript criteria (see criteria below).	A database is provided with no error.

Marks	HTML (5)	CSS (5)	JavaScript (10)	Database (5)
4 (8 for JavaScript)	HTML code provided meets all the HTML criteria (see criteria below).	CSS code provided meets all the CSS criteria (see criteria below).	JavaScript code provided meets all the JavaScript criteria (see criteria below).	A database is provided with no error and database query and update perform as required.
5 (10 for JavaScript)	HTML code provided meets all the HTML criteria (see criteria below) and additional efforts are made to improve front tned **accessibility**, e.g., support for vision impaired users.	CSS code provided meets all the CSS criteria (see criteria below) and additional efforts are made to make the front end **responsive**, i.e., adapts to different screen size.	JavaScript code provided meets all the JavaScript criteria (see criteria below) and **playwright tests** are created for all the conditions and exceptions for the third JavaScript and database criteria, such as owner exists/does not exist in the database, missing data, etc.	A database is provided with no error and database query and update perform as required. **The same as the extra requirement for JavaScript**, i.e., Playwright tests for the third criteria.

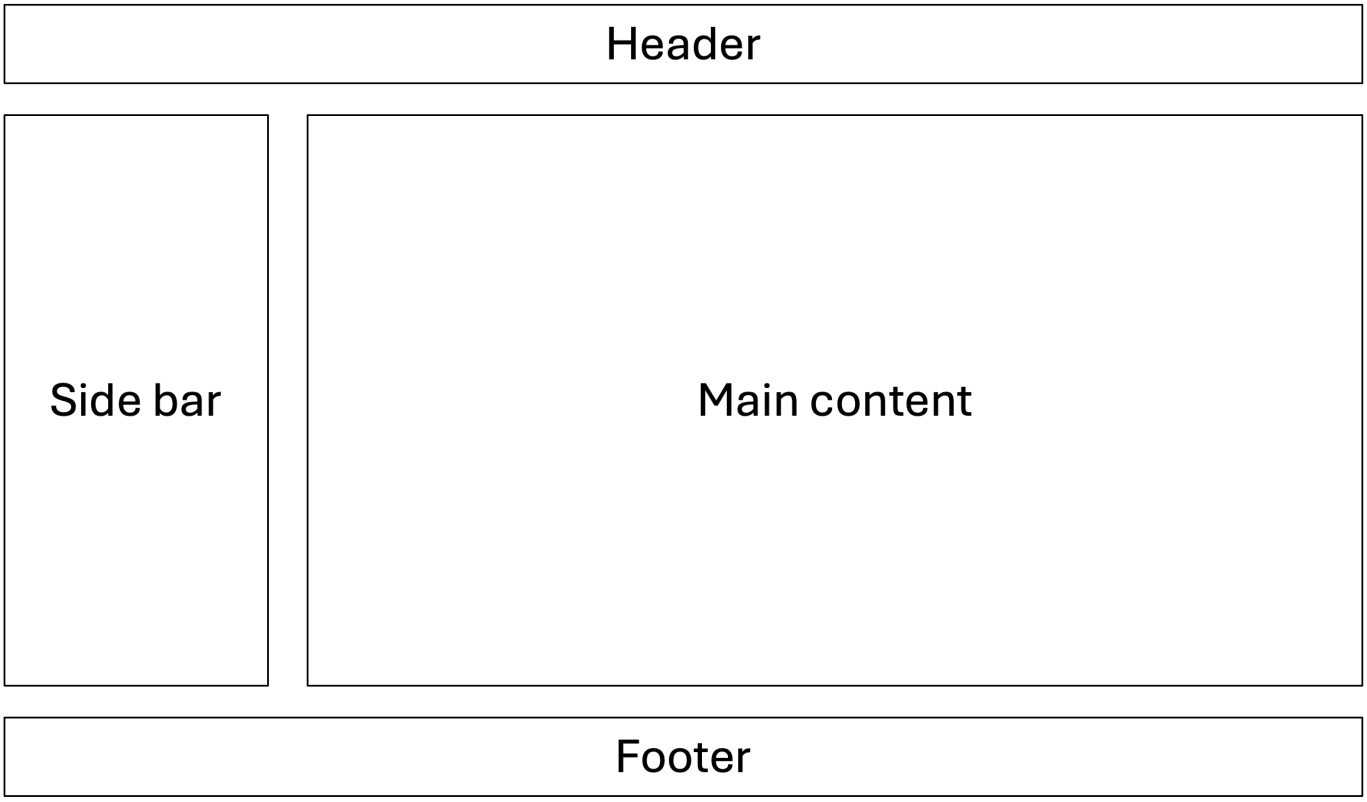
Criteria - HTML

1. There are at least three HTML pages, one for each database query/update.
2. The files are named correctly.
3. The page meta information includes language, character set, and title.
4. The heading and text elements are used correctly.
5. An unordered list `<ul>` is used to create the navigation links.
6. All the pages have the same navigation menu and the links works correctly.
7. Each page should have four sections: header, main, sidebar, and footer. These are marked up correctly using semantic elements.
8. There is at least one image or video for each page in the side bar and all the required information is present and correct.

Criteria - CSS

1. All the pages should share the same external CSS file. Justifications are provided (as code comment) for internal or inline CSS formatting.
2. CSS flex is used to place the navigation links horizontally.
    - Class is used to apply CSS flex to the navigation links only and not other `<ul>` element.
    - The links should use all the horizontal space.
3. Location selector is used to remove the bullet in front of the navigation links.
    - Class or ID is not allowed for this.

    - This should only remove the bullets from the navigation links and not any other unordered list items.
4. Add border, margin, and padding to header, main, sidebar, and footer.
5. Use CSS Grid to layout the page (see the figure below):
    - The header should be at the top of a page, occupying the fully width;
    - The side bar should be on the left of the main content;
    - The width ratio between the side bar and the main content should be 1:4.
    - The footer should be at the bottom of a page, occupying the fully width.



Criteria - JavaScript and database

1. The user should be able to look up people by their names or their driving licence number (by typing either of these in to the system). If the person is not in the system it should give an appropriate message. This search should not be case sensitive and it should work on partial names, e.g., “John”, “Smith” and “John Smith” would all find John Smith. If there are several people with the same name they should all be listed.
2. The user should be able to look up vehicle registration (plate) number. The system will then show details of the car (e.g., type, colour etc.), the owner’s name and license number. Allow for missing data in the system (e.g., the vehicle might not be in the system, or the vehicle might be in the system but the owner might be unknown).
3. The user should be able to enter details for a new vehicle. This will include the registration (plate) number, make, model and colour of the vehicle, as well as its owner. If the owner is already in the database, then it should be a matter of selecting that person and assigning them to the new vehicle. If the owner is not in the database they should be added (along with personal information including the license number).