

# 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

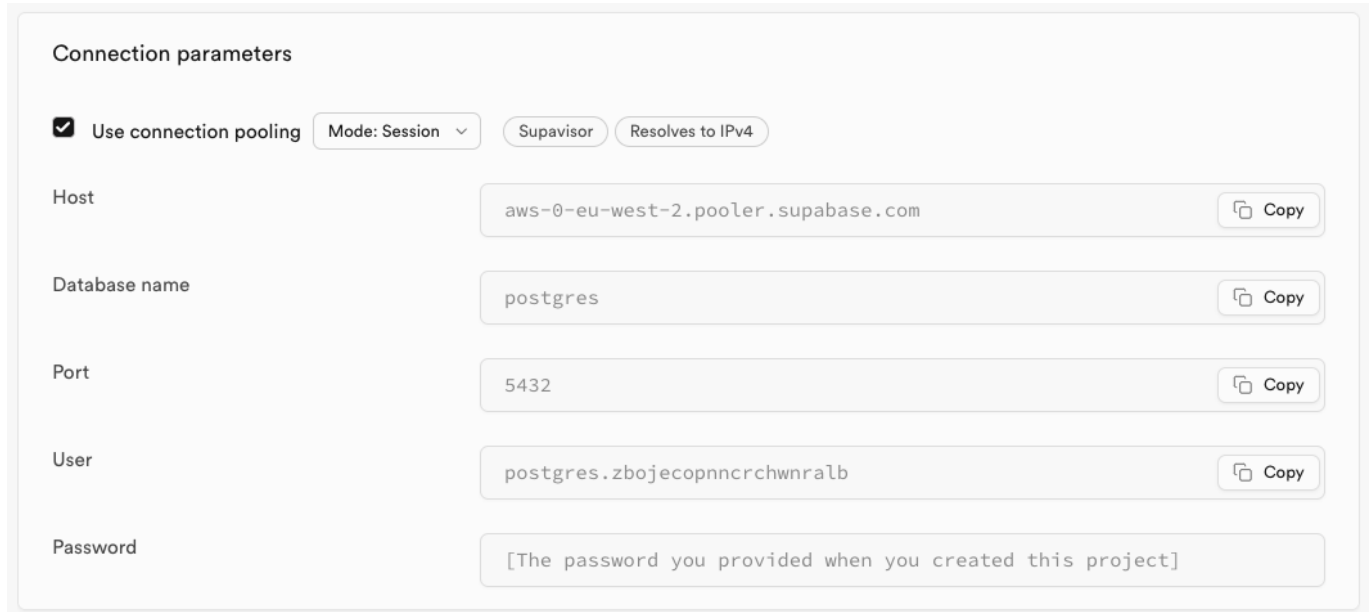
This coursework will use a simplified version of the database used for the module quiz with only two tables: **People** and **Vehicle**. The files are in the attached zip file in SQLite (**.db**) and CSV format.

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).



The screenshot shows the 'Connection parameters' section of the Supabase dashboard. It includes a checkbox for 'Use connection pooling' which is checked, and three buttons: 'Mode: Session', 'Supervisor', and 'Resolves to IPv4'. Below these are input fields for 'Host', 'Database name', 'Port', 'User', and 'Password'. Each field has a 'Copy' button to its right. The 'Host' field contains 'aws-0-eu-west-2.pooler.supabase.com', 'Database name' contains 'postgres', 'Port' contains '5432', and 'User' contains 'postgres.zbojecopnnrchwnralb'. The 'Password' field contains a placeholder text: '[The password you provided when you created this project]'.

## 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'.

Please see the last section for details.

- Please use services such as [cron-job.org](https://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) will get you 80% for each aspect (html/css/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 achieve 100% <b>accessibility</b> score in the Lighthouse test (more details below).	CSS code provided meets all the CSS criteria (see criteria below) and additional efforts are made to make the front end <b>responsive</b> , i.e., adapts to different screen size (more details below).	JavaScript code provided meets all the JavaScript criteria (see criteria below) and <b>playwright tests</b> 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. <b>The same as the extra requirement for JavaScript</b> , 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, i.e., all letters are in lower case and there is no special characters except hyphen.
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.
9. Accessibility: we will use the 'Lighthouse' tool in the browser development tools to measure the webpage accessibility.
  - For this requirement you only need to run the 'Accessibility' test.

1

Lighthouse

2

Accessibility

3

Analyze page load

Generate a Lighthouse report

Mode

Learn more

Navigation (Default)

Timespan

Snapshot

Device

Mobile

Desktop

Categories

Performance

Accessibility

Best practices

SEO

Progressive Web App

Plugins

Publisher Ads

13:56:08 - 127.0.0.1:3000

http://127.0.0.1:3000/index.html

93

Accessibility score

Accessibility

These checks highlight opportunities to [improve the accessibility of your web app](#). Automatic detection can only detect a subset of issues and does not guarantee the accessibility of your web app, so [manual testing](#) is also encouraged.

What can be improved

NAMES AND LABELS

▲ Form elements do not have associated labels

These are opportunities to improve the semantics of the controls in your application. This may enhance the experience for users of assistive technology, like a screen reader.

There are other accessibility issues that can't be checked automatically.

ADDITIONAL ITEMS TO MANUALLY CHECK (10)

Show

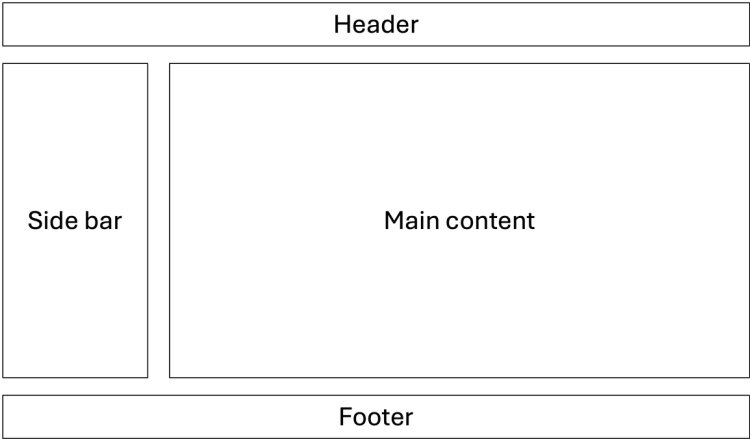
These items address areas which an automated testing tool cannot cover. Learn more in our guide on [conducting an accessibility review](#).

PASSED AUDITS (14)

Show

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 figures 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.



People Search

People search

[Vehicle search](#)

[Add a vehicle](#)



Search by driver name

Search by driving license number

Submit

(For result display)

This is a sample for the COMP1004 coursework.

6. Responsive layout: you need to change the page layout to the one shown below when the page width is less than 500px using CSS **media queries**.
1. The links are now vertically stacked instead of horizontally placed.
  2. The side bar is move under the **main**.
  3. The width ratio between the side bar and the footer is 1:4.

## People Search

People search  
[Vehicle search](#)  
[Add a vehicle](#)

Search by driver name

Search by driving license number



This is a sample for the COMP1004 coursework.

### 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).

### Testing with Playwright

Some of the functions will be tested with the Playwright library. There are some requirements needed to make this work (e.g., the output has a ID called `results`), so Playwright can find the right HTML element for the test.

A sample test spec file is attached to the submission page so you can make your work would work with the Playwright tests. This sample test covers most of the requirements below, but not all of them. Therefore you still need to make sure your code meet all the requirements. Failing a test because of not meeting the requirements will lead to lose of marks.

Change the `websiteURL` at the beginning of the test file to the URL of your people search page, this can be a local one (such as the one in the test file) or the GitHub Pages URL. The test will always start from this page.

As mentioned earlier, you need to keep the tables and records as they are provided. Otherwise some of the tests will fail because the returned results will be different from what is expected. This will lead to lose of mark.

Always include all the information (about a person or vehicle) in the search results. The test will check different fields.

Requirements (all the text and IDs are case sensitive):

- Page heading:
  - The people search page should have a `h1` heading with text 'People Search'
  - The vehicle search page should have a `h1` heading with text 'Vehicle Search'
  - The add vehicle page should have a `h1` heading with text 'Add a Vehicle '
- Navigation link text :
  - The link to the people search page should have text 'People search'
  - The link to the vehicle search page should have text 'Vehicle search'
  - The link to the add vehicle page should have text 'Add a vehicle'
- The `<ul>` used for navigation link must be inside the `<header>`.
- All the page sections (header, main, side bar, and footer) should have:
  - A solid black border with width 1px.
  - Margin and padding on all side with a value of 10px.
- The element(s) that CSS grid applies to should have the ID 'container'.
  - This means the HTML element(s) that has/have CSS grid in its/their CSS rules. Elements affected by the layout but do not have grid in their css rule does not count.
- For both searches (people and vehicle):
  - The button to start the search should have the label text 'Submit'
  - The HTML element showing the search results should have an ID 'results'.
  - Each search result record is shown in a `<div>`, inside the 'results' element.
  - There should be an element with ID 'message' (outside the 'results' element) that:
    - Displays the text 'Search successful' if the search is successful.
    - Displays the text 'No result found' if the search returns nothing.
    - Displays the text 'Error' if:
      - Both fields in the people search form are empty.
      - Both fields in the people search form are filled (only one can be used).
      - In the vehicle search form the registration number field is empty.



- For people search,
  - The text input field for driver name should have the ID 'name'.
  - The text input field for license number should have the ID 'license'.
- For vehicle search, the text input field for vehicle registration/plate number should have an ID 'rego'.
- For the add vehicle page,
  - The text input field should have the following IDs:
    - 'rego' for the vehicle registration/plate number
    - 'make' for the vehicle make
    - 'model' for the vehicle model
    - 'colour' for the vehicle colour
    - 'owner' for the vehicle owner
  - The button to add a vehicle should have the label text 'Add'.
  - If the owner does not exist in the people table, a new form should appear asking the user to enter the owner information, and the input fields in this form should have the following IDs:
    - 'personid' for the ID in the people table
    - 'name' for the owner name
    - 'address' for the owner address
    - 'dob' for the owner date of birth
    - 'license' for the owner's license number
    - 'expire' for the owner's license expiring date
  - The button to add a new owner should have the label text 'Add owner'
  - After entering all the information and clicking the 'Add' (and 'Add owner') button, there should be an element with ID 'message' that:
    - Displays the text 'Vehicle added successfully' if there is no error.
    - Displays the text 'Error' if any information is missing. All the information about the vehicle and the new owner is required, i.e., there cannot be any empty field.