COMP 636: Web App Assessment

Milestone submission due: 5pm Wednesday **24 January 2024**

Final submission due: 5pm Wednesday **14 February 2024**

Worth: 50% of COMP636 grade

Submit via Akoraka | Learn, with files set up and available on GitHub and pythonanywhere.

Introduction

Selwyn Panel Beaters have decided that they want to upgrade their internal system from the text-based system to a web-based system. Your task is to develop a small Web Application to help them manage customers, jobs, services, parts and billing. You will also write a report.

In the existing system, Selwyn Panel Beaters record multiple items (services, parts) by entering them multiple times. In the web system they want to be able to enter a single job for a vehicle and then record the quantities of services and parts within that job.

There are two different types of users who will use the system. The first is a technician who can select a customer job and add services and parts to that job.

The second is an office administrator who can book (create) jobs, add customers, services and parts, bill customers and enter payments.

Download the Web Application Project Files from the Assessment block on the Learn page. These will get you started, including for the Milestone. You will add more routes and templates as you develop your app.

Important

This is an **individual** assessment. You may not collaborate or confer with others. You may help others by *verbally* explaining concepts and making suggestions in general terms, but without directly showing or sharing your own code. You must develop the logical structure and the detail of your code on your own, even if you are working alongside others. Code that is copied or shares a similar logic to others *will receive zero marks for both parties*. The University policy on Academic Integrity can be found here.

Functional Requirements

There will be two interfaces:

- An interface for the technicians as the main users of the system. This allows technicians to update customer jobs with parts and services.
- An Administrator interface for admin staff to use to update, edit and add data. This interface requires administrators to access the system via a link from the menu system (*do not* add password functionality – security would be added to a full system to restrict access, but will lose marks if added for this assessment). The Admin link will provide a gateway to the Admin features. The Admin features must not be visible in any of the technician interface (apart from the Admin link in the menu).

Technician Interface on the default / route

- **Current Jobs:** Change the **currentjobs** page to display the customer's name, rather than the ID only. Modify or tidy the template as appropriate. This page only shows jobs that are not completed.
- **Job:** A job is selected (via a link) from the **currentjobs** page. On this page services and parts can be added to a job. The technician can select a part or service from existing lists (which are maintained by the Admins). The technician enters the number of units of each part or service used (e.g., 2 headlights or 1 respray). This page should also show the services and parts that have already been used in the job. A job can also be marked as completed.
  After a job is marked complete, the job total cost is calculated. Once a job is marked complete it cannot be modified.

Administrator Interface

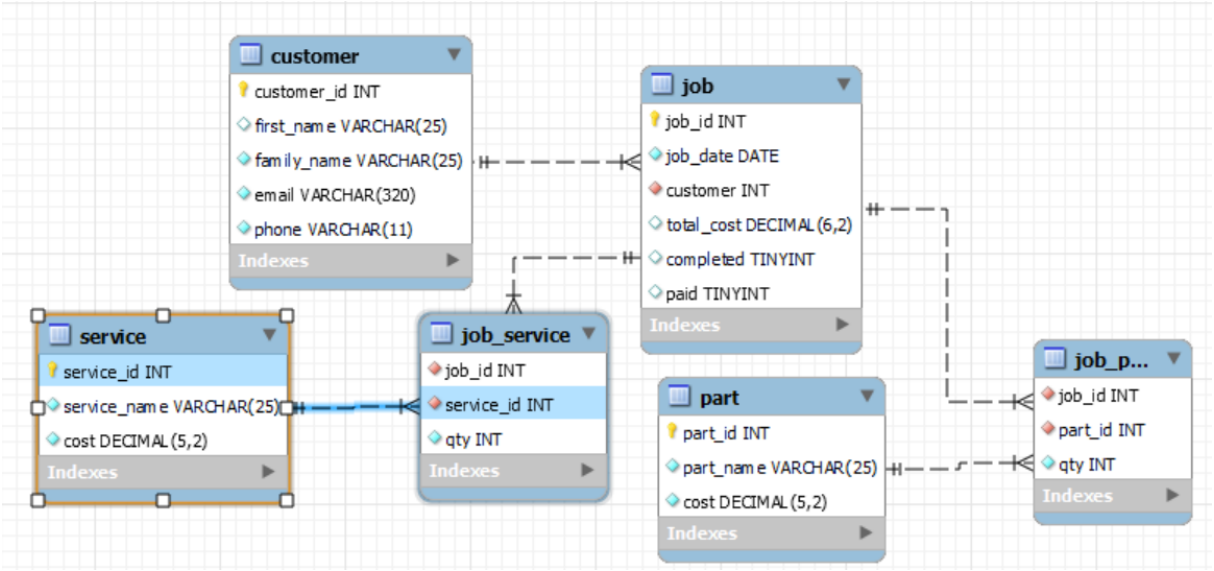
- **Customer list:** Display a list of customers, ordered by surname, then by first name.
- **Customer search:** Search for customers by first name or surname, allowing for partial text matches. Order the results appropriately.
- **Add customer:** Add a new customer to the system (surname, phone number and email address are required fields)
- **Add Service:** Add a new service to the system (name and cost are required)
- **Add Part**: Add a new part to the system (name and cost are required)
- **Schedule Job:** Select a customer and a date for the job to be booked for. The date must be today or in the future. Only the date is required – a time on that day is not required.
- **Unpaid bills & Pay Bills:** Shows all unpaid bills in date then customer order, which can be filtered by customer. A bill can then be selected so that it can be marked as paid by the Admin. Bills that are paid should not show on the list.
- **Billing History & Overdue Bills**: A report showing all bills that, with bills that are unpaid more than 14 days after the date of the job highlighted in red. The display should include the customer details, the date of the job, the total cost of the job. Bills should be grouped by customer, so that the customer details are only shown once above the list of bills for that customer. Customers should be shown in last name, first name order. Bills should be shown with the oldest bill first.

Report

Your report must be created using GitHub Markdown format and saved in the **README.md** file of your GitHub repository. It does not need to be a formal report – a tidy document using the following headings will be sufficient. Write a brief project report that includes:

- **Web application structure:**
  - Outline the **structure** of your solution (routes & functions, and templates). This should be brief and can be text-based or a diagram (as an image).
  - It must indicate how your **routes, functions** and **templates** relate to each other, as well as what **data** is being passed between them.
  - Do not just give a list of your routes. Do not include all of your code. This relates to the code, not the user experience – so do not describe the interface, user experience or HTML layout.
- **Design decisions**:
  - Discuss the **design decisions** you made when designing and developing your app: what design options you weighed up, why you designed your app the way that you did, your decisions about the routes, templates, navigation, broad layout, etc., that you made.
  - For example, when the edit button is clicked on a page, does that open a different template for editing or does it use the same template with IF statements to enable the editing? Did you use GET or POST to request and send data, and how and why? You will have considered many design possibilities. These are only two examples.
  - Note your decisions as you work, so you do not forget them!
- **Database questions**: Refer to the supplied ***spb_local.sql*** file to answer the following questions:
  1. What SQL statement creates the job table and defines its fields/columns? (Copy and paste the relevant lines of SQL.)
  2. Which line of SQL code sets up the relationship between the customer and job tables?
  3. Which lines of SQL code insert details into the parts table?
  4. Suppose that as part of an audit trail, the time and date a service or part was added to a job needed to be recorded, what fields/columns would you need to add to which tables? Provide the table name, new column name and the data type. (*Do not* implement this change in your app.)
  5. Suppose logins were implemented. Why is it important for technicians and the office administrator to access different routes? As part of your answer, give two specific examples of problems that could occur if all of the web app facilities were available to everyone.
- **Image sources:** It is not necessary to use any external images in your web app, but if you do, ensure you reference the image source in your report.

Data Model



Model Notes:

Child table.*field* *	(refers to)	**Parent table**.*field*
job.*customer*		**customer.*customer_id***
job_part.job_id		**job.job_id**
job_part.part_id		**part.part_id**
job_service.job_id		**job.job_id**
job_service.service_id		**service.service_id**

* the 'Foreign Key'

Project Requirements

You must:

- Use only the COMP636 technologies (Python & Flask, Bootstrap CSS, MySQL). Do not use SQLAlchemy or ReactJS (or other similar technologies) in your solution.

  Do not use any scripts, including JavaScript, except for the <script> at the bottom of base.html. Do not write your own CSS (use Bootstrap).

- Use the provided SQL files to create the database within your MySQL database & PythonAnywhere. This also creates initial data in the database.
  - You can re-run this SQL script at any time to reset your data back to the original version and remove any changes you made to it.
- Use the provided files to develop a Flask Webapp that:
  - **Must be in a folder called 'spb' (locally and on pythonanywhere)**
  - Meets the functional requirements.
  - Is appropriately commented.
  - Connects to your database.

    - Uses %s to insert values into SQL statements.
    - Provides appropriate routes for the different functions.
    - Provides templates and incorporates HTML forms to take input data.
    - Uses Bootstrap CSS to provide styling and formatting.
- Include your report as outlined above.
    - This report must be created using GitHub Markdown and saved in the **README.md** file of your GitHub repository.
- Create a **private** GitHub repository that contains:
    - All Python, HTML, images and any other required files for the web app.
    - A **requirements.txt** file showing the required pip packages.
    - Your project report as the **README.md** document.
    - Your repository must have a **.gitignore** file and therefore *not* have a copy of your virtual environment.
    - Add **lincolnmac (computing@lincoln.ac.nz)** as a collaborator to your GitHub repository.
    - **Your repository must show a minimum of two commits per week after the milestone submission.**
- Host your system (including database) using pythonanywhere.
    - Add **lincolnmac** as your "teacher" via Account > Education.
    - The webapp must be in a folder called '**spb'**

Project Hints

Create your GitHub repository first, and create all your required code and files in your local folder. You are required to commit and push changes from your local computer to your GitHub repository at least twice a week.

pythonanywhere is case sensitive so test your app early – **we will mark the pythonanywhere version of your app**.

Spend some time sketching the structure of your application before you start developing. Think about which features could share the same (or nearly the same) templates. Remember that you can nest templates (templates within templates).

Take note of your design decisions, compromises, workarounds, etc. for your report as you develop your web app. Otherwise you may struggle to remember all of the issues you worked through afterwards.

Milestone Submission (10 marks, due 24 January)

This milestone is to ensure that your app is correctly configured, and any set-up issues are resolved early. The milestone does not require any changes to the code and templates provided. By this date you only need to sync and share the provided files on GitHub, provide us teacher access to your PythonAnywhere and host the provided code on PythonAnywhere so that the web app and provided routes run correctly.

Submit the following via the link on the Learn COMP636 page:

- Your PythonAnywhere URL (e.g., joebloggs.PythonAnywhere.com/)
- Your GitHub repository name (e.g., joebloggs99)

For this submission you **must** have:

- Your GitHub repository set up correctly.
- The provided files loaded in GitHub and in PythonAnywhere.
- Your database set up on PythonAnywhere.
- Your app hosted on PythonAnywhere.
- The **/currentjobs** route working (as provided in the files).
- Granted access to your PythonAnywhere account (set **lincolnmac** as teacher).
- Granted access to your GitHub repository (**lincolnmac** or computing@lincoln.ac.nz as collaborator) .

IMPORTANT: Do not change your PythonAnywhere files until *after* you have received your Milestone marks. You may continue to work in the local copy on your computer in the meantime and you should also commit and push to your GitHub repository.

At this submission we will check your GitHub, PythonAnywhere and database setup.

Set-up Requirement	**Marks Available**
GitHub Repository set up and shared	3 marks
PythonAnywhere web app hosting correctly configured, including home URL and database setup, and teacher access granted	5 marks
/currentjobs route and page (as provided) running on PythonAnywhere	2 marks
TOTAL	**10 marks**

Final Submission (90 marks, due 14 February)

Submit your URLs *again* via the link on the Learn COMP636 page:

- Your pythonanywhere URL (e.g., joebloggs.pythonanywhere.com/)
- Your GitHub repository name (eg, joebloggs99)

This confirms where your work is, and tells us that your final submission is ready for marking.

Final Submission Marking

Report and General Project Aspects (25 marks):

Project Element	Marks Available
Project Report – Part 1: • Outline the structure of your solution (routes & functions). • Detail any assumptions that you made, discussing your design decisions. • Report created using GitHub Markdown and saved in the README.md file of your GitHub repository	9 marks • 4 marks assigned to appropriate and accurate structure outline. • 4 marks assigned to Assumptions and Design Decisions. • 1 mark for .md being in the right place and a reasonable length. (Just a heading, or a couple of sentences, is not a reasonable length.)
Project Report – Part 2: • Report Database Questions sufficiently answered.	10 marks
Spelling, presentation, etc	2 marks Spelling, punctuation, grammar, and presentation, and appropriate referencing of external images (if any).
Consistent 'Look and Feel' (interface, Bootstrap styling & templates, ease of use, etc)	4 marks
TOTAL	**25 marks**

Functional Project Aspects (65 marks):

Within each of the functional areas (see table below with indicative marks) we are looking for:

- Functionality working as specified in the requirements.
- Well commented and formatted HTML, SQL, and Python code throughout.
- A user interface that looks and functions to a professional standard, including Bootstrap colour and styling choices, sensible navigation and appropriate sorting of lists.
- ID numbers for table rows are mainly for internal system use only and should only be made visible to system users when mentioned in the requirements.
- Data Validation on forms. Wise choice of form elements.
- Well-structured SQL queries.
- Appropriate naming, both of variables and labels

An *indication* of marks (may be adjusted when marking) :

Item	**Interface**	**Functionality**	**Approx. Marks**
Access	Admin	Admin home page exists with appropriate links. Admin functionality only linked via menu.	1
Navigation	*All*	Sensible, well laid-out navigation throughout.	4
List currentjobs	Technician	Updated list to show customer name, formatting improvements	2
Job Details	Technician	Job selectable (from currentjobs). Shows part and services used, additional parts and services can be added.	6
		Job total cost updates after each change to a job.	6
Customer List & Search	Admin	A list of customers is displayed. Customers can be searched, partial text matches	6
Add Customers, Parts, Services	Admin	Appropriate interface, forms, validation for entry of new customers, parts and services	14
Schedule Job	Admin	Appropriate interface to choose the current or a future date and customer	6
Unpaid Bills and Bill Payment	Admin	Display of unpaid bills and ability to update payment status	8
Overdue Bills	Admin	Showing all bills with red highlight on those that are unpaid and overdue. Bills appropriately ordered and grouped by customer.	12
TOTAL			**65**

You may modify **data** in your database for testing purposes and may add new data, but you must **not** modify the schema.

Markers will modify or add alternative data to your database as part of the marking process.