SET Lab Management Tool

Description:

========

This is a web-based application designed for the SET Lab management team to view lab classes in various visualisations. The tool is implemented in the Python, Dash framework and Plotly Express. It allows users to upload course data in CSV or Excel format and interactively select courses to view data visualisations.

Features

- File upload: Users can upload CSV or Excel files containing course data directly into the web application.
- Filters data based on selected terms, courses, and date range for Course Selection or location, building, room, and date range for Location Selection.
- Generates various visualisations (Pie Chart, Table, Calendar, Timeline) to display information about class enrollment and room capacity.
- Pie Charts: Upon selecting a course, the application generates a pie chart displaying the ratio of enrolled students to the total room capacity.
- Table: displaying lab classes detail.
- Calendar: display lab classes in a calendar view, user can filter date range.
- Timeline: displaying lab classes time period in each date.

Prerequisites

- Operating System: Windows, macOS, or Linux
- Python: Version 3.6 or later (check with python --version in your terminal)
- Package Manager: pip (usually comes pre-installed with Python 3.6+)

Installation

Step 1: Forking the repository

To work on an open-source project, you will first need to make your copy of the repository. To do this, you should fork the repository and then clone it so that you have a local working copy.

Get your own Fork/Copy of repository by clicking the Fork button in right upper corner of Github.

Step 2: Clone the Forked Repository

After the repository is forked, you can now clone it so that you have a local working copy of the codebase.

To make your local copy of the repository follow the steps:

- Open the Command Prompt
- Type this command to clone the project:

Bash

```
$ git clone https://github.cs.adelaide.edu.au/MCI-Projects-2024/Team-8Q.git
```

Step 3: Setting up Project

1. **Verify Python Version:**
    - Open a terminal or command prompt.
    - Type `python --version` and press Enter.
    - If you have Python 3.6 or later, proceed to the next step. If not, download and install the latest version from https://www.python.org/downloads/.
2. **Install pip (if not already installed):**
    - **Windows:**
        - Download `get-pip.py` from https://bootstrap.pypa.io/get-pip.py.

        - Open a command prompt as administrator, navigate to the downloaded file's location, and run `python get-pip.py`.

    - **macOS:**

        - Open a terminal application (usually found in Applications > Utilities).

        - Type the following command and press Enter:

          ```
          python -m ensurepip --upgrade
          ```

          This command uses Python's built-in `ensurepip` module to install or upgrade pip if it's not already present.

    - **Linux:**

        - Type the following command and press Enter:

          ```
          sudo apt-get install python3-pip
          ```

3. **Install Required Libraries:**

    - Inside the activated virtual environment, run the following command in your terminal:

      Bash

      ```
      pip install dash dash-renderer dash-core-components dash-html-components dash-bootstrap-components plotly pandas datetime base64 io calendar
      ```

    - or run below bash command to install all required libraries

      ```
      pip install -r requirements.txt
      ```

4. **Verification**

    - After installation, try importing the libraries to ensure everything is set up correctly:

      Python

      ```
      import dash

      print(dash.__version__)
      ```

      The line of code `print(dash.__version__)` is used to verify the installation of a Python library and print its version number.

5. **Run the Python script**

   Bash

   ```
   python main.py
   ```

6. **Accessing the App**

   - Once the script runs successfully, you should see output in your terminal indicating that the server is running, typically on port 8050. The exact message might vary slightly depending on your Dash version.
   - Open a web browser and navigate to http://127.0.0.1:8050/ (or localhost:8050/) to access your Dash application.

7. **Additional Tips**

   - Consider using a code editor or IDE with Python support for a more streamlined development experience (e.g., Visual Studio Code, PyCharm, Spyder).
   - Dash documentation (https://dash.plotly.com/) for effective app building and to familiarise with Dash.

Deploy on Google Cloud Service

Prerequisites

- Google Cloud Account
- Google Cloud SDK: Install the Google Cloud SDK on your local machine.
- Docker: Install Docker on your local machine.
- Billing: Enable billing for your Google Cloud project.

Install the Google Cloud SDK on your local machine:

1. Download the installer:

- Head to the Google Cloud official documentation for installing the gcloud CLI https://cloud.google.com/sdk/docs/install.
- The page will offer downloads for different operating systems (Windows, macOS, Linux). Choose the one that matches your machine.

2. Install based on your OS:

- **Windows:** There's a friendly installer for Windows. Download it, run the installer, and follow the on-screen instructions. You can choose to install Python during this process (recommended as it's a dependency).
- **macOS/Linux:** The installation involves downloading a compressed archive and extracting it. The specific commands will differ slightly depending on your system type (e.g., 64-bit vs 32-bit). Refer to the Google Cloud documentation for detailed instructions based on your OS.

3. (Optional) Add gcloud to your PATH:

- By default, the installer might not automatically add gcloud commands to your system path. This path determines which programs your terminal can find.
- The Google Cloud documentation provides instructions for adding the path manually for different operating systems https://cloud.google.com/sdk/docs/install.

4. Verify the installation:

- Open a terminal window.
- Type gcloud --version and press enter.
- If the installation was successful, you should see the installed gcloud version information.

Install Docker

1. Install Docker:

- If you haven't already, you'll need to install Docker on your machine. Head over to the official Docker website: https://www.docker.com/products/docker-desktop/
- The website offers downloads for different operating systems (Windows, macOS, Linux). Choose the one that matches your system and follow the installation instructions.

2. Check Installation Path (Windows & macOS):

- After installation, there's a chance the Docker directory might not be added to your system path by default (particularly on Windows and macOS). This path determines which programs your terminal can find.
- Refer to the official Docker documentation for your OS on how to add the Docker path to your environment variables: https://docs.docker.com/engine/install/

3. Restart Terminal (Optional):

- Sometimes, changes to environment variables might not take effect immediately in your current terminal session. Try restarting your terminal window after making path adjustments.

4. Verify Installation:

- Once you've installed and configured Docker (if necessary), open a terminal window and type:

Step 1: Set Up Google Cloud Project

- Go to the Google Cloud Console.
- Create a new project.
- Enable Required APIs (Cloud Run Admin API, Cloud Build API)

Step 2: Prepare Your Dash App

- main.py: Dash app's main script.
- Dockerfile: Instructions to containerise the app.
  A Dockerfile specifies the instructions for building a Docker image. This image will contain your Dash app's code (from main.py) and all its dependencies (listed in requirements.txt).
- requirements.txt: List of dependencies.

Step 3: Containerise Your App with Docker

- Change to the project directory and build the Docker Image by input the command:

  ```
  docker build -t your-app .
  ```

- Run the Docker Container Locally:

  ```
  docker run -p 8080:8080 your-app
  ```

- Access your app at http://localhost:8080.

Step 4: Deploy to Google Cloud Run

- Authenticate with Google Cloud:

  ```
  gcloud auth login
  ```

- Set Your Google Cloud Project:

```
gcloud config set project your-project-id
```

- Build the Docker Image with Google Cloud Build:

```
gcloud builds submit --tag gcr.io/your-project-id/your-app
```

- Deploy to Cloud Run:

```
gcloud run deploy your-app --image gcr.io/your-project-id/your-app --platform
managed --region your-region --allow-unauthenticated
```

Step 5: Access Your Deployed App

After deployment, Google Cloud Run will provide you with a URL for your app. Visit the URL to access your deployed Dash app.

Addition Detail:

After you update your source code, you'll typically run the following commands again to deploy the changes to Google Cloud Run:

```
gcloud builds submit --tag gcr.io/your-project-id/your-app
```

This command below deploys the newly built image to Cloud Run. It's optional because if you already have a deployment with the same name (your-app), Cloud Run will automatically update it with the new image:

```
gcloud run deploy your-app --image gcr.io/your-project-id/your-app --platform
managed --region your-region --allow-unauthenticated
```

Local Development:

To run and test your app locally for faster development cycle and easier debugging, you need to change arguments of the function `app.run_server()` as following:

```
# run locally
    app.run_server(debug=False, port=8080)
```

When you're ready to deploy your updated app to Cloud Run, you'll still need to rebuild the Docker image and deploy it to the Cloud Run.

If you want to build and deploy the project on cloud, please change the code to:

```
# run on Cloud
    app.run_server(debug=False, host='0.0.0.0', port=port)
```

Code Breakdown

The code is structured using various components:

1. **Libraries:**
    - dash: Framework for building analytical web apps.
    - dash.dependencies: Handles reactivity in Dash applications.
    - dcc and html: Components for layout and interactivity.
    - callback: Decorator for defining callbacks in Dash apps.
    - PreventUpdate: Prevents unnecessary updates in callbacks.
    - dash_table: Component for creating interactive tables.
    - datetime: Module for date and time manipulation.
    - pandas: Library for data analysis and manipulation.
    - base64: For base64 encoding/decoding.
    - io: For working with input/output streams.
    - calendar: Module for working with calendar data.
    - dash_bootstrap_components: Components from Dash Bootstrap library.
2. **Global Variables:**
    - app: The Dash app instance.
    - weekday_mapping: Dictionary mapping weekday abbreviations to corresponding integers (0 for Monday, 4 for Friday).
    - tech_team_mapping: Dictionary mapping abbreviations for technical teams to their full names.
3. **Layout:**
    - The app.layout defines the overall layout of the web application, including:
        - A navbar with selection options for courses or locations.

- A heading for the application title.
- File upload section for uploading a data file.
- Reset button to clear selections and filters.
- Container for term/course selection (depending on the selected page).
- Container for buttons to choose between different data visualisations (pie chart, table, calendar, timeline).
- A hidden element to store the current toggle state (e.g., 'pie').
- Containers for day and location dropdown menus (initially hidden).
- Containers for displaying data based on selections.

4. **Functions:**

   - file_feedback: Handles file upload validation and displays feedback messages.
   - display_page: Renders the layout based on the selected page (course selection or location selection).
   - course_selection_layout and location_selection_layout: Define the layout for the respective selection pages.
   - parse_contents: Parses the uploaded data file and validates its format.
   - reset and reset_location: Reset functionalities for course and location selection pages.
   - clear_output: Clears output sections when switching pages.
   - store_data: Stores the uploaded data in a hidden container.
   - set_course_term_options and set_location_term_options: Set options for term dropdown menus based on uploaded data.
   - generate_options: Generates options for dropdown menus.

- set_tech_team_options: Sets options for the Tech Team dropdown menu.
- update_location_dropdown: Updates the location dropdown based on the selected day and uploaded data.
- update_course: This is a large callback that handles several things:
    - Updates course dropdown options based on selected terms.
    - Generates visualisations (Pie Chart, Table, Timeline) based on user selection and last clicked button.
    - Updates the allowed date range for filtering data based on the available class information.
- check_dates: Determines if a set of dates (potentially a list) falls within a specified date range (start and end dates).
- create_timeline_for_selected_course: generates a timeline visualisation for courses offered within a selected date range.
- generate_course_dates: Generates a list of datetimes for a course considering meeting start/end times and weekdays offered.
- format_datetime: Formats datetime objects into a specific string representation.
- make_pie_chart_for_selected_day: Creates a pie chart to visualise room capacity compared to enrolled capacity for a selected day.
- make_pie_chart_for_group: Creates a pie chart to visualise room capacity compared to enrolled capacity for a group of courses.
- update_last_clicked_button: Updates a hidden variable to track the most recently clicked button.
- update_calendar: Updates the calendar view based on selected terms, courses, and date range.
- format_event: Formats HTML content for displaying event details in the calendar.
- parse_time: Parses a time string into a time object.

        - update_location: Updates the location view based on selected terms, tech teams, buildings, rooms, and date range. It also generates visualisations (pie chart or table) based on the last clicked button.

        - set_building_options: Sets the available building options in the dropdown menu based on the loaded data and selected tech teams (if any).

        - set_room_options: Sets the available room options in the dropdown menu based on the selected building and tech teams (if any).

Problems and Troubleshooting for Deploying on Google Cloud Service

Here are some potential problems and troubleshooting tips for developers deploying on Google Cloud Service (GCP) based on the provided information:

1. Google Cloud SDK Installation:

- **Problem:** The Google Cloud SDK installation fails or isn't found in your PATH.
- **Troubleshooting:**
    - Verify you downloaded the correct installer for your operating system (Windows, macOS, Linux).
    - Ensure you followed the installation instructions carefully, including adding gcloud to your PATH if necessary (refer to Google Cloud documentation for specific OS instructions).
    - Check your internet connection during download and installation.
    - Restart your terminal window after installation for changes to take effect.

2. Docker Installation:

- **Problem:** Docker installation fails or isn't accessible from your terminal.
- **Troubleshooting:**
    - Download the appropriate Docker Desktop version for your OS from the official website.
    - Follow Docker's installation instructions, including adding the Docker path to your environment variables (especially on Windows and macOS).
    - Restart your terminal after configuration changes.
    - Verify the installation by running docker --version in your terminal.

3. Project Setup and Permissions:

- **Problem:** Issues creating a project, enabling required APIs, or authentication problems.
- **Troubleshooting:**
  - Ensure you have a valid Google Cloud account and billing enabled for your project.
  - Double-check the API names (Cloud Run Admin API, Cloud Build API) when enabling them in the Google Cloud Console.
  - Verify your Google Cloud account has the necessary permissions to create projects and enable APIs.

4. Docker Image Build Issues:

- **Problem:** The docker build command fails or produces unexpected results.
- **Troubleshooting:**
  - Check for typos or errors in your Dockerfile syntax.
  - Ensure you have Docker installed and running.
  - Verify your Dockerfile has the correct instructions to install dependencies and copy your application code.
  - Consider using a multi-stage build for a smaller image size (advanced technique).

5. Cloud Run Deployment Issues:

- **Problem:** The gcloud builds submit or gcloud run deploy commands fail.
- **Troubleshooting:**
  - Make sure you've authenticated with gcloud auth login and set the project using gcloud config set project your-project-id.
  - Verify the image tag (gcr.io/your-project-id/your-app) matches your Dockerfile and repository name.
  - Check for errors in your internet connection or GCP service availability.
  - Ensure you have the required permissions to deploy to Cloud Run in your project.

6. Local Development vs. Cloud Deployment:

- **Problem:** The application behaves differently when deployed to Cloud Run compared to local development.
- **Troubleshooting:**
  - Double-check your code changes between local and deployed versions.
  - Ensure your local development environment (Python version, libraries) matches the one used in your Docker image.
  - Consider using environment variables to configure settings differently for local development and deployment.

    - Utilise logging statements or debuggers to inspect application behavior on Cloud Run.

7. Security Considerations:

- **Problem:** The deployment exposes your application publicly with --allow-unauthenticated flag.
- **Troubleshooting:**
    - This flag is for demonstration purposes only. In production, implement proper authentication and authorisation mechanisms to secure your application.
    - Consider using Cloud Identity and Access Management (IAM) to control access to your Cloud Run service.