Project Setup for IT2002 - Database Management Application Prototype

Mehdi Yaminli

February 2023

1 Introduction

The role of this small web relation management application is to be a standard where the student may extend the capabilities by integrating more relational operations. The view of the application is written in a compact view (front-end) library React, in TypeScript. Feel free to check the detailed comments on the code to get to know how the code operates line by line. The main part, the back-end operations are written in Python, using the library flask, and PostgreSQL is used as the database engine. The directory structure and program architecture are given below and to activate/set up the environment, follow the instructions below.

2 Virtual Machine Setup

2.1 Accessing SoC Virtual Machines

You will be provided virtual Ubuntu machines with certain applications pre-installed on them by the School of Computing IT Office. Alternatively, VM requests can be done via https://rt.comp.nus.edu.sg/, selecting the "Virtual Machine" option from the available elements.

Additionally, you will need to expose **port 80** of your virtual server in order to access the web pages as a client. For this purpose, you will have to submit a request to SoC IT services through this link and select "Application Systems" while explicitly denoting your request.

To access the virtual server while you are outside the SoC network, you will need **VPN software**. Refer to this link and set FortiClient up for your respective operating system. After successfully installing FortiClient and logging in using your respective credentials, you should be able to access the server using **SSH** while connected over VPN.

2.2 Connecting to SoC Virtual Machines via SSH

Secure Shell (SSH) usually comes on the installed OS. On Windows machines, you can directly run the command from the PowerShell console. Linux-based OSs and MacOS's newer versions should already have the ssh command available in their bash console. In case you cannot access the ssh command, please refer to the links below to install openssh-client:

- Windows - https://learn.microsoft.com/en-us/windows-server/administration/openssh/openssh_install_firstuse?tabs=gui
- Ubuntu - https://ubuntu.com/server/docs/service-openssh
- MacOS - https://formulae.brew.sh/formula/openssh

The connection command will be in the form `ssh sadm@<your-given-id>.comp.nus.edu.sg`. You will be prompted to the server shell after you typed the given/set password of your VM. Keep in mind sticking with the user 'sadm' to keep things simpler.

2.3 Working on the VM using VSCode Remote Server

Note that you're free to use any tool (such as remote displays, shell-only) to access the server and deploy your code there. The reason we're introducing VSCode and its remote tools is it may make remote development and debugging way easier for you.

VSCode provides a set of tools to write and debug code on remote servers. The extension "Remote - SSH" allows you to do it. After installing the extension (the link):

1. Access the command palette using "**Ctrl (Cmd) + Shift + P**" and type "**Remote-SSH: Connect to Host**"
2. Select "**+ Add new host**" and type "**sadm@<your-given-id>.comp.nus.edu.sg**". Press "Enter" to add it to your list of available hosts
3. To connect, select your machine, enter your "sadm" password, and continue
4. On the file explorer bar (usually on the left of the VSCode window), you will see the "Open Folder" button. Click on it to open your code folder or the root folder of your VM
5. OPTIONAL: You may refer to this link to learn more about remote development using Visual Studio Code

2.4 Git and Synchronizing Your Work with the VM

You may prefer to do the development on your local machines too, before deploying them to the server. For this purpose, you will have to add your SSH public key to GitHub SSH/GPG keys section. In order to generate SSH/GPG keys on your local machine, you can use the ssh-keygen command. If the command is not available in your OS by default, refer to these links to set it up:

- Windows - `https://www.purdue.edu/science/scienceit/ssh-keys-windows.html`

In MacOS and Ubuntu cases, ssh-keygen is installed under openssh-client by default.

To be able to interact with GitHub, copy your public ssh key (stored in '~/.ssh/id_rsa.pub') fully and add it to GitHub `https://github.com/settings/keys` by clicking the "New SSH Key" button. *Note that* the same thing will be done for the VM server.

In the VM server, generate the SSH keys running the same command (ssh-keygen) in the terminal. Go to the same place ('~/.ssh/id_rsa.pub'), fully copy the key, and add it to the same place under a different name (refer to the page for clarifications). Also, note that this tutorial assumes that you have basic knowledge of Git and GitHub.

2.5 Setting Up PostgreSQL and Accessing it via pgAdmin

We have already set up the DBMS server with the username "postgres" and password "postgres" on the VM. A database called "postgres" also has been created in it. If you want to create several databases, please access the postgres user's interface for PostgreSQL with the `sudo -u postgres psql` command. Then run the `CREATE DATABASE <your-database-name>;` command to proceed. For visual interactions with your DBMS, please refer to this link in order to download pgAdmin to your corresponding machine. After successful installations, go to `Object→Register→Server` from the navigation bar. You will be prompted a subwindow. In the `General` tab, give your preferred name for the DBMS, literally anything possible. Then navigate to the `Connection` tab, to enter the credentials. The default port on our virtual servers for PostgreSQL DMBS is set to 5432. Maintenance Database, Username, and Password fields are all postgres. After that, you will be able to manage your remote database. Note that you may face firewall or access-related errors when trying to access your remote DBMS. On your VM, run the command:

```
sudo ufw allow from <server-ip> to <your-machine-ip> port 3306 proto tcp comment "pgAdmin access"
```

Note that your machine's IP should be the IP given to it in the SoC network.

In case pgAdmin is heavy for your machine, or you cannot use it for any reason, you may proceed to go with adminer, an open-source, web-based and lightweight alternative for pgAdmin.

2.6 Installing Necessary Libraries on the VM

The virtual machine provided to you is supposed to have all the necessary libraries pre-installed. In case some of them miss, or you face any errors while executing, you may refer to the given commands:

- For the errors related to the 'psycopg2' library, used to access the PostgreSQL ecosystem through in Python, if you face incompatibility errors, please uninstall and reinstall it as:
  ```
  pip uninstall psycopg2
  pip install --no-binary :all: psycopg2
  ```

The main reason that we use a front-end library for the view is because of the dynamic operations done by the entry cells. It is possible to achieve the exact same thing using pure JavaScript, even using a single HTML file. But the complexities it would bring would make the application harder to extend, and understand.

The view code is commented clearly, which means, all the variables, functions, and necessary terms have explanations to guide you in case you may want to expand the code. Another note is that TypeScript has been used as the main language instead of JavaScript. That would make the debugging of the application logic way easier since the code is strongly typed (i.e. all the variables are assigned a type and functions are strict about their input types).

3 Application Development & Deployment

3.1 Structure of the Files in Directory

1. `app.py` - codebase of the backend API and database integration. SQL statements to be written here
2. `view/src/api.ts` - codebase communicating the view with the main API (namely functionalities of app.py)
3. `view/App.tsx` - main view file where the relation data gathered from the database is rendered
4. `view/components/<View>.tsx` - auxiliary view components used in `App.tsx` to render the relation and the editor application structurally (check the comments in the files for a detailed cover)

And here are the miscellaneous files you won't need to modify manually:

1. `main.tsx` and `index.css` - entrance point files not to be modified: all content of `App.tsx` and `App.scss` are compiled into them respectively
2. Files including `vite` in their names - config files for the fast build tool called Vite
3. `tsconfig.json` - compiler configuration for TypeScript
4. `package.json` - includes names and versions of the necessary view libraries used in the application
5. `.gitignore; index.html; node_modules` - keeps unnecessary files away from version control; main entry point for the browser; folder storing necessary installed JavaScript libraries

   In case you want to get familiar with the code, please refer to the repository and read the comments on top of the corresponding lines.

3.2 Fetching the Repository from GitHub

The code repository is stored in this link. Below are the instructions for you to pull it to your virtual machine. Keep in mind that you may have to clone/fork it to your own private repository. Run the command `git clone ???` in your `~/Projects` folder.

After the cloning is completed, you will have the `origin` as the given URL, and whatever you push/pull will be represented as your *fork*. In order to keep your codebase private, create a new private repository in GitHub, and make your teammates collaborators by adding their GitHub usernames to the repo too (refer to this link). To change the origin to your private repository, copy the SSH link to it (it should be the green button on the top-right corner of the repo), and use the command `git remote set-url origin <your-origin-here>`. This way, you can do basic git operations on your local and remote repositories. Please keep in mind that git and GitHub knowledge is not necessary for this project. This is a way to help you keep track of your code and collaborate.

3.3 Running Your Code

We'll begin the main source code - `app.py`. The version of Python used to write it was `Python 3.8`, the same as the default version installed on the server. In case you get library errors, run the `pip install -r requirements.txt` command. Running `app.py` only requires you to type the command `python3 app.py`. Keep in mind that you have to be in the same directory in the shell as the `app.py` file.

To run the view application, you have to navigate to the `view/` directory and run the command `npx vite`. Both of the commands will show you the IP address of the running instances so that you can access the view and the API from the browser and your favorite API test & development application respectively.

3.4 Debugging Your Code

You can use any preferable IDE to debug your code. Here, we will cover how to set up Visual Studio Code to do it. Keep in mind that you have to open 2 separate VSCode windows - one in the main directory, which contains `app.py`, and one in the `view/` directory. You can alternatively run the `npm run dev` command to run the application in the debug mode. The details about how the code functions are given in explanatory comments above each line.

To debug the Python code, you have to install the Python software development extension from the VSCode extensions store (here in the link). To debug the file, click the Debug button, or press the F5 key. You can place breakpoints (where your code will pause when the runtime reaches that line) inside the respective HTTP handlers to debug individual database interactions. To debug the view application, go to the `package.json` file, and click the Debug button on the "scripts" header. Again, keep in mind that your VSCode should be opened in the `view/` folder, not the main directory.

Note that you can take advantage of API development tools such as Postman (from here) or Insomnia (from here). This will help you to focus on the database development side rather than managing annoying CSS errors.

3.5 Deploying Your Code

We will avoid configuring background service files in `/etc/systemd/system/` or utilizing server packages such as nginx or apache since it is not our area of focus. But you are still free to use them in case you feel comfortable.

First, you have to make sure your ports 2222 and 4173 are not occupied by some other services. To do so, run `sudo kill -9 ‘sudo lsof -t -i:<your-port>‘` command. Replace `<your-port>` by 2222 and 4173, or any other custom port you've given.

Instead, we will go with something very primitive. After finishing the given tasks on the prototype application, you will use the `tmux` interface for background task management. Assuming you have

verified that your application runs well both locally on your machines and on the server in debug mode, you will need to create 2 separate tmux terminal interfaces - one for the view, and one for the Python application.

Run the `tmux new-session -s db-view` command to open a tmux terminal for the view (HTML/CSS interface). Please make sure you are in the `view/` directory. Run the `npm run build` command. It will compile your application in a few seconds (3-4 secs). Run `npm run preview -- --host` command, which will make your application available inside the SoC network too. You will see a prompt with your IP address and the terminal will halt there. Press `Ctrl+B`, and then `D` to detach from that terminal. Run the `tmux ls` command to make sure your session is on, where you will see "db-view" listed.

You will have to do the same for the flask application. Run `tmux new-session -s db-flask` to get another session for the flask app. When you are inside, you can simply run the `python3 app.py` to activate the application. Press `Ctrl+B`, and then `D` to detach from that terminal, and you should be able to see it after `tmux ls`.

Note that the above will run your code in debug mode. In case you want to make it fully production mode, you will have to do `pip install waitress`, add `export PATH="$ PATH: /.local/bin"` to your `~/.bashrc` file and run `waitress-serve --port=2222 --call app:create_app` instead of `python3 app.py`.

To access your tmux session, use `tmux attach -t <your-session> (such as db-flask)`. To terminate it, press `Ctrl+B`, and then `X` keys. To learn more about how to interact with tmux, refer to this link.