CS2102 Web App Development Guide

The objective of this guide is to let you start with your first database project using Windows. This guide uses NodeJS + PostgreSQL for back-end and Bootstrap for styling at the front-end. You are welcomed to use any other back-end/front-end pairs of your choosing.

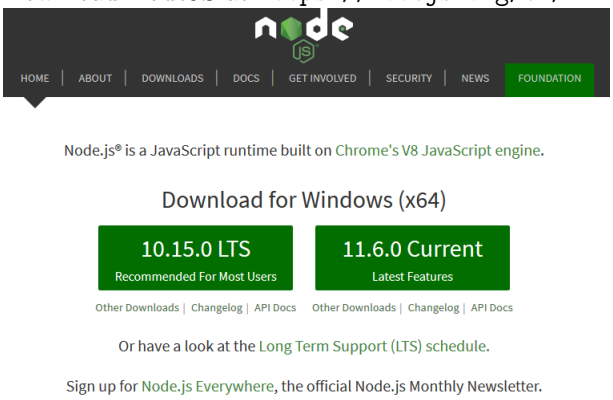
We recommend NodeJS because you can program in JavaScript. As such, you are guaranteed compatibility when sending data using JSON format should you wish to design an *interactive* web page instead of static web pages as shown in this guide. Furthermore, this reduces the number of programming language you have to learn in comparison to using PHP.

1 Installing NodeJS and Setting Up PostgreSQL Data

REQUIREMENTS: Installation of PostgreSQL.
OBJECTIVES: Installation of NodeJS and setting up data on PostgreSQL.

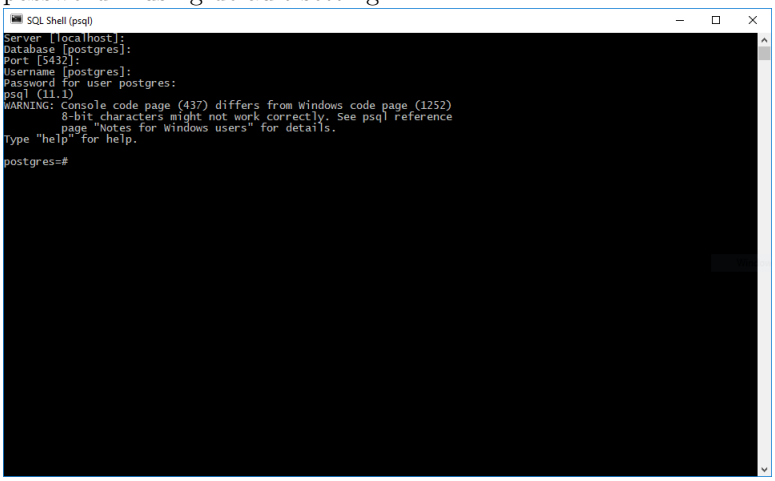
You can follow the instruction on file `install-postgresql.pdf` in **IVLE SQL Workbin** to install PostgreSQL. The guide file `guide_files.zip` can be downloaded from **IVLE Project Workbin**.

1. Download NodeJS at `https://nodejs.org/en/`.



2. Follow the installation guide. You may need to run the installer with administrator rights or log in as an administrator.

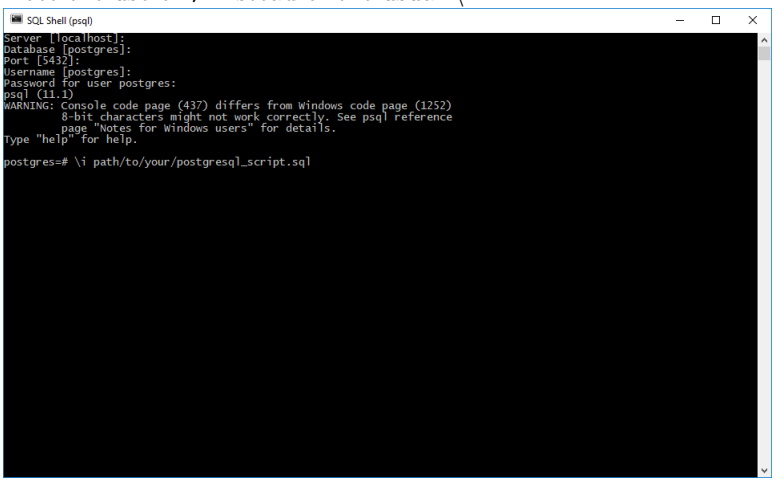
3. Open `SQL Shell (psql)`. Fill in the necessary details, or simply press `Enter` for all fields except password if using default setting.



4. Run the `postgresql_script.sql` using the command:

   `\i path/to/your/postgresql_script.sql`

   Note the use of `/` instead of the usual \.

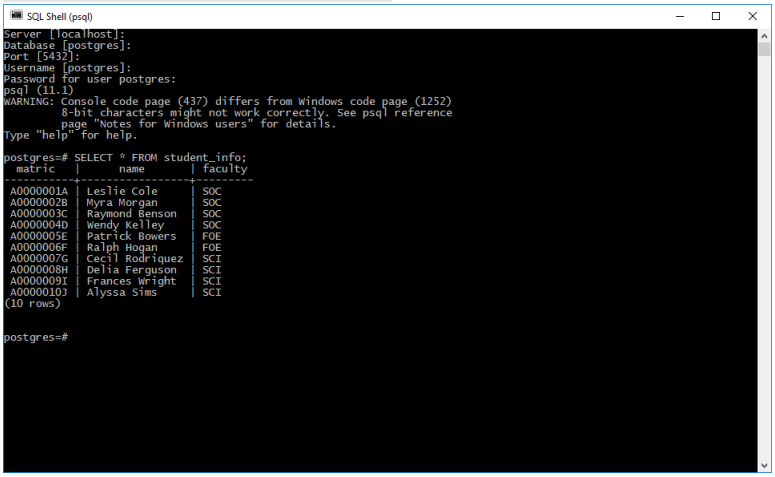


   Alternative: Copy paste the code in `postgresql_script.sql` into the `SQL Shell (psql)` console.

   This will set up and populate the database with dummy data.

5. You can check if the database is populated correctly by typing the command:

```
SELECT * FROM student_info;
```



You should see the values above.

2 Starting a Web Page

REQUIREMENTS: Installation of PostgreSQL, NodeJS, and setting up data on PostgreSQL.
OBJECTIVES: Running a simple web server with a welcome page.

`express-generator` is a tool to generate a skeleton files for a simple web app using `express` framework. It will generate the necessary files and folders to start your own web server.

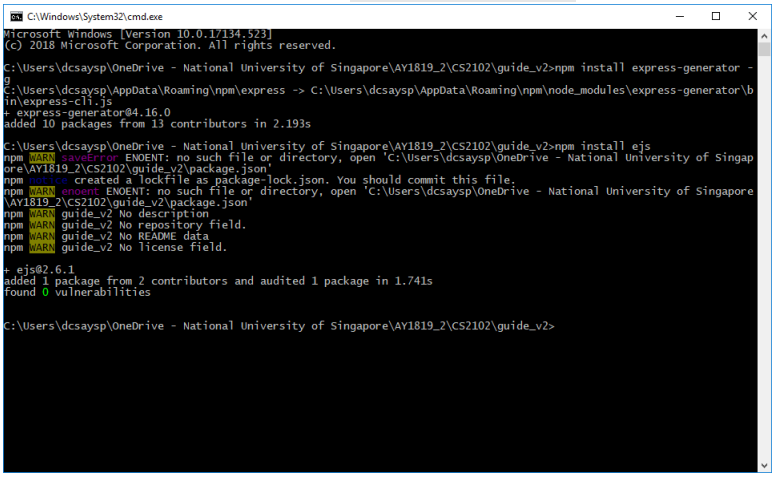
`ejs` is a templating engine that is close to HTML. While you can simply use HTML and rename the file to `.ejs` extension, it can also embed JavaScript language to generate a different `HTML` file depending on the value passed to the generator. To embed JavaScript code, you need to enclose the code in a tag `<% JS CODE %>` In this way, you can have a single template file to display different information. For instance, you may want to display different web page depending on the user that is logged in.

`express` uses the MVC architectural pattern. MVC stands for Model - View - Controller and they are tightly connected.

- The model is typically stored in database. It is used to update the view and it can be updated using the controller.
- The view handles what the user sees. It is updated by the model. In this guide, the view is handled by the templating engine `ejs`.
- The controller handles user interaction. This is the part of the architecture that the user uses to manipulate the model.

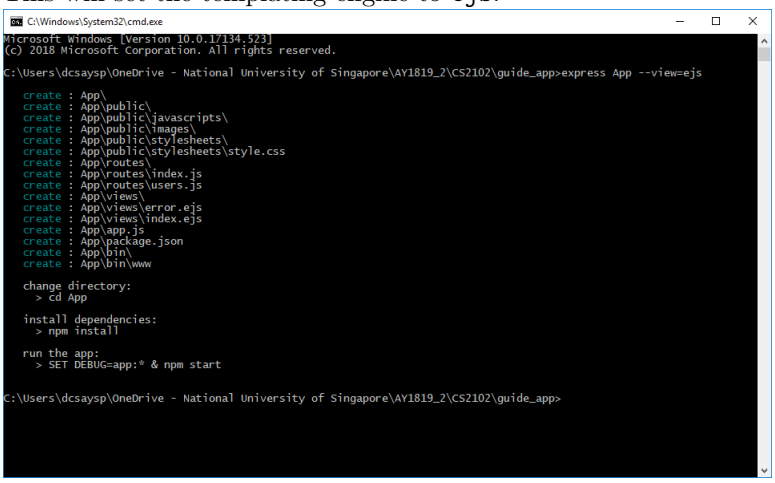
To read more about MVC architectural pattern, you are advised to read `https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller`

1. Install `express-generator` from NPM using: `npm install express-generator -g`
   Install `ejs` from NPM using: `npm install ejs`

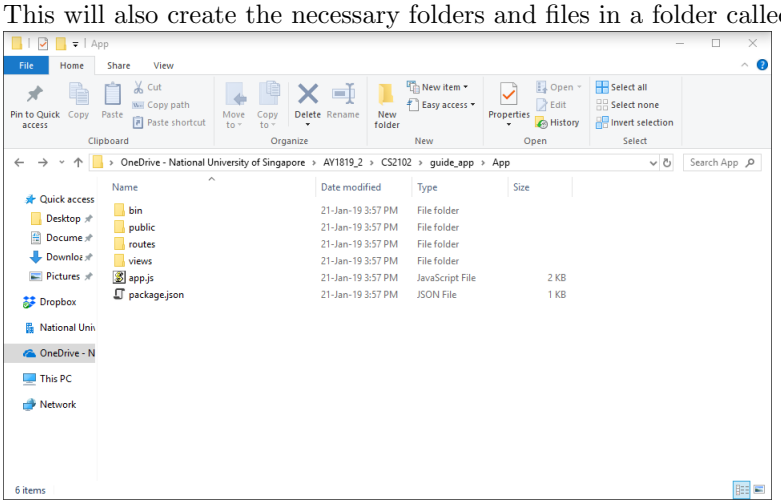


2. To create a web page (also called web app nowadays) called `App`, use the command: `express App --view=ejs`.

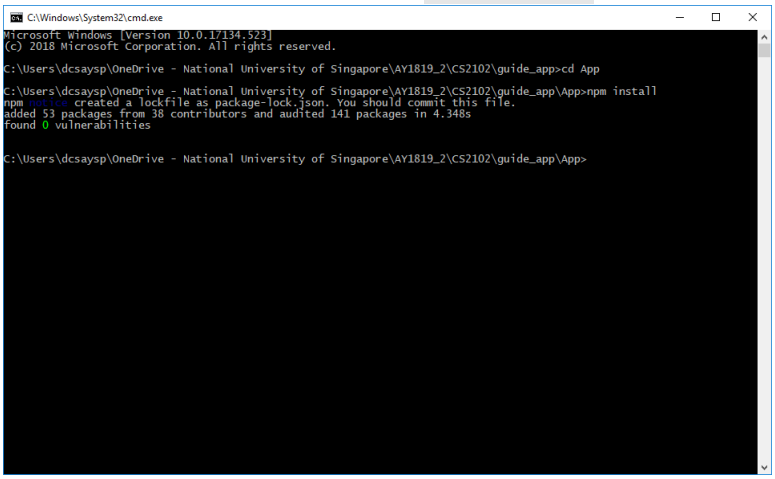
   This will set the templating engine to `ejs`.



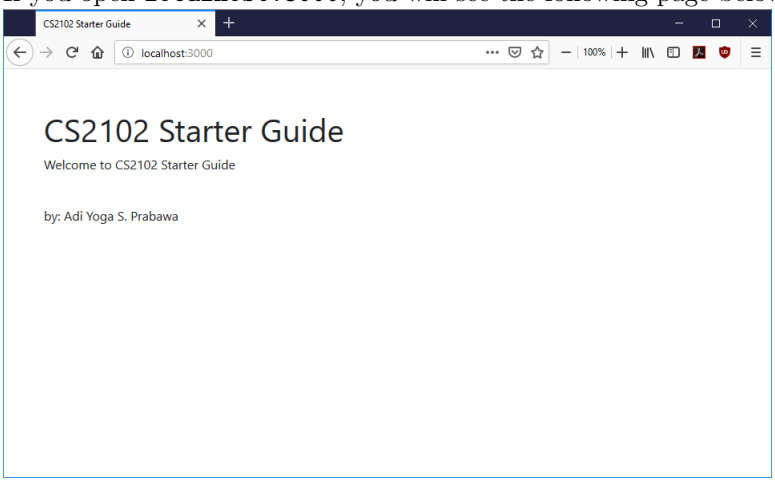
   This will also create the necessary folders and files in a folder called `App`.



3. Go to the newly created folder using: `cd App`
   Install the required packages using: `npm install` which will install all the missing packages.



4. Replace `App/views/index.ejs` with `guide_files/v1/index.ejs`.
   You can then run your server using: `node bin\www`.
   If you open `localhost:3000`, you will see the following page below.



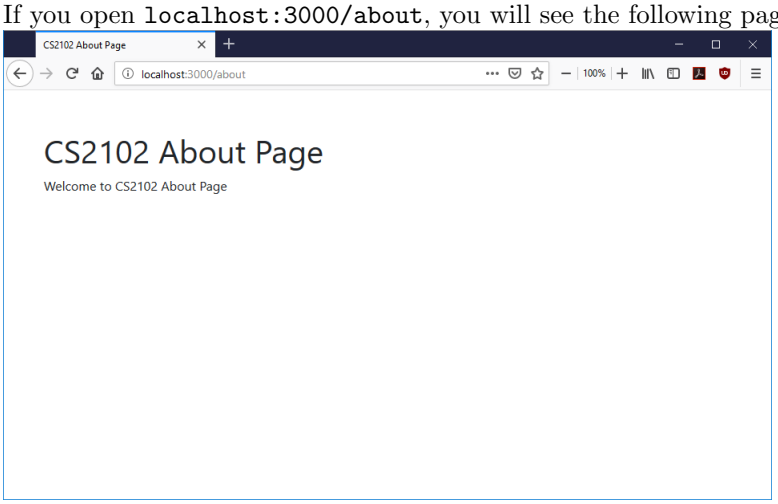
   NOTE: Our current `index.ejs` is written entirely in HTML. However, `ejs` provides functionalities that make templating easier. If you know HTML, you can simply use the HTML feature of `ejs`. If you wish to know more about `ejs`, please visit `https://www.ejs.co/#docs`. I will discuss some of the simple functionalities in later subsections.

5. If you want to stop the server, press `CTRL + C`.

3 Adding Web Pages

REQUIREMENTS: Running a simple web server with a welcome page.
OBJECTIVES: Adding more static web pages into our web server.

1. Replace `App/app.js` with `guide_files/v2/app.js`.
   Copy `guide_files/v2/about.js` to `App/routes` folder.
   Copy `guide_files/v2/about.ejs` to `App/views` folder.
   You can then run your server using: `node bin\www`.
   If you open `localhost:3000/about`, you will see the following page below.



2. **Explanation:** See the respective files for annotations.
   NOTE: In this example, `page` is the name of path whereas the file given is using the path `about`.

   - In `App/app.js`, you need to add the following lines:
     `var pageRouter = require('./routes/page');` before `var app = express();`
     and `app.use('/page', pageRouter)` before `module.exports = app;`
     This will instruct NodeJS to the appropriate file for routing. This will call the file `routes/page.js` below.
   - Create a file `page.js` in the folder `App/routes`. This is the file called by `app.use('/page', pageRouter)` above.
   - In the file `App/routes/page.js`, the function `router.get(...);` specifies the action when GET API[1] is used.
   - The function `router.get` takes two arguments:
     (a) The first specifies the `PATH`. In this case, we use `'/'` since this has been routed from `'/page'` path. In essence, this handles `'/page'` path.

[1]For more information about GET, POST, and other REST API, please refer to the link given in the Additional Information section at the end of the guide.

(b) The second specifies what to do when this page is accessed. This is given as a function. The command `res.render('page', {title: 'Page'});` will use the template `views/page.ejs`. You can pass the title and other arguments you wish besides title. For instance, `res.render('page', {title: 'Page', author: 'Adi'});` will pass both title and author to the template.

- Create `views/page.ejs`. This is the template to be rendered above. `ejs` can use standard HTML tags, but it also provides a way to customize your web page. In this example, we pass the value of title into the template using `res.render('page', {title: 'Page'});`. We can display the title by adding a snippet `<%= title%>`.

  This template is called by the function `res.render('page', params)`. The name of the file has to match, e.g., `page.ejs`.

3. If you want to stop the server, press `CTRL + C`.

4 Basic Template

REQUIREMENTS: Adding more static web pages into our web server.
OBJECTIVES: Getting familiar with the templating engine and Bootstrap.

1. Replace `App/app.js` with `guide_files/v3/app.js`.
   Copy `guide_files/v3/table.js` to `App/routes` folder.
   Copy `guide_files/v3/loops.js` to `App/routes` folder.
   Copy `guide_files/v3/table.ejs` to `App/views` folder.
   Copy `guide_files/v3/loops.ejs` to `App/views` folder.
   You can then run your server using: `node bin\www`.

2. Open `localhost:3000/table` to view code and some tutorials on table. Since most data in database can be represented as table, you can use them as the basis of your web page.

3. Open `localhost:3000/loops` to view code and some tutorials on tables generated with loops. The data that is passed is:

   ```
   data: [
     {matric: 'A0000001A', name: 'Leslie Cole'   , faculty: 'SOC'},
     {matric: 'A0000002B', name: 'Myra Morgan'   , faculty: 'SOC'},
     {matric: 'A0000003C', name: 'Raymond Benson', faculty: 'SOC'},
   ]
   ```

   NOTE: This data can be obtained from database. Which is the topic for the next section.

4. If you want to stop the server, press `CTRL + C`.

5 Connecting to Database

REQUIREMENTS: Getting familiar with the templating engine and Bootstrap.
OBJECTIVES: Getting data from database and display them.

1. Install `pg` module with the command: `npm install pg`

2. Replace `App/app.js` with `guide_files/v4/app.js`.
   Copy `guide_files/v4/select.js` to `App/routes` folder.
   Copy `guide_files/v4/select.ejs` to `App/views` folder.
   You can then run your server using: `node bin\www`.
   Open `localhost:3000/select` to view the result.

3. **Explanation:**

   - To connect to the database, you need to either create a `Client` or a `Pool`. Since we will be connecting many times, we use a `Pool`.
   - The code to connect to your database is as follows:

     ```
     const { Pool } = require('pg')
     const pool = new Pool({
       user: 'postgres',
       host: 'localhost',
       database: 'postgres',
       password: '********',
       port: 5432,
     })
     ```

     NOTE: You may need to change certain parameters depending on your configuration. In particular, you need to change the `password`.
   - The `SQL query` used in this example is basic `SELECT * FROM student_info`. You may need to provide a more complex query for your need.
   - Using the `SQL query` and the `Pool`, you can send the query to SQL using:
     `pool.query(sql_query, (err, data) =>  /* callback */ );`
     Callback is the operation that you are going to perform once you retrieve the data from the SQL database.
   - In our example, we will only render the result. This can be done without modifying `loops.ejs` from previous subsection (but we still need to copy to a new file).
     To pass the result into the template `select.ejs`, we need to pass it as `data:  data.rows` since the result is stored in `data.rows`.

4. If you want to stop the server, press `CTRL + C`.

6 Forms and Interaction

REQUIREMENTS: Getting familiar with the templating engine and Bootstrap.
OBJECTIVES: Adding forms into your web page.

1. Replace `App/app.js` with `guide_files/v5/app.js`.
   Copy `guide_files/v5/forms.js` to `App/routes` folder.
   Copy `guide_files/v5/forms.ejs` to `App/views` folder.
   Copy `guide_files/v5/formsScript.ejs` to `App/public/javascripts` folder.
   You can then run your server using: `node bin\www`.
   Open `localhost:3000/forms` to view the result.

2. **Explanation:**

   - To use the file `formsScript.js` in `App/public/javascripts` folder, add:
     `<script src="javascripts/formsScript.js"></script>` to the HTML head in `view/forms.ejs`.
     This contains our interactive code written in JavaScript.
   - The static files to be served to the user are by default located in `App/public` folder. To change the location of the static files, you can change:
     `app.use(express.static(path.join(__dirname, 'public')));`
     to another location such as:
     `app.use(express.static(path.join(__dirname, 'path/to/static/folder')));`.
     This line can be found in `App/app.js` file.
   - To retrieve the input, we need a way to identify the input. This can be done by specifying the `id` of the input:
     `<input class="form-control" id="matric">`
     The `id` of the input above is `matric`. You can check the `id` of the other inputs by yourself.
   - Once you identified the input `id`, you can retrieve the value by:
     `var value = document.getElementById('id').value;`
     where `id` is the `id` of the input. For instance, to retrieve the input with `id="matric"` above, you can use:
     `var value = document.getElementById('matric').value;`.

     If you are more familiar with jQuery, you can use `$('#matric').val()` instead. You can find more information about jQuery in the Additional Information section at the end of the guide.
   - Our current interactivity is limited to displaying the inputs in `alert`.

3. If you want to stop the server, press `CTRL + C`.

7 Modifying Database

REQUIREMENTS: Getting data from database and display them AND Adding forms into your web page.
OBJECTIVES: Modifying the database by inserting new rows.

1. Install `body-parser` with the command:
   `npm install body-parser --save`

2. Replace `App/app.js` with `guide_files/v6/app.js`.
   Copy `guide_files/v6/insert.js` to `App/routes` folder.
   Copy `guide_files/v6/insert.ejs` to `App/views` folder.
   Copy `guide_files/v6/insertScript.ejs` to `App/public/javascripts` folder.
   You can then run your server using: `node bin\www`.
   Open `localhost:3000/insert` to view the result.

3. **Explanation:**

   - We need to add more interactivity in our page. First, we setup an event listener to prevent the page from doing anything if the forms are not filled to standard. This is done by the script at the bottom of `insert.ejs`.
     The script basically attach the function `check` whenever the submit button is pressed.
     - The check is done by checking the length since we know that matric number should have a length of 9, faculty typically has a length of 3, and name cannot be empty.
     - To stop the page from submitting the form, add: `event.preventDefault();` and `event.stopPropagation();`.
   - In the routing, we need to handle `POST` request, which is done by having `router.post`.
     - To retrieve the parameters of the `POST` request, use `req.body.param_name`.
     - After we are done with SQL query, we can redirect instead of displaying the original web page. To redirect, use: `res.redirect(/other_page)`
   - In the forms, we need to add the action to be performed.

     - The web page to be visited when submit button is pressed is encoded in `action="page"`.
     - The method encodes the protocol to be used. In our case, we use `method="post"`. For other methods, you can read up on `https://www.w3schools.com/tags/ref_httpmethods.asp`.
   - Also in the forms, each input should have a `name` such as `name="param_name"`.
     - This corresponds to the `req.body.param_name` above.
   - You can try filling in the form with your information. You will see the `/select` web page will be updated with your information. In particular, your information will be in the database.

4. If you want to stop the server, press `CTRL + C`.

8 Using `dotenv`

REQUIREMENTS: Modifying the database by inserting new rows.
OBJECTIVES: Using `dotenv` to avoid leaking password in source code.

1. Install `dotenv` with the command:
   `npm install dotenv --save`

2. Replace `App/app.js` with `guide_files/v7/app.js`.
   Copy `guide_files/v7/select.js` to `App/routes` folder.
   Copy `guide_files/v7/.env` to `App` folder.
   You can then run your server using: `node bin\www`.
   Open `localhost:3000/select` to view the result.

3. **Explanation:**

   - The `dotenv` package allows us to load the string stored in `.env` file.
     This is the connection string that contains all the necessary information to connect to your database.
     The connection string has the following format:
     `DATABASE_URL=postgres://username:password@host_address:port/database_name`
     Modify the values as necessary according to your PostgreSQL configuration.
   - To avoid passwords being leaked when using versioning software such as Git, SVN, or Mercurial, you need to include the `.env` file in the ignore list.
   - Once the connection string is loaded using the command: `process.env.DATABASE_URL` you can create a connection `Pool` using:
     `const pool = new Pool({connectionString:process.env.DATABASE_URL})`
     instead of explicitly stating the username and password.

4. If you want to stop the server, press `CTRL + C`.

9 Additional Information

The following links are provided for your reading if you wish to know more about the tools used in this guide.

- JQuery
  - `https://api.jquery.com/`
  - `https://www.w3schools.com/jquery/default.asp`
- Bootstrap
  - `https://getbootstrap.com/docs/4.2/getting-started/introduction/`
  - `https://www.tutorialspoint.com/bootstrap/`
  - `https://www.w3schools.com/bootstrap/`
- EJS
  - `https://www.ejs.co/#docs`
  - `https://www.npmjs.com/package/ejs`
  - `https://ionicabizau.github.io/ejs-playground/` to test your template
- Express
  - `https://expressjs.com/en/starter/installing.html`
  - `https://expressjs.com/en/starter/basic-routing.html` on routing
  - `https://expressjs.com/en/starter/static-files.html` on serving static files
  - `https://expressjs.com/en/resources/frameworks.html` on other frameworks based on `express`
- Miscellaneous
  - JSON:
    `https://www.tutorialspoint.com/json/`
    `https://www.w3schools.com/js/js_json_intro.asp`
  - XMLHttpRequest (AJAX):
    `https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest`
    `https://www.w3schools.com/xml/xml_http.asp`
    `https://api.jquery.com/jQuery.ajax/`
  - MVC:
    `https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller`
  - REST API:
    `https://restfulapi.net/`

- SQL Injection
  - Prevention using Bind Parameter:
    `http://docs.sequelizejs.com/manual/tutorial/raw-queries.html#bind-parameter`
  - TL;DR:
    SQL Query = `SELECT * FROM student_info WHERE matric=$1`
    Code = `pool.query(sql_query, ['A0000001A'], (err, data) => /* omitted */)`
    `$1` will be replaced with `A0000001A`.
- Session and Login
  - Passport.js:
    `http://www.passportjs.org/`
  - Aqua:
    `https://jedireza.github.io/aqua/`
  - node-login:
    `https://github.com/braitsch/node-login`