

ASSIGNMENT 2 FRONT SHEET

Qualification	BTEC Level 5 HND Diploma in Computing		
Unit number and title	Unit 16: Cloud Computing		
Submission date	31/10/2022	Date Received 1st submission	
Re-submission Date		Date Received 2nd submission	
Student Name	Huynh Minh Huy	Student ID	GCD210173
Class	GCD1001	Assessor name	Tran Trong Minh
Student declaration <p>I certify that the assignment submission is entirely my own work and I fully understand the consequences of plagiarism. I understand that making a false declaration is a form of malpractice.</p>			
		Student's signature	<i>Huy</i>

Grading grid

P5 ✓	P6 ✓	P7 ✓	P8 ✓	M3	M4	D2	D3

⚙ **Summative Feedback:**

⚙ **Resubmission Feedback:**

Grade:

Assessor Signature:

Date:

Lecturer Signature:

Acknowledgement

First and foremost, I want to thank the curators at the University of Greenwich for making these courses available to students. I'd want to thank all of the writers and academics that have investigated Cloud Computing. Furthermore, I'd like to thank speaker Tran Trong Minh for his excellent and professional lectures and tutorials. Finally, I'd want to thank our university's professors and administrative personnel.

Table of Contents

CHAPTER 1: INTRODUCTION	9
CHAPTER 2: CONFIGURE A CLOUD COMPUTING PLATFORM WITH A CLOUD SERVICE PROVIDER'S FRAMEWORK (P5).	9
I. Tools	9
1. Integrated Development Environment (IDE).	9
2. Framework.	11
3. Database.	11
II. Configure.....	12
1. Configure Github.	12
2. Configure Heroku.....	15
3. Connect Heroku To Github.	18
4. Configure PostgreSQL On Heroku.	19
CHAPTER 3: IMPLEMENT A CLOUD PLATFORM USING OPENSOURCE TOOLS (P6).	23
I. Setup Pgadmin And Create Database For ATN.	23
II. Diagram.	27
III. Source code for ATN Shop.	28
1. Models.	29
2. Controller (Route).	38
3. View.....	43
IV. Commit your code to Github.....	47
V. Deploy app on Heroku.....	49
VI. Link of the project.....	54

CHAPTER 4: ANALYSING THE MOST COMMON PROBLEMS WHICH ARISE IN A CLOUD COMPUTING PLATFORM AND DISCUSS APPROPRIATE SOLUTIONS TO THESE PROBLEMS (P7).

.....	55
I. Optimizing Cloud Expenses.....	55
II. Migrating Existing Applications onto the Cloud.....	55
III. Cost Calculations Within Limits.	56
IV. Proprietary Lock – in.....	56
V. Software as a Service (SaaS) Security Concerns.....	56
VI. Platform as a Service (PaaS) Security ConcernS.....	57
VII. Infrastructure as A Service (IaaS) Security Issues.	58
VIII. Data Security Violation.	59
IX. Consistent Results.....	59

CHAPTER 5: ASSESS THE MOST COMMON SECURITY ISSUES IN CLOUD ENVIRONMENTS (P8).

.....	60
I. Misconfiguration.....	60
II. Data Privacy.....	60
III. Data Problem.	61
IV. Insecure Interfaces/APIs.....	61
V. Hijacking of Accounts.....	61
VI. Notifying Customers Affected by Data Breaches.	62
VII. A Lack of Visibility/Control.....	62
VIII. Unauthorized Access.....	62
References.....	63

Table of Figures

Figure 1: Visual Studio Code.....	9
Figure 2: ExpressJS Framework.....	11
Figure 3: PostgreSQL.....	12
Figure 4: GitHub homepage.....	12
Figure 5: Github Login Page.	13
Figure 6: Profile.....	13
Figure 7: Create new repository.....	14
Figure 8: Heroku sign up page.....	15
Figure 9: Login Page in Heorku.....	16
Figure 10: Heroku Page.....	16
Figure 11: Create new app.....	17
Figure 12: Create app.....	17
Figure 13: Finished create new app.....	18
Figure 14: Connect Heroku to Github.....	18
Figure 15: Connect success to Github.....	19
Figure 16: Connect to the repository.....	19
Figure 17: Select app.....	19
Figure 18: Select the suggested Heroku Postgres add-on.....	20
Figure 19: Add postgresQL.....	20
Figure 20: Select a database pricing plan.....	21
Figure 21: Create PostgreSQL Success.....	21
Figure 22: Setting.....	22
Figure 23: Database Credentials.....	23
Figure 24: Install Postgres/pgAdmin.....	23
Figure 25: Database information.....	24
Figure 26: Create table.....	26
Figure 27: Test database on Heroku.....	26
Figure 28: Use case diagram.....	27

Figure 29: Model.....	28
Figure 30: View.....	28
Figure 31: Controller.....	28
Figure 32: Source Code For Connect Database Server.....	29
Figure 33: Source Code Authentication.	30
Figure 34: Source Code For Product Table 1.	30
Figure 35: Source Code For Product Table 2.	31
Figure 36: Source Code For Product Table 3.	32
Figure 37: Source Code For Product Table 4.	33
Figure 38: Source Code For Product Table 5.	34
Figure 39: Source Code For Select Box.....	35
Figure 40: Source Code For CRUD Function 1.....	36
Figure 41: Source Code For CRUD Function 2.....	36
Figure 42: Source Code For CRUD Function 3.....	37
Figure 43: Source Code For CRUD Function 4.....	37
Figure 44: Admin.	38
Figure 45: Source Code For Admin 1.	38
Figure 46: Source Code For Admin 2.	39
Figure 47: User.	39
Figure 48: Source Code For User 1.	40
Figure 49: Source Code For User 2.	40
Figure 50: Index.	41
Figure 51: Source Code For Index 1.	42
Figure 52: Source Code For Index 2.	42
Figure 53: Source Code For Index 3.	42
Figure 54: Some Source Code For Admin Interface.	43
Figure 55: JavaScript for Admin Page.....	43
Figure 56: Some Source Code For User Interface.....	44
Figure 57: Some Source Code For Login Interface.....	44

Figure 58: Some Source Code For Index Interface.	45
Figure 59: CSS for landing page in index interface.	45
Figure 60: CSS for login page.	46
Figure 61: CSS for user and admin interface.	46
Figure 62: git init.	47
Figure 63: git add.	47
Figure 64: git commit.	47
Figure 65: git remote.	47
Figure 66: git push.	47
Figure 67: Push GitHub success.	48
Figure 68: Index page.	49
Figure 69: Login page.	49
Figure 70: Login as user.	50
Figure 71: Login as admin.	50
Figure 72: Wrong username and password.	51
Figure 73: The sub menu.	51
Figure 74: Log Out.	53
Figure 75: Log Out Success.	54

CHAPTER 1: INTRODUCTION.

I explored cloud computing and service provider models in report one. The research and selection of remedies to ATN's difficulties was also discussed. As a result, in this report, I will discuss the process of designing a cloud application for ATN in order to address the issues raised in report one.

CHAPTER 2: CONFIGURE A CLOUD COMPUTING PLATFORM WITH A CLOUD SERVICE PROVIDER'S FRAMEWORK (P5).

I. Tools.

1. Integrated Development Environment (IDE).

Visual Studio Code is a free editor that helps programmers write code, debug it, and fix it using the intelli-sense technique. In layman's terms, it makes it easier for people to generate code. Many people believe it's half an IDE and half an editor, but that's entirely up to the developers.



Figure 1: Visual Studio Code.

I'm going to utilize Visual Studio Code because:

- **Cross-Platform Support:** Editors have traditionally supported either Windows, Linux, or Mac platforms. Visual Studio Code, on the other hand, is cross-platform. As a result, it is compatible with all three platforms. Furthermore, the code operates on all three platforms; previously, the open-source and commercial software codes were separate.
- **Multiple programming languages are supported:** Several programming languages are available. Previously, programmers need Web-Support: a separate editor for each language; however, multi-language support is now integrated in. This also means that it can quickly identify any errors or cross-language references.
- **Extensions & Support:** Generally, all programming languages are supported; however, if a user/programmer chooses to use a programming language that is not supported, he can download and use an extension. Furthermore, because the addon operates as a distinct process, it does not slow down the editor.
- **Terminal Support:** When a user has to start from the directory's root to complete a job, the built-in terminal or console lets the user to avoid moving between two screens.
- **Web Application Support:** Web application support is included in. VSC can therefore create and maintain web applications.
- **Git Support:** Online resources from Git Hub Repo may be obtained, and vice versa; saving is also allowed. Cloning code that is freely available on the internet is known as resource pulling. This code can be changed and saved at a later time.

2. Framework.

For this project, I intend to use the Express framework. Express is a Node.js web application framework with a robust feature set for both online and mobile projects. With a wealth of HTTP utility methods and middleware development tools at your disposal, creating a powerful API is quick and simple. Express provides a solid foundation for basic web application capabilities without concealing the Node.js features you know and love.



Figure 2: ExpressJS Framework.

3. Database.

For this project, I intend to use PostgreSQL instead of MongoDB and here is the reasons why I choose PostgreSQL:

- Multi-Version Concurrency Control (MVCC), point in time recovery, tablespaces, asynchronous replication, nested transactions, online/hot backups, a strong query planner/optimizer, and write ahead logging for fault tolerance are among the advanced features of PostgreSQL.
- PostgreSQL is compatible with virtually all Linux and Unix variations, as well as Windows and Mac OS X. Because it is open-source, it is simple to upgrade and improve. In PostgreSQL, you may construct your own data types, custom functions, and even write code in another programming language (such as Python) without having to recompile the database. PostgreSQL is, of course, free on Heroku.

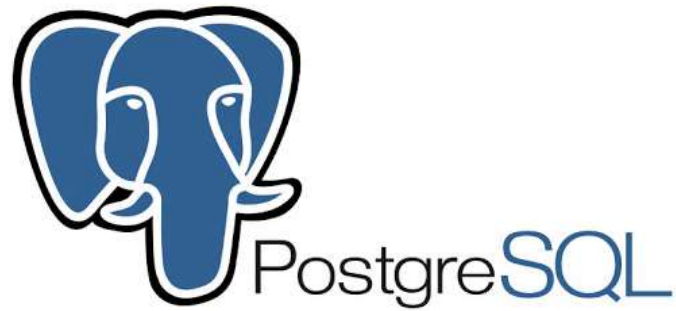


Figure 3: PostgreSQL.

II. Configure.

1. Configure Github.

I'll start by visiting the GitHub homepage.

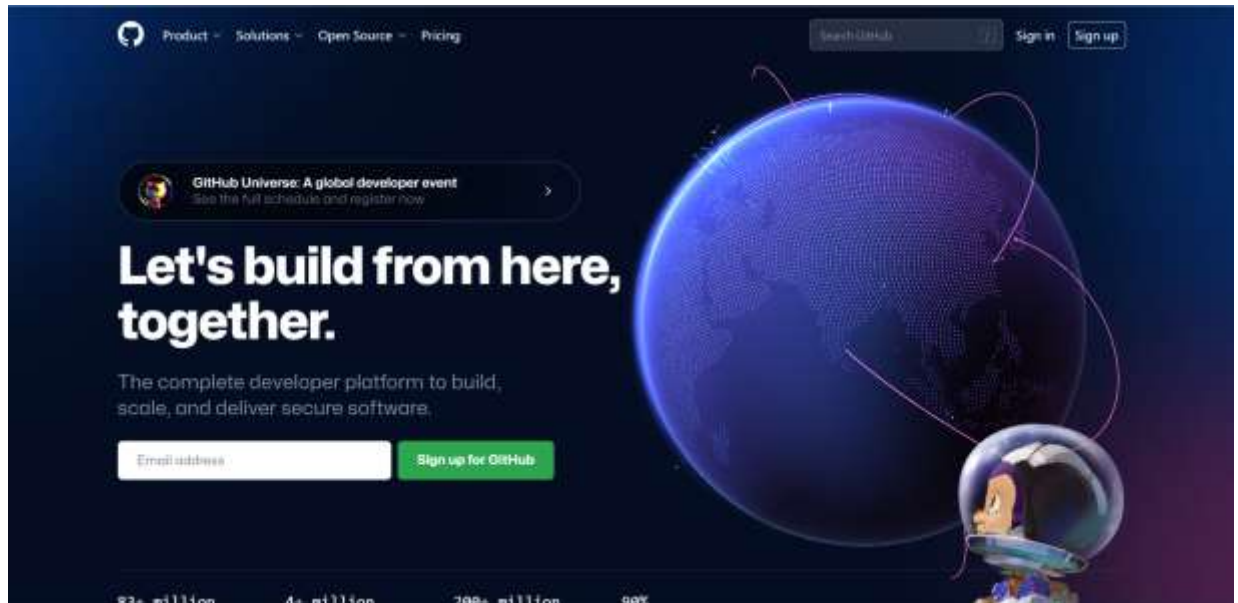


Figure 4: GitHub homepage.

I'll then click the login box to gain access to my GitHub account. I'll go to my profile after I've signed onto the GitHub site.

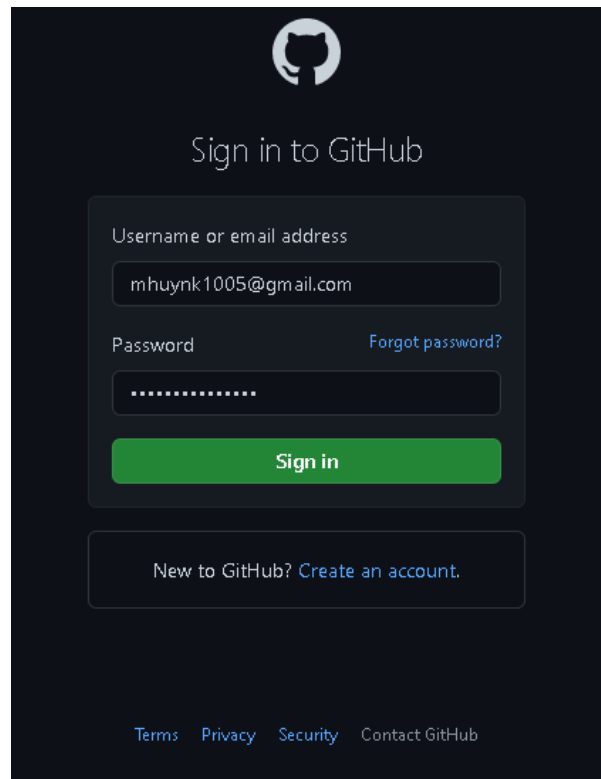


Figure 5: Github Login Page.

When I finished login into the Github site, I will go to my profile.

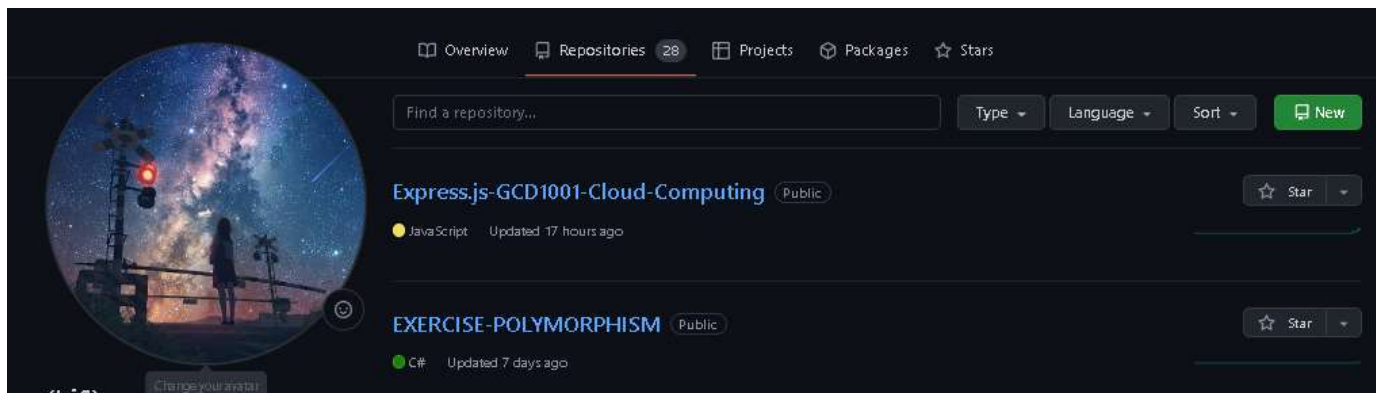


Figure 6: Profile.

To create a new repo, click the "New" button in the right corner of the profile page.

Figure 7: Create new repository.

The system will take me to the repo creation page, where I will provide the necessary repo information. Then, on the following screen, click the "Create Repository" button.

```
Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH https://github.com/1lr-is/Cloud-Computing-Project-Assignment.git

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line

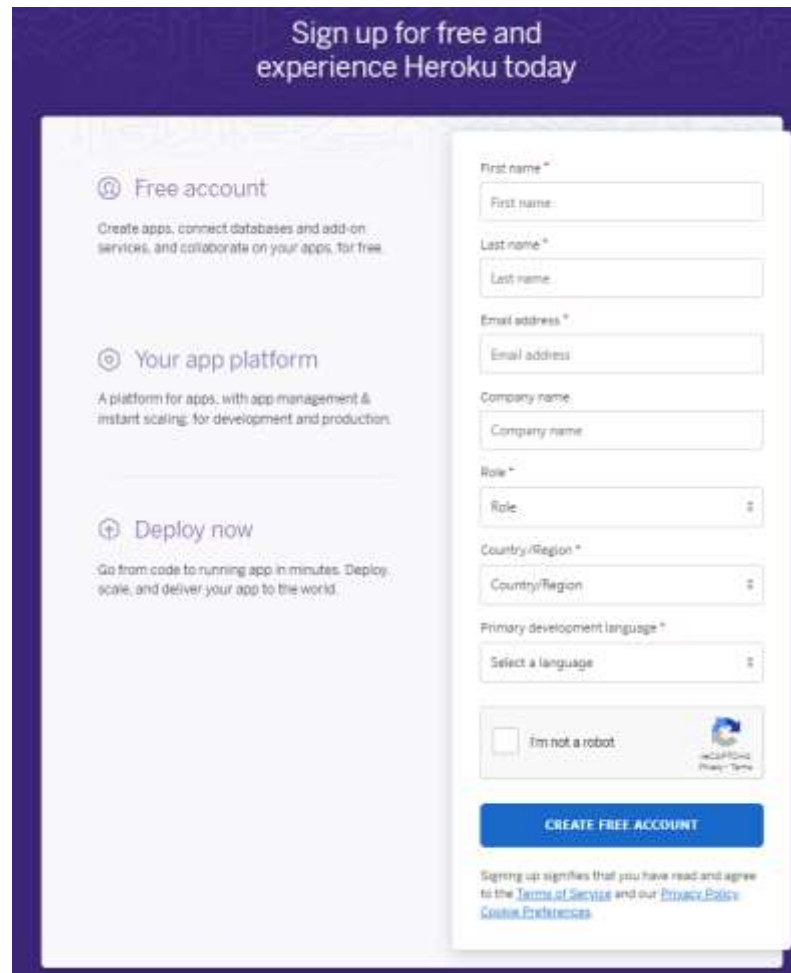
echo "# Cloud-Computing-Project-Assignment" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/1lr-is/Cloud-Computing-Project-Assignment.git
git push -u origin main

...or push an existing repository from the command line

git remote add origin https://github.com/1lr-is/Cloud-Computing-Project-Assignment.git
git branch -M main
git push -u origin main
```

2. Configure Heroku.

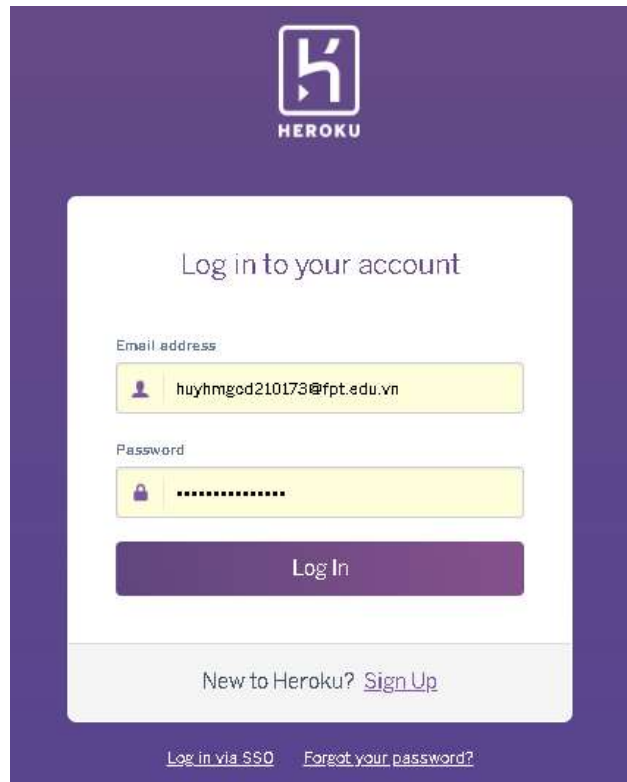
To begin, go to Heroku's site and create an account.



The image shows the Heroku sign-up page. The header reads "Sign up for free and experience Heroku today". The page is divided into two main sections. On the left, there are three bullet points: "Free account" (Create apps, connect databases and add-on services, and collaborate on your apps, for free), "Your app platform" (A platform for apps, with app management & instant scaling, for development and production), and "Deploy now" (Go from code to running app in minutes. Deploy, scale, and deliver your app to the world). On the right, there is a form with the following fields: "First name *" (text input), "Last name *" (text input), "Email address *" (text input), "Company name" (text input), "Role *" (dropdown menu), "Country/Region *" (dropdown menu), and "Primary development language *" (dropdown menu). Below these fields is a checkbox labeled "I'm not a robot" and a reCAPTCHA widget. At the bottom of the form is a blue button labeled "CREATE FREE ACCOUNT". Below the button, there is a small disclaimer: "Signing up signifies that you have read and agree to the [Terms of Service](#) and our [Privacy Policy](#) [Cookie Preferences](#)".

Figure 8: Heroku sign up page.

Proceed to log in to the system if you already have an account on Heroku.



The image shows the Heroku login page. At the top, there is the Heroku logo (a stylized 'H' in a square) and the word 'HEROKU'. Below this, the text 'Log in to your account' is centered. There are two input fields: 'Email address' with the value 'huyhmgod210173@fpt.edu.vn' and 'Password' with masked characters. A 'Log In' button is below the password field. At the bottom, there is a link 'New to Heroku? Sign Up' and two links: 'Log in via SSO' and 'Forgot your password?'.

Figure 9: Login Page in Heorku.

After successfully login in, the interface will look like this.

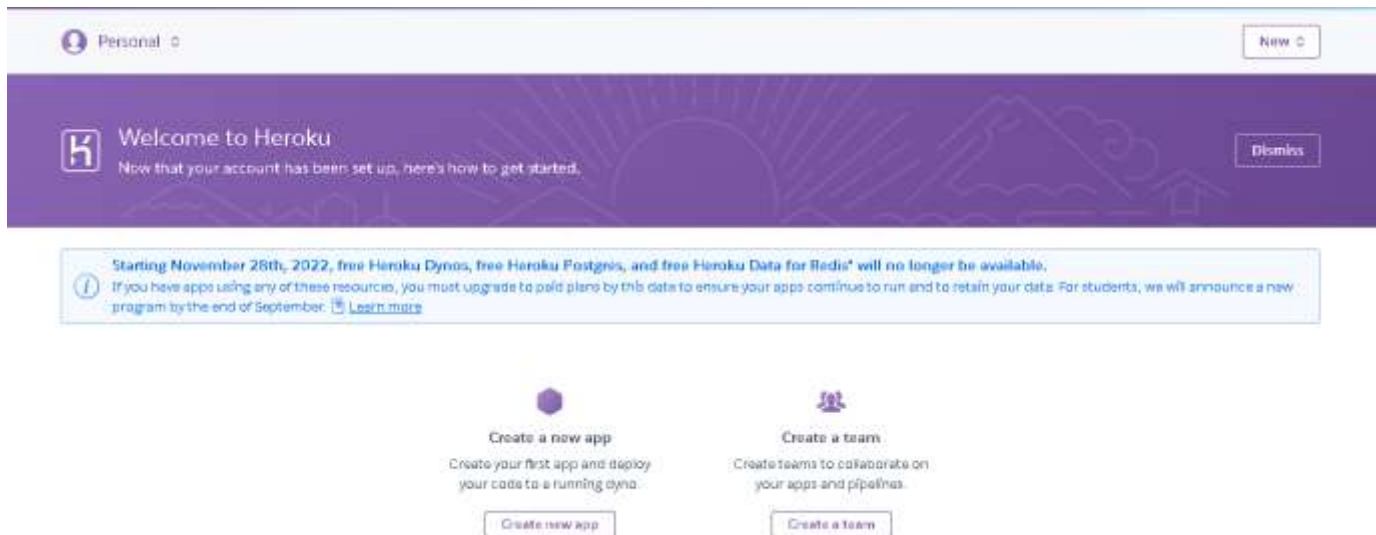


Figure 10: Heroku Page.

Select “New” to create a new cloud app.

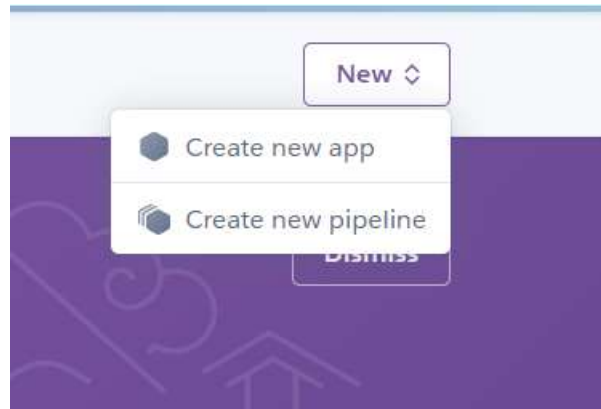


Figure 11: Create new app.



Create the app and give it a name.

App name

atntoystore-app 

atntoystore-app is available

Choose a region

 United States 

Add to pipeline...

Create app

Figure 12: Create app.

After successfully building the app, the screen will look like this.

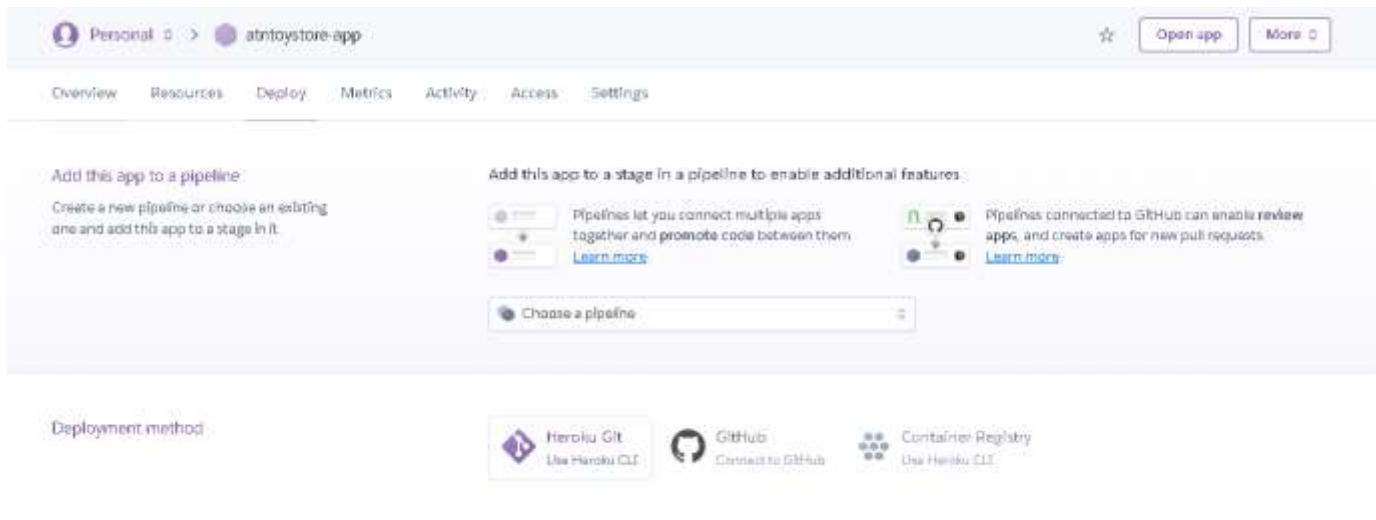


Figure 13: Finished create new app.

3. Connect Heroku To Github.

Select Github to locate the repository connected with your Github account, and then select Connect to connect heroku to Github.

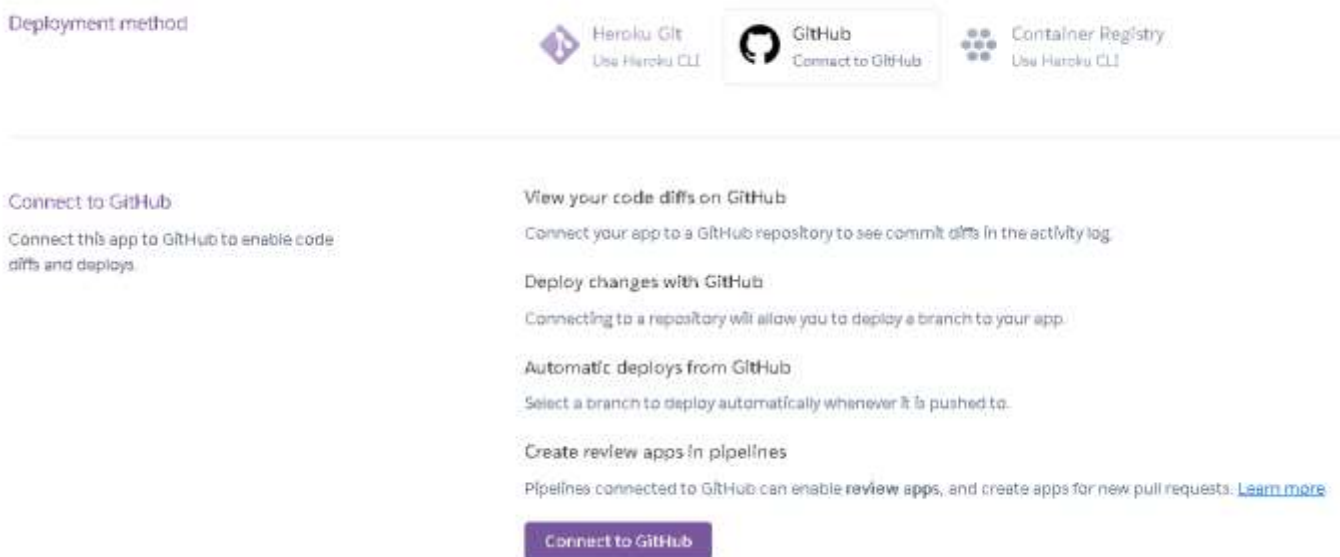


Figure 14: Connect Heroku to Github.

When you connect success Heroku with Github, choose the repositories and connect.

Deployment method



Heroku Git
Use Heroku CLI



GitHub
Connect to GitHub



Container Registry
Use Heroku CLI

Connect to GitHub

Connect this app to GitHub to enable code
diffs and deploys.

Search for a repository to connect to

Missing a GitHub organization? [Ensure Heroku Dashboard has team access](#)

Figure 15: Connect success to Github.

After connecting success with repo in GitHub.



Heroku Git
Use Heroku CLI



GitHub
Connected



Container Registry
Use Heroku CLI

Connected to [11r-is/Cloud-Computing-Project-Assignment](#) by [11r-is](#)

Disconnect...

Releases in the [activity feed](#) link to GitHub to view commit diffs

Figure 16: Connect to the repository.

4. Configure PostgreSQL On Heroku.

Select an app to configure PostgreSQL on Heroku.

Starting November 28th, 2022, free Heroku Dynos, free Heroku Postgres, and free Heroku Data for Redis[®] will no longer be available. If you have apps using any of these resources, you must upgrade to paid plans by this date to ensure your app continues to run and to retain your data. For students, we will announce a new program by the end of September. [Learn more](#)

Filter apps and pipelines



atintoystore-app

heroku-22 • United States



firstexpress-gcd

Node.js • heroku-22 • United States



Figure 17: Select app.

When the option displays, select the suggested Heroku Postgres add-on.

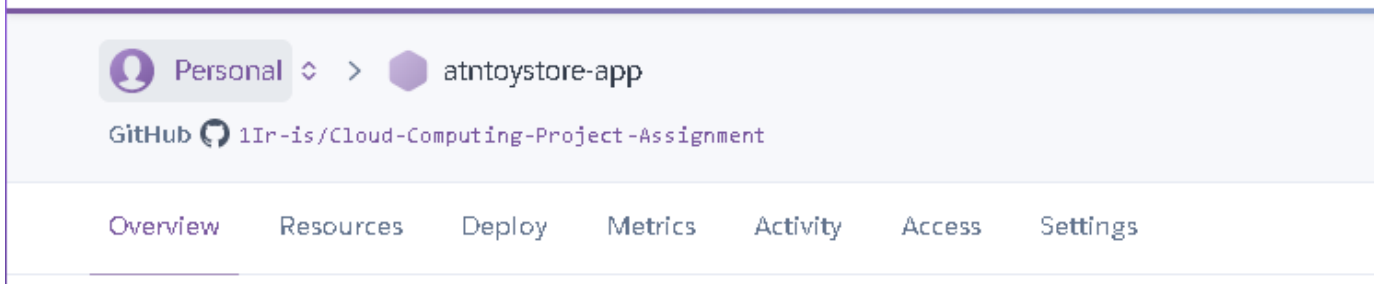


Figure 18: Select the suggested Heroku Postgres add-on.

To connect a PostgreSQL database to your freshly developed app's dashboard, navigate to the Resources tab in the dashboard's header. Then type Heroku Postgres into the Add-ons search box.

Add-ons

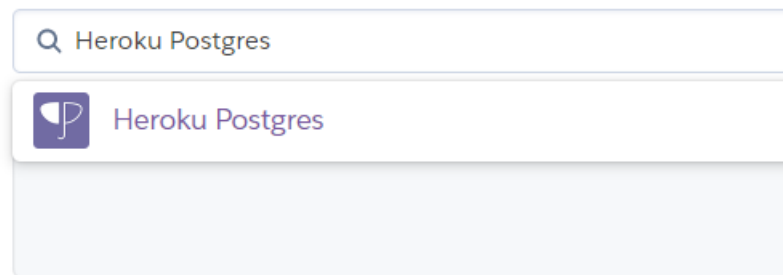


Figure 19: Add postgresQL.

You are prompted to select a database pricing plan in the following popup window. Click Provision: after choosing the Hobby Dev - Free plan.

Figure 20: Select a database pricing plan.

Create PostgreSQL Success

Figure 21: Create PostgreSQL Success.

To see the login details and connection URL for the PostgreSQL database, head to the Resources page in your app's dashboard and choose the Heroku Postgres resource:



To enter the settings page, click the "Settings" tab in the header.



Figure 22: Setting.

To view your PostgreSQL database's credentials, click the View Credentials button.

ADMINISTRATION

Database Credentials

Get credentials for manual connections to this database.

Cancel

Please note that these credentials are not permanent.

Heroku rotates credentials periodically and updates applications where this database is attached.

Host	ec2-3-220-207-90.compute-1.amazonaws.com
Database	d3ph1st5druklm
User	bsrcfbtdzclgbk
Port	5432
Password	aa9b5f69ce006df64266a1c43ce2880a129429adae7db54b010982db76ef1761
URI	postgres://bsrcfbtdzclgbk:aa9b5f69ce006df64266a1c43ce2880a129429adae7db54b010982db76ef1761@ec2-3-220-207-90.compute-1.amazonaws.com:5432/
Heroku CLI	heroku pg:psql postgresql-clear-86343 --app atntoystone-app

Figure 23: Database Credentials.

CHAPTER 3: IMPLEMENT A CLOUD PLATFORM USING OPENSOURCE TOOLS (P6).

I. Setup Pgadmin And Create Database For ATN.

We must first install Postgres/pgAdmin locally.

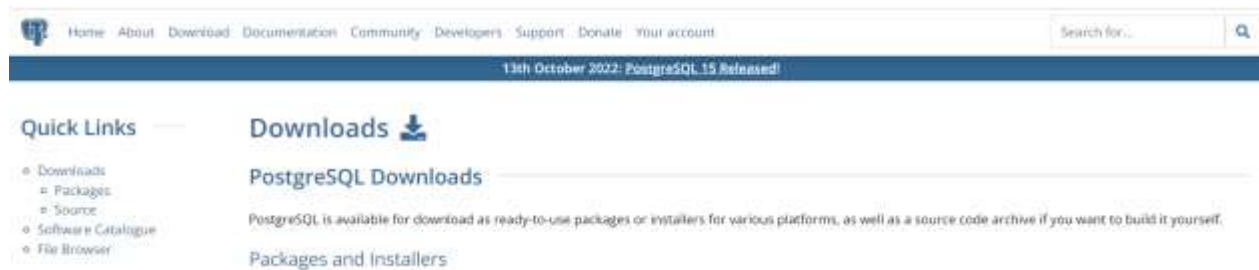


Figure 24: Install Postgres/pgAdmin.

- As stated on the downloads page, pgAdmin should be included in the installation (among various other clients). Check this during installation and make sure these choices are selected.
- Use one of the built-in clients to rapidly determine whether Postgres is present. — version.
- Add a new server to link Heroku SQL to PgAdmin4.



Select Connection and fill out the form SQL Heroku:

- Database information for an app on Heroku.

Host	ec2-3-220-207-90.compute-1.amazonaws.com
Database	d3phixt5drukim
User	bsrcibtdzclgbk
Port	5432
Password	aa9b5f69ce006df64266a1c43ce2880a129429adae7db54b010982db76ef1781
URI	postgres://bsrcibtdzclgbk:aa9b5f69ce006df64266a1c43ce2880a129429adae7db54b010982db76ef1781@ec2-3-220-207-90.compute-1.amazonaws.com:5432/?sslmode=require
Heroku CLI	heroku pgsql:postgres:clear-86343 --app atmtoystore-app

Figure 25: Database information.

- Enter details and save to establish a Heroku server with database information.

Register - Server

General

Connection

SSL

SSH Tunnel

Advanced

Host name/address

ec2-3-220-207-90.compute-1.amazonaws.com

Port

5432

Maintenance database

d3ph1st5druk1m

Username

bsrcibtdzclgbk

Kerberos authentication?

☐

Password

Save password?

☒

Role

Service

Close

Reset

Save

Register - Server

General

Connection

SSL

SSH Tunnel

Advanced

Host address

DB restriction

d3ph1st5druk1m x |

Password file

Connection timeout (seconds)

10

Close

Reset

Save

- When I finished create server and database, I will going to create database table for the project.

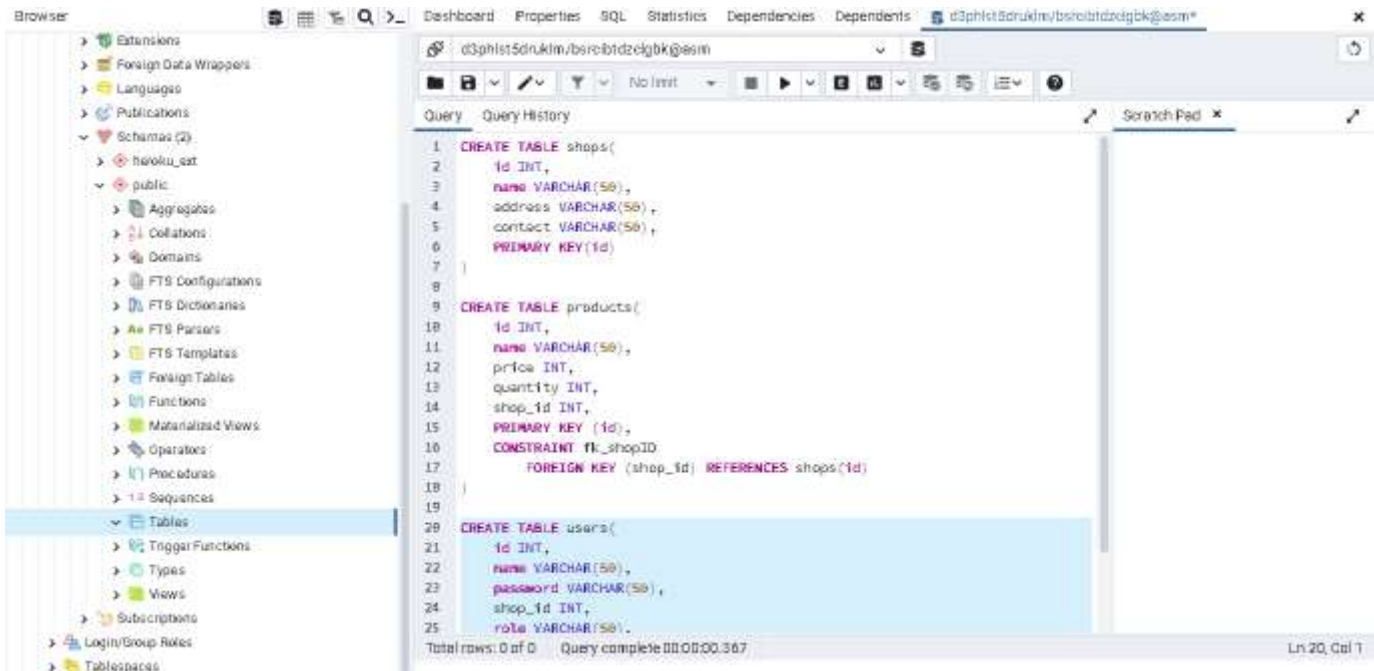


Figure 26: Create table.

- Last but not least, I will going to test database on Heroku.

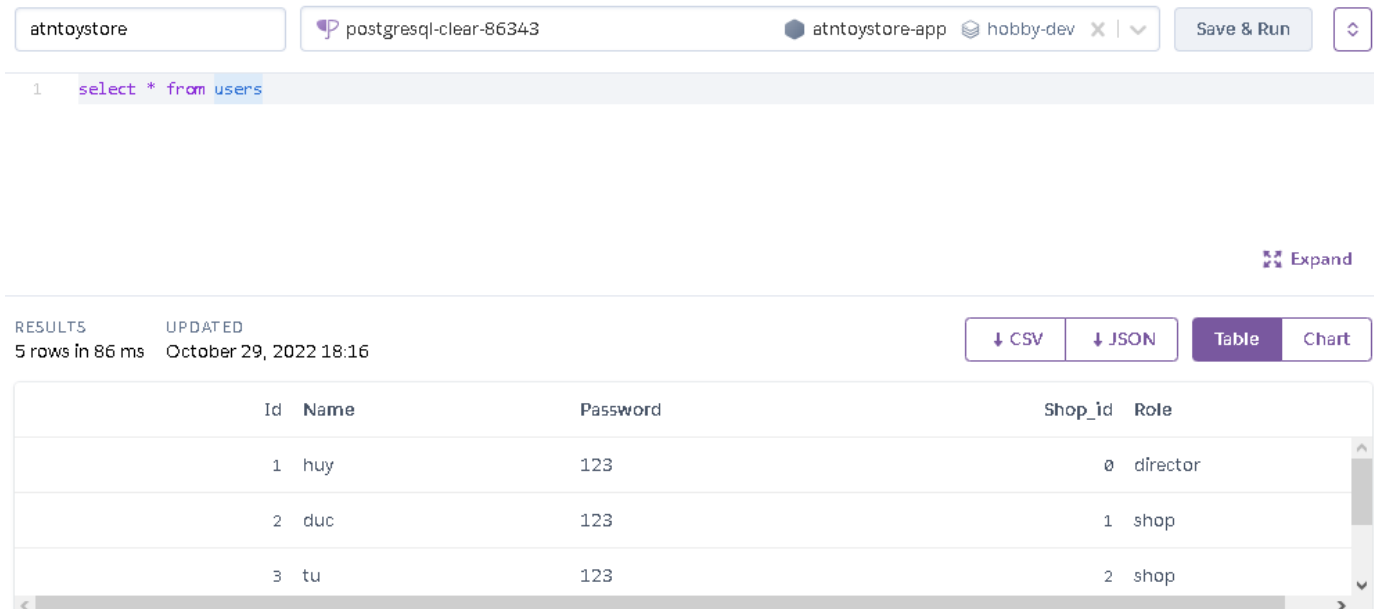


Figure 27: Test database on Heroku.

II. Diagram.

Use case diagram of ATN shop development

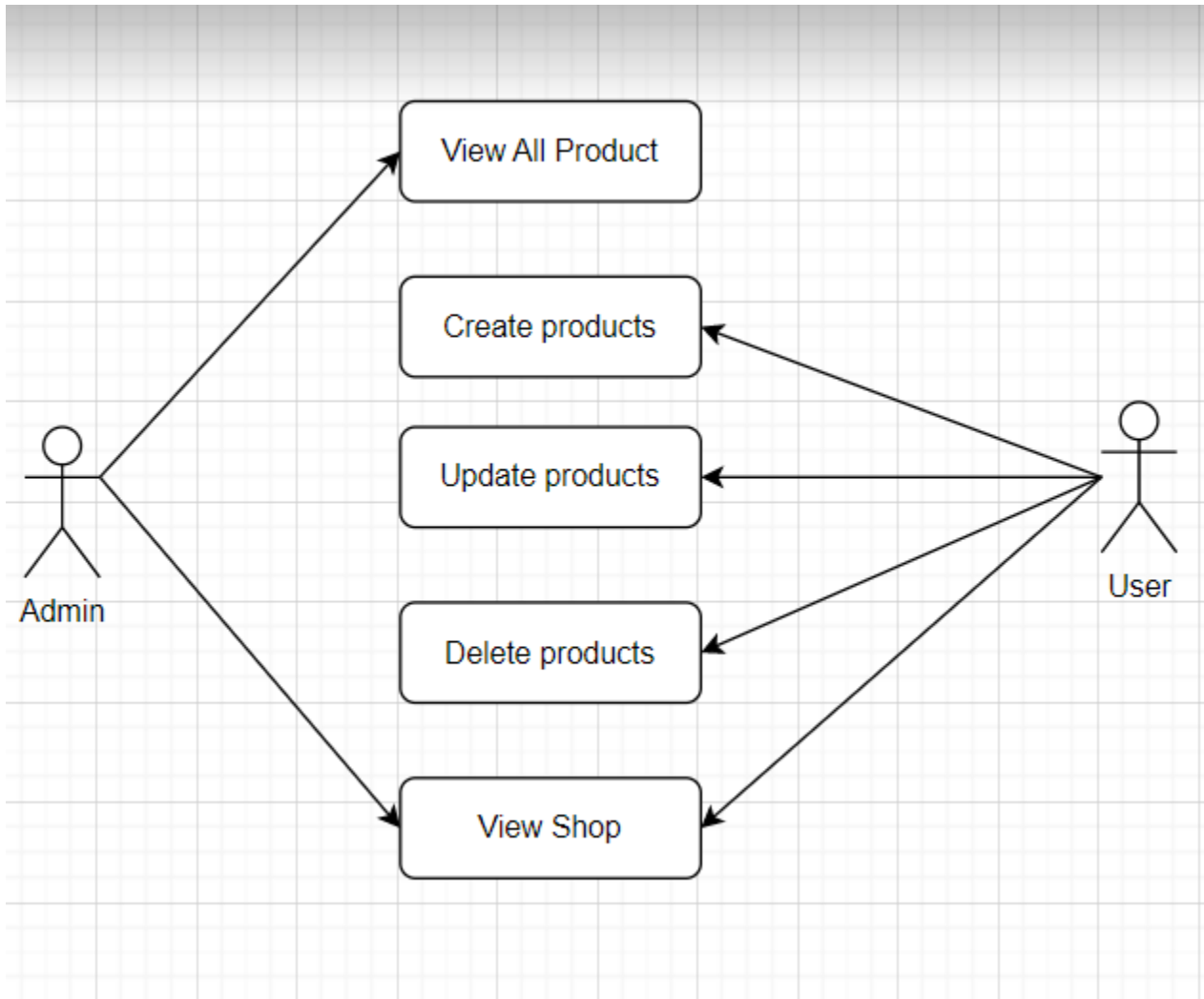


Figure 28: Use case diagram.

We should use the database name exactly as it appears in the Heroku postgres info option.

III. Source code for ATN Shop.

This project following MVC model so that I can handle source code easier.

- **Model (M):** This part is responsible for all functions of the project.

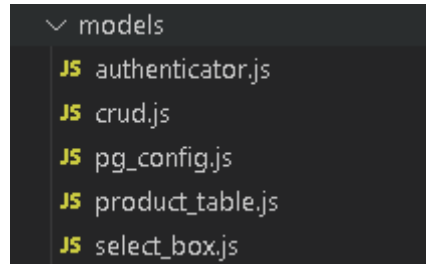


Figure 29: Model.

- **View (V):** This part is used for display view for the project.

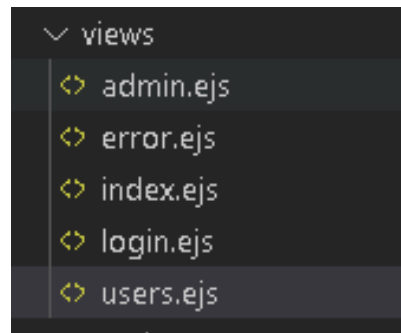


Figure 30: View.

- **Controller (C):** This part has folder “routes”, which implemented all functions of the project.



Figure 31: Controller.

1. Models.

a. Database Connection.

This 'pg_config.js' class is in charge of connecting the project to Heroku's PostgreSQL database; other classes can call it and run queries against it.

```
1  const Pool = require('pg').Pool;
2  const pg_conn = new Pool (
3    {
4      user: 'bsrcibtdzclgbk',
5      password: 'aa9b5f69ce006df64266a1c43ce2880a129429adae7db54b010982db76ef1781',
6      host: 'ec2-3-220-207-90.compute-1.amazonaws.com',
7      database: 'd3phlst5druk1m',
8      port: 5432,
9      ssl: {
10        rejectUnauthorized: false
11      },
12    });
13
14  module.exports = pg_conn;
```

Figure 32: Source Code For Connect Database Server.

b. Authentication.

This is the logic of the 'Authenticate' method in the class users.js and admin.js, which manages which user and admin logs in. If the number of rows returned by the query is equal to 1, and if so it sets the variables authenticated, shop_id and role to true, shop_id and role respectively, the function will render the user.ejs and admin.js class by shifting the data to that class for display. To force the user and admin to login again, it will return the message "Incorrect username or password".

```

1  var pg_conn = require("../pg_config");
2
3  async function authen(user, pass)
4  {
5      let authenticated = false;
6      let shop_id;
7      let role;
8      const auth_query =
9      {
10         text: 'SELECT * FROM users WHERE name = $1 AND password = $2',
11         values: [user, pass]
12     };
13     var query_data = await pg_conn.query(auth_query);
14     if (query_data.rowCount==1)
15     {
16         authenticated = true;
17         shop_id = query_data.rows[0].shop_id;
18         role = query_data.rows[0].role;
19     }
20     return [authenticated, shop_id, role];
21 }
22 module.exports = authen;

```

Figure 33: Source Code Authentication.

c. Product Table.

```

1  var pg_conn = require('../pg_config');
2  // Define a function to display products table for a shop
3  async function display_products(shop_id,session){
4      // Query DB to get the table data
5      console.log(session.shop_id);
6      var products_query;
7      if (shop_id==0){
8          products_query = 'SELECT * FROM products ORDER BY id ASC ';
9      }
10     else{
11         products_query = {
12             text: 'SELECT * FROM products WHERE shop_id=$1 ORDER BY id ASC',
13             values: [shop_id]
14         }
15     }
16     const data = await pg_conn.query(products_query);

```

Figure 34: Source Code For Product Table 1.

This is a function that displays the products table for a shop. It takes two parameters, the shop_id and session. The first line of code defines a variable called products_query which will be used to query the database. The second

line of code checks if the shop_id is equal to 0 or not. If it is equal to 0 then it will select all rows from the products table ordered by id in ascending order, otherwise it will select all rows from the products table where shop_id equals \$1 (the value passed as an argument) ordered by id in ascending order. The third line of code executes this query using pg-promise and stores its result in data variable.

```
let table_string = `
<style>
  table {
    font-family: arial, sans-serif;
    border-collapse: collapse;
    width: 100%;
  }
  .insert-button {
    background-color: #4dff00;
  }
  .edit-button {
    padding-left: 4px;
    background-color: #e8ff00;
  }
  .delete-button {
    background-color: #ff0032;
  }
  td, th {
    border: 1px solid #dddddd;
    text-align: left;
    padding: 8px;
  }
  tr:nth-child(even) {
    background-color: #dddddd;
  }
  .heading {
    font-weight: bold;
    text-align: center;
    font-size: 40px;
    margin-bottom: 20px;
    margin-top: 20px;
    color: blue;
  }
  .table {
    margin-left: 50px;
    margin-right: 50px;
  }
</style>
</head>
<body>
<h2 class="heading">Table products</h2>
<table class="table">
<tr>`;
```

Figure 35: Source Code For Product Table 2.

This is a string that contains the HTML code for the table. It will be displayed in the browser.

```
//display all headers of table
let num_fields = data.fields.length;
for (let i=0; i<num_fields; i++) {
    //if shop owner then don't need shop_ID
    if(session.shop_id!=0 && data.fields[i].name != 'shop_id' )
    {
        table_string += `<th>${data.fields[i].name}</th>`;
    }
    if(session.shop_id==0){
        table_string += `<th>${data.fields[i].name}</th>`;
    }
}

if(session.shop_id !=0){
    table_string += `<th>actions</th>`
}

table_string += `</tr>`;
```

Figure 36: Source Code For Product Table 3.

This is a code for display all headers of table, in here I have declared `num_fields = data.fields.length` to holds the number of fields in the table. The for loop that iterates through the `data.fields` array and displays all headers of table. In addition, there is a conditional statement in the for loop that checks if the `shop_id` of the session is not equal to 0 and if the name of the field in `data.fields[i]` is not equal to 'shop_id'. If both conditions are true, then it will add a table header with the name of that field. If either condition is false, then it will check if `session.shop_id` equals 0 and if so, it will add a table header with the name of that field. Last but not least, there is a conditional statement that checks if the `shop_id` of the user is not equal to 0. If it is not equal to zero, then it adds a new column called “actions” to the table.


```
//display all rows of table
let num_rows = data.rowCount;
console.log("Num rows: " + num_rows)
for (let i=0; i<num_rows; i++) {
    table_string += `<form action="/users/crud" method="post">
    <tr>`;
    for (let j=0; j<num_fields; j++) {
        let field_name = data.fields[j].name
        let cell = data.rows[i][field_name];
        if(session.shop_id!=0 && field_name != 'shop_id' )
        {
            table_string += `<td><input type="text" name= ${field_name} value=${cell}></td>`;
        }
        if(session.shop_id==0){
            table_string += `<td>${cell}</td>`;
        }
    }
    //since only shop owners can input new data then the buttons should only on their side
    if(session.shop_id!=0){
        table_string +=
        `<td>
        <button type="submit" name='crud'
            class="bg-sky-900 hover:bg-sky-500 text-white font-bold py-2 px-4 rounded" value='update'>Update</button>
        <button type="submit" name='crud'
            class="bg-violet-900 hover:bg-violet-500 text-white font-bold py-2 px-4 rounded"
            value='delete'>Delete</button>
        </td>
        </tr></form>`
    }
}
```

Figure 37: Source Code For Product Table 4.

This code is a function that displays all rows of the table, in here I have declared `num_rows = data.rowCount` to stores the number of rows in the table. The first for loop that loops through the number of rows in the table. It adds a form tag to each row, with an action and method attribute. The action attribute specifies where to send the form-data when a form is submitted. The method attribute specifies how to send form-data (GET or POST). The second for loop that iterates over the number of fields in the table. It then assigns the name of each field to a variable called `field_name` and it also assigns the value of each cell to another variable called `cell`. Then, it checks if `session.shop_id` is not equal to 0 and if `field_name` is not equal to `shop_id`. If both conditions are true, then it adds an input tag with type text, name = `${field_name}` and value = `${cell}`. If only one condition is true or none of them are true, then it adds a td tag with the value of cell inside it. Additionally, it also have if statement to

check that if that is user table will have button and if that is admin it will not have button.

```
//since only shop owner can add product, director does not need it
if(session.shop_id!=0){
  const temp = await pg_conn.query('SELECT * FROM products ORDER BY id DESC');
  let tempName = temp.fields[0].name;
  last_id = temp.rows[0][tempName];
  console.log(last_id);
  // Add an empty row and insert button at the end of row
  table_string += `<tr><form action="/users/crud" method="post">`
  for (let j=0; j<num_fields; j++) {
    let field_name = data.fields[j].name

    // console.log(field_name);
    if(field_name!='shop_id'){
      //automatic id
      if(field_name=='id'){
        table_string += `<td><input type="text" name= ${field_name} value = ${last_id+1}></td>`;
      }
      else{
        table_string += `<td><input type="text" name= ${field_name} ></td>`;
      }
    }
  }
  table_string +=
  `<td>
  <button type="submit"
  class="bg-rose-900 hover:bg-rose-500 text-white font-bold py-2 px-4 rounded"
  name='crud'
  value='insert'>Insert</button>
  </td>`
  table_string += `</tr></form>`;
}
```

Figure 38: Source Code For Product Table 5.

This coded is a function to insert new product into the table for users. If the user is login into the system it will have the insert place to insert new product at the end of the row.

d. Select Box.

```

let pg_conn = require('./pg_config');
async function gen_box(selectValue=-1){
  //Query DB to get the table data
  let shops_query = `SELECT shops.id, shops.name, users.role FROM shops
  .....JOIN users ON shops.id = users.shop_id`;
  const data = await pg_conn.query(shops_query);
  let box_string = `
  ....<form action="/admin/select_box" method="post">
  .....<label for="shop">Choose a shop:</label>
  .....<select name="shop" id="shop">
  .....<option value=0 ${selectValue== -1? `selected` : ''}>All shops</option>;
  let select_items = data.rowCount;
  for (let i = 0; i < select_items; i++) {
  ...if (data.rows[i].role != "director") {
  .....box_string += `<option value=${data.rows[i].id} ${selectValue==data.rows[i].id? `selected` : ''}>${data.rows[i].name}</option>`;
  ...}
  }
  box_string += `</select>
  ....<input type="submit" value="view">
  </form>`;
  return box_string;
}
module.exports = gen_box;

```

Figure 39: Source Code For Select Box.

This is a function that generates a select box. It takes in one parameter, `selectValue`, which is the value of the option that should be selected by default. If no value is passed in, it defaults to -1. The function queries the database for all shops and their corresponding users' roles. It then creates an HTML string with a form containing a select box with options for each shop (except those whose user role is "director"). The first option has text "All shops" and its value is 0. The other options have values equal to their respective shop's id and text equal to their respective shop's name.

e. CRUD Function.

```
var pg_conn = require('./pg_config');

async function crud(req_body, session){
  let id = req_body.id;
  let product_name = req_body.name;
  let price = req_body.price;
  let quantity = req_body.quantity;
  let shop_id = session.shop_id;
  if (req_body.crud == 'update'){
    var query = {
      text: `UPDATE products
            SET name = $2, price = $3, quantity = $4
            WHERE id = $1;`,
      values: [id, product_name, price, quantity]
    }
  }
  else if (req_body.crud == 'delete'){
    var query = {
      text: `DELETE FROM products
            WHERE id = $1;`,
      values: [id]
    }
  }
  else {
    var query = {
      text: `INSERT INTO products VALUES
            ($1, $2, $3, $4, $5);`,
      values: [id, product_name, price, quantity, shop_id]
    }
  }
  results = await pg_conn.query(query);
  return results;
}
module.exports = crud;
```

Figure 40: Source Code For CRUD Function 1.

```
if (req_body.crud == 'update'){
  var query = {
    text: `UPDATE products
          SET name = $2, price = $3, quantity = $4
          WHERE id = $1;`,
    values: [id, product_name, price, quantity]
  }
}
```

Figure 41: Source Code For CRUD Function 2.

This is a code that checks if the crud property of the req_body object is equal to 'update'. If it is, then it sets a variable called query to an object with two properties: text and values. The text property contains a string that represents an SQL query. The values property contains an array of values that will be used in the SQL query.

```
else if (req_body.crud == 'delete'){  
  var query = {  
    text: `DELETE FROM products  
          WHERE id = $1;`,  
    values: [id]  
  }  
}
```

Figure 42: Source Code For CRUD Function 3.

This is a code that checks if the crud property of the req_body object is equal to 'delete'. If it is, then it sets a variable called query to an object with two properties: text and values. The text property contains a string that represents an SQL query. The values property contains an array of values that will be used in the SQL query.

```
else {  
  var query = {  
    text: `INSERT INTO products VALUES  
          ($1, $2, $3, $4, $5);`,  
    values: [id, product_name, price, quantity, shop_id]  
  }  
}
```

Figure 43: Source Code For CRUD Function 4.

This is a function that inserts data into the database. It takes in 5 parameters and inserts them into the products table. The parameters are: id, product_name, price, quantity and shop_id.

2. Controller (Route).

a. Admin.

```
var express = require('express');
var gen_box = require('../models/select_box');
var display_product = require('../models/product_table');
const { render } = require('ejs');
var router = express.Router();

/* GET home page. */
var session;
router.get('/', async function(req, res, next) {
  session = req.session;
  console.log(session.shop_id)
  if(session.user_id){
    let shop_id = session.shop_id;
    let username = session.user_id;
    let table = await display_product(shop_id,session);
    let box_string = await gen_box(shop_id);
    res.render('admin', {title: 'Admin Page', name: username, select_box: box_string, table_string: table})
  }
  else{
    res.redirect('/login');
  }
});

router.post('/select_box', async function(req, res, next) {
  let shop_id = req.body.shop;
  username = req.session.user_id;
  let table = await display_product(shop_id,session);
  let box_string = await gen_box(shop_id);
  res.render('admin', { title: 'Admin Page',
    name: username,
    select_box: box_string,
    table_string: table});
});

module.exports = router;
```

Figure 44: Admin.

```
router.get('/', async function(req, res, next) {
  session = req.session;
  console.log(session.shop_id)
  if(session.user_id){
    let shop_id = session.shop_id;
    let username = session.user_id;
    let table = await display_product(shop_id,session);
    let box_string = await gen_box(shop_id);
    res.render('admin', {title: 'Admin Page', name: username, select_box: box_string, table_string: table})
  }
  else{
    res.redirect('/login');
  }
});
```

Figure 45: Source Code For Admin 1.

This is a function that handles the GET request to the admin page. It checks if there is a user logged in and if so, it renders the admin page with all of the products from the shop. If not, it redirects to login.

```
router.post('/select_box', async function(req, res, next) {
  let shop_id = req.body.shop;
  username = req.session.user_id;
  let table = await display_product(shop_id,session);
  let box_string = await gen_box(shop_id);
  res.render('admin', { title: 'Admin Page',
    name: username,
    select_box: box_string,
    table_string: table});
});
```

Figure 46: Source Code For Admin 2.

This is a post request that will be called when the user selects a shop from the dropdown menu. It will then call two functions, one to generate the table and one to generate the select box.

b. User.

```
var express = require('express');
var router = express.Router();
var display_product = require('../models/product_table');
var crud = require('../models/crud');
var session;

/* GET users listing. */
router.get('/', async function(req, res, next) {
  session = req.session;
  if(session.user_id){
    let username = session.user_id;
    let shop_id = session.shop_id;
    let table = await display_product(shop_id,session);
    res.render('users', { title: 'ATN Shop',
      name: username,
      table_string: table})
  }
  else{
    res.redirect('/login');
  }
});

router.post('/crud', async function(req, res, next) {
  console.log(req.body);
  let results = await crud(req.body,session);
  //refresh the page
  let table = await display_product(session.shop_id,session);
  res.render('users', { title: 'ATN Shop',
    name: session.user_id,
    table_string: table})
});

module.exports = router;
```

Figure 47: User.

```
router.get('/', async function(req, res, next) {  
  session = req.session;  
  if(session.user_id){  
    let username = session.user_id;  
    let shop_id = session.shop_id;  
    let table = await display_product(shop_id,session);  
    res.render('users', { title: 'ATN Shop',  
      name: username,  
      table_string: table})  
  }  
  else{  
    res.redirect('/login');  
  }  
});
```

Figure 48: Source Code For User 1.

This is a function that will be called when the user accesses the `/users` route. It gets the session and checks if there is a `user_id` in it. If there is, then it will get the `username` and `shop_id` from the session, call another function to display products, and render a page with those values. Otherwise, it will redirect to login page.

```
router.post('/crud', async function(req, res, next) {  
  console.log(req.body);  
  let results = await crud(req.body,session);  
  //refresh the page  
  let table = await display_product(session.shop_id,session);  
  res.render('users', { title: 'ATN Shop',  
    name: session.user_id,  
    table_string: table})  
});
```

Figure 49: Source Code For User 2.

This is a post request that will be called when the user done they crud function. It will send data to the server and then refresh the page with new data.

c. Index.

```
1 var express = require('express');
2 var authen = require('../models/authenticator')
3 var router = express.Router();
4 /* GET home page. */
5 router.get('/', function(req, res, next) {
6   res.render('index', { title: 'ATN Shop' });
7 });
8 router.get('/login',function(req,res,next){
9   res.render('login', { title: 'Login',
10     message: ''})
11 }
12 /* POST home page */
13 router.post('/',function(req,res,next){
14   res.render('login', { title: 'Login',
15     message: 'Enter your account'})
16 }
17 /* POST login page */
18 router.post('/login', async function(req,res,next){
19   let username = req.body.username;
20   let password = req.body.password;
21   // console.log(username+" "+password);
22   var session = req.session;
23   let [authenticated, shop_id, role] = await authen(username, password);
24   if(authenticated){
25     session.user_id = username;
26     session.shop_id = shop_id;
27     session.role = role;
28     console.log(session.user_id,session.shop_id,session.role);
29     if(role == 'shop'){
30       res.redirect('/users');
31     }
32     if(role == 'director'){
33       res.redirect('/admin')
34     }
35   }
36   else{
37     res.render('login', { title: 'Login',
38       message: 'Incorrect username or password'});
39   }
40 }
41 router.get('/logout', function(req, res, next) {
42   req.session.destroy();
43   res.redirect('/');
44 });
45
46 module.exports = router;
47
```

Figure 50: Index.

This is a code for render the landing page and the login page for the project

```
/* POST home page */  
router.post('/', function(req, res, next){  
  res.render('login', { title: 'Login',  
    message: 'Enter your account'})  
})
```

Figure 51: Source Code For Index 1.

This is a function that renders the login page with a message. It is called when the user clicks on the login button in the home page.

```
/* POST login page */  
router.post('/login', async function(req, res, next){  
  let username = req.body.username;  
  let password = req.body.password;  
  // console.log(username+" "+password);  
  var session = req.session;  
  let [authenticated, shop_id, role] = await authen(username, password);  
  if(authenticated){  
    session.user_id = username;  
    session.shop_id = shop_id;  
    session.role = role;  
    console.log(session.user_id, session.shop_id, session.role);  
    if(role == 'shop'){  
      res.redirect('/users');  
    }  
    if(role == 'director'){  
      res.redirect('/admin')  
    }  
  }  
  else{  
    res.render('login', { title: 'Login',  
      message: 'Incorrect username or password'});  
  }  
})
```

Figure 52: Source Code For Index 2.

This is a route handler for the login page. It handles the POST request to /login. The code checks if the username and password are correct, then redirects to either /users or /admin depending on the role of user. If it's incorrect, it renders the login page with an error message.

```
router.get('/logout', function(req, res, next) {  
  req.session.destroy();  
  res.redirect('/');  
});
```

Figure 53: Source Code For Index 3.

This is a function that handles the logout request. It destroys the session and redirects to home page.

3. View.

a. Admin Interface Code.

```
<div class="hero">
  <nav>
    <div class="shop_name"><a href="">ATM SHOP</a></div>
    <ul>
      <li><a href="">Home</a></li>
      <li><a href="">Features</a></li>
      <li><a href="">About</a></li>
      <li><a href="">Contact</a></li>
    </ul>
    

    <div class="sub-menu-wrap" id="subMenu">
      <div class="sub-menu">
        <div class="user-info">
          
          <h2>%= name %</h2>
        </div>
        <hr>
        <a href="#" class="sub-menu-link">
          
```

Figure 54: Some Source Code For Admin Interface.

```
<script>
  window.setInterval(() => {window.location.reload();}, 10000);

  let subMenu = document.getElementById("subMenu");
  function toggleMenu(){
    subMenu.classList.toggle("open-menu");
  }
</script>
```

Figure 55: JavaScript for Admin Page.

This is a script tag that contains javascript code. It's used to add functionality to the page. In this case, it adds a function called `toggleMenu()` this function use to display sub menu when user click on the user profile and I will show how it work in the screenshot part. Additionally, there is another function that reloads the page every 10 seconds.

b. User Interface Code.

```
<!DOCTYPE html>
<html>
<head>
<title>%= title %</title>
<link rel="stylesheet" href="/stylesheets/style.css">
<script src="https://cdn.tailwindcss.com"></script>
</head>
<body>
<div class="hero">
<nav>
<div class="shop_name"><a href="">ATM SHOP</a></div>
<ul>
<li><a href="">Home</a></li>
<li><a href="">Features</a></li>
<li><a href="">About</a></li>
<li><a href="">Contact</a></li>
</ul>

<div class="sub-menu-wrap" id="subMenu">
<div class="sub-menu">
<div class="user-info">

<h2>%= name %</h2>
</div>
```

Figure 56: Some Source Code For User Interface.

The source code for user interface is the same as admin interface it also have the function called `toggleMenu()` to display the sub menu and it not have the fuction that reloads the page every 10 seconds because that function for admin and user does not need it.

c. Login Interface Code.

```
<!DOCTYPE html>
<html>
<head>
<title>%= title %</title>
<link rel="stylesheet" type="text/css" href="fontawesome/css/all.min.css">
</head>
<body>
<div class="container">
<div class="header">
<h1>Login</h1>
</div>
<div class="main">
<form action="login" method="post">
<span>
<i class="fa fa-user"></i>
<input type="text" placeholder="Username" name="username">
</span><br>
<span>
```

Figure 57: Some Source Code For Login Interface.

There is some source code for login page when user click on the button login.

d. Index Interface Code.

```
<a href="#" class="logo"><span>A</span>TN</a>
<div class="navbar-menu">
  <a href="#">home</a>
  <a href="#">about</a>
  <a href="#">products</a>
  <a href="#">contact</a>
</div>
</div>
</div>
<section id="home">
  <div class="content1">
    <div class="content">
      <p>Lorem ipsum dolor sit amet, consectetur elit. Proin ante tellus, interdum vitae consequat maximus, sagittis ac lacus. Phasellus finibus rutrum diam.</p>
      <form action="" method="post">
        <button type="submit">get started</button>
      </form>
    </div>
    <div class="social-container">
      <div class="social-icons">
        <li><a href="#"><i class="fa fa-facebook-f"></i></a></li>
        <li><a href="#"><i class="fa fa-instagram"></i></a></li>
        <li><a href="#"><i class="fa fa-twitter"></i></a></li>
      </div>
    </div>
  </div>
</section>
```

Figure 58: Some Source Code For Index Interface.

There is some source code for landing page in index interface.

I also utilize some CSS for this project and here is some CSS for the project.

➤ CSS for landing page in index interface.

```
<style>
@import url(https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.6.3/css/font-awesome.css);
@import url('https://fonts.googleapis.com/css2?family=Great+Vibes&display=swap');

*{
  margin: 0;
  padding: 0;
  text-decoration: none;
  box-sizing: border-box;
}

.navbar{
  position: fixed;
  background-color: transparent;
  width: 100%;
  padding: 30px 0;
  top: 0;
  z-index: 100;
  line-height: 20px;
}
```

Figure 59: CSS for landing page in index interface.

➤ **CSS for login page.**

```
<style>
body {
  font-family: sans-serif;
  background-image: url(images/bg.jpg);
  background-repeat: no-repeat;
  overflow: hidden;
  background-size: cover;
}

.alert{
  position: relative;
  padding: 0.75rem 1.25rem;
  margin-bottom: 1rem;
  border: 1px solid transparent;
  border-radius: 0.25rem;
}
```

Figure 60: CSS for login page.

➤ **CSS for admin and user interface.**

```
@import url('https://fonts.googleapis.com/css2?family=Poppins:wght@100;200&display=swap');

*{
  margin: 0;
  padding: 0;
  font-family: 'Poppins', sans-serif;
  box-sizing: border-box;
}

.hero{
  width: 100%;
  min-height: 100vh;
  background: #e6eaff;
  color: #525252;
}

.table{
  margin-top: 60px;
}
```

Figure 61: CSS for user and admin interface.

IV. Commit your code to Github.

Initialize the “git init” command in the company's code folder ATN.

```
found 0 vulnerabilities
Admin@Iris 99% Cloud Computing @ 16.17.1
> git init
Initialized empty Git repository in E:/Cloud Computing/Cloud Computing/.git/
```

Figure 62: git init.

Add File to GitHub.

```
Admin@Iris 99% Cloud Computing #master @ 16.17.1
> git add .
warning: in the working copy of 'README.md', LF will be replaced by CRLF the next time
warning: in the working copy of 'app.js', LF will be replaced by CRLF the next time
warning: in the working copy of 'bin/www', LF will be replaced by CRLF the next time
```

Figure 63: git add.

```
Admin@Iris 99% Cloud Computing #master @ 16.17.1
> git commit -m 'Add file'
```

Figure 64: git commit.

Enter the command git remote to add origin to confirm the path to the folder you want to save, the command git branch to add into the branch you want and enter the command git push -u origin main to push the code file you want to add to GitHub.

```
create mode 100644 views/users.ejs
Admin@Iris 99% Cloud Computing #master @ 16.17.1
> git remote add origin https://github.com/11r-is/cloud-computing.git
> git branch -M main
> git push -u origin main
```

Figure 65: git remote.

Push code to GitHub.

```
> git branch -M main
> git push -u origin main
Enumerating objects: 2579, done.
Counting objects: 100% (2579/2579), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2556/2556), done.
Writing objects: 100% (2579/2579), 10.14 MiB | 4.33 MiB/s, done.
Total 2579 (delta 198), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (198/198), done.
To https://github.com/11r-is/cloud-computing.git
* [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
Admin@Iris 99% Cloud Computing #main @ 16.17.1
```

Figure 66: git push.

Code after pushing to GitHub success.

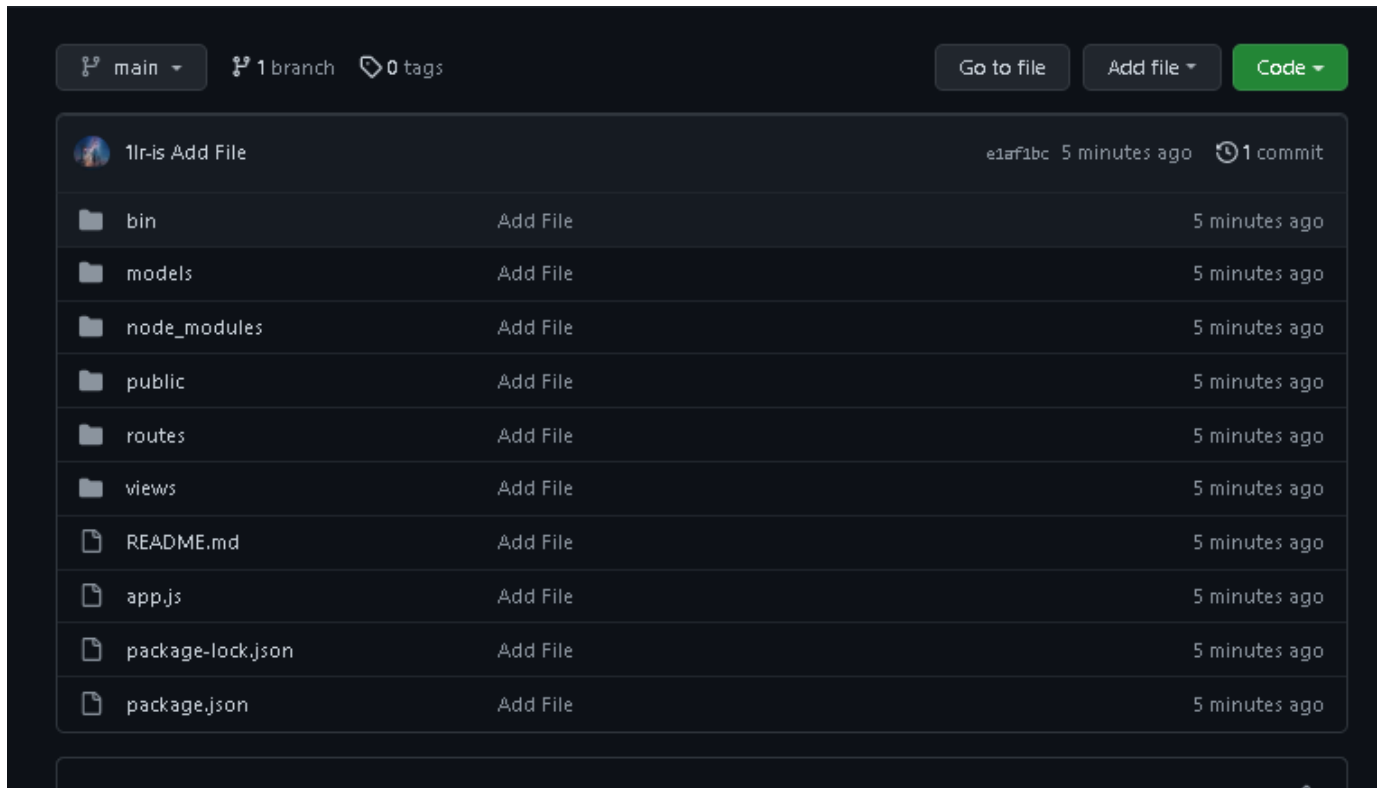


Figure 67: Push GitHub success.

V. Deploy app on Heroku.

The Index Page for ATN Store.

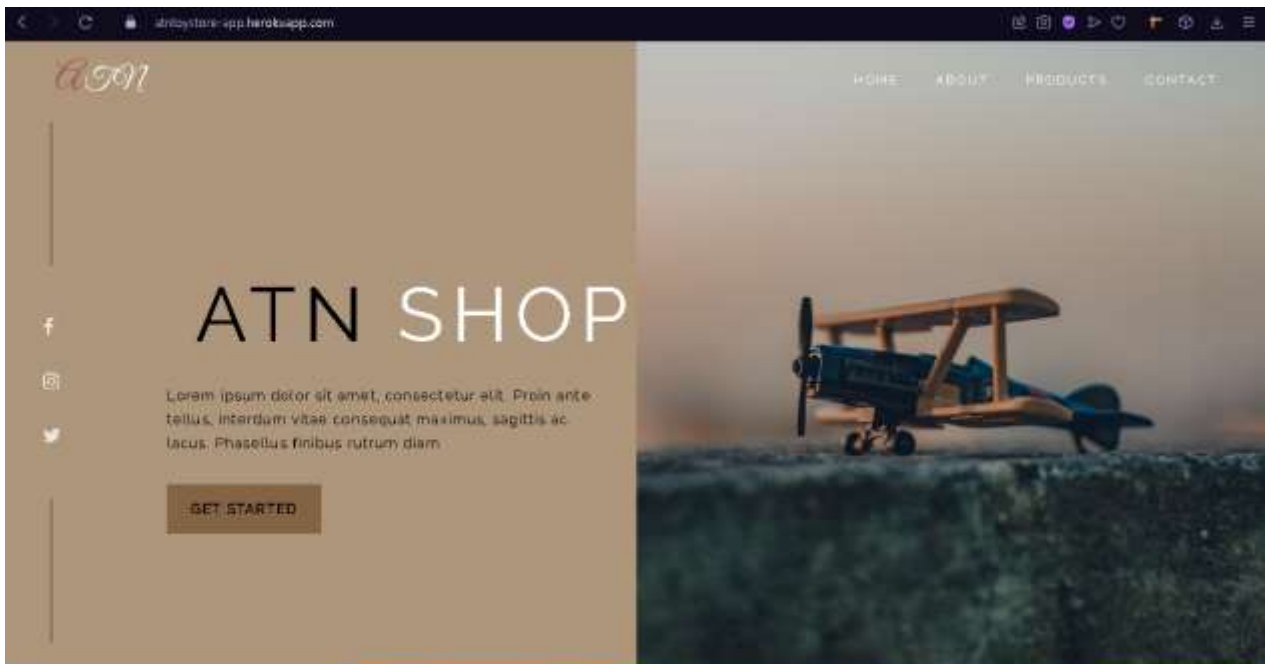


Figure 68: Index page.

The Login Page of ATN Store.

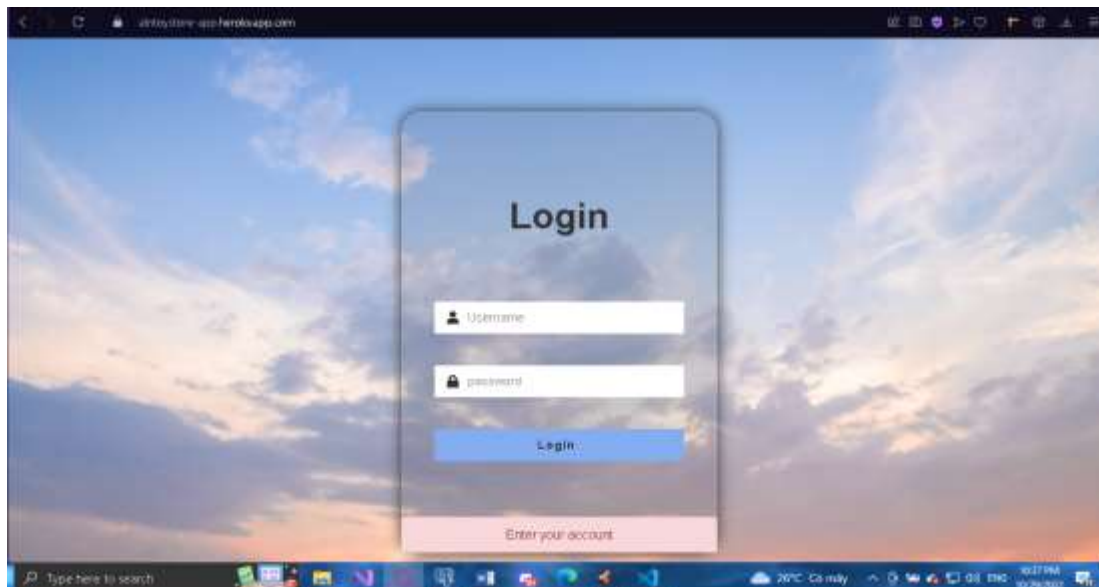


Figure 69: Login page.

Login as user.

ATN SHOP

Home Features About Contact

Table products

id	name	price	quantity	actions
3	barbie	12	12	Update Delete
5	superman	12	12	Update Delete
8				Insert

Figure 70: Login as user.

Login as admin.

ATN SHOP

Home Features About Contact

Choose a shop: All shops - View

Table products

id	name	price	quantity	shop_id
1	mega	10	10	1
2	robot	10	10	2
3	barbie	12	12	3
4	doll	10	10	4
5	superman	12	12	3

Figure 71: Login as admin.

When the user/admin enter an incorrect username and password the login page will display the alert and require user/admin to enter username and password again.

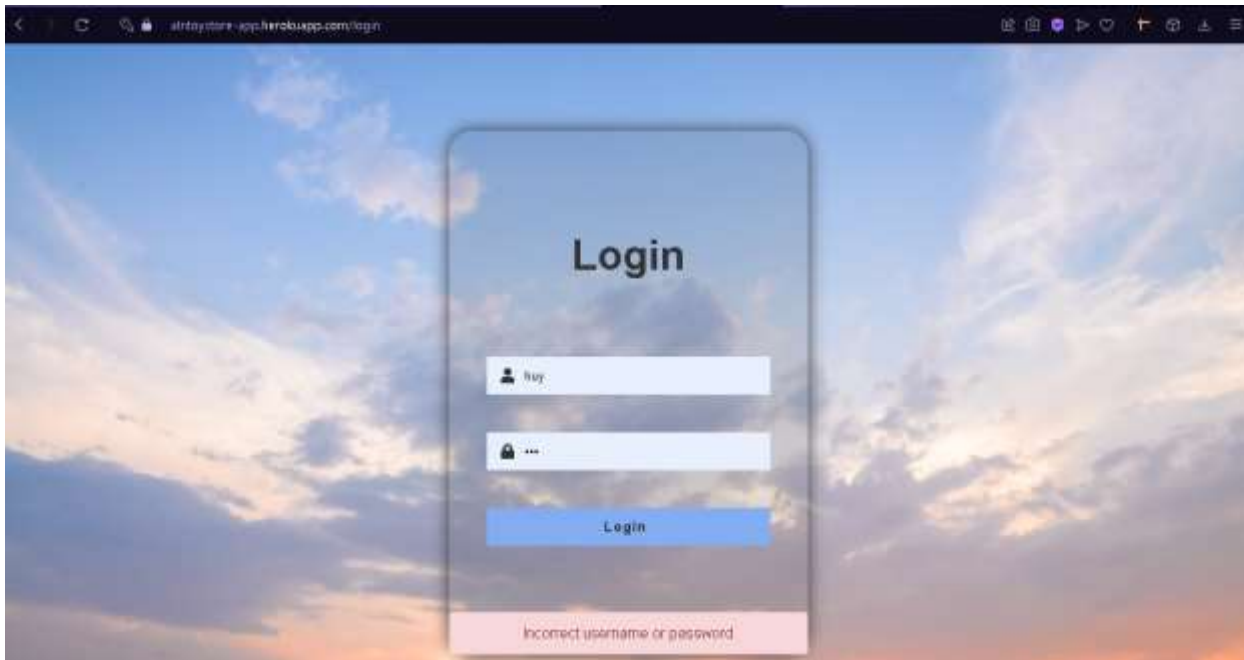


Figure 72: Wrong username and password.

As I mentioned before in admin and user interface have script to display the sub menu when the user click on the avatar.

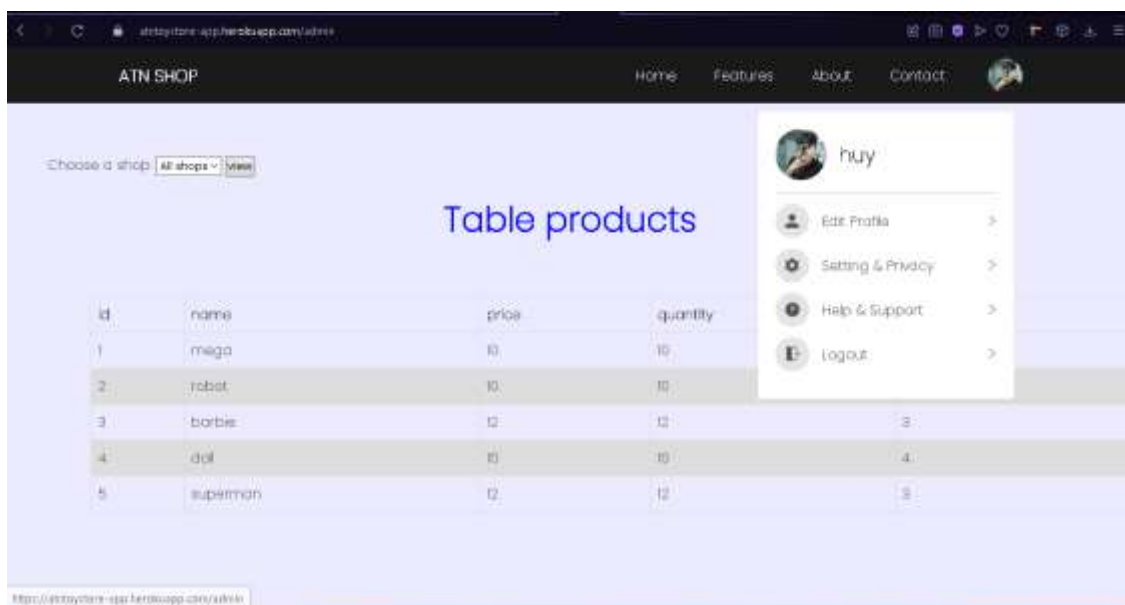
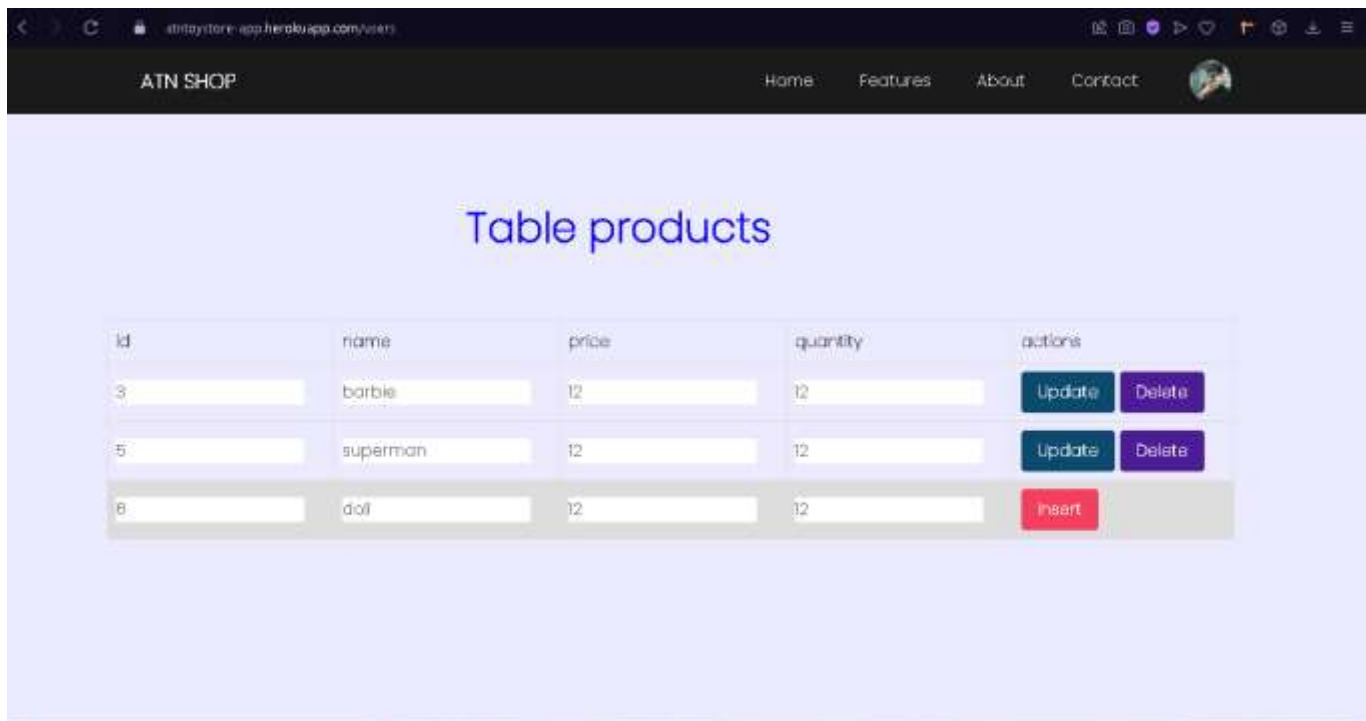


Figure 73: The sub menu.

And last but not least, I also create a script that call auto refresh which I also mentioned above. This script works by updating the admin page every time a user adds a new product or do something new into the database, and every 10 seconds the admin page will automatically refresh once to ensure that the admin knows all information has been updated as soon as possible.

Now I will insert new product which is “doll” into the system.



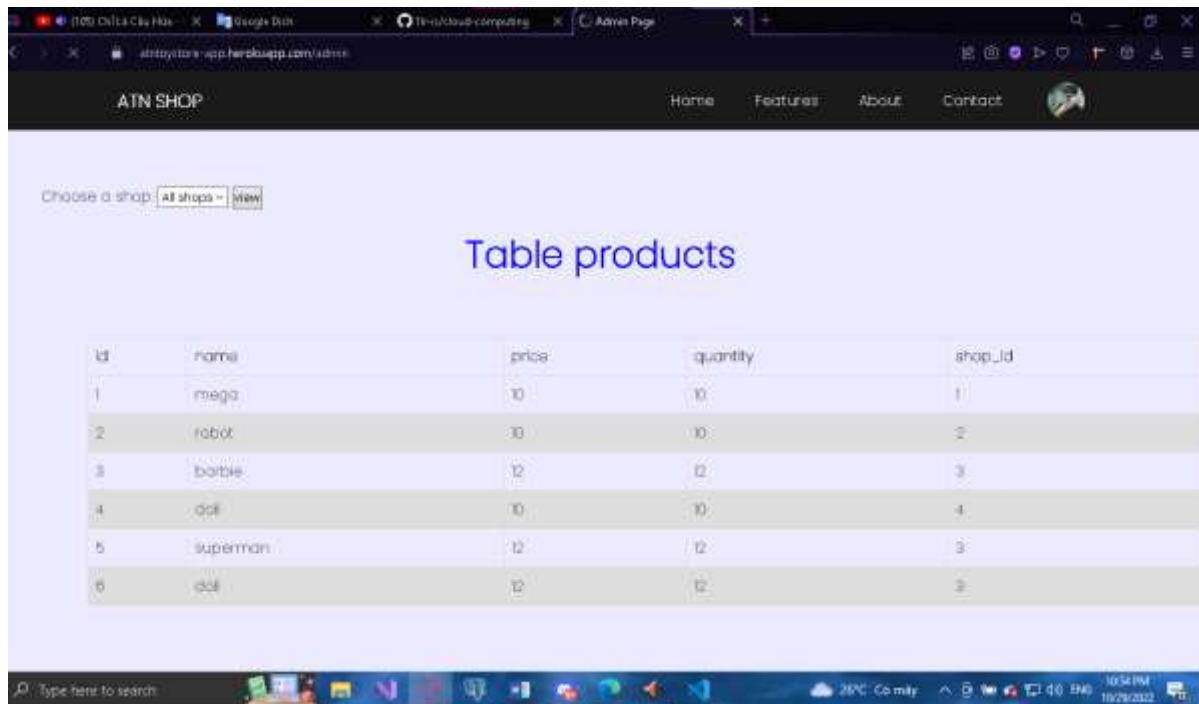
ATN SHOP

Home Features About Contact

Table products

id	name	price	quantity	actions
3	barbie	12	12	<button>Update</button> <button>Delete</button>
5	superman	12	12	<button>Update</button> <button>Delete</button>
6	doll	12	12	<button>insert</button>

When the user insert success the new product will be insert into the database and admin will know the type of item has been added to the system quickly by auto-refreshing every 10 seconds.



The screenshot shows the Admin Page of the ATN SHOP. The page has a navigation bar with links for Home, Features, About, and Contact. Below the navigation bar, there is a section titled "Table products" which contains a table with the following data:

id	name	price	quantity	shop_id
1	mega	10	10	1
2	robot	10	10	2
3	barbie	12	12	3
4	apple	10	10	4
5	superman	12	12	3
6	doll	12	12	3

You can see the admin page has automatically reloaded and one new item has been added which is "doll".

Finally, to log out the system I just click on the avatar and choose log out.

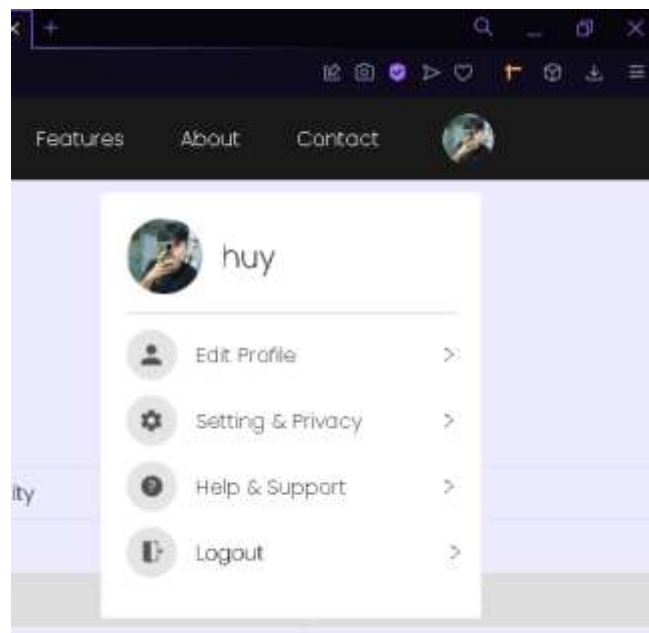


Figure 74: Log Out.

After that, the system will return to the landing page.

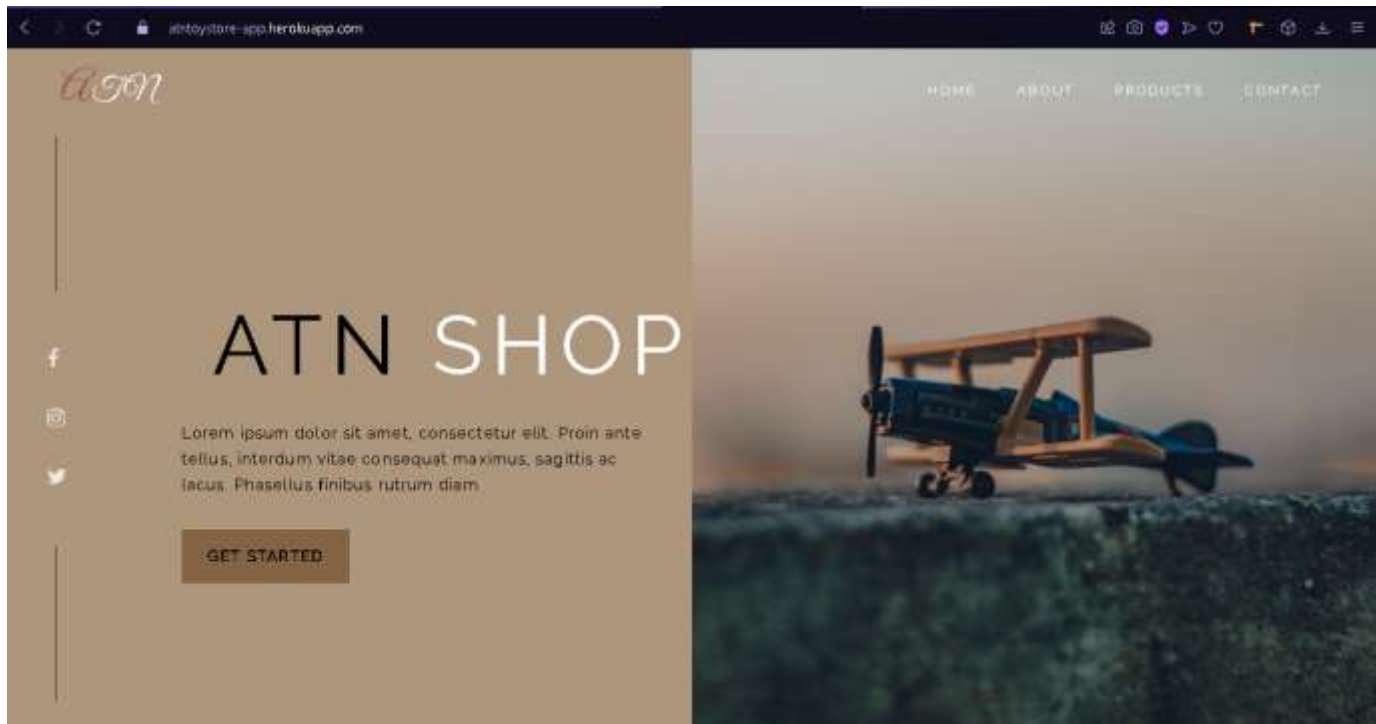


Figure 75: Log Out Success.

VI. Link of the project.

- **Link Github:** <https://github.com/1lr-is/Cloud-Computing-Project-Assignment.git>
- **Link Website:** <https://atntoystore-app.herokuapp.com>

CHAPTER 4: ANALYSING THE MOST COMMON PROBLEMS WHICH ARISE IN A CLOUD COMPUTING PLATFORM AND DISCUSS APPROPRIATE SOLUTIONS TO THESE PROBLEMS (P7).

Experts and professionals are concerned about the security of cloud computing in today's increasingly remote world. According to various research studies, some of these security vulnerabilities emerge as data breaches, while others concern access control. Whatever the issue, it is critical for decision makers to consider when picking software or a solution.

I. Optimizing Cloud Expenses.

This has been one of the most onerous tasks for cloud users. Managing cloud expenses is a complex task that extends beyond security considerations. For a number of reasons, businesses waste a considerable percentage of their budget on unnecessary cloud-related activities. The expenditures connected with cloud computing surpass restrictions, whether due to ignorance, a lack of competence, hurried operations, or unskilled resources.

➤ **Solution:**

- Seeking assistance from a variety of technological solutions for cloud cost control.
- Including a cloud computing partner that is well-versed and skilled in cloud solution administration.
- Putting up a centralized cloud team to go through financial details.

II. Migrating Existing Applications onto the Cloud.

When you compare building a completely new cloud application to migrating an existing software to the cloud, the answer is simple. Migrating an existing one has its own set of challenges, drawbacks, and issues. Security settings, time consumption, budget overflow, mismatched demands, downtime, and other difficulties have hampered cloud migration.

➤ **Solution:**

- Pre-migration testing centered on migration criteria.
- Setting a suitable project timetable and budget while accounting for migration issues.
- Hiring cloud service providers with migration experience.

III. Cost Calculations Within Limits.

Because cloud operations are still in their infancy, businesses commonly underestimate their costs. Every company's operational costs differ, and budgets shift. Certain unanticipated charges may lead the budget to be surpassed. Even small businesses struggle to deploy cloud technologies on a budget.

➤ **Solution:**

- Developing a cost estimating technique from the beginning.
- Incorporating cost specialists who can forecast all connected costs.
- Putting money aside in the budget for unanticipated events.

IV. Proprietary Lock – in.

When a client is unable to readily move to a competitive product, this is referred to as vendor lock-in. It is primarily the result of incompatible technologies or standards. This is what cloud computing providers face. Global behemoths such as AWS, Microsoft Azure, and Google Cloud Platform may represent a vendor lock-in danger to IT executives.

➤ **Solution:**

- Using care when receiving cloud services from a variety of cloud service providers.
- Assuring from the start that the services involved are compatible with other suppliers.
- Understanding the criteria from start to finish.

V. Software as a Service (SaaS) Security Concerns.

Multiple concurrent users are supported by SaaS-based apps. Because SaaS products are accessible through the internet, web browser security is crucial. Enterprise data is stored on the SaaS platform.

The following are the most common SaaS pain points:

- Network safety.
- Locality of resources.
- Cloud specifications.

- Data separation.
- Data accessibility.

➤ **Solution:**

We will go through a useful SaaS security check list. All users should ask the following questions to assess the security risks and capabilities of third-party SaaS providers:

Which metrics are suitable for reporting? Will the service providers be able to produce reports that persuade the CIO, board, and auditors that corporate data is safe and fulfills regulatory requirements?

What are the access controls like? The most well-known type of data breach nowadays is the unlawful or inadvertent usage of user credentials. The capacity to observe individual user activity, including administrative changes, is required for data security. Is the data given easily integrated into internal monitoring systems to minimize data silos? You must monitor both SaaS and internal business apps from a centralized control panel to make the report simple and easy to understand. How significant and crucial is the company's data? Every SaaS program must understand the importance of the data it uses for business goals. Is the SaaS application managing sensitive customer data correctly? Inventory compliance measures can then be taken.

VI. Platform as a Service (PaaS) Security Concerns.

PaaS provides a ready-to-use platform by leveraging operating systems that operate on the vendor's infrastructure. Because CSPs own the infrastructure, the proliferation of user objects on cloud servers is responsible for the majority of the security vulnerabilities associated with centralized design. It is definitely forbidden to allow objects to access resources and to safeguard them from corrupt or malicious providers in a way that appropriately decreases the risks involved. Concerns concerning secure communications and access control are brought together by network access and service measurement. Follow tried-and-true methods, enforce allowable item sizes, and apply trustworthy traceability solutions to relieve concerns. Aside from the above mentioned concerns, user privacy in the shared cloud must be safeguarded. As a result, the options presented must be discussed in secret.

➤ **Solution:**

Understanding the PaaS concept, kinds, and security issues discussed in this article will assist you in designing a safe PaaS cloud. With a deeper grasp of the cloud-based PaaS environment, the various PaaS features may be correctly employed. Along with PaaS security issues, PaaS features and criteria for selecting PaaS vendors have been developed. The final section discussed the security concerns in PaaS and the suitable remedies to offer a comprehensive grasp of data security issues and other challenges encountered while running an application on a PaaS platform. Clients that have solutions and are aware of the risk can use PaaS safely.

VII. Infrastructure as A Service (IaaS) Security Issues.

Cloud computing has great promise in terms of enhancing flexibility and agility, cutting potential expenses, and providing developers with a competitive advantage in terms of quickly and effectively constructing infrastructure and producing software that supports commercial success. Private clouds, in particular, have the ability to solve a variety of issues, but they are dangerous. However, the following are some of the most prevalent challenges experienced in a private cloud environment:

- Hypervisor security.
- Multitenancy.
- Access control and identity management
- Network safety.

➤ **Solution:**

The virtual network security platform, cloud workload protection platform, cloud security posture management, and cloud access security broker are the four fundamental IaaS security solutions.

VIII. Data Security Violation.

It was and continues to be a source of worry for cloud computing. Data security has always been a primary priority, particularly with cloud-based services. As a wide range of industrial sectors adopt the cloud culture, data breaches may wreak havoc on end users. A lack of security measures, whether in the public, private, hybrid, poly, or single cloud, can have a direct impact on the organization's operations.

➤ ***Solution:***

- Having a stringent security procedure in place for the cloud solution.
- Changing corporate culture to ensure data security.
- Training and certification of IT employees in order to address security risks with a proper solution.

IX. Consistent Results.

In the case of mission-critical solutions, performance in cloud-based applications is vital. This is due to the fact that any interruption in the cloud has an immediate impact on the application's performance. Cloud outages may happen to any system. After all, no cloud service is without flaws. At such times, the client cannot survive the mission-critical application.

➤ ***Solution:***

- Developing comprehensive cloud-based application recovery solutions.
- Having enough catastrophe recovery systems in place.
- Third-party vendors should provide failover alternatives in the event of unavailability.

CHAPTER 5: ASSESS THE MOST COMMON SECURITY ISSUES IN CLOUD ENVIRONMENTS (P8).

I. Misconfiguration.

Cloud data breaches are frequently caused by incorrectly configured cloud security settings. Cloud security posture management approaches used by many businesses are insufficient for safeguarding their cloud-based infrastructure.

Several factors have a part in this. Because cloud infrastructure is designed to be simple to use and to enable data sharing, it is difficult for businesses to ensure that data is only accessible to authorized individuals. Furthermore, cloud-based infrastructure users lack extensive access and control over their infrastructure, mandating reliance on security measures provided by their cloud service provider (CSP) to create and protect their cloud installations.

Because many firms are inexperienced with cloud infrastructure protection and often have multi-cloud deployments, each with its own set of vendor-provided security measures, a misconfiguration or security oversight might expose an organization's cloud-based resources to attackers.

II. Data Privacy.

Data privacy has long been a concern for corporate leaders, but it has grown increasingly important as the cybersecurity situation deteriorates and grows more complex. GDPR, HIPAA, PCI DSS, and a slew of other data protection rules are now in operation and were enacted to safeguard client information.

Only 12% of global IT organizations are aware of how GDPR may effect their cloud services, according to a Commvault research. This finding suggests that organizations will be more susceptible if they are not GDPR-compliant in the cloud.

Businesses that fail to meet these rules risk facing harsh penalties such as significant fines or, in the worst-case scenario, a data leak. A managed cloud provider may share compliance responsibilities. Companies should choose a partner who understands data protection and compliance regulations to ensure ongoing security for their business and clients.

III. Data Problem.

Data saved in cloud-based systems is freely accessible to others. These setups enable simple data exchange with third parties by direct email invites or the posting of a public link to the data, and they are instantly available from the open Internet.

Although a valuable asset and necessary for cloud collaboration, the ease with which data may be shared raises serious concerns about data loss or leakage. In fact, 69 percent of businesses are most concerned about cloud security. Anyone with access to the link can examine data shared over public links or made publicly available in a cloud-based repository, and there are applications that can search the internet for these unsecured cloud installations.

IV. Insecure Interfaces/APIs.

CSPs usually provide a range of application programming interfaces (APIs) and interfaces to customers. In general, these APIs are well-documented and easy to use for CSP clients.

However, if a customer has not adequately protected the interfaces of their cloud-based infrastructure, this may cause issues. A cybercriminal can utilize customer documentation to identify and exploit potential strategies for accessing and exfiltrating sensitive data from a company's cloud infrastructure.

V. Hijacking of Accounts.

Many people have extremely poor password security, including password repetition and the use of weak passwords. This vulnerability exacerbates the effect of phishing efforts and data breaches by allowing a single stolen password to be used on several accounts.

Account hijacking is one of the most significant cloud security threats as businesses increasingly rely on cloud-based infrastructure and apps for critical business processes. By using an employee's credentials, an attacker can get access to critical data or functionality, and compromised consumer credentials allow complete control over their online account. Furthermore, enterprises in the cloud usually lack the capacity to detect and respond to these risks in the same manner that they can with on-premises technology.

VI. Notifying Customers Affected by Data Breaches.

One disadvantage of not having total control and visibility over a network is that it may be difficult to know which resources and data have been impacted if the network is attacked. It may be difficult to ascertain which customers were affected by a data breach and what data was exposed if a cloud service does not provide appropriate visibility features and access to event logs.

If a breach occurred under such conditions, it would be necessary to plan for the worst-case situation and notify everyone whose data may have been stored on the cloud platform. It would be the only way to assure that all data breach notifications reached everyone.

VII. A Lack of Visibility/Control.

One of the most major benefits of using cloud-based technology is that the client does not have to manage the resources needed to keep it running (such as servers). Delegating responsibility for the day-to-day maintenance of a program, platform, or computing asset, on the other hand, may result in less visibility and control over that asset.

- Examine the efficacy of their security precautions (since there is no access to the cloud platform's tools and data).
- Implement incident response plans (because to the fact that they may not have total control over cloud-based assets)
- Analyze information about their products, services, and clients (which is often necessary to recognize abnormal use patterns inherent to a security breach).

VIII. Unauthorized Access.

Unlike on-premises infrastructure, cloud-based infrastructure is located outside the network perimeter and is immediately accessible over the public Internet. While this enhances worker and customer access to this infrastructure, it also enables illegal access to an organization's cloud-based services simpler. Improperly setup security or compromised credentials may allow an attacker to get direct access, sometimes without the awareness of the enterprise.

References

Chandrasekaran, K., 2014. *Essentials of Cloud Computing*. s.l.:s.n.

Checkpoint, n.d. *Top 15 Cloud Security Issues, Threats and Concerns*. [Online]

Available at: <https://www.checkpoint.com/cyber-hub/cloud-security/what-is-cloud-security/top-cloud-security-issues-threats-and-concerns/>

[Accessed 29 10 2022].

Educba, 2022. *What is Visual Studio Code?*. [Online]

Available at: <https://www.educba.com/what-is-visual-studio-code/>

[Accessed 29 10 2022].

Kumar, 2018. *10 Key Challenges in Cloud Computing and How to Overcome..* [Online]

[Accessed 29 10 2022].

Sharma, K., 2021. *What is PostgreSQL and why do enterprise developers and start-ups love it?*

[Online]

Available at: <https://ubuntu.com/blog/what-is-postgresql>

[Accessed 29 10 2022].