Name – **Onkar Kajrolkar**

CampbuzzId – **2020Sep@03308**

Email – **onkarkajrolkar7798@gmail.com**

**Web Based Case Study**

**LBJ Cycle-2**

1. **Synopsis –**

Web Development has been an increasing technology with engineers mastering the skills of frontend as well as backend. This Case study gave us the opportunity to experience the tip of the iceberg that is Web Development. The Frontend for this case study was built with HTML5 for web pages and CSS for styling of the said web pages. The server side code was written using Python by implementing the Flask module for setting up a local server. The HTML5 web pages are stored in the “templates” folder and the CSS styling file is stored in the “static” folder. The Flask will search for the templates and static folders for rendering the webpages and styling files respectively. Using python and the data received through the web pages, we can write to/read from a CSV file.

1. **Source Code –**
2. **“~/students-server.py” –**

from flask import \*

import csv

app = Flask(\_\_name\_\_)

app.secret\_key = "wubbalubbadubdub"

@app.route("/")

def index():

return render\_template("index.html")

@app.route("/add", methods=["POST", "GET"])

def add():

if request.method == "POST":

result = list(request.form.values())

split = result[3].split('-')

result[3] = split[2] + "-" + split[1] + "-" + split[0]

with open('students.csv', 'a', newline='') as csv\_file:

csv\_writer = csv.writer(csv\_file)

csv\_writer.writerow(result)

flash("Student added to Database!")

return render\_template("add-student.html")

@app.route("/search", methods=["POST", "GET"])

def search():

if request.method == "POST":

id = list(request.form.values())

success = False

with open('students.csv', 'r') as csv\_file:

csv\_reader = csv.reader(csv\_file)

for line in csv\_reader:

if line[0] == id[0]:

details = line

success = True

if not success:

flash(f"Student ID: {id[0]} not found, Please try again!", "info")

else:

result = dict(zip(keys, details))

return render\_template("search-student.html", data=True, result=result)

return render\_template("search-student.html", data=False, result="")

@app.route("/display")

def display():

with open('students.csv', 'r') as csv\_file:

csv\_reader = list(csv.reader(csv\_file))

temp = csv\_reader.copy()

if len(list(temp)) == 0:

flash("No Students to Display", "info")

return render\_template("display-student.html", result="")

else:

return render\_template("display-student.html", result=csv\_reader)

if \_\_name\_\_ == "\_\_main\_\_":

keys = []

with open('students.csv', 'r') as csv\_file:

csv\_reader = csv.DictReader(csv\_file)

keys = csv\_reader.fieldnames

app.run()

1. **“~/templates/index.html” –**

<!DOCTYPE html>

<html>

<link rel="stylesheet" type="text/css" href="{{ url\_for('static', filename='style.css')}}">

<head>

<title>Student Dashboard</title>

</head>

<body>

<ul id="list" type="none" align="center">

<li><button class="button" onclick="window.location='add'">Add Student</button></li>

<li><button class="button" onclick="window.location='search'">Search Student</button></li>

<li><button class="button" onclick="window.location='display'">Display all Students</button></li>

</ul>

</body>

</html>

1. **“~/templates/add-student.html”**

<!DOCTYPE html>

<html>

<link rel="stylesheet" type="text/css" href="{{ url\_for('static', filename='style.css')}}">

<head>

<title>Add A Student</title>

</head>

<body>

<div id="entry">

<ul type="none">

<form action="" method="post">

<li>Student Id</li>

<li><input type="text" name="id" required></li>

<li>Student Name</li>

<li><input type="text" name="name" required></li>

<li>Gender</li>

<li><input type="radio" name="gender" value="Male" required>Male</li>

<li><input type="radio" name="gender" value="Female" required>Female</li>

<li>Date of Birth</li>

<li><input type="date" name="date" required></li>

<li>City</li>

<li><input type="text" name="city" required></li>

<li>State</li>

<li><input type="text" name="state" required></li>

<li>Email Id</li>

<li><input type="text" name="email" required></li>

<li>Qualification</li>

<li><input type="text" name="qual" required></li>

<li>Stream</li>

<li><input type="text" name="stream" required></li>

<li><input type="submit" class="submit" value="Submit"></li>

</form>

{% with messages = get\_flashed\_messages() %}

{% if messages %}

{% for message in messages %}

<li>{{message}}</li>

{% endfor %}

{% endif %}

{% endwith %}

<li><button class="submit" onclick="window.location='/'">Home</button></li>

</ul>

</div>

</body>

</html>

1. **“~/templates/search-student.html” –**

<head>

<title>Search Student by College Id</title>

</head>

<body>

<div class="search" align="center">

<form action="" method="post">

<p style="font-family:sans-serif">Student ID</p>

<input type="text" name="id" required><br>

<input type="submit" class="submit" name="Submit">

</form>

{% with messages = get\_flashed\_messages() %}

{% if messages %}

{% for message in messages %}

<p>{{message}}</p>

{% endfor %}

{% endif %}

{% endwith %}

{% if data %}

{% for key,value in result.items() %}

<table class="table-search" >

<tr>

<th>{{key}}</th>

<td>{{value}}</td>

</tr>

</table>

{% endfor %}

{% endif %}

<button class="submit" onclick="window.location='/'">Home</button>

</div>

</body>

</html>

1. **“~/templates/display-student.html” –**

<!DOCTYPE html>

<html>

<link rel="stylesheet" type="text/css" href="{{ url\_for('static', filename='style.css')}}">

<head>

<title>Display all Students</title>

</head>

<body>

<div id="display-list" align="center">

<table class="table-display">

<tr>

{% for key in result[0] %}

<th>{{key}}</th>

{% endfor %}

</tr>

{% for lines in result[1:] %}

<tr>

{% for value in lines %}

<td>{{value}}</td>

{% endfor %}

</tr>

{% endfor %}

</table>

{% with messages = get\_flashed\_messages() %}

{% if messages %}

{% for message in messages %}

<p style="color:#000000";>{{message}}</p>

{% endfor %}

{% endif %}

{% endwith %}

<button class="submit" onclick="window.location='/'">Home</button>

</div>

</body>

</html>

1. **“~/static/style.css” –**

body{

background: #007fff;

}

#list{

background: #e6e6e6;

color: white;

font-family: sans-serif;

align-self: center;

text-align: center;

width: 400px;

margin: 100px auto;

border-radius: 10px;

box-shadow: 0 0 20px rgba(0,0,0,2);

}

#display-list{

background: #e6e6e6;

color: white;

font-family: sans-serif;

align-self: center;

text-align: center;

width: 1020px;

margin: 100px auto;

border-radius: 10px;

box-shadow: 0 0 20px rgba(0,0,0,2);

}

.button{

background: white;

color: black;

font-family: 'Nunito',sans-serif;

font-size: 200%;

margin: 50px 20px;

padding: 10px;

text-align: center;

position: relative;

border-radius: 10px;

box-shadow: 0 0 5px rgba(0,0,0,0.5);

}

.button:hover,.submit:hover{

background: #004dff;

color: white;

cursor: pointer;

}

#entry{

font-family: sans-serif;

background: #e6e6e6;

color: black;

width: 250px;

align-self: left;

text-align: left;

margin: 100px auto;

padding: 10px;

border-radius: 10px;

box-shadow: 0 0 20px rgba(0,0,0,2);

}

.submit{

background: white;

color: black;

font-family: 'Nunito',sans-serif;

font-size: 100%;

margin: 10px 50px;

padding: 10px;

text-align: center;

position: relative;

border-radius: 10px;

box-shadow: 0 0 5px rgba(0,0,0,0.5);

}

.search{

font-family: sans-serif;

background: #e6e6e6;

color: black;

padding: 20px;

width: 500px;

margin: 100px auto;

border-radius: 10px;

box-shadow: 0 0 20px rgba(0,0,0,2);

}

.table-search{

width: 400px;

overflow: hidden;

}

.table-search th{

background-color: #007fff;

color: white;

text-align: left;

font-weight: bold;

}

.table-search th,.table-search td{

padding: 12px 15px;

width:16.6%;

word-break: break-all;

border: 1px solid black;

}

.table-search tbody tr{

border-bottom: 1px solid #e6e6e6;

}

.table-search tbody tr:last-of-type {

border-bottom: 2px solid #007fff;

}

.table-display{

border-collapse: collapse;

color: black;

margin: 25px 0;

font-size: 90%;

min-width: 400px;

border-radius: 10px 10px 0 0;

overflow: hidden;

box-shadow: 0 0 20px rgba(0, 0, 0, 0.15);

}

.table-display th{

background-color: #007fff;

color: #ffffff;

text-align: left;

font-weight: bold;

}

.table-display th,

.table-display td {

padding: 12px 15px;

}

.table-display tbody tr {

border-bottom: 1px solid #dddddd;

}

.table-display tbody tr:nth-of-type(even) {

background-color: #f3f3f3;

}

.table-display tbody tr:last-of-type {

border-bottom: 2px solid #007fff;

}

1. **Approach –**

The approach towards this problem statement was fairly simple. First the basic templates for the web pages were created using HTML and the entry fields, buttons, forms, etc were added to the web pages. Styling for the web pages was done by using CSS file to make it more user-friendly. Then to make a local server on which these web pages will be rendered and the functionalities to be carried out, python was used. Python offers Flask module which is used to create a local server. The python code also governs the behaviour of the web pages. It carries out the functions on the backend. It sends/receives data to the client side as per the request generated by the client. The python file then depending on the request, reads and write data to the CSV file. The web pages form the face of the whole web application and the python code forms the main brain of the application.

1. **Coding Practices –**

Python offers Flask module which is used here to form a local server that’ll render web pages. Flask is a lightweight WSGI (Web Server Gateway Interface) web application framework. It is designed to getting started quick and easy, with the ability to scale up to complex applications. The requirement for this particular application was basic so the Flask framework was chosen.

The Flask module carries different functions based on the application route specified for the app (The URL on the webpage) and distinguishes different webpages. The home or “/” route redirects to home function which renders the index.html file. The display route redirects to display-student.html file. The search route redirects to the search-student.html file.

In the Search section of our WebApp, when the user enters a student ID to search a student, the client page (i.e. search-student.html) will generate a request to the server. Now, the requests generated by a client are of two types, 1) GET 2) POST. When a client generates request using GET method, the information about the request is displayed on the URL bar so that we can jump to the response generated by the server. But this leads to insecurity of the data, as sensitive information is displayed on the URL. The GET method is useful when we want to redirect between web pages, but to send sensitive information using GET can be risky. So, to prevent that from happening, POST method is used to generate requests. In POST, the request generated by the client is not displayed on the URL. Thus, privacy is ensured. So when anybody wants to search a student, a POST request is generated. This request is recognized by the server side code and then it searches for the information about the searched student in the CSV file and displays it for the user. Similarly in the Add Student section, the data collected from the user to create a new entry in the CSV file, is sent to the server using POST method.

In all the HTML templates excluding index.html, we display some elements which are not present in the actual HTML code, but they are displayed dynamically on the web page by the server code using “Jinja”. Jinja is a modern and designer-friendly templating language for Python, modelled after Django’s templates. It is fast, widely used and secure with the optional sandboxed template execution environment. Using Jinja, we were able to display a table showing student details of the searched student. It also helped in displaying all the students in tabular format in the display students section. The data read from the CSV by the server side code was directly rendered on the HTML templates. This made the JavaScript files obsolete for this particular application. All the data fetching, displaying and the overall manipulation was handled by the server code itself. That’s why there is no need of any JavaScript file for this. Also, since all the data manipulation is carried out on the server side rather than on the client side, the load on the user’s end is minimum.

Accessing the CSV file was very easy thanks to the csv module available in python. Reading and Writing to the CSV file was a crucial task as it made the database of our application. Thus with a few lines of codes, the desired goal was achieved.

The HTML and CSS files were designed in such a way that they are pleasing to the user as well as carry out certain tasks right on the user end. This made traversing through the webpages very easy. The user is indicated if any mishaps occur during any operations with appropriate messages and prompts.

1. **Screenshots –**

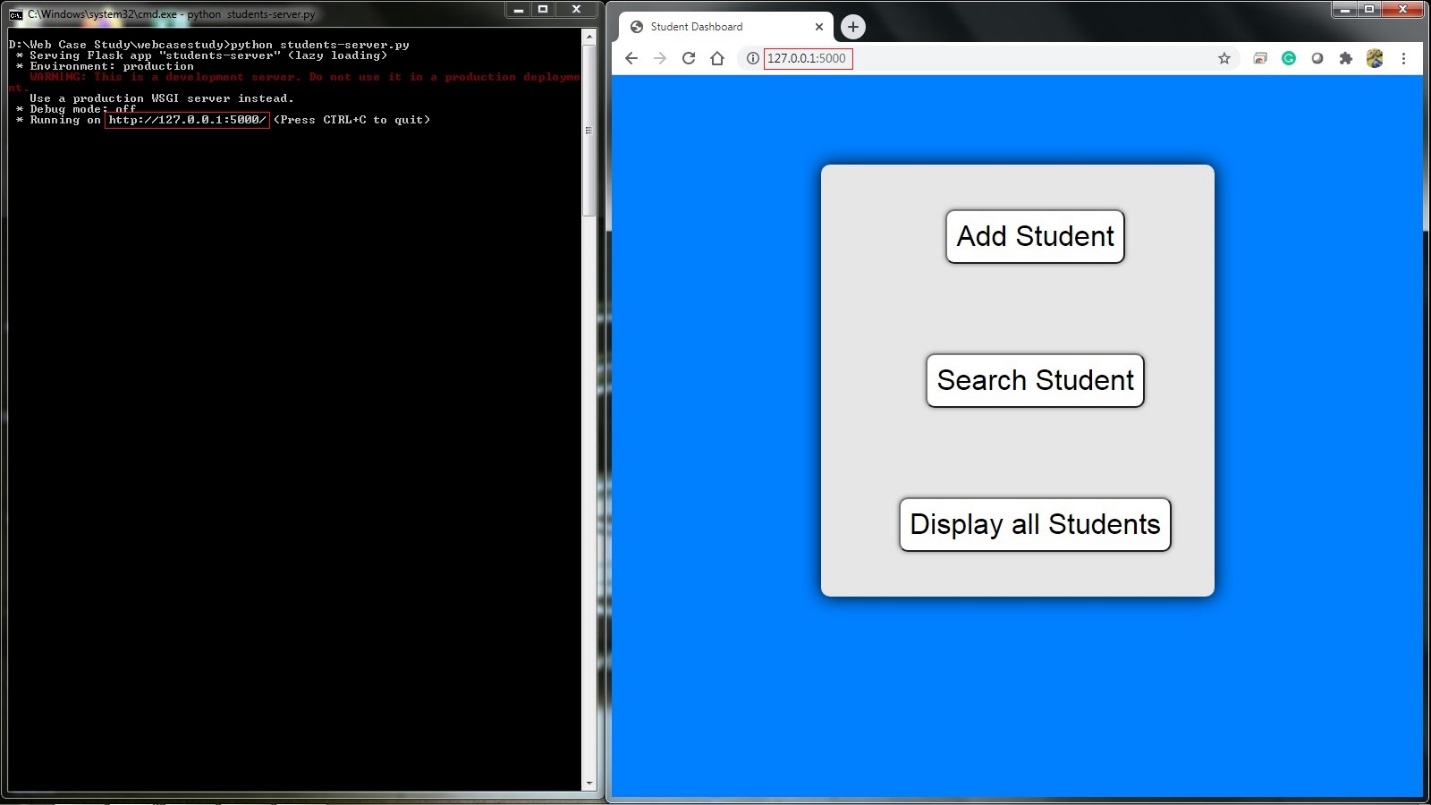


Fig. 1 – Index.html with address port from the server

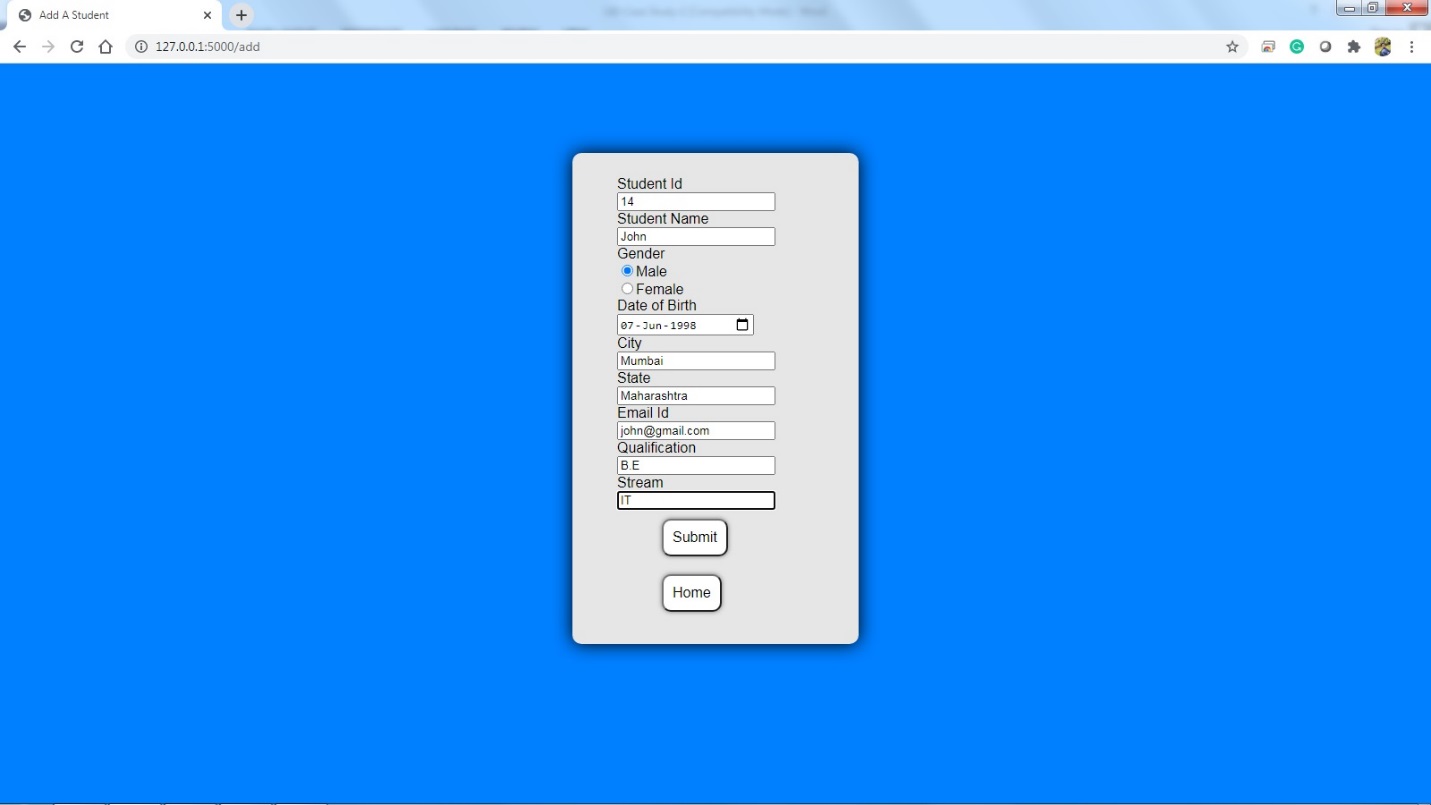


Fig. 2 – add-student.html

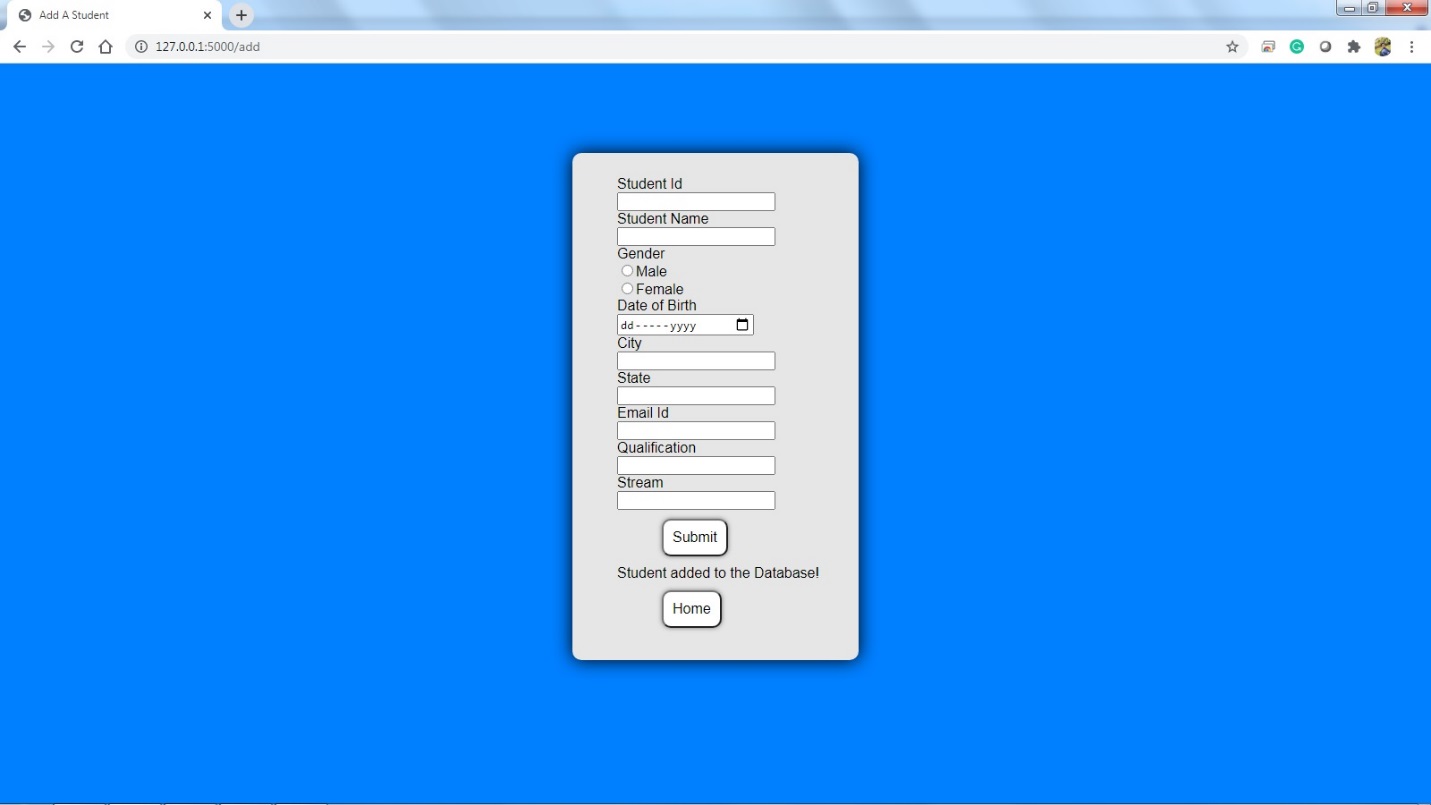


Fig. 3 – After adding a Student

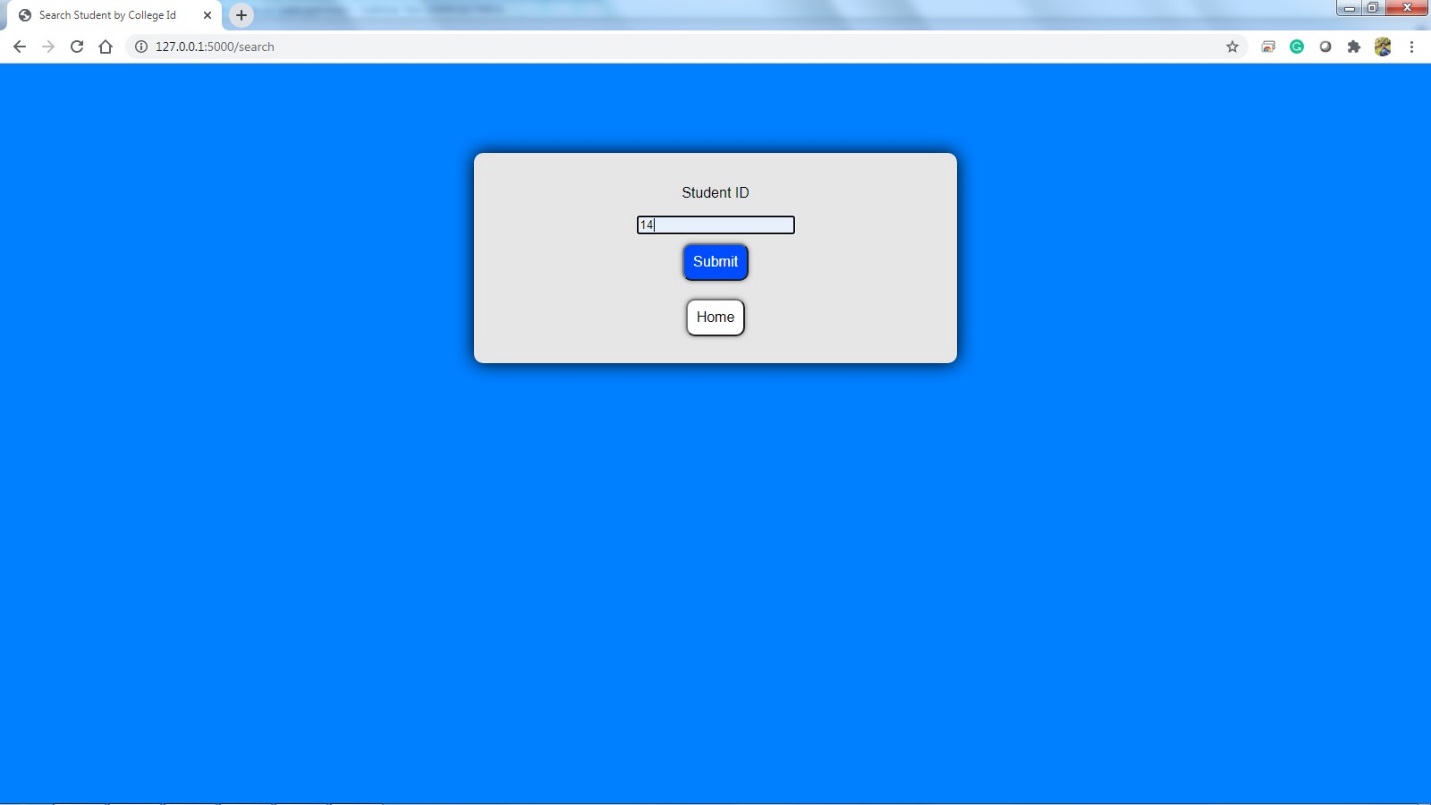


Fig. 4 – search-student.html

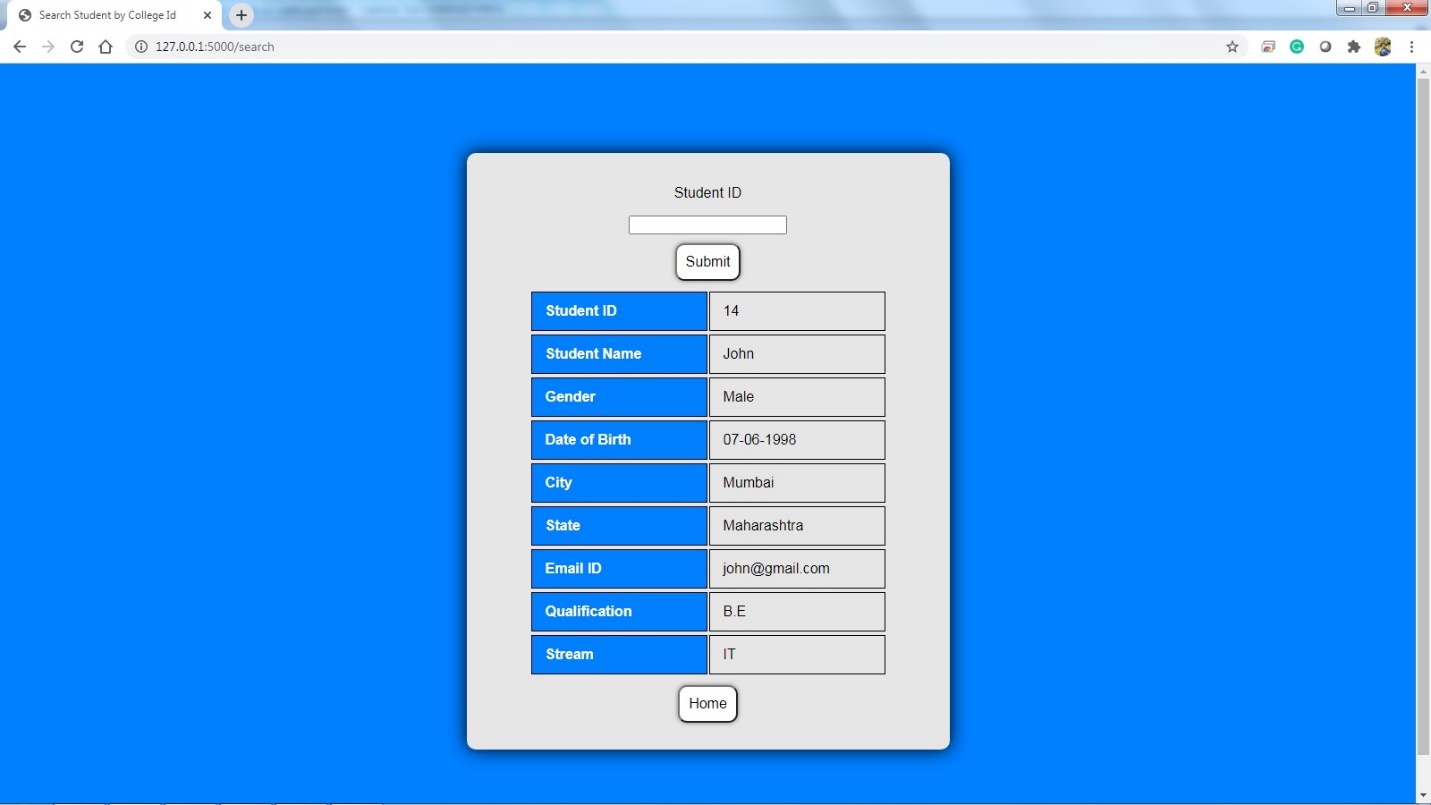


Fig. 5 – After searching a Student

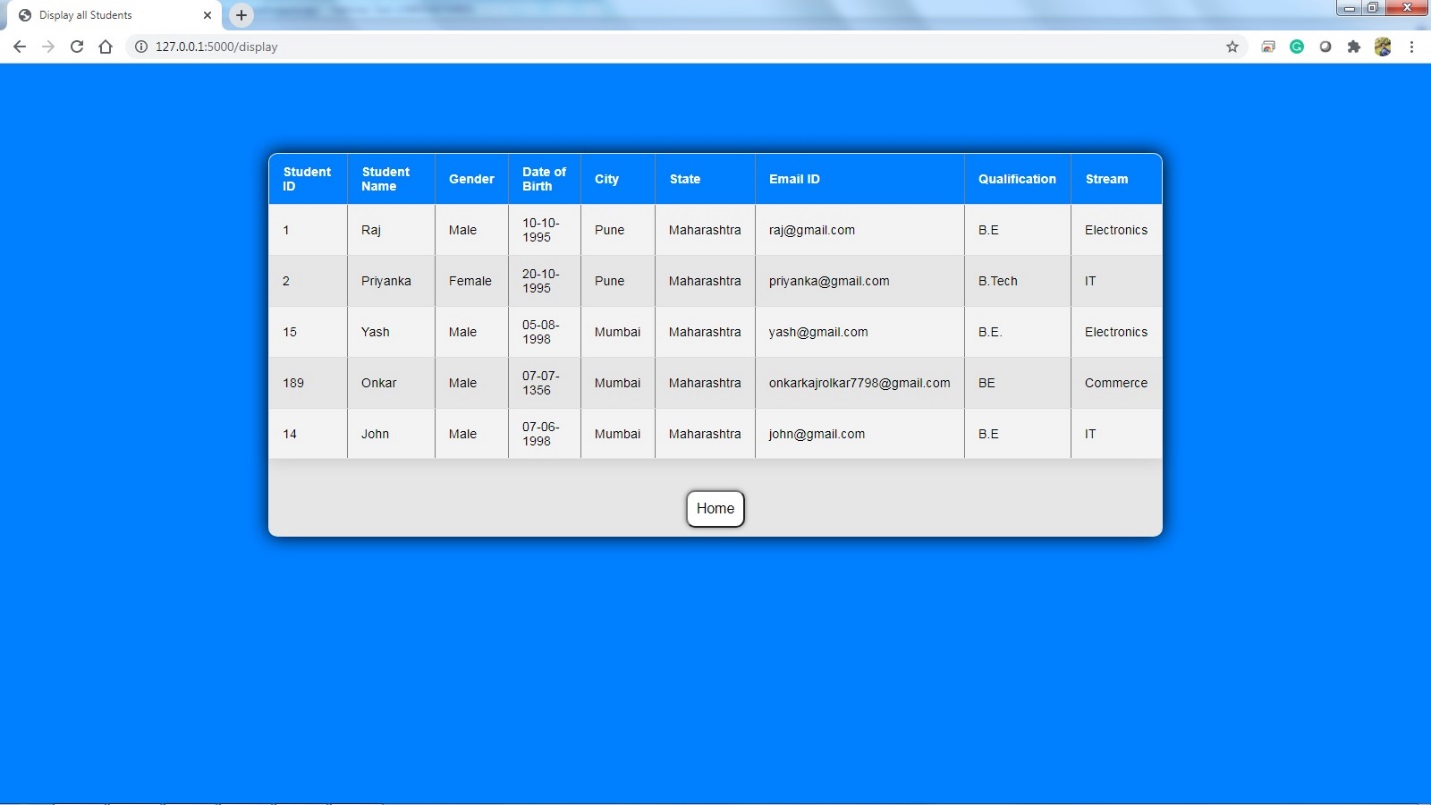


Fig. 6 – display-student.html