KLE Society's

KLE Technological University, Hubballi.

An Internship Project Report on:

ORGAN DONATION AND TRANSPLANT MATCHING SYSTEM

Report on Phase 1 Training

Submitted in partial fulfillment of the requirement for the degree of

BACHELOR OF COMPUTER APPLICATIONS

Submitted by:

Akshata Saunshi

[01FE22BCA032]

Under the guidance of:

Prof. Maheshwari Kittur

DEPARTMENT OF COMPUTER APPLICATIONS

KLE TECHNOLOGICAL UNIVERSITY, HUBBALLI–580031

The academic year 2024-25

1. Training Report & Documentation

Introduction

During the one-month industry training, I gained hands-on experience in Python programming and SQLite database management. The training focused on essential concepts such as writing Python scripts, handling data structures, and working with file operations. I also learned how to create and manage an SQLite database, including creating tables, inserting data, and executing queries for data retrieval and manipulation.

Additionally, I gained practical knowledge in **Django installation and setup**, including creating necessary files and folders, setting up models, and managing database migrations. The training was divided into two phases of 15 days each, allowing for a structured learning approach and real-time application of acquired skills.

What is python?

Python is a high-level programming language that is easy to read and write, making it great for beginners. It is used in web development, data science, automation, and AI. Python has a simple syntax and runs code line by line, making it easy to debug. It is versatile, supporting different coding styles and offering many built-in tools and libraries. With its flexibility and strong community support, Python is one of the most popular programming languages today.

Example:-

```
# Print a message
Input: print("Hello, Python!")
Output: Hello, Python!
```

Add two numbers Input:

a = 5

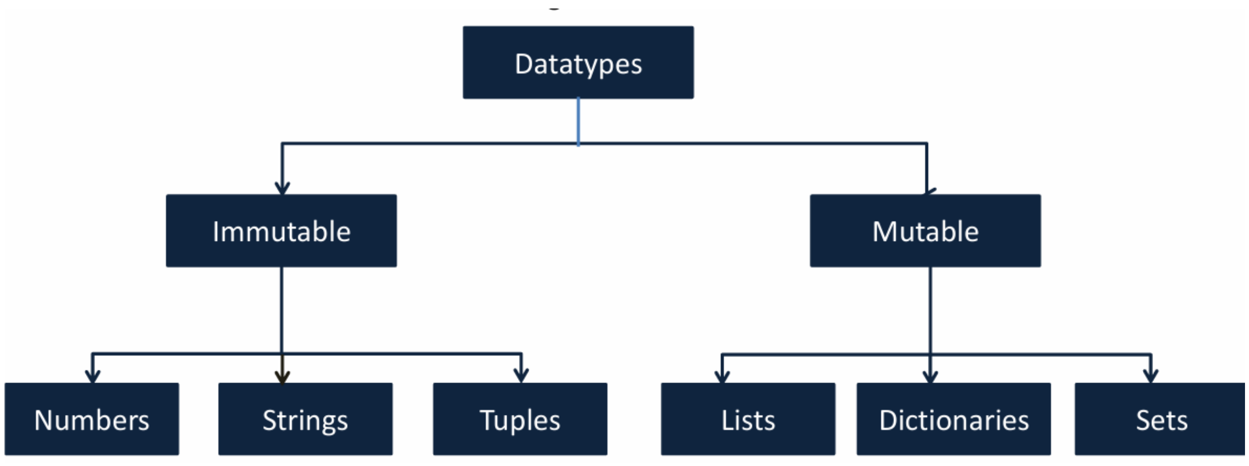
b = 3

sum = a + b

Output: The sum is: 8

Datatypes:

Python is a loosely typed language. Therefore, no need to define the datatype of variables No need to declare variables before using them



Types of Operators

Arithmetic Operators in Python:-

Python provides arithmetic operators to perform mathematical operations.

Here are the main types:

1. Addition(+),
2. Subtraction(-),
3. Multiplication(*),
4. Division(/),

5. Modulus(%),
6. Power(**).

Example:

Input:

```
a = 30

b = 20

print("Sum is: ", a+b)
print("Difference is: ", a-b)

print("Product is: ", a*b)

print("Division is: ", a/b)

print("Modulus is: ", a%b)

print("Power is: ", a**2)
```

Output:

```
Sum is: 50

Difference is: 10

Product is: 600

Division is: 1.5

Modulus is: 10

Power is: 900
```

Relational Operators in Python:-

Relational operators (also called comparison operators) are used to compare two values. They return True or False based on the condition.

They are: Equal to(==), Not equal to(!=), Greater than(>), Less than(=), Less than or equal to(<=).

Example:

```
Input:   a = "abc"

             b = "abc"
             print("Less Than:", a<b)
             print("Greater Than:", a>b)
             print("Less Than Equal:", a<=b)
             print("Greater Than Equal:", a>=b)
             print("Equal:", a==b)
             print("Not Equal:", a!=b)
```

```
Output: Less Than: False
           Greater Than: False
           Less Than Equal: True
           Greater Than Equal: True
           Equal: True
           Not Equal: False
```

Conditional Statements in Python:-

Conditional statements allow a program to make decisions based on conditions. Python provides

different types of conditional statements:

- if-else Statement:

The if-else statement runs one block of code if the condition is True, and another if the condition is False.

Example:

Input: a = 30 b

= 40

min = a if a<b else b

print("Minimum is:", min)

Output: Minimum is: 30

- Nested if Statement:

A nested if statement is an if statement inside another if. It allows multiple levels of conditions.

Example:

Input: num = 10

if num > 0:

print("Number is positive") if

num % 2 == 0:

print("Number is even")

Output: Number is positive.

Number is even

- Multi-line if-else (elif):

When there are multiple conditions, we use elif (else if).

Example:

Input: marks = 85

```
if marks >= 90:
    print("Grade: A")
elif marks >= 80:
    print("Grade: B")
elif marks >= 70:
    print("Grade: C")
else:
    print("Grade: D")
```

```
Output:  Grade: B
```

Looping Statements in Python:-

Loops in Python are used to execute a block of code multiple times until a condition is met.

Python has two main types of loops:

- for Loop:

The for loop is used to iterate over a sequence (list, tuple, string, range, etc.).

Example:

```
Input:  fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)
```

```
Output:   apple
  banana
```

cherry

- while Loop:

The while loop keeps running as long as the condition is True.

Example:

```
Input: num = 1
while num <= 5:
        print(num)
        num += 1
```

```
Output:    1
    2
    3
    4
    5
```

Objectives

- To understand and implement Python programming concepts effectively.
- To understand how to create and manage an SQLite database, including making tables, adding data, and retrieving information.
- To install and set up Django, including creating necessary files and folders.
- To practice database migrations and connect Python with SQLite.

Methodologies Used

- Hands-on coding and project-based learning for better understanding.

- Practical implementation of database management, query execution, and data handling.
- Use of industry-relevant tools like Visual Studio Code, SQLite Database Browser, and Django Admin Panel for effective development.

1. Technical Skills Acquired

Skills Developed

- **Python Scripting:** Writing efficient Python programs to handle data processing and automation tasks.
- **Data Structures & File Handling:** Managing lists, dictionaries, and file operations to store and retrieve data efficiently.
- **SQLite Database Management:** Creating databases, defining tables, inserting records, and executing SQL queries for data retrieval and manipulation.
- **Django Framework:** Setting up Django projects, creating models, handling views, and configuring URLs for seamless web application development.
- **Project Organization:** Structuring Django applications by creating necessary files and folders for a well-maintained codebase.
- **Development Tools:** Utilizing Visual Studio Code for coding, SQLite Database Browser for managing databases, and Django Admin Panel for backend operations.

Demonstrations

- **Code Samples:**
    - **Basic Python Web Output:** Displaying **"Hello, World!"** in the browser using Django views and templates.

- **Member Details Form:** Creating a simple form where users can enter member details (name, email, etc.), store the data in an SQLite database, and display it on another page.

- **Student Data Entry:** Implementing a form to collect student details (name, date of birth, email), save the submitted information in an SQLite database, and display the stored data on a separate page.

- **Screenshots & Project Files:**
  - Attached examples of Django view, template, and URL configuration for displaying "Hello, World!" in the browser.

3. Code Snippets and Outputs

Code Samples

Urls.py

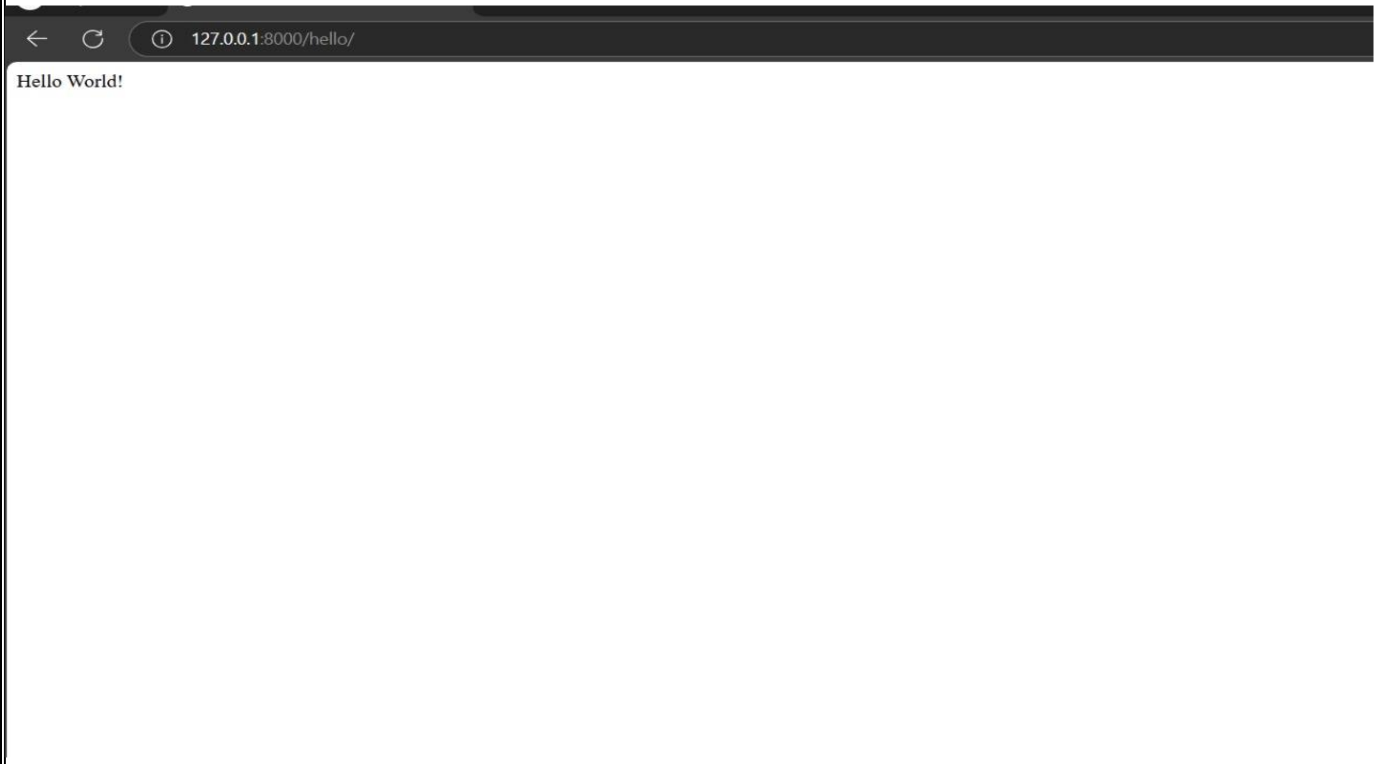
```
from django.contrib import admin from
django.urls import include,path

urlpatterns = [
path('admin/', admin.site.urls),
path('',include('members.urls')),
]
```

views.py

```
from django.shortcuts import render

# Create your views here.
from django.http import HttpResponse

def hello(request):
    return HttpResponse("Hello world!")
```

(members)Urls.py

```
from django.urls import path
from  .import views

urlpatterns = [
    path('hello/',views.hello,name='hello'),]
```



127.0.0.1:8000/hello/

```
Hello World!
```

- Attached examples of Django view, template, and URL configuration for displaying "member details " in the browser.

Views.py

```
def datapush(request):
if request.method=='POST':
firstname=request.POST.get('firstname')
lastname=request.POST.get('lastname')
x=MemberInfo.objects.create(firstname=firstname,lastname=lastname)
return redirect('members_data')
return render(request,'datapush.html')
def members_data(request):
mymembers = MemberInfo.objects.all()
return render(request, 'members_data.html', {'mymembers': mymembers})
```

datapush.html

```
<html>
<h1>Enter Employee Information</h1>
<form method="POST">  <!-- Fixed from <from> to <form> -->
{% csrf_token %}
<label for="firstname">First Name:</label>
<input type="text" id="firstname" name="firstname" required>
<br><br>
<label for="lastname">Last Name:</label>
<input type="text" id="lastname" name="lastname" required>
<br><br>
```

```
<button type="submit">Submit</button>
</form>
</html>
```

Member_data.html

```
<html>
<h1>Members</h1>
<ul>
{% for x in mymembers %}
<a href="/member_detail/{{x.id}}"><li>{{x.firstname}}{{x.lastname}}</li></a>
{% endfor %}
</ul>
</html>
```

Models.py

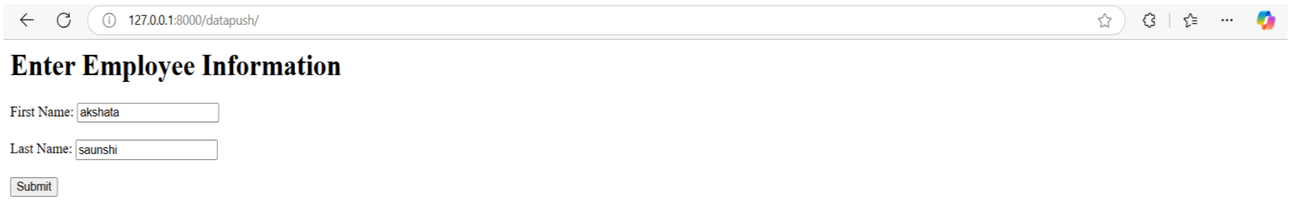
```
from django.db import models
# Create your models here.
class MemberInfo(models.Model):
firstname=models.CharField(max_length=255)
lastname=models.CharField(max_length=225)
```

urls.py

```
from django.contrib import admin
from django.urls import include,path
urlpatterns = [path('admin/', admin.site.urls),
path('',include('members.urls')),]
```

(members)urls.py

```
from django.urls import path
from . import views
from members import views
urlpatterns = [
path('datapush/',views.datapush,name='datapush'),
path('members_data/', views.members_data, name='members_data'),
]
```

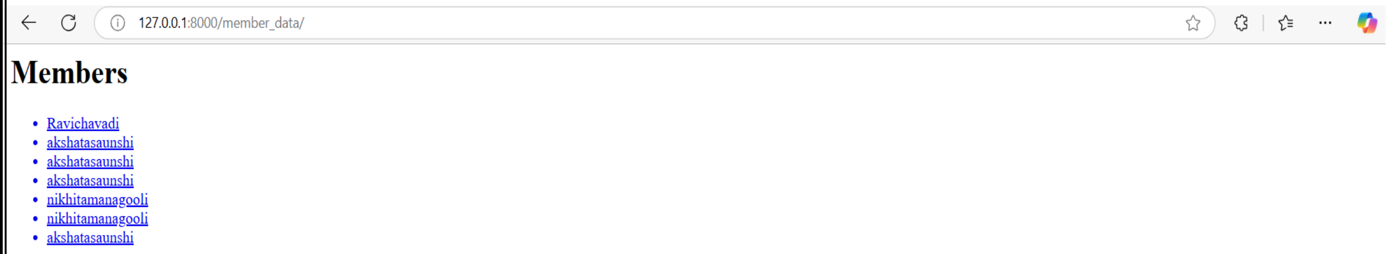


127.0.0.1:8000/datapush/

Enter Employee Information

First Name: akshata

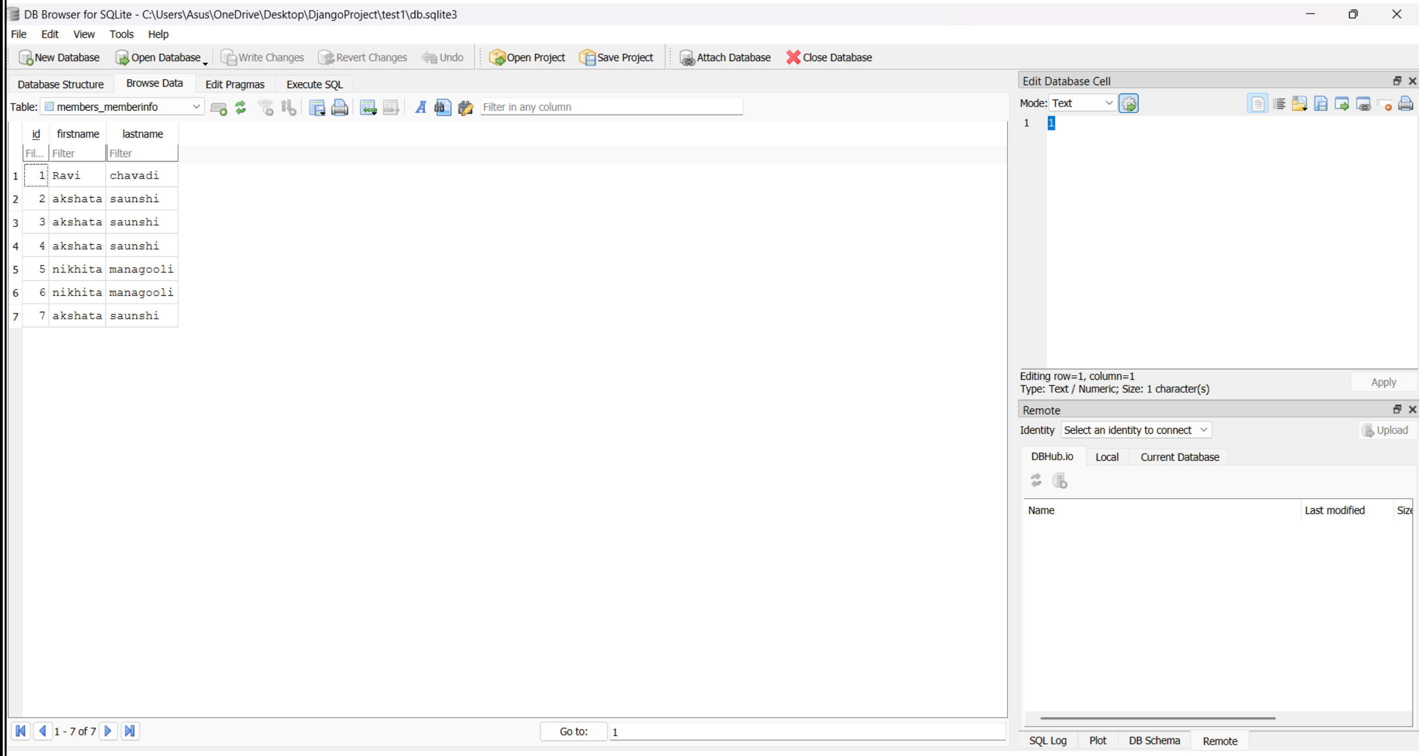
Last Name: saunshi



127.0.0.1:8000/member_data/

Members

- Ravichavadi
- akshatasaunshi
- akshatasaunshi
- akshatasaunshi
- nikhitamanagooli
- nikhitamanagooli
- akshatasaunshi



DB Browser for SQLite - C:\Users\Asus\OneDrive\Desktop\DjangoProject\test1\db.sqlite3

Table: members_memberinfo

	id	firstname	lastname
1	1	Ravi	chavadi
2	2	akshata	saunshi
3	3	akshata	saunshi
4	4	akshata	saunshi
5	5	nikhita	managooli
6	6	nikhita	managooli
7	7	akshata	saunshi

1 - 7 of 7

Editing row=1, column=1
Type: Text / Numeric; Size: 1 character(s)

- Attached examples of Django view, template, and URL configuration for displaying "student details " in the browser.

Views.py

```
from django.shortcuts import render, redirect
from .forms import StudentForm
from .models import Student

def student_form(request):
if request.method == "POST":
form = StudentForm(request.POST) if
form.is_valid():
form.save()
return redirect('student_list')  # Redirect to list page else:
form = StudentForm()
return render(request, 'students/student_form.html', {'form': form})

def student_list(request):
students = Student.objects.all()
return render(request, 'students/student_list.html', {'students': students})
```

models.py

```
from django.db import models
class Student(models.Model):
firstname = models.CharField(max_length=100)
lastname = models.CharField(max_length=100)
email = models.EmailField(unique=True)
date_of_birth = models.DateField()
```

```
def __str__(self):
return f"{self.firstname} {self.lastname}"
from django.db import connection
with connection.cursor() as cursor:
cursor.execute("SELECT name FROM sqlite_master WHERE type='table';")
tables = cursor.fetchall()
for table in tables:
print(table[0])
```

forms.py

```
from django import forms
from .models import Student
class StudentForm(forms.ModelForm):
class Meta:
model = Student
fields = ['firstname', 'lastname', 'email', 'date_of_birth']
widgets = {
   'date_of_birth': forms.DateInput(attrs={'type': 'date'})
}
```

Urls.py

```
from django.urls import path
from .views import student_form, student_list
urlpatterns = [
path('', student_form, name='student_form'),
path('students/', student_list, name='student_list'),
]
```

(Student_project)urls.py

```
from django.contrib import admin
from django.urls import include, path
urlpatterns = [
path('admin/', admin.site.urls),
path('', include('students.urls')),
]
```

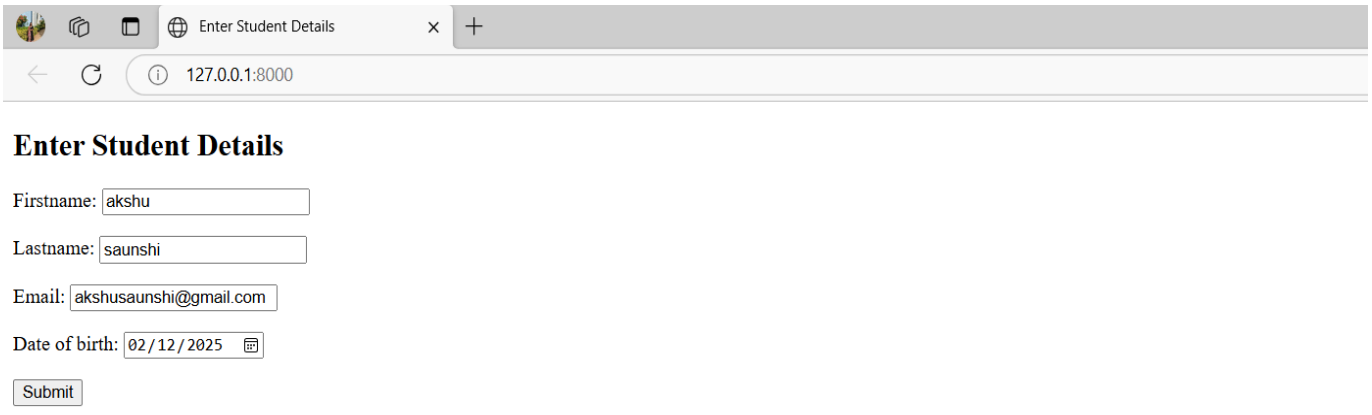
Student_form.html

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>Enter Student Details</title>
</head>
<body>
<h2>Enter Student Details</h2>
<form method="post">
{% csrf_token %}
{{ form.as_p }}
<button type="submit">Submit</button>
</form>
</body>
</html>
```

Student_list.html

```
<!DOCTYPE html>
```

```
 <html lang="en">
<head>
<title>Entered Student Details</title>
</head>
<body>
<h2>Entered Student Details</h2>
<table border="1">
<tr>
   <th>Firstname</th>
   <th>Lastname</th>
   <th>Email</th>
   <th>Date of Birth</th>
</tr>
   {% for student in students %}
<tr>
   <td>{{ student.firstname }}</td>
   <td>{{ student.lastname }}</td>
   <td>{{ student.email }}</td>
   <td>{{ student.date_of_birth }}</td>
</tr>
   {% endfor %}
  </table>
  </body>
  </html>
```



Enter Student Details

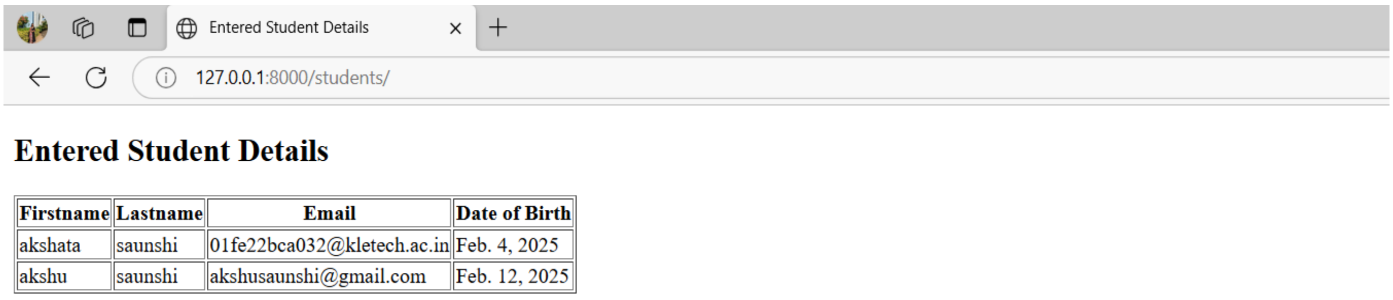
Firstname: akshu

Lastname: saunshi

Email: akshusaunshi@gmail.com

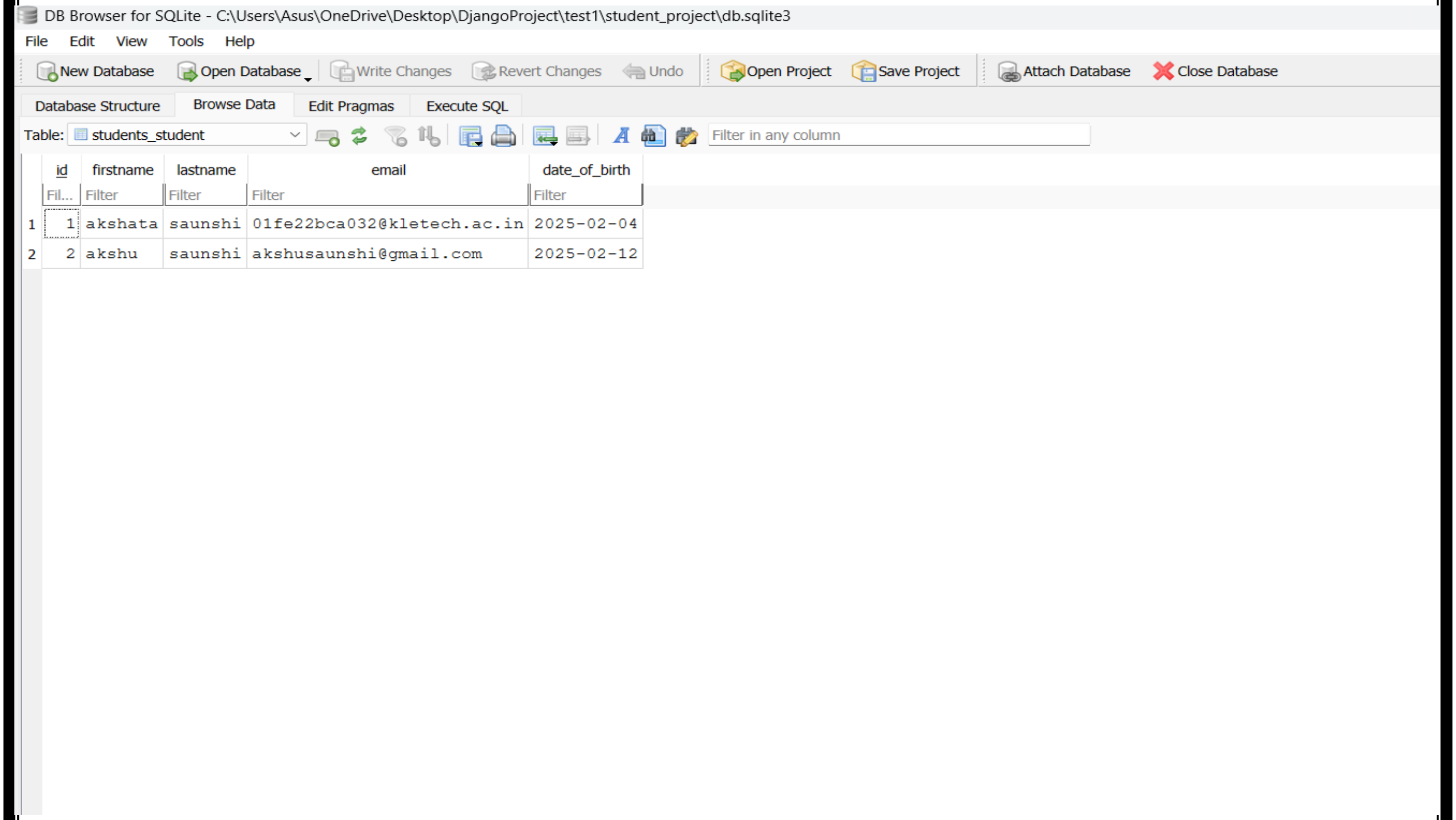
Date of birth: 02/12/2025

Submit



Entered Student Details

Firstname	Lastname	Email	Date of Birth
akshata	saunshi	01fe22bca032@kletech.ac.in	Feb. 4, 2025
akshu	saunshi	akshusaunshi@gmail.com	Feb. 12, 2025



	id	firstname	lastname	email	date_of_birth
1	1	akshata	saunshi	01fe22bca032@kletech.ac.in	2025-02-04
2	2	akshu	saunshi	akshusaunshi@gmail.com	2025-02-12

4. Problem-Solving & Innovation

Challenges Faced & Solutions

- **Issue:** Difficulty in setting up Django and connecting it with SQLite.
  - **Solution:** Followed Django documentation and used migration commands (makemigrations & migrate) to set up the database properly.
- **Issue:** Form data was not saving in the database.
  - **Solution:** Checked form validation, used POST method correctly, and ensured models were properly linked.
- **Issue:** Errors in URL routing while navigating between pages.
  - **Solution:** Defined correct path() functions in urls.py and used {% url %} in templates for proper linking.
- **Issue:** Displaying saved data dynamically on the webpage.
  - **Solution:** Used Django ORM to fetch data with Model.objects.all()

5. Final Conclusion

The industry training was a valuable learning experience that significantly enhanced my skills in Python, Django, and SQLite. Through hands-on practice and project-based learning, I gained in-depth knowledge of developing dynamic web applications, managing databases, and handling form data efficiently. The structured training approach enabled me to apply core concepts such as database migrations, URL routing, and form validation, strengthening both my technical expertise and problem-solving abilities.

During the training, I overcame challenges such as setting up Django, ensuring seamless data storage and retrieval, and resolving URL navigation errors. By implementing Django's ORM for database management and optimizing form handling, I improved the functionality and user experience of my projects.

This experience has greatly boosted my confidence in web development, equipping me with the necessary skills to build scalable and interactive applications. With a solid foundation in Python, Django, and database management, I am well-prepared to tackle more complex development projects and contribute effectively to the field of web development.