Vivekanand Education Society's

Institute of Technology

(Autonomous Institute Affiliated to University of Mumbai, Approved by AICTE &

Recognized by Govt. of Maharashtra, NAAC accredited with 'A' Grade)

Department of Information

Technology CERTIFICATE

This is to certify that **Kaushik Kotian** of **D15B** having Roll No_**30**__ in semester **VI,** have successfully completed necessary experiments in the **Web X.0 Lab** under my supervision in **VES Institute of Technology** during the academic year **2023-2024.**

Lab Teacher
Mrs. Sukanya Roychowdhury

Head of Department
Dr. Shalu Chopra

Signature: Signature:

Course: Title: Web Lab (Web X.0) **Year:-2023-24**

Branch: INFT. (VI) Class: D15B

Faculty In-charge: Mrs. Sukanya Roychowdhury

Lab In-charge: Mrs. Sukanya Roychowdhury

Email : sukanya.roychowdhury@ves.ac.in

Lab Outcomes (LO): At the end of the course the student will be able to:

LO1	Open Source Tools for Web Analytics and Semantic Web.
LO2	Programming in TypeScript for designing Web Applications.
LO3	AngularJS Framework for Single Page Web Applications.
LO4	AJAX for Rich Internet Applications.
LO5	REST API and MongoDB for Frontend and Backend Connectivity.
LO6	Flask Framework for building web applications.

INDEX

Module	Sr. No	Topics Of Coverage	LO's	DOP	DOS	Grade	Sign
1	1	Study Any 1 tool in each 1. Study web analytics using open source tools like Matomo, Open Web Analytics, AWStats, Countly, Plausible, Google Analytics.	LO1	23/1/24	6/2/24	O	
2	2	Small code snippets for programs like Hello World, Calculator using TypeScript.	LO2	6/2/24	20/2/24	O	
	3	Inheritance example using TypeScript					
	4	Access Modifiers example using TypeScript					
3	5	Create a simple HTML "Hello World" Project using AngularJS Framework and apply ng-controller, ng-model and expressions.	LO3	20/2/24	5/3/24	O	
	6	Perform Events and Validations in AngularJS					
4	7.	Build a RESTful API using MongoDB. Implementation of all CRUD operations in MongoDB	LO5	5/3/24	12/3/24	O	
5	9	Case study on Rich Internet Application-AJAX	LO4	12/3/24	19/3/24	O	
6	10	To create a Website using FLASK. Show routing and templating in Flask.	LO6	19/3/24	26/4/24	O	

Subject In-charge

Web X.0 Lab

Experiment No. 1

Aim: Exploring Web Analytics Tools

Introduction: Web analytics plays a pivotal role in understanding and optimizing website performance, aiding businesses in making informed decisions. This study delves into the significance of web analytics and focuses on Plausible, a GDPR-compliant and privacy-friendly alternative to traditional tools.

Understanding Web Analytics: Web analytics involves tracking and analyzing visitor behavior on a website, encompassing various aspects such as traffic sources, page views, conversion rates, and more. The data collected serves as a valuable resource for making informed decisions in areas like customer relationship management (CRM), product analytics, social media analytics, and marketing analytics.

Importance of Web Analytics:

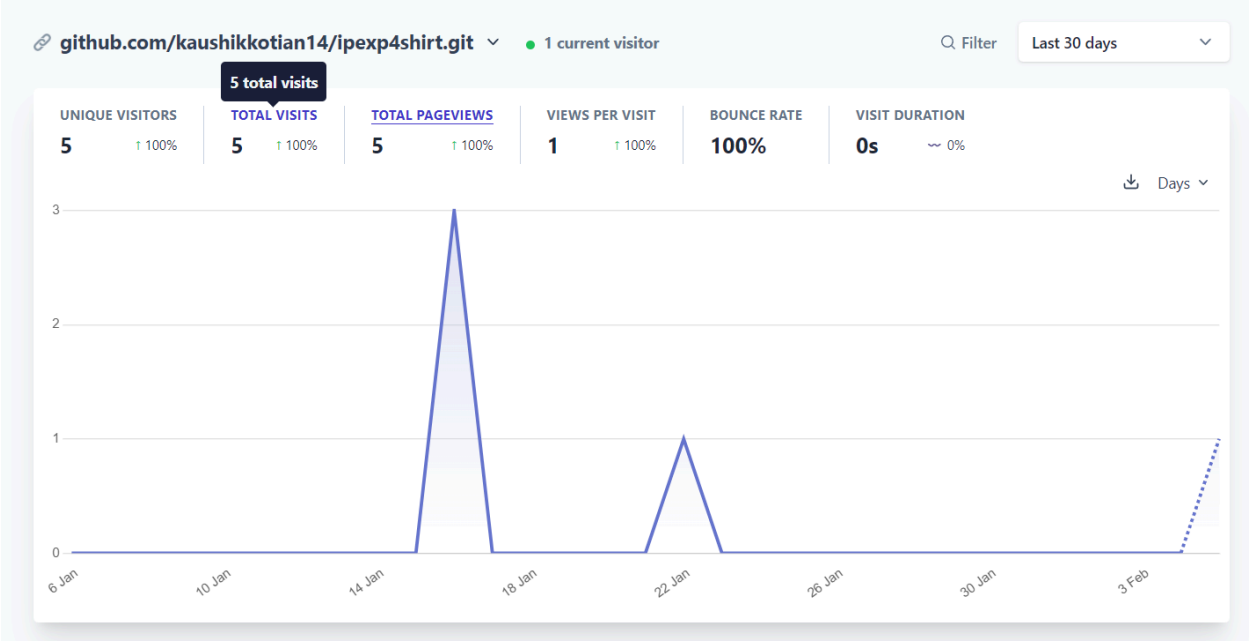
Access to Relevant Data: Google Analytics provides valuable insights, helping businesses discover hidden trends and make data-driven decisions.
Audience Understanding: Improving user experience requires a deep understanding of the website audience, including their devices, language, and preferences.
ROI Tracking: Web analytics aids in assessing the return on investment by monitoring the performance of various campaigns and strategies.
SEO Improvement: Identifying issues like slow loading and browser compatibility, web analytics contributes to enhancing a website's search engine optimization (SEO).
PPC Performance Optimization: Web analytics tools, including Plausible, enhance the performance of Google ads through advanced remarketing capabilities and tracking of ecommerce transactions.
Identifying Pain Points: Businesses can use web analytics to identify and rectify pain points in the user experience, ensuring a seamless journey for visitors.

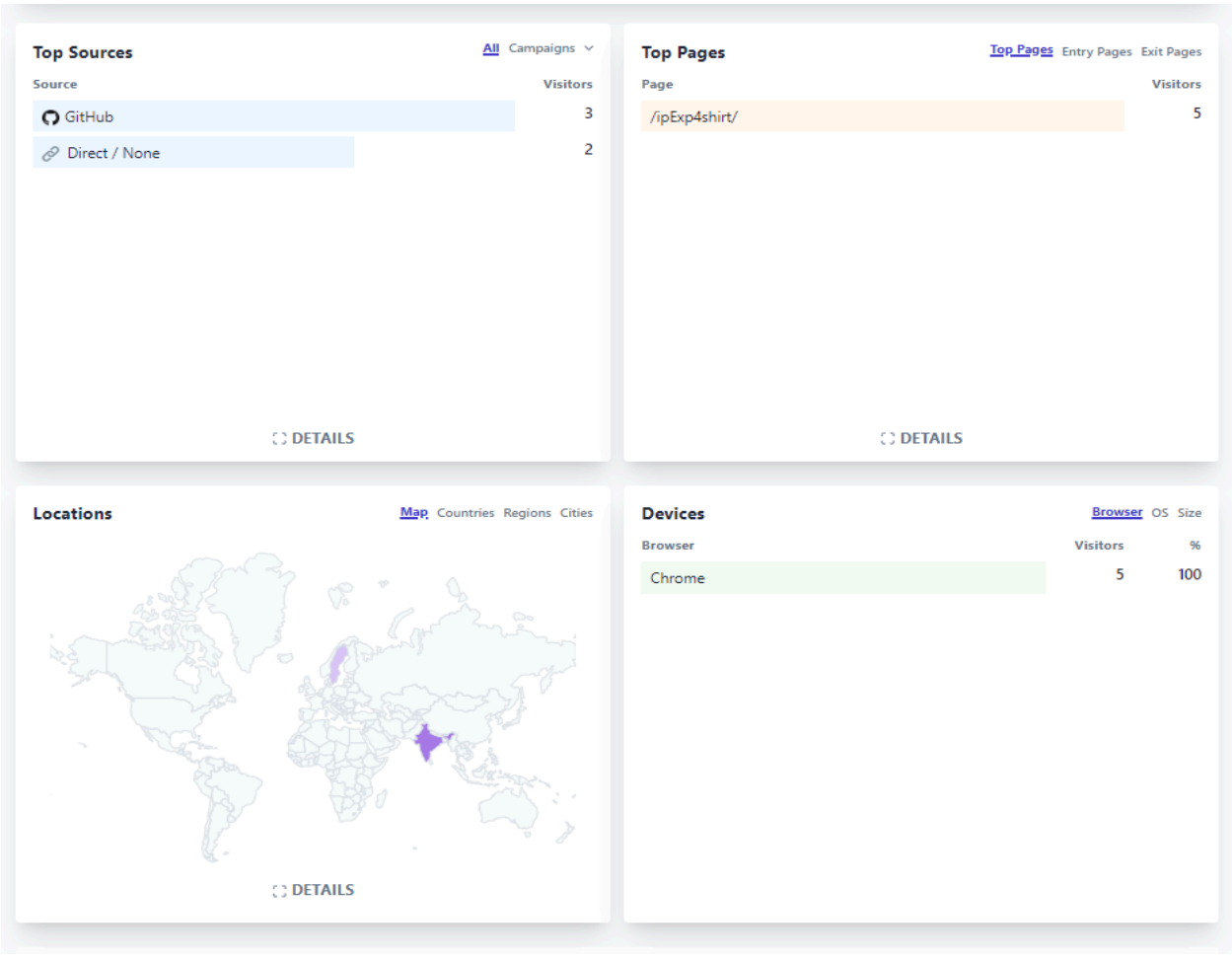
Conversion Funnel Optimization: Setting goals and tracking user actions with web analytics helps in optimizing conversion funnels for essential business objectives.
Data Reporting: Web analytics refines and optimizes data, providing businesses with visually appealing representations for better comprehension.

Example Tools:

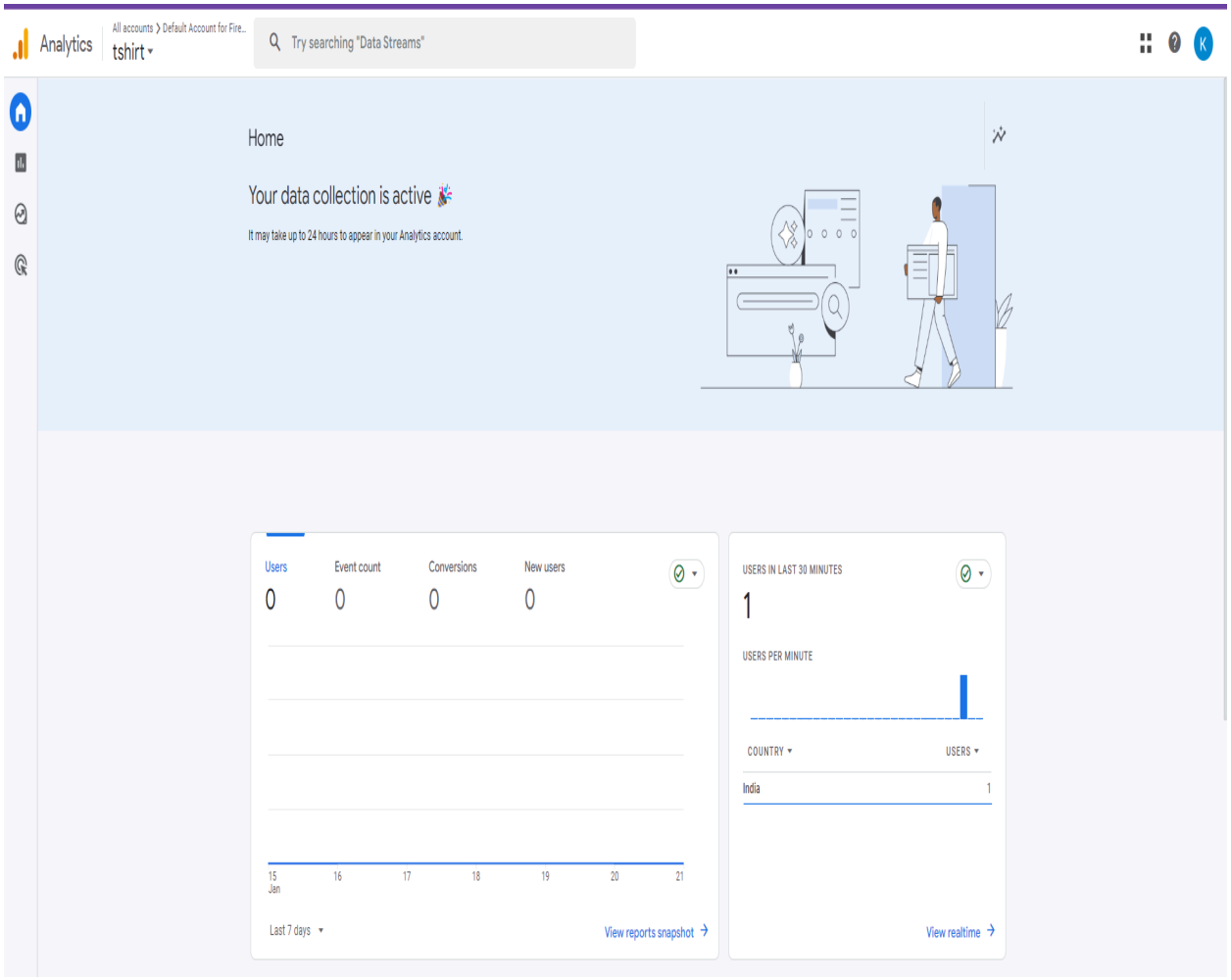
Google Analytics: A comprehensive platform tracking various metrics related to website traffic, behaviors, and conversions.
Plausible: A lightweight, GDPR-compliant, and privacy-focused alternative that doesn't use cookies and allows self-hosting.
Optimizely: A customer experience and A/B testing platform for optimizing online experiences.
Kissmetrics: A customer analytics platform offering deep insights into customer behavior and enhancing website and marketing campaigns.
Crazy Egg: A tool tracking customer clicks on a page, providing valuable insights into visitor interactions through heatmaps and user session recordings.

Plausible:


github.com/kaushikkotian14/ipexp4shirt.git
1 current visitor
Filter
Last 30 days
5 total visits
UNIQUE VISITORS
5
↑ 100%
TOTAL VISITS
5
↑ 100%
TOTAL PAGEVIEWS
5
↑ 100%
VIEWS PER VISIT
1
↑ 100%
BOUNCE RATE
100%
VISIT DURATION
0s
〰 0%
Days
3
2
1
0
6 Jan
10 Jan
14 Jan
18 Jan
22 Jan
26 Jan
30 Jan
3 Feb


Top Sources
All
Campaigns
Source
Visitors
GitHub
3
Direct / None
2
DETAILS
Top Pages
Top Pages
Entry Pages
Exit Pages
Page
Visitors
/ipExp4shirt/
5
DETAILS
Locations
Map
Countries
Regions
Cities
DETAILS
Devices
Browser
OS
Size
Browser
Visitors
%
Chrome
5
100

Google Analytics:



Conclusion:

In the dynamic landscape of web analytics, understanding the importance of data-driven decisions is crucial for businesses. Plausible emerges as a privacy-friendly alternative, aligning with the evolving expectations for secure and transparent web analytics.

EXPERIMENT NO.:2

Aim:

1.Small code snippets for programs like Hello World, Calculator using TypeScript.

2.Inheritance example using TypeScript

3.Access Modifiers example using TypeScript

1. Calculator.ts:

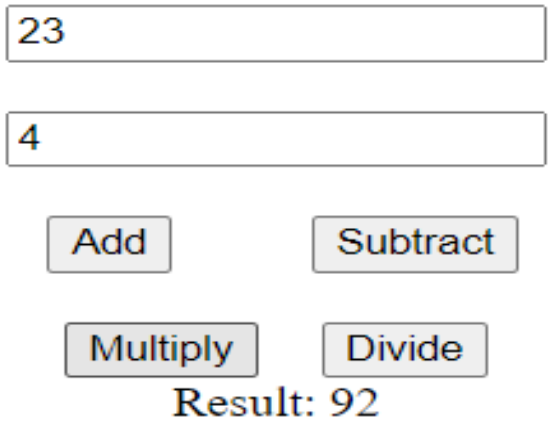
Html code:

```
<!DOCTYPE html>
<html>
<head>
   <title>Calculator</title>
</head>
<body align="center" >
   <br><br><br><br><br><br><br>
   <input id="num1" type="number" placeholder="Number 1"><br><br>
   <input id="num2" type="number" placeholder="Number 2"><br><br>
    <button id="add">Add</button>          
   <button id="subtract">Subtract</button><br><br>
   <button id="multiply">Multiply</button>     
   <button id="divide">Divide</button>
   <div id="result"></div>

   <script src="app.js"></script>
</body>
</html>
```

TypeScript code:

```
window.onload = () => {
    const num1 = document.getElementById("num1") as HTMLInputElement;
    const num2 = document.getElementById("num2") as HTMLInputElement;
    const addButton = document.getElementById("add") as HTMLButtonElement;
    const subtractButton = document.getElementById("subtract") as
HTMLButtonElement;
    const multiplyButton = document.getElementById("multiply") as HTMLButtonElement;
    const divideButton = document.getElementById("divide") as HTMLButtonElement;
    const resultDiv = document.getElementById("result");

    function add(a: number, b: number): number {
        return a + b;
    }

    function subtract(a: number, b: number): number {
        return a - b;
    }

    function multiply(a: number, b: number): number {
        return a * b;
    }

    function divide(a: number, b: number): string | number {
        if (b === 0) {
            return "Cannot divide by zero";
        }
        return a / b;
    }

        function performOperation(operation: (a: number, b: number) => string | number) {
        const a = parseFloat(num1.value);
        const b = parseFloat(num2.value);
        const result = operation(a, b);
        resultDiv!.innerText = `Result: ${result}`;
    }
```

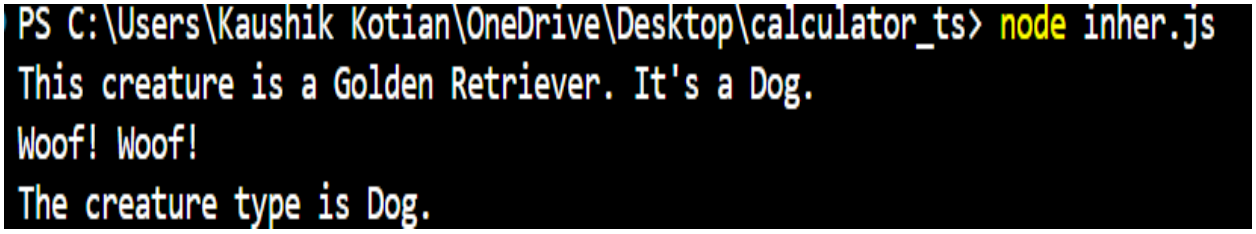
```
    addButton.addEventListener("click", () => performOperation(add));
    subtractButton.addEventListener("click", () => performOperation(subtract));
    multiplyButton.addEventListener("click", () => performOperation(multiply));
    divideButton.addEventListener("click", () => performOperation(divide));
};
```



2. Inheritance.ts:

```
interface Animal {
    makeSound(): void;
}

class Creature {
    protected name: string;
    private creatureType: string;
    constructor(name: string, creatureType: string) {
        this.name = name;
        this.creatureType = creatureType;
    }

    public display(): void {
        console.log(`This creature is a ${this.name}. It's a ${this.creatureType}.`);
    }

    protected getCreatureType(): string {
        return this.creatureType;
```

```
    }
}

class Dog extends Creature implements Animal {
    constructor(name: string) {
        super(name, "Dog");
    }

    public makeSound(): void {
        console.log("Woof! Woof!");
    }

    public showCreatureType(): void {
        console.log(`The creature type is ${this.getCreatureType()}.`);
    }
}

let myDog: Dog = new Dog("Golden Retriever");
myDog.display();
myDog.makeSound();
myDog.showCreatureType();
```



EXPERIMENT NO.:3

Aim: To study and implement angularjs.

1. Create a simple html hello world project using angular js framework and apply ng-controller, ng-model and expressions

```
<!DOCTYPE html>

<html>

<head>

    <title>Hello World Project</title>

    <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js">
</script>

</head>

<body ng-app="helloWorldApp">

    <div ng-controller="HelloWorldController">

        <input type="text" ng-model="name" placeholder="Enter your
name">

        <p>Hello, {{name}}!</p>

    </div>

    <script>

        var app = angular.module('helloWorldApp', []);

        app.controller('HelloWorldController', function($scope) {

            $scope.name = 'World';

        });
```

```
    </script>
</body>
</html>
```

Hello, Kaushik!

2. Events and form validations in angular js.Create functions and add events, add html validators and use $valid property of angular.js:

```
<!DOCTYPE html>
<html>
<head>
    <title>Student Registration Form</title>
    <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js">
</script>
    <style>
        body {
            font-family: Arial, sans-serif;
            background-color: #f4f4f4;
        }
```

```
.container {
    display: flex;
    justify-content: center;
    align-items: flex-start;
    padding: 20px;
    flex-wrap: wrap;
}

form, .preview {
    margin: 10px;
    padding: 20px;
    border: 1px solid #ccc;
    border-radius: 5px;
    background-color: #fff;
}

form {
    border-color: #007bff;
}

.preview {
    border-color: #28a745;
}
```

```
label {
    color: #333;
}

input, select, button {
    margin-top: 5px;
    margin-bottom: 15px;
    padding: 10px;
    border-radius: 5px;
    border: 1px solid #ccc;
}

button {
    background-color: #007bff;
    color: white;
    cursor: pointer;
}

button:hover {
    background-color: #0056b3;
}
```

```
        @media (max-width: 768px) {
            .container {
                flex-direction: column;
                align-items: center;
            }
        }
    </style>
</head>
<body>

<div ng-app="studentApp" ng-controller="StudentFormController"
class="container">

    <form name="studentForm" ng-submit="submitForm()" novalidate>
        <div>
            <label for="name">Name:</label>
            <input type="text" id="name" name="name"
ng-model="student.name" required>
            <span ng-show="studentForm.name.$touched &&
studentForm.name.$invalid">Name is required.</span>
        </div> <br>

        <div>
            <label for="rollno">Roll No:</label>
```

```
            <input type="text" id="rollno" name="rollno"
ng-model="student.rollno" required>

            <span ng-show="studentForm.rollno.$touched &&
studentForm.rollno.$invalid">Roll No is required.</span>

        </div><br>



        <div>

            <label for="class">Class:</label>

            <input type="text" id="class" name="class"
ng-model="student.class" required>

            <span ng-show="studentForm.class.$touched &&
studentForm.class.$invalid">Class is required.</span>

        </div><br>



        <div>

            <label for="email">Email:</label>

            <input type="email" id="email" name="email"
ng-model="student.email" required>

            <span ng-show="studentForm.email.$touched &&
studentForm.email.$invalid">Valid email is required.</span>

        </div><br>



        <div>

            <label for="phone">Phone Number:</label>
```

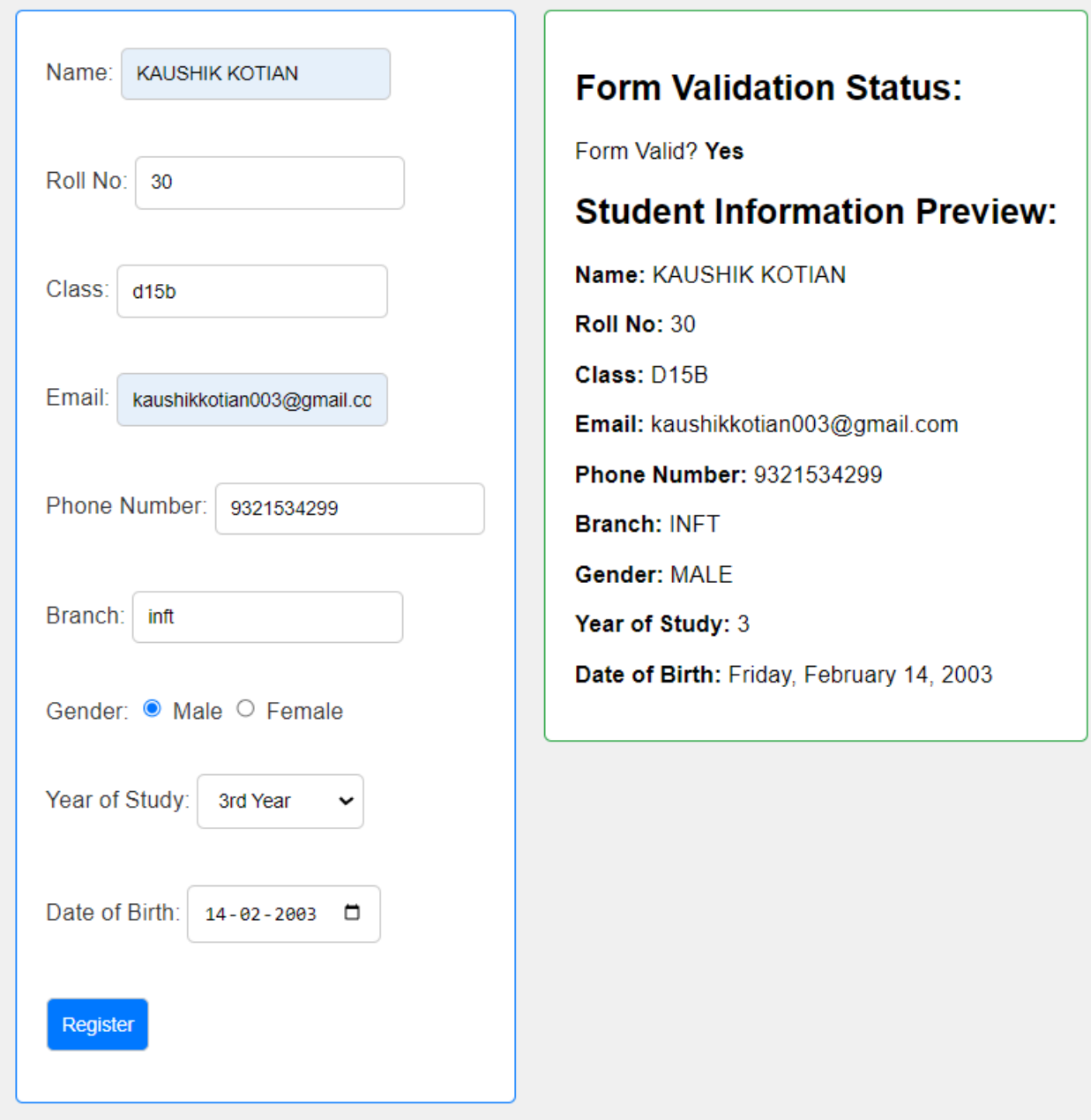
```
            <input type="tel" id="phone" name="phone"
ng-model="student.phone" required pattern="[0-9]{10}">

            <span ng-show="studentForm.phone.$touched &&
studentForm.phone.$invalid">Valid phone number is required.</span>

        </div><br>



        <div>

            <label for="branch">Branch:</label>

            <input type="text" id="branch" name="branch"
ng-model="student.branch" required>

            <span ng-show="studentForm.branch.$touched &&
studentForm.branch.$invalid">Branch is required.</span>

        </div><br>



        <div>

            <label>Gender:</label>

            <input type="radio" id="male" name="gender"
ng-model="student.gender" value="Male" required>

            <label for="male">Male</label>

            <input type="radio" id="female" name="gender"
ng-model="student.gender" value="Female" required>

            <label for="female">Female</label>

            <span ng-show="studentForm.gender.$touched &&
studentForm.gender.$invalid">Gender is required.</span>

        </div><br>
```

```
        <div>
            <label for="year">Year of Study:</label>
            <select id="year" name="year" ng-model="student.year" required>
                <option value="">Select Year</option>
                <option value="1">1st Year</option>
                <option value="2">2nd Year</option>
                <option value="3">3rd Year</option>
                <option value="4">4th Year</option>
            </select>
            <span ng-show="studentForm.year.$touched &&
studentForm.year.$invalid">Year of Study is required.</span>
        </div><br>

        <div>
            <label for="dob">Date of Birth:</label>
            <input type="date" id="dob" name="dob" ng-model="student.dob"
required>
            <span ng-show="studentForm.dob.$touched &&
studentForm.dob.$invalid">Date of birth is required.</span>
        </div><br>

        <button type="submit"
ng-disabled="studentForm.$invalid">Register</button>
```

```
    </form>


    <div class="preview">
        <h2>Form Validation Status:</h2>
        <p>Form Valid? <strong>{{studentForm.$valid ? 'Yes' :
'No'}}</strong></p>
        <h2>Student Information Preview:</h2>
        <p><strong>Name:</strong> {{student.name | uppercase}}</p>
        <p><strong>Roll No:</strong> {{student.rollno | uppercase}}</p>
        <p><strong>Class:</strong> {{student.class | uppercase}}</p>
        <p><strong>Email:</strong> {{student.email }}</p>
        <p><strong>Phone Number:</strong> {{student.phone |
uppercase}}</p>
        <p><strong>Branch:</strong> {{student.branch | uppercase}}</p>
        <p><strong>Gender:</strong> {{student.gender| uppercase}}</p>
        <p><strong>Year of Study:</strong> {{student.year|
uppercase}}</p>
        <p><strong>Date of Birth:</strong> {{student.dob |
date:'fullDate'}}</p>
    </div>


</div>


<script>
```

```
angular.module('studentApp', [])
.controller('StudentFormController', ['$scope', function($scope) {
    $scope.student = {
        name: '',
        rollno: '',
        class: '',
        email: '',
        phone: '',
        branch: '',
        gender: '',
        year: '',
        dob: ''
    };
    $scope.submitForm = function() {
        if ($scope.studentForm.$valid) {
            alert('Form submitted successfully!');
        } else {
            alert('Please fill out the form correctly!');
        }
    };
}]);
</script>
</body>
```

</html>


Name: KAUSHIK KOTIAN
Roll No: 30
Class: d15b
Email: kaushikkotian003@gmail.cc
Phone Number: 9321534299
Branch: inft
Gender: Male Female
Year of Study: 3rd Year
Date of Birth: 14-02-2003
Register
Form Validation Status:
Form Valid? Yes
Student Information Preview:
Name: KAUSHIK KOTIAN
Roll No: 30
Class: D15B
Email: kaushikkotian003@gmail.com
Phone Number: 9321534299
Branch: INFT
Gender: MALE
Year of Study: 3
Date of Birth: Friday, February 14, 2003

Experiment no.:4

index.js:

```
    .then(() => {
        console.log("MONGO CONNECTION OPEN!!!")
    })
    .catch(err => {
        console.log("OH NO MONGO CONNECTION ERROR!!!!")
        console.log(err)
    })

app.set('views', path.join(
____dirname, 'views')); app.set('view
engine', 'ejs');


app.use(express.urlencoded({ extended: true }));
app.use(methodOverride('_method'))


const categories = ['fruit', 'vegetable', 'dairy'];


    app.get('/products', async (req,
        res) => { const { category } =
        req.query;

    if (category) {

        const products = await Product.find({ category })
        res.render('products/index', { products, category })

    } else {
```

```
        const products = await Product.find({})
        res.render('products/index', { products, category: 'All'

})

    }
})


    app.get('/products/new', (req, res) => {
        res.render('products/new', { categories })

})


app.post('/products', async (req, res) => {

    const newProduct = new
    Product(req.body); await
    newProduct.save();
    res.redirect(`/products/${newProduct.
    _id}`)

})

    app.get('/products/:id', async (req,
        res) => { const { id } =
        req.params;

    const product = await Product.findById(id)
    res.render('products/show', { product })

})

    app.get('/products/:id/edit', async (req,
        res) => { const { id } = req.params;

    const product = await Product.findById(id);
    res.render('products/edit', { product, categories })
```

```
})

    app.put('/products/:id', async (req,
        res) => { const { id } =
        req.params;
    const product = await
Product.findByIdAndUpdate(id, req.body, {
runValidators: true, new: true });

    res.redirect(`/products/${product._id}`);

})
    app.delete('/products/:id', async (req,
        res) => { const { id } = req.params;

    const deletedProduct = await Product.findByIdAndDelete(id);
    res.redirect('/products');

})

app.listen(3000, () => {
    console.log("APP IS LISTENING ON PORT 3000!")

})
```

a. Previously added products

All Products!

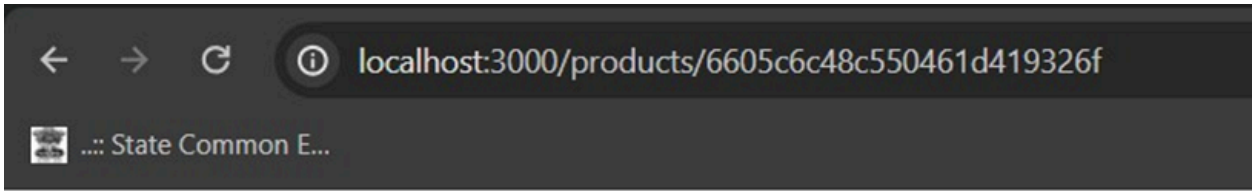
- Apple
- Capsicum

New Product

b. Adding a new product

Add A Product

Product Name Cabbage Price (Unit) 15 Select Category vegetable Submit

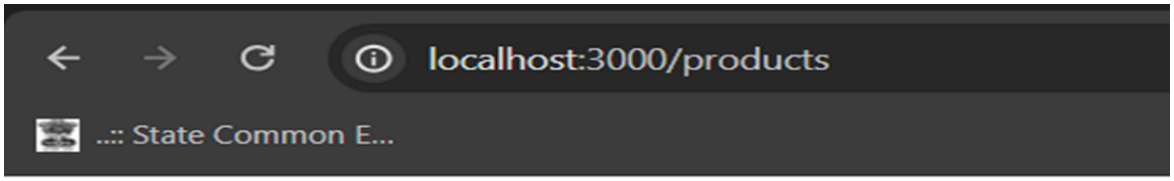


Cabbage

- Price: $15
- Category: vegetable

All Products Edit Product
Delete

c. List of all products after adding a new item

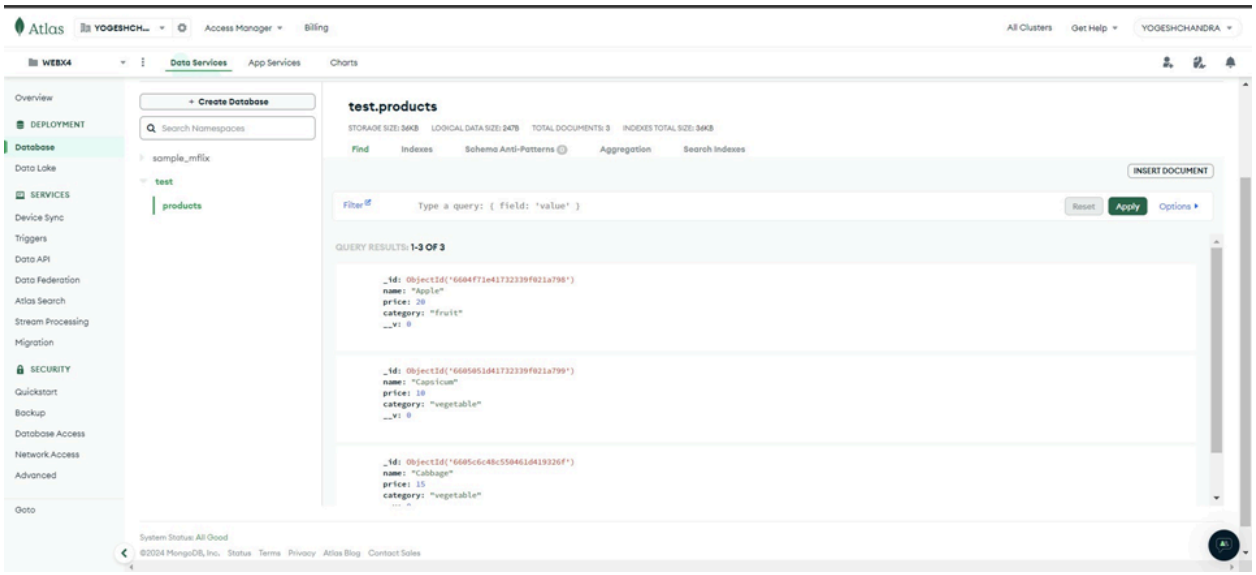


All Products!

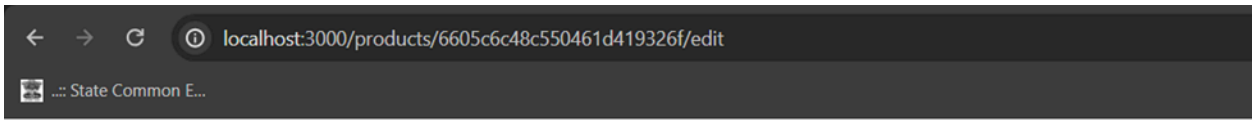
- Apple
- Capsicum
- Cabbage

New Product

d. Changes reflected in MongoDB Atlas after adding a new product



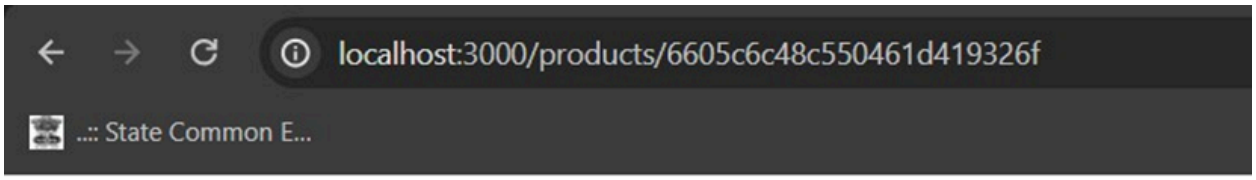
e. Editing the recently added product



Edit Product

Product Name Cabbage Price (Unit) 20 Select Category vegetable Submit

Cancel



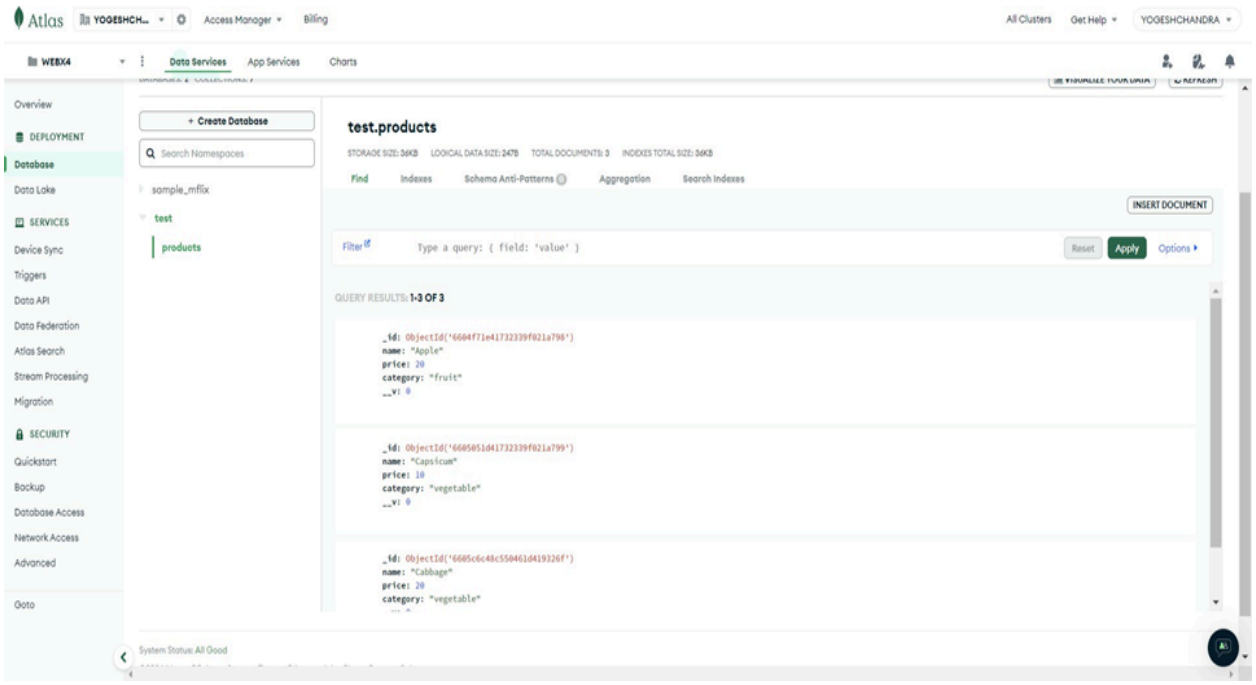
Cabbage

- Price: $20
- Category: vegetable

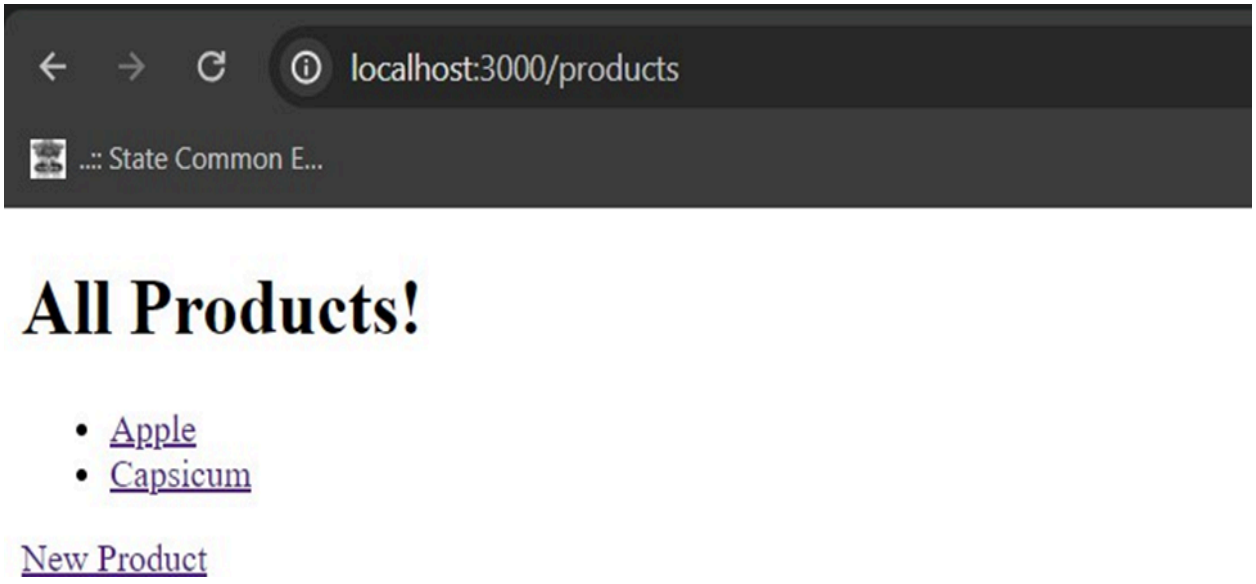
All Products Edit Product

Delete

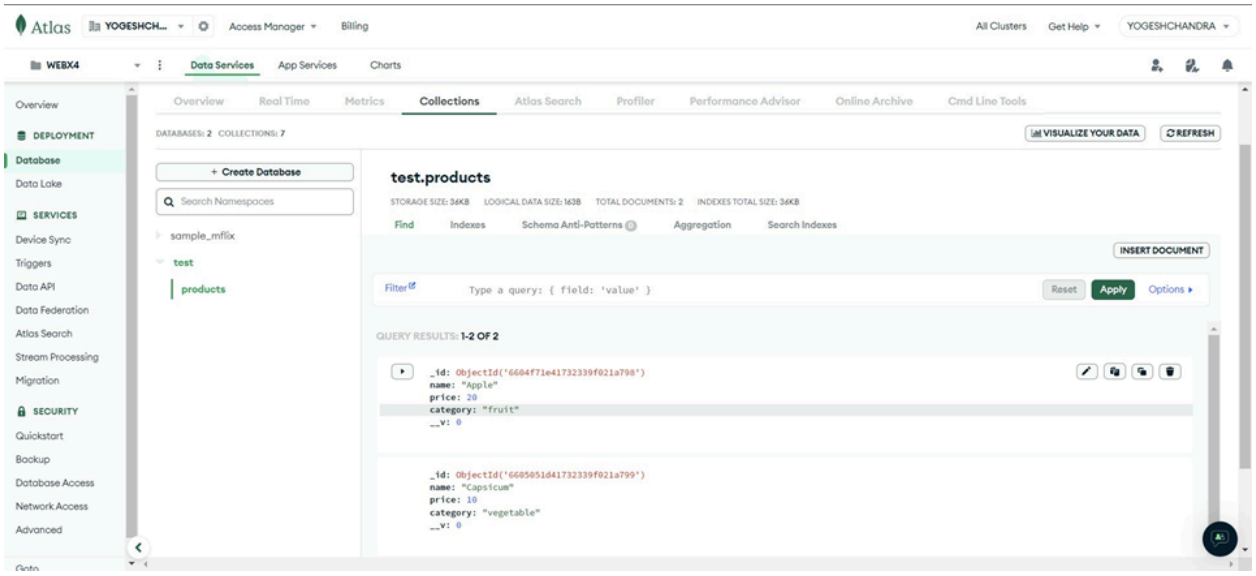
f. Changes reflected in MongoDB Atlas after editing the added product



g. Deleting the added product



h. Changes reflected in MongoDB Atlas after deleting the added product



Conclusion : In this experiment, the creation of a RESTful API with MongoDB showcased the robust capabilities of modern web development. Implementing CRUD operations provided a comprehensive understanding of data manipulation. Leveraging MongoDB's flexibility and scalability alongside RESTful architecture highlights the efficient management and accessibility of data in

Experiment No. 6

Aim: Design Flask application to demonstrate the usage of routing and template

Theory:

Flask Web Framework
Flask is a micro web framework for Python based on Werkzeug and Jinja2. It is called a micro framework because it does not require particular tools or libraries. Despite its simplicity, Flask is extensible and can be used to build complex web applications. It provides the basics for web development, such as request handling, routing, and template rendering, while allowing developers to choose their tools for database integration, form validation, authentication, and more.

Project Structure
The project is structured into two main components: the Python script (app.py) that defines the Flask application and its behavior, and the HTML templates (index.html, home.html, login.html) that provide the user interface.

- app.py: This is the core of the Flask application. It sets up the Flask app, defines routes, handles requests, and renders templates. It includes essential Flask concepts like application initialization, route decorators, view functions, and the use of render_template for integrating Python logic with HTML content.

- templates/: A directory that stores HTML templates. Flask automatically looks for templates in a folder named templates within the application root. These templates use Jinja2 syntax to allow dynamic content generation based on the Python code and logic defined in app.py.

Core Concepts
- Application Setup and Configuration: The Flask application is initialized with app = Flask(__name__), and various configurations, including debug mode, are set to tailor the app's behavior.

- Routing and View Functions: Flask uses routes to bind URLs to Python functions, known as view functions. These functions define the responses for specific HTTP requests. The project uses decorators (@app.route()) to map view functions to routes.

- Template Rendering: Flask integrates with the Jinja2 templating engine, allowing Python variables and logic to be included in HTML templates. The render_template function is used to render templates and dynamically pass data from the Flask application to the HTML frontend.

- Request Handling: The project demonstrates handling GET and POST requests using the request object. This object provides access to request data, such as form fields submitted via a POST request.

- User Authentication: Basic user authentication is implemented to demonstrate form handling and conditional logic based on user input. It involves checking submitted credentials and redirecting users based on the authentication outcome.

Development and Testing
The application is run in a development environment, facilitated by Flask's built-in development server. This server is suitable for development and testing but not for production use. Developers are advised to use a production WSGI server for deploying Flask applications.

Security Considerations
While this project illustrates basic concepts, it simplifies aspects like user authentication for educational purposes. In real-world applications, security measures such as password hashing, secure session management, and protection against common web vulnerabilities (e.g., SQL injection, cross-site scripting) are essential.

Code:

Index.html:

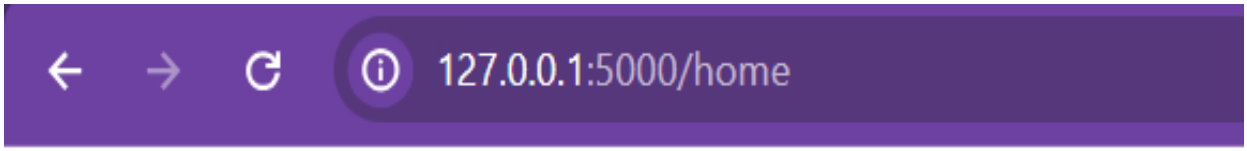
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Welcome Page</title>
</head>
<body>
    <h1>Welcome to the Flask Application</h1>
    <nav>
        <ul>
            <li><a href="{{ url_for('home') }}">Home</a></li>
            <li><a href="{{ url_for('login') }}">Login</a></li>
        </ul>
    </nav>
</body>
</html>
```

Welcome to the Flask Application

- Home
- Login

Home.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Home Page</title>
</head>
<body>
    <h1>Home Page</h1>
    <p>Welcome to the home page. You're logged in as admin.</p>
    <a href="{{ url_for('index') }}">Back to Welcome Page</a>
</body>
</html>
```



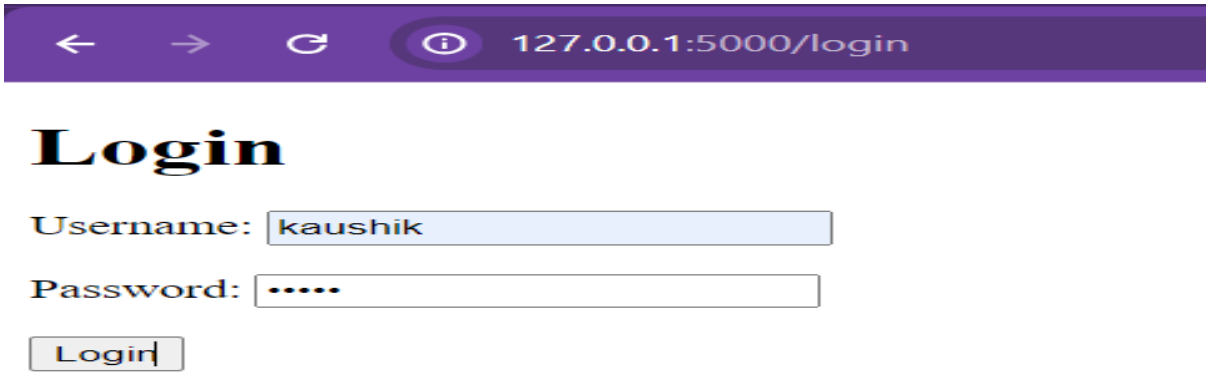
Home Page

Welcome to the home page. You're logged in as admin.

Back to Welcome Page

Login.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Login Page</title>
</head>
<body style="display: flex; justify-content: center; align-items:
center; flex-direction: column; height: 100vh;">
    <h1>Login</h1>
    <form method="POST" action="{{ url_for('login') }}">
        <div>
            <label for="user">Username:</label>
            <input type="text" id="user" name="user" required>
        </div><br/>
        <div>
            <label for="password">Password:</label>
            <input type="password" id="password" name="password"
required>
        </div><br/>
        <button type="submit" >Login</button>
    </form>
</body>
</html>
```



app.py:

```
from flask import Flask, render_template, request, redirect,
url_for

app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/home')
def home():
    return render_template('home.html')

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        user = request.form['user']
        password = request.form['password']
        if user == 'admin' and password == 'admin':
            return redirect(url_for('home'))
        else:
            return redirect(url_for('login'))
    else:
        return render_template('login.html')

if __name__ == '__main__':
    app.run(debug=True)
```

```
PS C:\Users\Kaushik Kotian\OneDrive\Desktop\WebX\exp6> python app.py
 >>
  * Serving Flask app 'app'
  * Debug mode: on
WARNING: This is a development server. Do not use it in a production deploy
  * Running on http://127.0.0.1:5000
Press CTRL+C to quit
  * Restarting with watchdog (windowsapi)
  * Debugger is active!
  * Debugger PIN: 197-367-356
127.0.0.1 - - [22/Mar/2024 18:47:04] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [22/Mar/2024 18:47:12] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [22/Mar/2024 18:47:14] "GET /home HTTP/1.1" 200 -
127.0.0.1 - - [22/Mar/2024 18:47:16] "GET /login HTTP/1.1" 200 -
127.0.0.1 - - [22/Mar/2024 18:47:39] "GET /login HTTP/1.1" 200 -
127.0.0.1 - - [22/Mar/2024 18:48:06] "GET /login HTTP/1.1" 200 -
127.0.0.1 - - [22/Mar/2024 18:48:29] "GET /login HTTP/1.1" 200 -
127.0.0.1 - - [22/Mar/2024 18:48:30] "GET /login HTTP/1.1" 200 -
127.0.0.1 - - [22/Mar/2024 18:59:20] "GET /home HTTP/1.1" 200 -
127.0.0.1 - - [22/Mar/2024 18:59:25] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [22/Mar/2024 18:59:27] "GET /home HTTP/1.1" 200 -
127.0.0.1 - - [22/Mar/2024 18:59:29] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [22/Mar/2024 18:59:30] "GET /login HTTP/1.1" 200 -
127.0.0.1 - - [22/Mar/2024 18:59:53] "GET /login HTTP/1.1" 200 -
127.0.0.1 - - [22/Mar/2024 19:00:03] "POST /login HTTP/1.1" 302 -
127.0.0.1 - - [22/Mar/2024 19:00:03] "GET /login HTTP/1.1" 200 -
127.0.0.1 - - [22/Mar/2024 19:11:16] "GET /home HTTP/1.1" 200 -
127.0.0.1 - - [22/Mar/2024 19:13:04] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [22/Mar/2024 19:13:05] "GET /login HTTP/1.1" 200 -
127.0.0.1 - - [22/Mar/2024 19:13:45] "GET /login HTTP/1.1" 200 -
```

Conclusion:

This Flask project provides a foundation for understanding web application development with Flask, from setting up the application and handling HTTP requests to rendering dynamic content and implementing simple authentication. It serves as a stepping stone towards building more complex and secure web applications with Flask.