A INDUSTRIAL TRAINING REPORT

On

“

“ FULL STACK WEB DEVELOPMENT”

Submitted in the fulfilment of the requirementfor the award of the degree of

Bachelor of Technology

in

COMPUTER SCIENCE

Submitted To:
Mr. Pawan Sen
HOD CSE

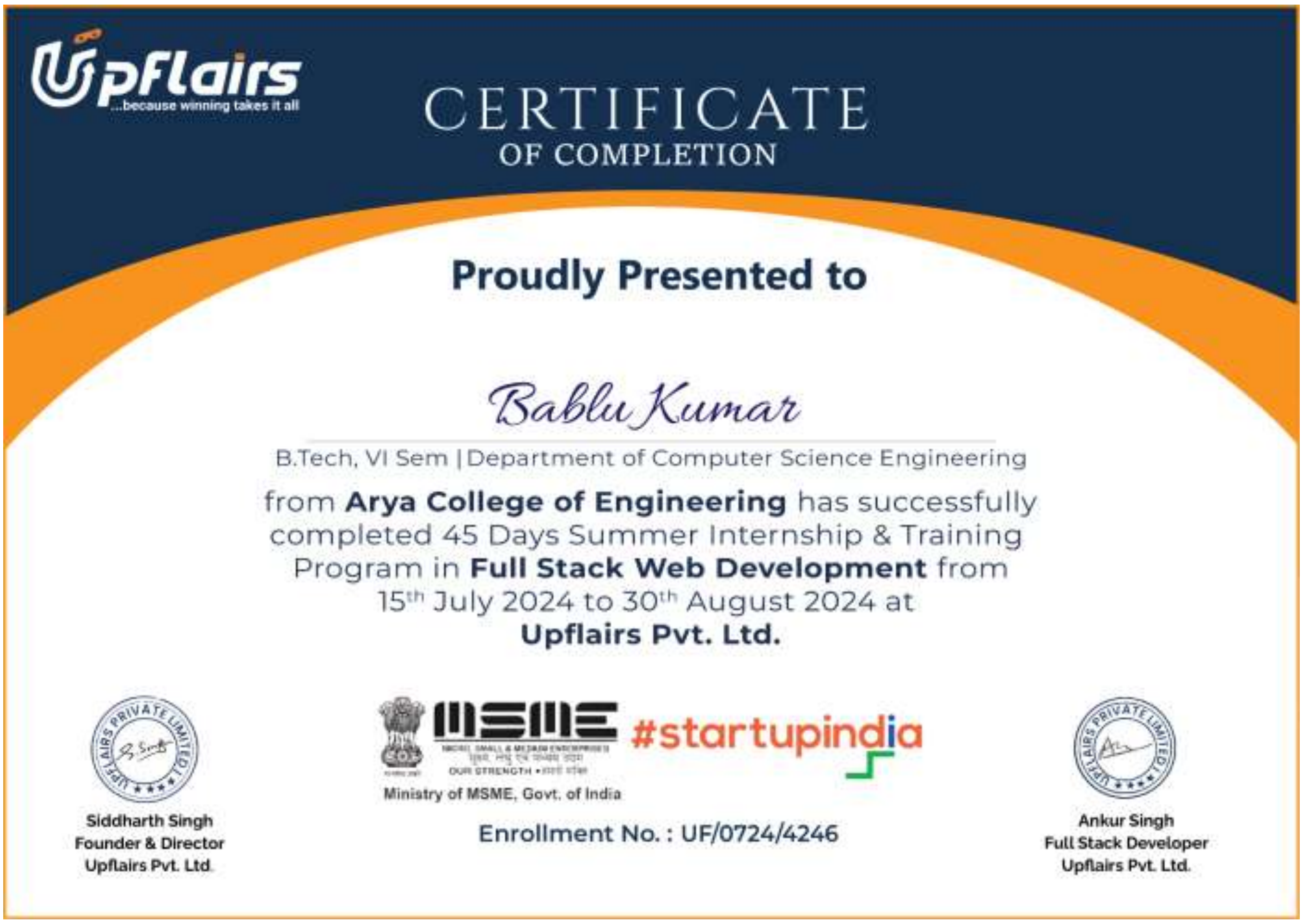
Submitted By:
Bablu Kumar
V Sem CSE
21EAYCS032

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

ARYA COLLEGE OF ENGINEERING AND RESEARCH CENTER ,JAIPUR

RAJASTHAN TECHNICAL UNIVERSITY,KOTA

RICCO KUKAS 302028,JAIPUR RAJASTHAN
2024-2025



Upflairs ...because winning takes it all

CERTIFICATE OF COMPLETION

Proudly Presented to

Bablu Kumar

B.Tech, VI Sem | Department of Computer Science Engineering

from **Arya College of Engineering** has successfully completed 45 Days Summer Internship & Training Program in **Full Stack Web Development** from 15th July 2024 to 30th August 2024 at **Upflairs Pvt. Ltd.**

UPFLAIRS PRIVATE LIMITED

[illegible]

Siddharth Singh
Founder & Director
Upflairs Pvt. Ltd.

MSME #startupindia

Ministry of MSME, Govt. of India

Enrollment No. : UF/0724/4246

UPFLAIRS PRIVATE LIMITED

[illegible]

Ankur Singh
Full Stack Developer
Upflairs Pvt. Ltd.

Candidate's Declaration

I hereby declare that the work, which is being presented in the Industrial Training report entitled**"FULL STACK WEB DEVELOPMENT" I**n partial fulfilment for the award of Degree of "Bachelor of Technology" in Department of Computer Science & Engineering with Specialization in Computer Engineering and submitted to the **Department of Computer Science & Engineering,** Arya College of Engineering And Research Center is a record of my own investigations carried under the Guidance of **Mr Pawan Sen** Assistant Professor, Department of Computer Science & Engineering.

(Signature of Candidate)

Bablu Kumar

Roll No: 21EAYCS032

Acknowledgements

This work is just not an individual contribution till its completion. I take this opportunity to express a deep gratitude towards my teacher for providing excellent guidance encouragement and inspiration throughout the Training work. Without his invaluable guidance, this work would never have been a successful one.

I would like to express deepest appreciation towards **PAWAN SEN**, Head ofDepartment of Computer Science.

At last we must express our sincere heartfelt gratitude to all the staff members of Information Technology Department who helped me directly or indirectly during this course of work.

Bablu Kumar

ABSTRACT

The aim of this Bachelor’s thesis was to develop a mobile-first style website for the customer, Pohjois-Suomen Pesis. The main purpose of the devel- opment was to learn website designing principles and create a responsive website for the mobile and desktop platforms. The development process began defining the requirements of the website and creating the requirements document. Then next step was learning how to design a website layout and to choose the colour scheme for the site. The website was constructed by Mern and Bootstrap. The result of the website was as desired. The website scaled all the different platforms, and all the required requirements were fulfilled.

Contents

Chapter 1

Introduction

1.1 WEB DEVELOPMENT

Web development is the work involved in developing a web site for the Internet oran intranet. Web development can range from developing a simple single static pageof plain text to complex web-based internet applications, electronic businesses, andsocial network services

Chapter 2

WEB

2.1 HTTP

HTTP stands for Hyper Text Transfer Protocol WWW is about communication between web clients and servers Communication between client computers and web servers is done by sending HTTP Requests and receiving HTTP Responses

2.1.1 HTTP Request / Response

Communication between clients and servers is done by requests and responses:

a. A client (a browser) sends an HTTP request to the web

b. An web server receives the request

c. The server runs an application to process the request

d. The server returns an HTTP response (output) to the browser

e. The client (the browser) receives the response

2.1.2 The HTTP Request Circle

A typical HTTP request / response circle:

a. The browser requests an HTML page. The server returns an HTML file.

b. The browser requests a stylesheet. The server returns a CSS file.

c. The browser requests an JPG image. The server returns a JPG file.

d. The browser requests JavaScript code. The server returns a JS file

e. The browser requests data. The server returns data (in XML or JSON).

2.2 HTML

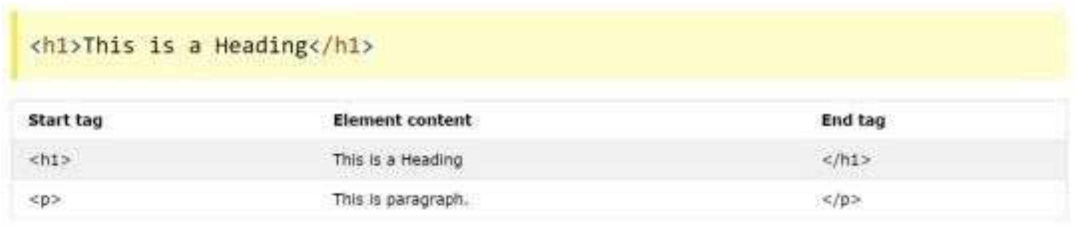
HTML stands for Hyper Text Markup Language HTML is the standard markup language for Web pages HTML elements are the building blocks of HTML pages HTML elements are represented by tags BASIC TERMS:

* Project stucture:

```
<!Doctype>
<html>
<body>
...........
............
...........
...........
</body>
</html>
```

2.2.1 HTML Elements

An HTML element is a start tag and an end tag with content in between:



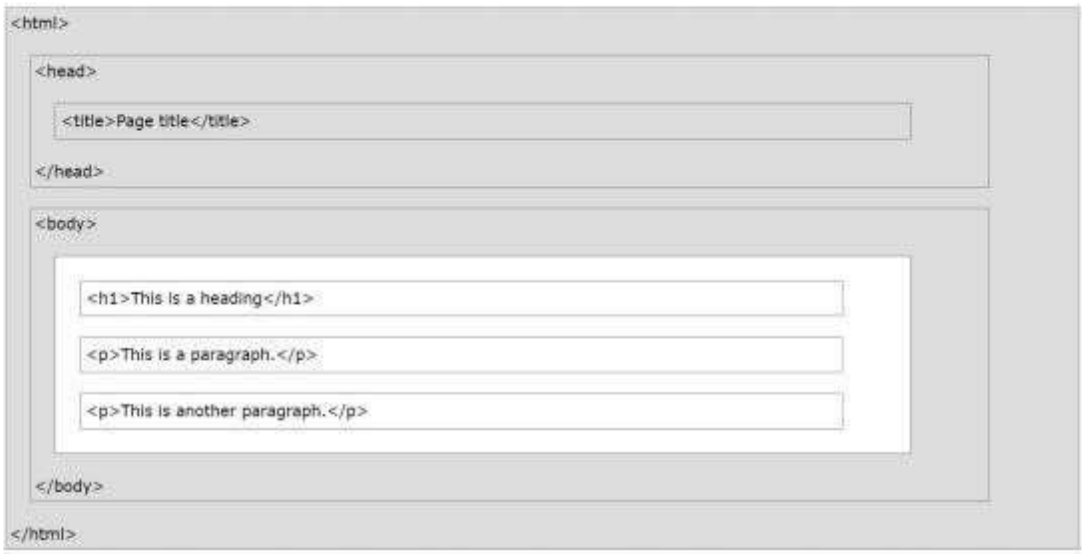
```
<h1>This is a Heading</h1>
```

Start tag	Element content	End tag
<h1>	This is a Heading	</h1>
<p>	This is paragraph.	</p>

Figure 2.1: Heading Tag

2.2.2 HTML Documents

The HTML document itself begins with html tag and ends with html tag followed by ’/’ forward slash. The visible part of the HTML document is between body tag.



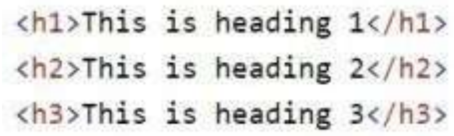
```
<html>
<head>
<title>Page title</title>
</head>
<body>
<h1>This is a heading</h1>
<p>This is a paragraph.</p>
<p>This is another paragraph.</p>
</body>
</html>
```

. <HTML> tag:

The HTML <title> tag is used for declaring the title, or name, of the HTML document. The title is usually displayed in the browser's title bar (at the top).It is also displayed in browser bookmarks and search results. The title tag is placed between the opening and closing <head> tags. The <link> element is used to define a relationship between an HTML document and an external re- source. This element is most commonly used to define the relationship between a document and one or more external CSS stylesheets.

. HTML Headings

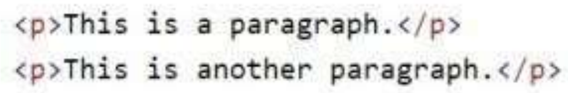
HTML headings are defined with h1 to h6 tags.



```
<h1>This is heading 1</h1>
<h2>This is heading 2</h2>
<h3>This is heading 3</h3>
```

. HTML Paragraphs

HTML paragraphs are defined with p tags:



```
<p>This is a paragraph.</p>
<p>This is another paragraph.</p>
```

. HTML Links

HTML links are defined with a tags:

```
<a href="https://www.w3schools.com">This is a link</a>
```

. HTML Images

HTML images are defined with img tags.

The source file (src), alternative text (alt), width, and height are provided as attributes:

```
<img src="img_w3schools.jpg" alt="W3Schools" style="width:120px;height:150px"
```

. HTML Buttons

HTML buttons are defined with button tags:

```
<button>Click me</button>
```

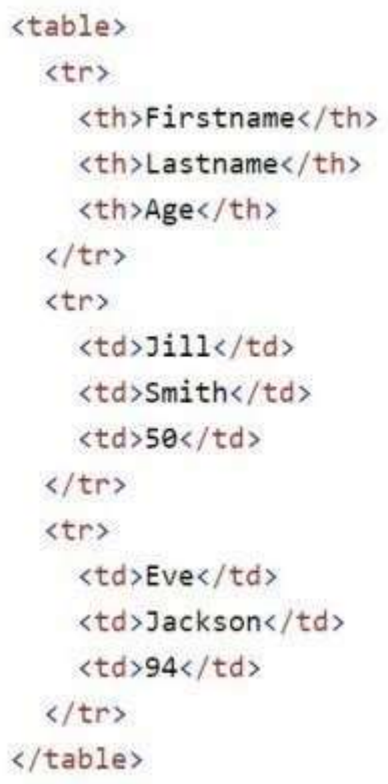
. HTML Lists

HTML lists are defined with ul tag (unordered/bullet list) or ol tag (ordered/numbered list) tags, followed by li tags (list items):

```
<ul>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

. HTML Tables

An HTML table is defined with a table tag. Table rows are defined with tr tags. Table headers are defined with th tags. (bold and centered by default). Table cells (data) are defined with td tags.



```
<table>
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Jill</td>
    <td>Smith</td>
    <td>50</td>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
    <td>94</td>
  </tr>
</table>
```

. <Body> tag:

Description. The HTML ¡body¿ tag defines the main content of the HTML document or the section of the HTML document that will be directly visible on your web page. This tag is also commonly referred to as the <body> element.

. <header>:

he <header> element is intended to usually contain the section’s heading(an < h1 > - < h6 > element or an < hgroup> element), but this is not required. The <header> element can also be used to wrap a section’s table of contents,a search form, or any relevant logos.

. <div> tag:

The <div> tag defines a division or a section in an HTML document. The <div> element is often used as a container for other HTML elements to style them with CSS or to perform certain tasks with JavaScript.The div tag is known as Division tag. The Div tag is used in HTML to make divisions of content in the web page like (text, images, header, footer, navigation bar etc). Div tag has both open(<) and closing (>) tag and it is mandatory to close the tag. The Div is the most usable tag in web development because it helps us to separate out data in the web page and we can create a particular section for particular dataor function in the web pages.

Div tag is Block level tag

It is a generic container tag

It is used to the group of various tags of HTML so that sections can be created and style can be Applied on them.

Left center right.

. <br> tag:

The HTML anchor tag defines a hyperlink that links one page to another page. The ”href” attribute is the most important attribute of the HTML a tag. An unvisited link is displayed underlined and blue. A visited link displayed underlined and purple. An active link is underlined and red.

. <footer> tages:

HTML5 <footer> Element. The <footer> elementspecifies a footer for a document or section. A <footer> element should contain information about its containing element. A footer typically contains the author of the document, copyright information, links to terms of use, contact information, etc.

. <form> tag:

The <form>tag is used in conjunction with form-associated elements. To create a form,you can nest form-associated elements inside the opening/closing <form> tags. You can also use the form attribute within those elements to reference the ID of the form to use.

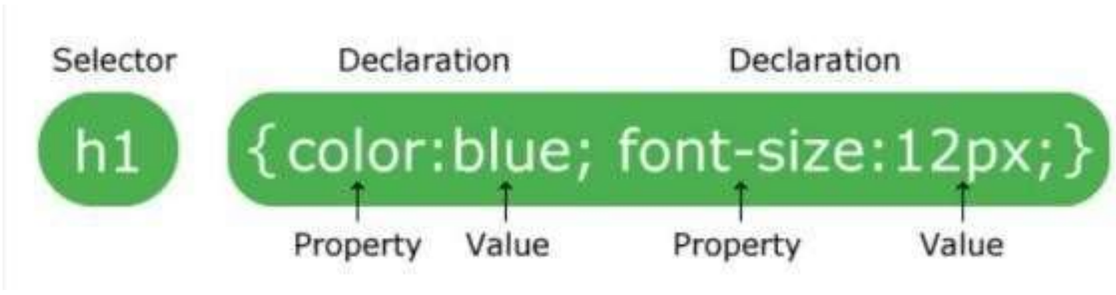
2.3 CSS

CSS stands for Cascading Style Sheets.

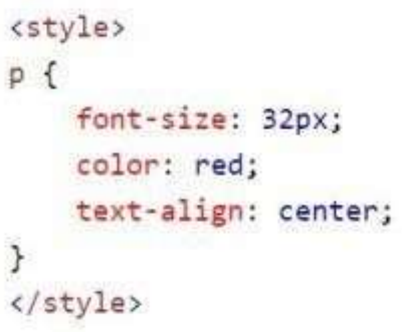
Cascading Style Sheets is a style sheet language used for describing the presenta- tion of a document written in a markup language like HTML. CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript.

2.3.1 CSS Syntax

A CSS rule consists of a selector and a declaration block:



The selector points to the HTML element to style (h1). The declaration block (in curly braces) contains one or more declarations separated by semicolons. Each declaration includes a CSS property name and a value, separated by a colon. In the following example all p tag elements will be 32px wide, center-aligned, and with red. Example:

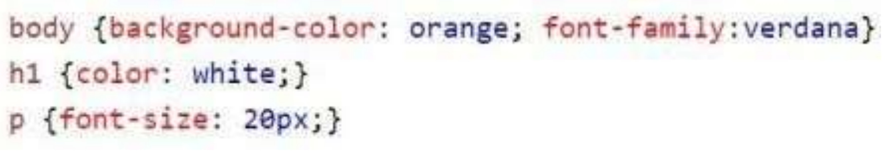


```
<style>
p {
    font-size: 32px;
    color: red;
    text-align: center;
}
</style>
```

2.3.2 External Style Sheet

The external style sheet is generally used when you want to make changes on mul- tiple pages. It is ideal for this condition because it facilitates you to change the look of the entire website by changing just one file. It uses the link tag on every pages and the link tag should be put inside the head section.

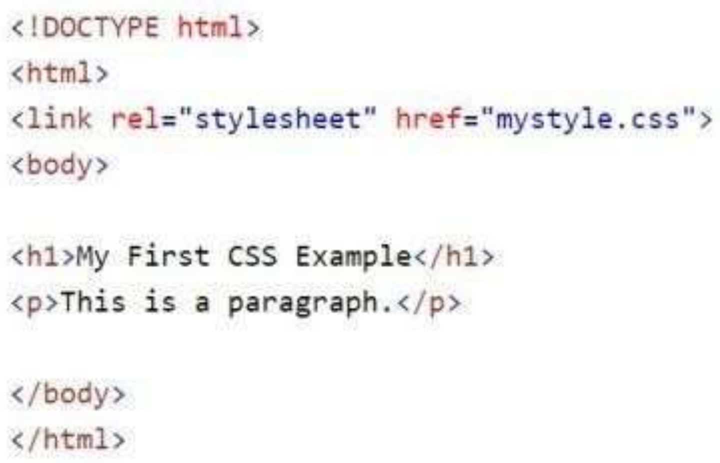
Example:



```
body {background-color: orange; font-family:verdana}
h1 {color: white;}
p {font-size: 20px;}
```

Figure 2.3: .css file

The external style sheet may be written in any text editor but must be saved with a .css extension. This file should not contain HTML elements.



```
<!DOCTYPE html>
<html>
<link rel="stylesheet" href="mystyle.css">
<body>

<h1>My First CSS Example</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

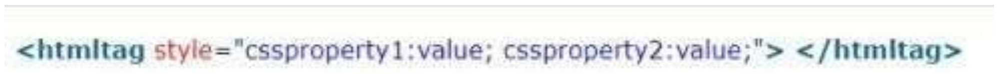
Figure 2.4: .css file linked with .html file

2.3.3 Inline Style

We can apply CSS in a single element by inline CSS technique.

The inline CSS is also a method to insert style sheets in HTML document. This method mitigates some advantages of stylesheets so it is advised to use this method sparingly.

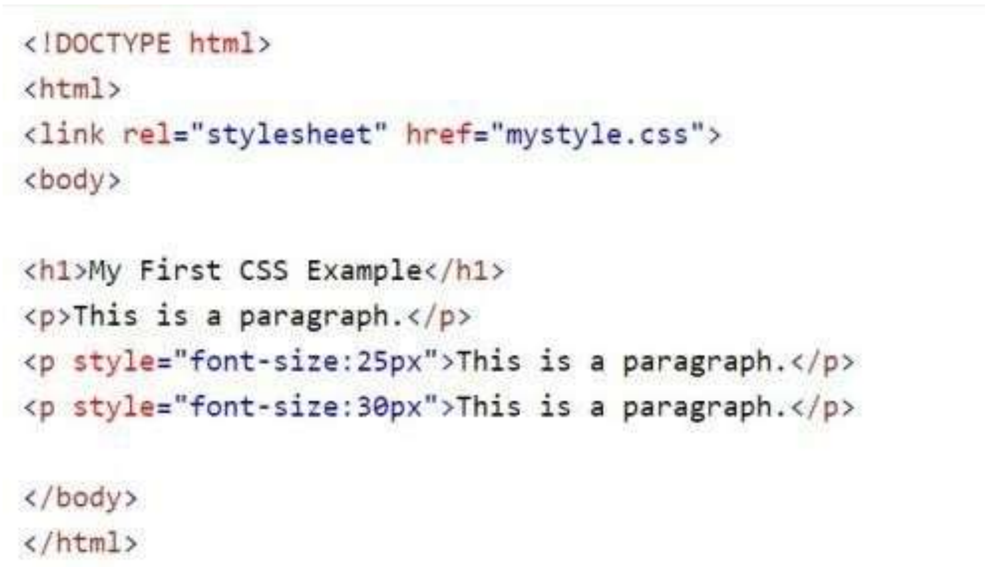
If you want to use inline CSS, you should use the style attribute to the relevant tag.



```
<htmltag style="cssproperty1:value; cssproperty2:value;"> </htmltag>
```

Figure 2.5: Inline Syntax .css file

Example:

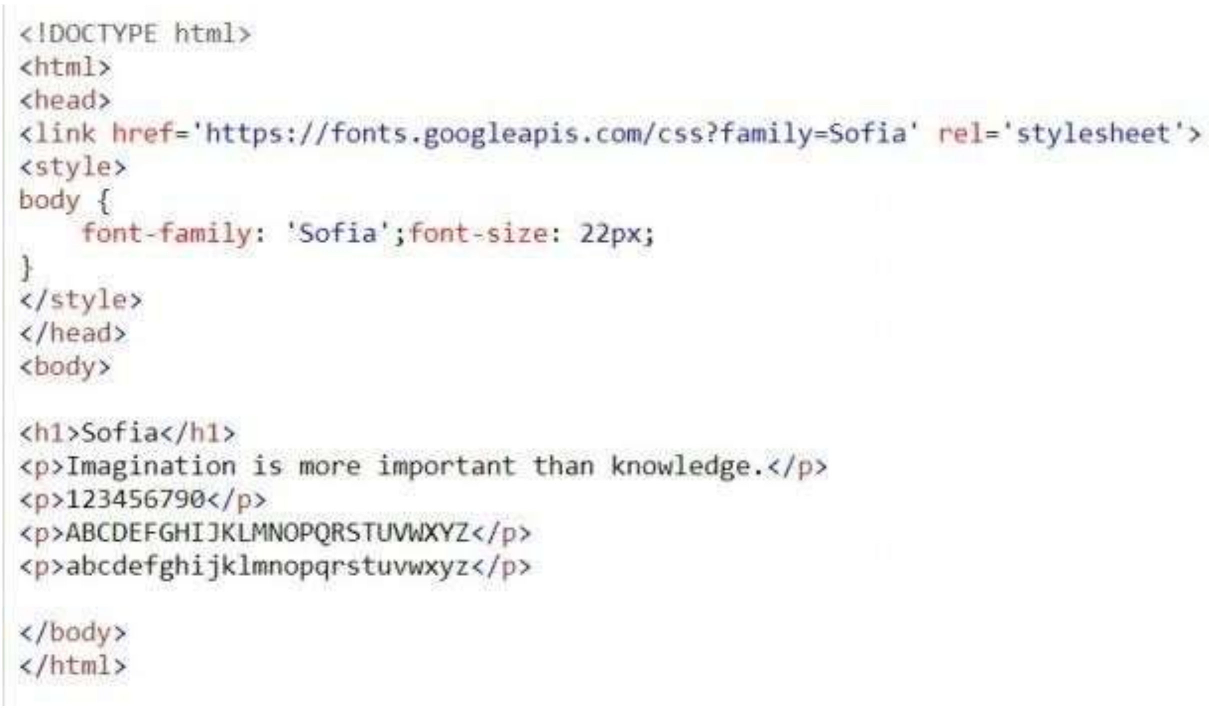


```
<!DOCTYPE html>
<html>
<link rel="stylesheet" href="mystyle.css">
<body>

<h1>My First CSS Example</h1>
<p>This is a paragraph.</p>
<p style="font-size:25px">This is a paragraph.</p>
<p style="font-size:30px">This is a paragraph.</p>

</body>
</html>
```

Figure 2.6: Inline Style Sheet

2.4 Google fonts

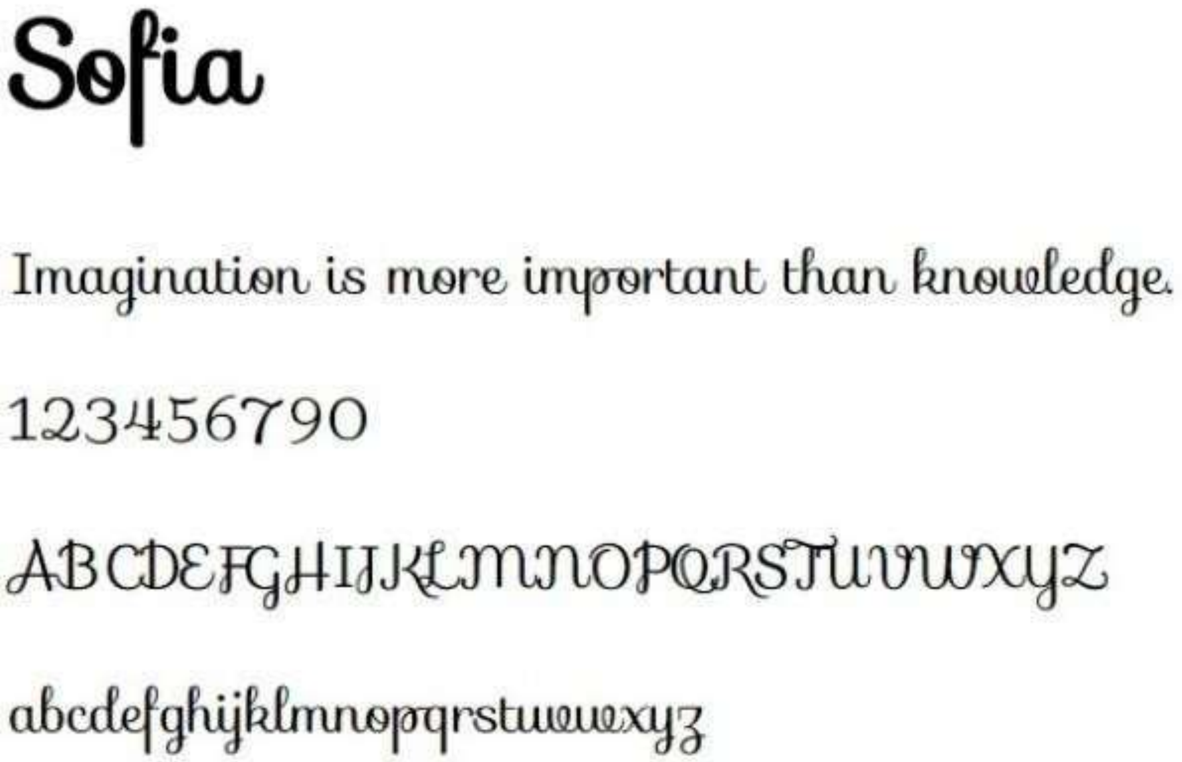
Google Fonts is a Google API.
We can use Google Fonts in our Website design.



```
<!DOCTYPE html>
<html>
<head>
<link href='https://fonts.googleapis.com/css?family=Sofia' rel='stylesheet'>
<style>
body {
    font-family: 'Sofia';font-size: 22px;
}
</style>
</head>
<body>

<h1>Sofia</h1>
<p>Imagination is more important than knowledge.</p>
<p>123456790</p>
<p>ABCDEFGHIJKLMNOPQRSTUVWXYZ</p>
<p>abcdefghijklmnopqrstuvwxyz</p>

</body>
</html>
```

Figure 2.7: Google font using in .html file

When we use google fonts in designing webpage it will be viewed as:


Sofia

Imagination is more important than knowledge.

123456790

ABCDEFGHIJKLMNOPQRSTUVWXYZ

abcdefghijklmnopqrstuvwxyz

Figure 2.8: Google font viewpage

What is Javascript?

- a lightweight programming language ("scripting language")
- used to make web pages interactive
- insert dynamic text into HTML (ex: user name)
- react to events (ex: page load user click)
- get information about a user's computer (ex: browser type)
- perform calculations on user's computer (ex: form validation)

- a web standard (but not supported identically by all browsers)
- NOT related to Java other than by name and some syntactic similarities

Javascript vs Java

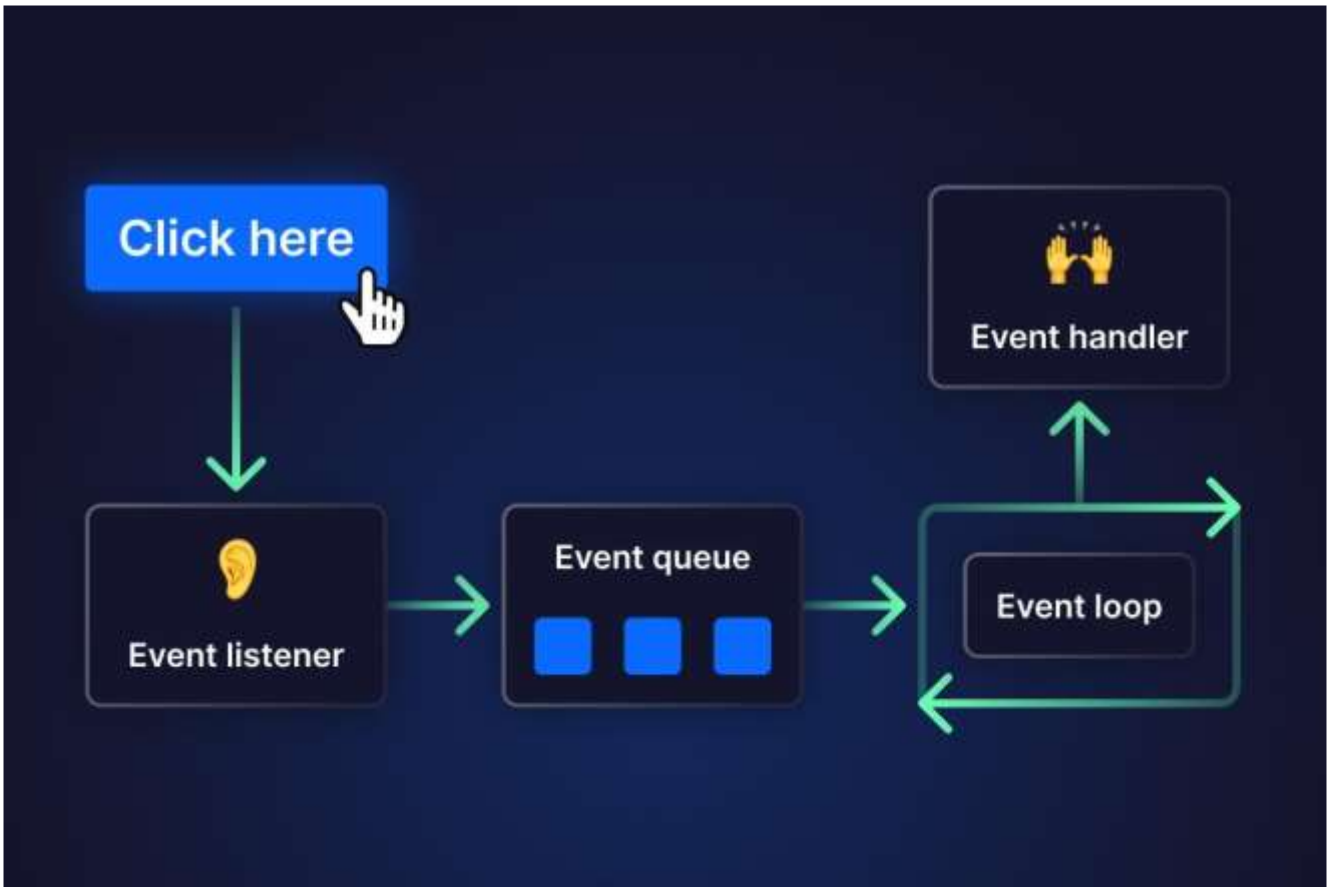
- interpreted, not compiled
- more relaxed syntax and rules
- fewer and "looser" data types
- variables don't need to be declared
- errors often silent (few exceptions)
- key construct is the function rather than the class
- "first-class" functions are used in many situations
- contained within a web page and integrates with its HTML/CSS content

Linking to a JavaScript file: script

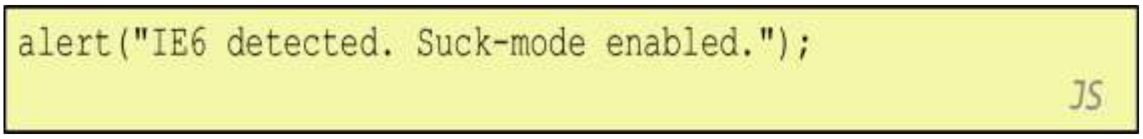
```
<script src="filename" type="text/javascript"></script>
                                                        HTML
```

- script tag should be placed in HTML page's head
- script code is stored in a separate .js file
- JS code can be placed directly in the HTML file's body or head (like CSS)
- but this is bad style (should separate content, presentation, and behavior

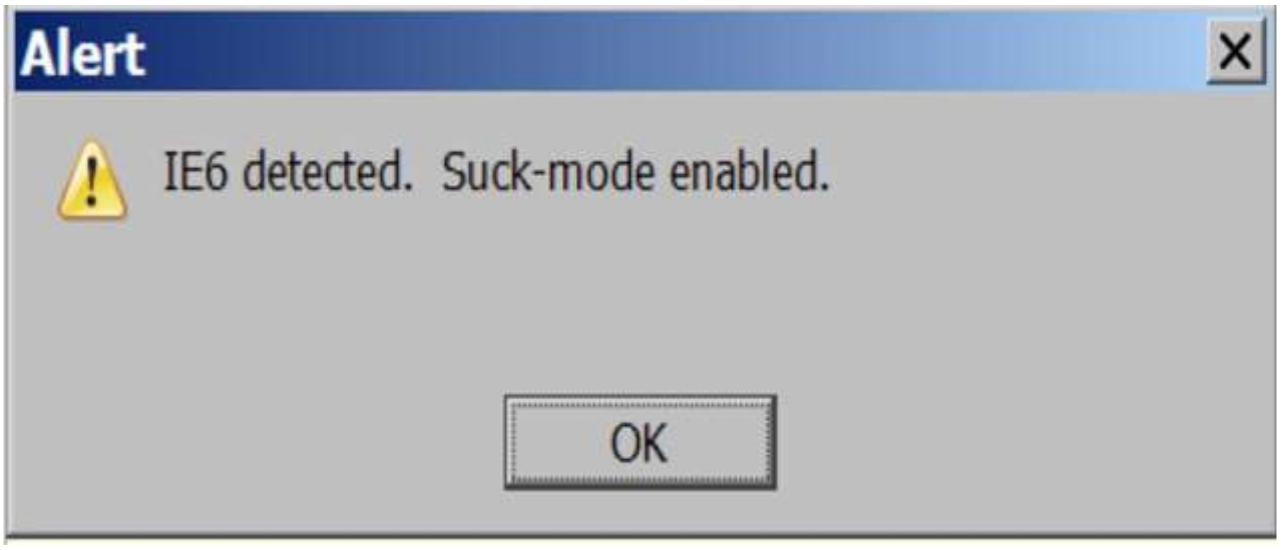
Event-driven programming



A JavaScript statement: alert



```
alert("IE6 detected. Suck-mode enabled.");
```



Alert

IE6 detected. Suck-mode enabled.

- ❑ a JS command that pops up a dialog box with a message

Event-driven programming

- ❑ you are used to programs start with a main method (or implicit main like in PHP)
- ❑ JavaScript programs instead wait for user actions called events and respond to them
- ❑ event-driven programming: writing programs driven by user events
- ❑ Let's write a page with a clickable button that pops up a "Hello, World" window...

Event Types

- ❑ Mouse Events: Click, double-click, hover, etc.
- ❑ Keyboard Events: Keydown, keyup, etc.
- ❑ Form Events: Submit, change, etc.
- ❑ Document Events: DOMContentLoaded, load, etc.
- ❑ 2.2 Event Listeners
- ❑ Functions that wait for a specific event to occur and then execute a predefined action.

Event Propagation

- ❑ Events can propagate through the DOM in two phases: capturing and bubbling.

Event Object

- ❑ An object containing information about the event, passed to the event handler.

Handling Events:

- ❑ Inline Event Handlers
- ❑ Assigning event handlers directly in HTML attributes.

- ❑ Traditional Event Handling
- ❑ Using addEventListener to attach event handlers dynamically.

Event-driven programming

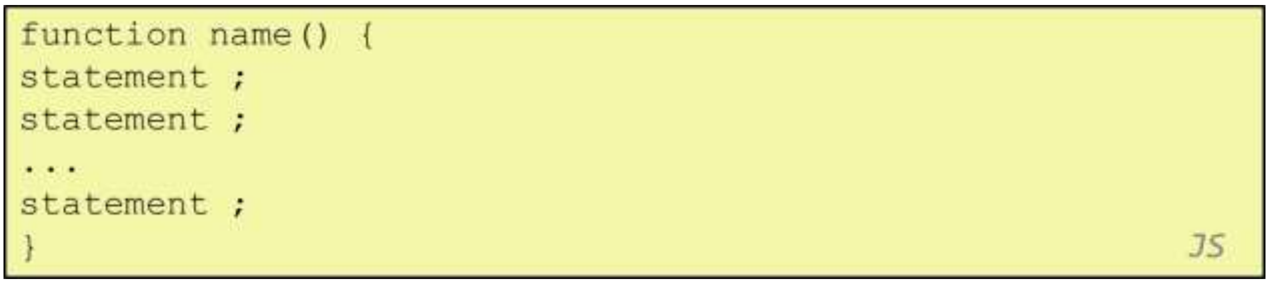
- you are used to programs start with a main method (or implicit main like in PHP)
- JavaScript programs instead wait for user actions called events and respond to them
- event-driven programming: writing programs driven by user events
- Let's write a page with a clickable button that pops up a "Hello, World" window...

Buttons

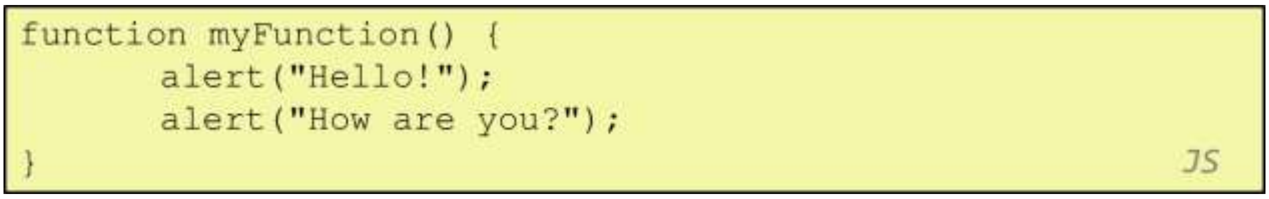
```html
<button>Click me!</button>                                HTML
```

- button's text appears inside tag; can also contain images
- To make a responsive button or other UI control:
- choose the control (e.g. button) and event (e.g. mouse 1. click) of interest
- write a JavaScript function to run when the event occurs
- attach the function to the event on the control

JavaScript functions



```
function name() {
statement ;
statement ;
...
statement ;
}
```



```
function myFunction() {
        alert("Hello!");
        alert("How are you?");
}
```

❑ the above could be the contents of example.js linked to our HTML page
❑ statements placed into functions can be evaluated in response to user events

Event handlers

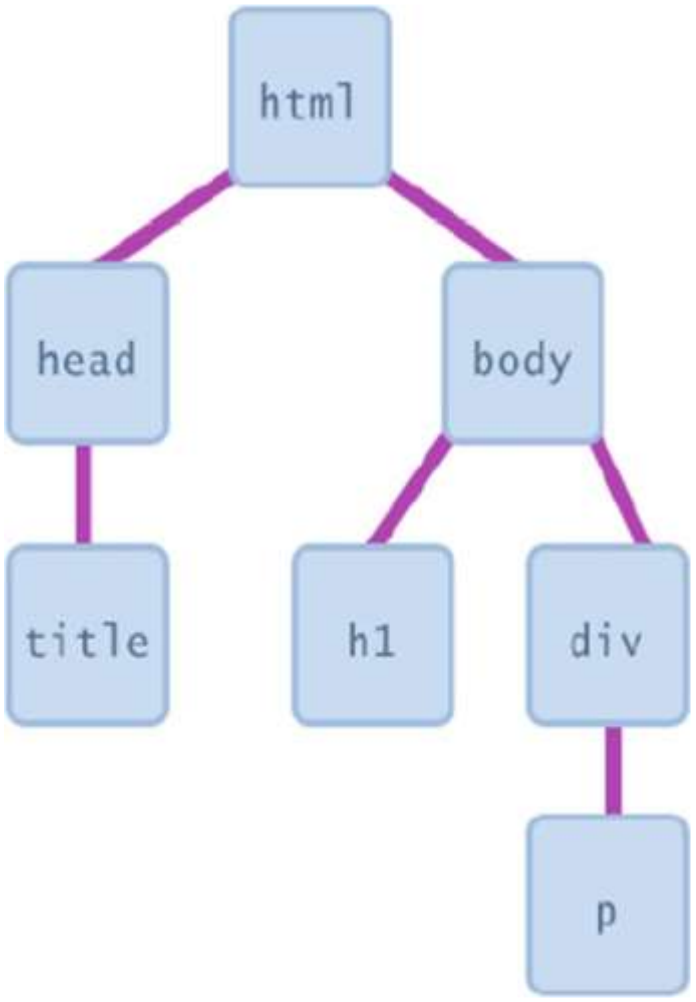
```html
<element attributes onclick="function();">...
```
HTML

```html
<button onclick="myFunction();">Click me!</button>
```
HTML

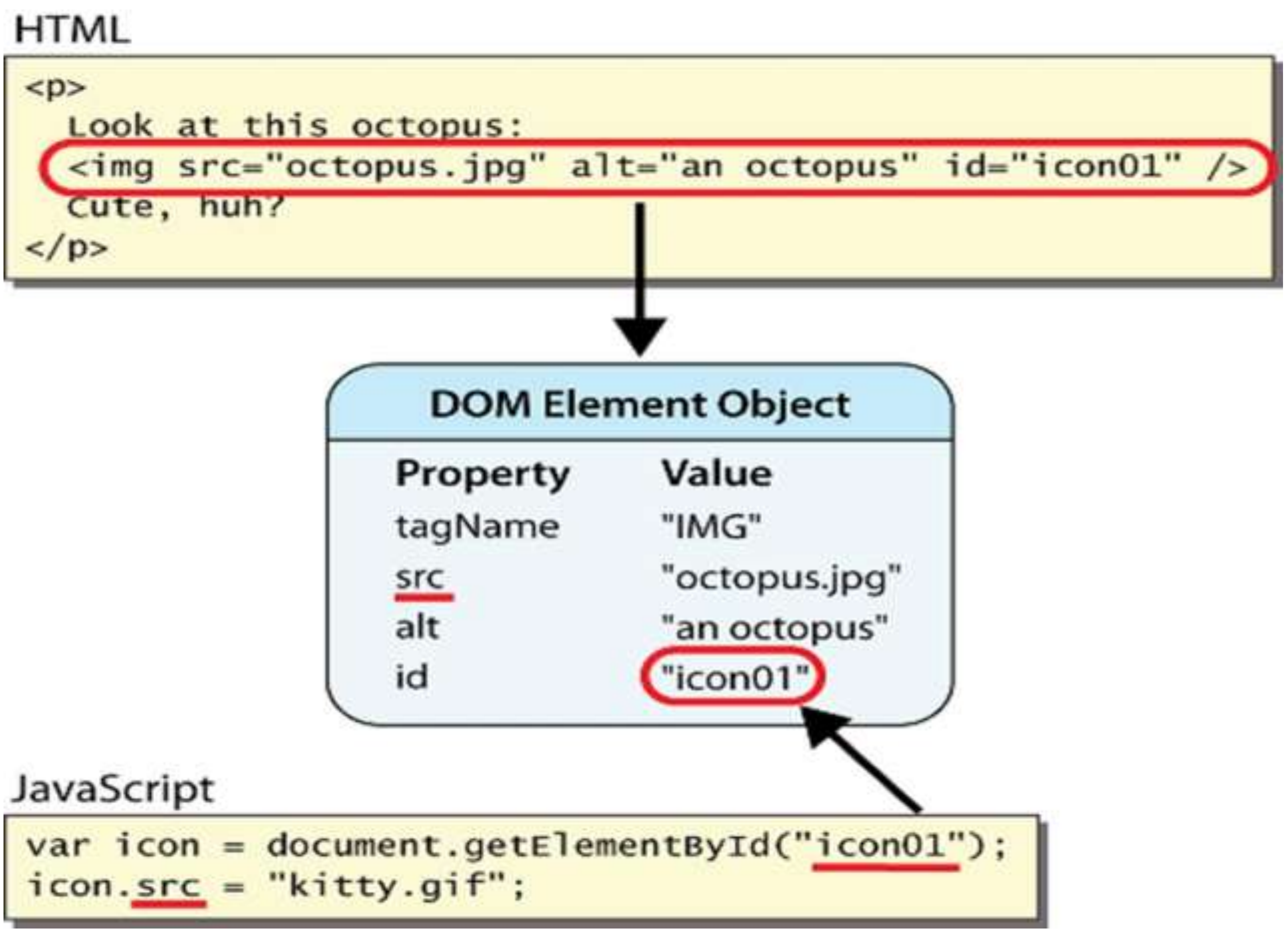
❑ JavaScript functions can be set as event handlers
❑ when you interact with the element, the function will execute
❑ onclick is just one of many event HTML attributes we'll use
❑ but popping up an alert window is disruptive and annoying
❑ A better user experience would be to have the message appear on the page...

Document Object Model (DOM)

- ❑ most JS code manipulates elements on an HTML page
- ❑ we can examine elements' state
- ❑ e.g. see whether a box is checked
- ❑ we can change state
- ❑ e.g. insert some new text into a div
- ❑ we can change styles
- ❑ e.g. make a paragraph red



DOM element objects


HTML

```
<p>
  Look at this octopus:
  <img src="octopus.jpg" alt="an octopus" id="icon01" />
  Cute, huh?
</p>
```

DOM Element Object

Property	Value
tagName	"IMG"
src	"octopus.jpg"
alt	"an octopus"
id	"icon01"

JavaScript

```
var icon = document.getElementById("icon01");
icon.src = "kitty.gif";
```

Accessing elements:

document.getElementById

```
var name = document.getElementById("id");
                                                          JS
```

```
<button onclick="changeText();">Click me!</button>
<span id="output">replace me</span>
<input id="textbox" type="text" />                              HTML
```

```
function changeText() {
       var span = document.getElementById("output");
       var textBox = document.getElementById("textbox");

        textbox.style.color = "red";

}                                                                 JS
```

Accessing elements: document.getElementById

document.getElementById returns the DOM object for an element with a given id
can change the text inside most elements by setting the innerHTML property
can change the text in form controls by setting the value property

Variables

```
var name = expression;                                JS
```

```
var clientName = "Connie Client";
var age = 32;
var weight = 127.4;                                   JS
```

- ❑ variables are declared with the var keyword (case sensitive)
- ❑ types are not specified, but JS does have types ("loosely typed")
- ❑ Number, Boolean, String, Array, Object, Function, Null, Undefined
- ❑ can find out a variable's type by calling typeof

Comments (same as Java)

```
// single-line comment
/* multi-line comment */
                                                      JS
```

- ❑ identical to Java's comment syntax
- ❑ recall: 4 comment syntaxes
- ❑ HTML: <!-- comment -->
- ❑ CSS/JS/PHP: /* comment */
- ❑ Java/JS/PHP: // comment
- ❑ PHP: # comment

Chapter 3

Software Requirements Specification

3.1 Software

3.1.1 Browser

Firefox has always been known for its flexibility and support for extensions, but in recent years it had started to lag behind the competition in terms of speed. Firefox Quantum, first released last year, represented a total overhaul of the browser's code base, with speeds now comparable with Google Chrome. That's not just on top-end computers, either – the new Firefox makes frugal use of RAM, even with masses of tabs open.

. Mozilla Firefox

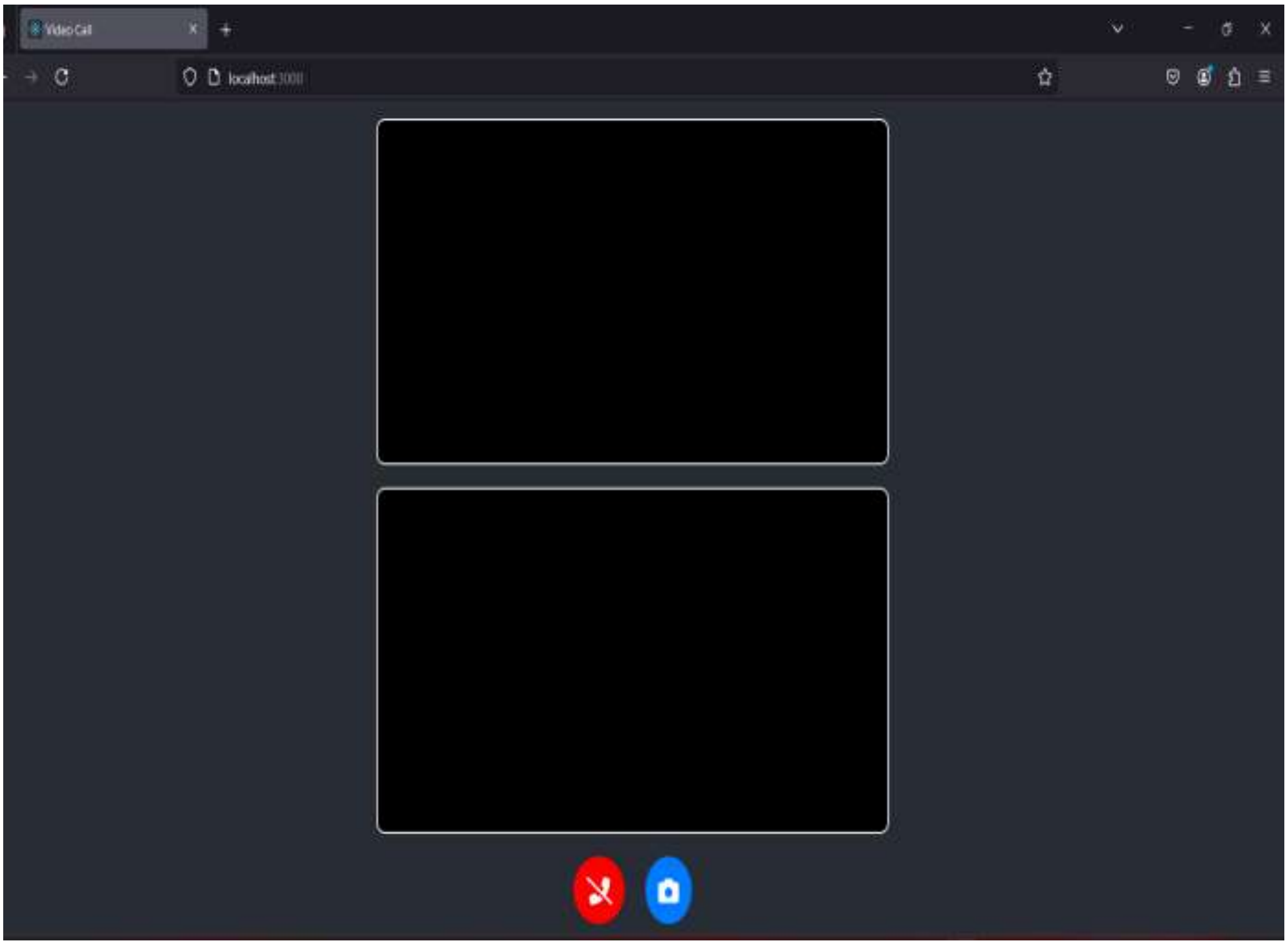
Mozilla Firefox is a free and open-source web browser developed by The Mozilla Foundation and its subsidiary, Mozilla Corporation. Firefox is avail- able for Windows, macOS, Linux, BSD, illumos and Solaris operating systems. Its sibling, Firefox for Android, is also available.

. Google Chrome

Project Details 4.1

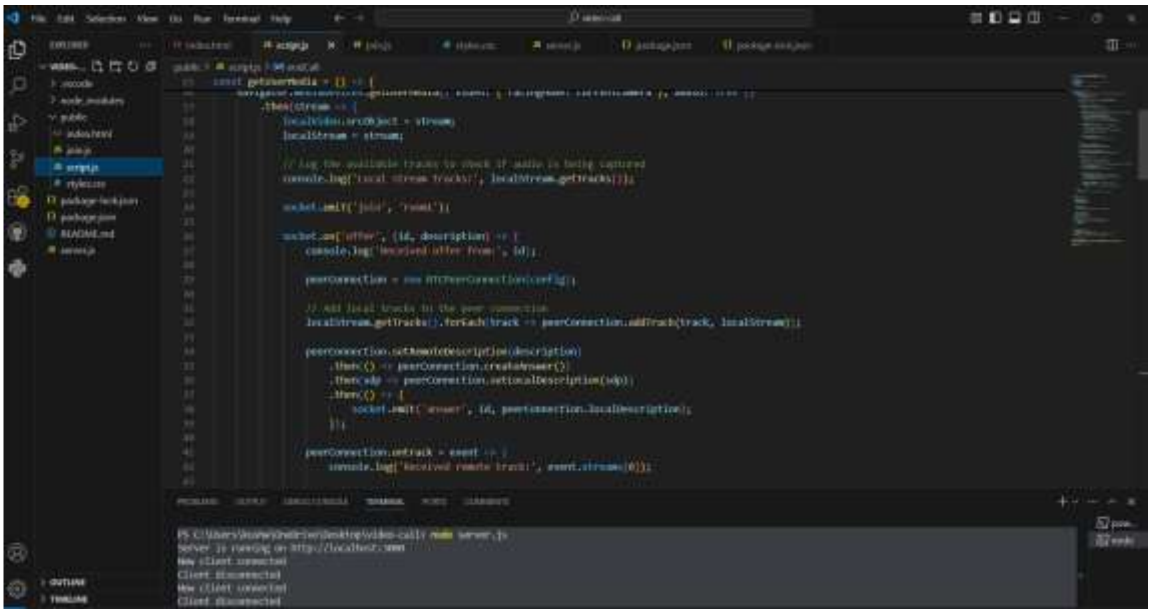
4.1 VIDEO CALL

4.1a Front-End Design



Chapter 4.1

Source Code



```
const getUserMedia = () => {
    navigator.mediaDevices.getUserMedia({ video: { facingMode: currentCamera }, audio: true })
        .then(stream => {
            localVideo.srcObject = stream;
            localStream = stream;

            // Log the available tracks to check if audio is being captured
            console.log('Local stream tracks:', localStream.getTracks());

            socket.emit('join', 'room1');

            socket.on('offer', (id, description) => {
                console.log('Received offer from:', id);

                peerConnection = new RTCPeerConnection(config);

                // Add local tracks to the peer connection
                localStream.getTracks().forEach(track => peerConnection.addTrack(track, localStream));

                peerConnection.setRemoteDescription(description)
                    .then(() => peerConnection.createAnswer())
                    .then(sdp => peerConnection.setLocalDescription(sdp))
                    .then(() => {
                        socket.emit('answer', id, peerConnection.localDescription);
                    });

                peerConnection.ontrack = event => {
                    console.log('Received remote track:', event.streams[0]);
```

```
PS C:\Users\[illegible]\OneDrive\Desktop\video-call> node server.js
Server is running on http://localhost:3000
New client connected
Client disconnected
New client connected
Client disconnected
```

Project Description

Project Title: Video Call Web Application

Overview:

This dynamic web application facilitates real-time video calling between users. The project implements core features like initiating and ending video calls, switching between front and rear cameras, and managing peer-to-peer connections. The system uses modern web technologies like HTML, CSS, JavaScript, and WebRTC for real-time communication, with Node.js, Express.js, and Socket.IO for backend and signaling functionalities.

Application Key Components:

1. index.html – The main video call interface.
2. join.js – Facilitates joining a specific video call room.
3. script.js – Handles WebRTC connections and video call functionalities.
4. server.js – The backend server managing WebSocket signaling.

Code Explanation

1. `index.html` (Video Call Interface)

Purpose:
This HTML file provides the user interface for initiating and managing video calls.

Key Features:

Video Elements: Two video elements display the local and remote video streams.
Buttons:
End Call Button – Ends the current video call.
Switch Camera Button – Switches between the front and back cameras.

2. `join.js` (Room Joining Logic)

Purpose:
This file allows users to join specific video call rooms.
Key Features:

- **Room Selection**: Users can join a room by entering a room ID.
- **Redirection:** Redirects users to the main call page with the selected room

3 `Script.js` (Video Call Logic)

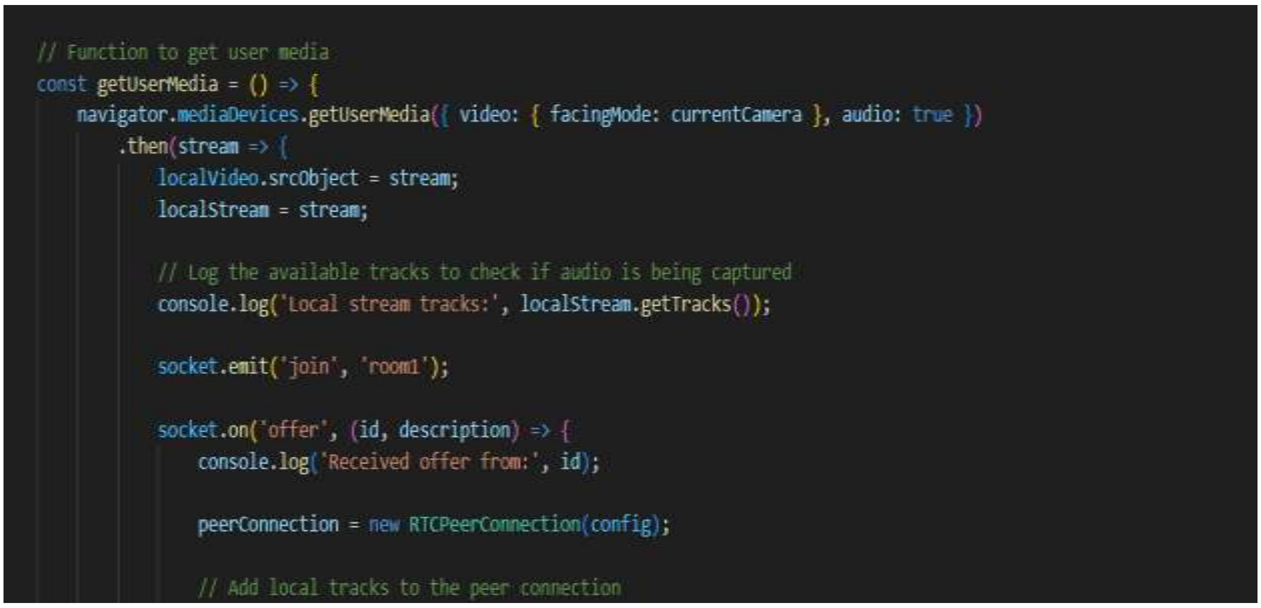
Purpose:

This file handles WebRTC peer-to-peer video call functionalities, including camera switching and call termination.

Key Features:

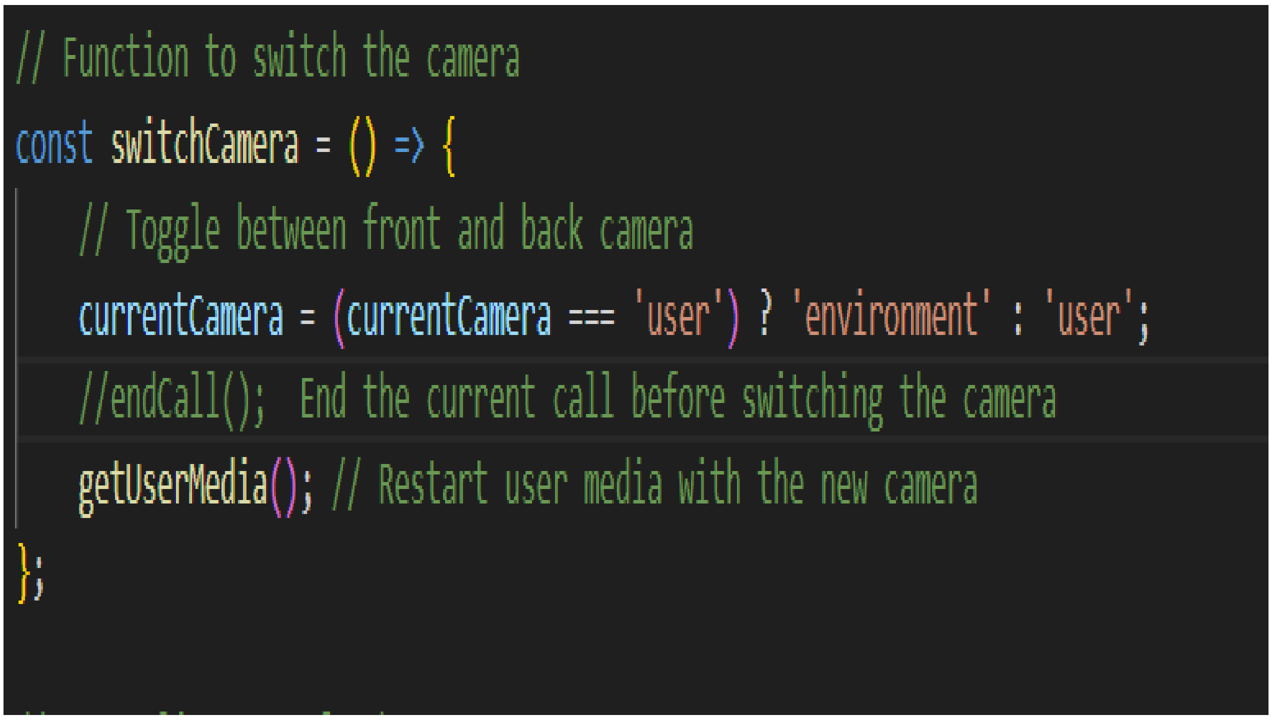
- WebRTC Connection: Establishes a peer-to-peer connection using STUN servers.
- Camera Switching: Toggles between front and rear cameras.
- Call Termination: Ends the call and releases media resources.
- Signaling with Socket.IO: Manages communication with the signaling server.

Getting User Media:



```
// Function to get user media
const getUserMedia = () => {
    navigator.mediaDevices.getUserMedia({ video: { facingMode: currentCamera }, audio: true })
        .then(stream => {
            localVideo.srcObject = stream;
            localStream = stream;

            // Log the available tracks to check if audio is being captured
            console.log('Local stream tracks:', localStream.getTracks());

            socket.emit('join', 'room1');

            socket.on('offer', (id, description) => {
                console.log('Received offer from:', id);

                peerConnection = new RTCPeerConnection(config);

                // Add local tracks to the peer connection
```

Switching Cameras:



```
// Function to switch the camera
const switchCamera = () => {
    // Toggle between front and back camera
    currentCamera = (currentCamera === 'user') ? 'environment' : 'user';
    //endCall();  End the current call before switching the camera
    getUserMedia(); // Restart user media with the new camera
};
```

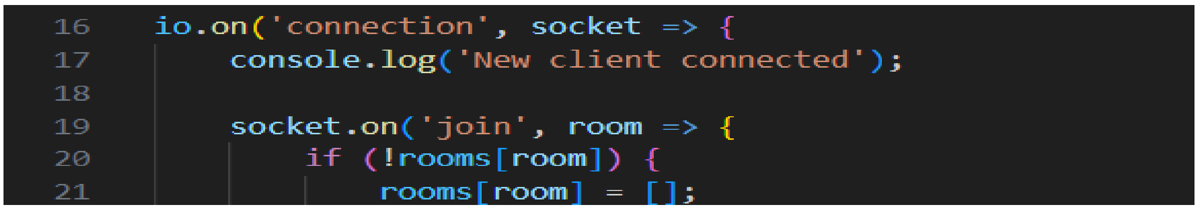
4 Script.js

Purpose:
Manages WebSocket signaling for the video calls and handles room connections.

Key Features:

- Signaling: Handles offer, answer, and candidate events for WebRTC connections.
- Room Management: Manages users joining and leaving rooms.
- Socket.IO Events: Facilitates real-time communication.

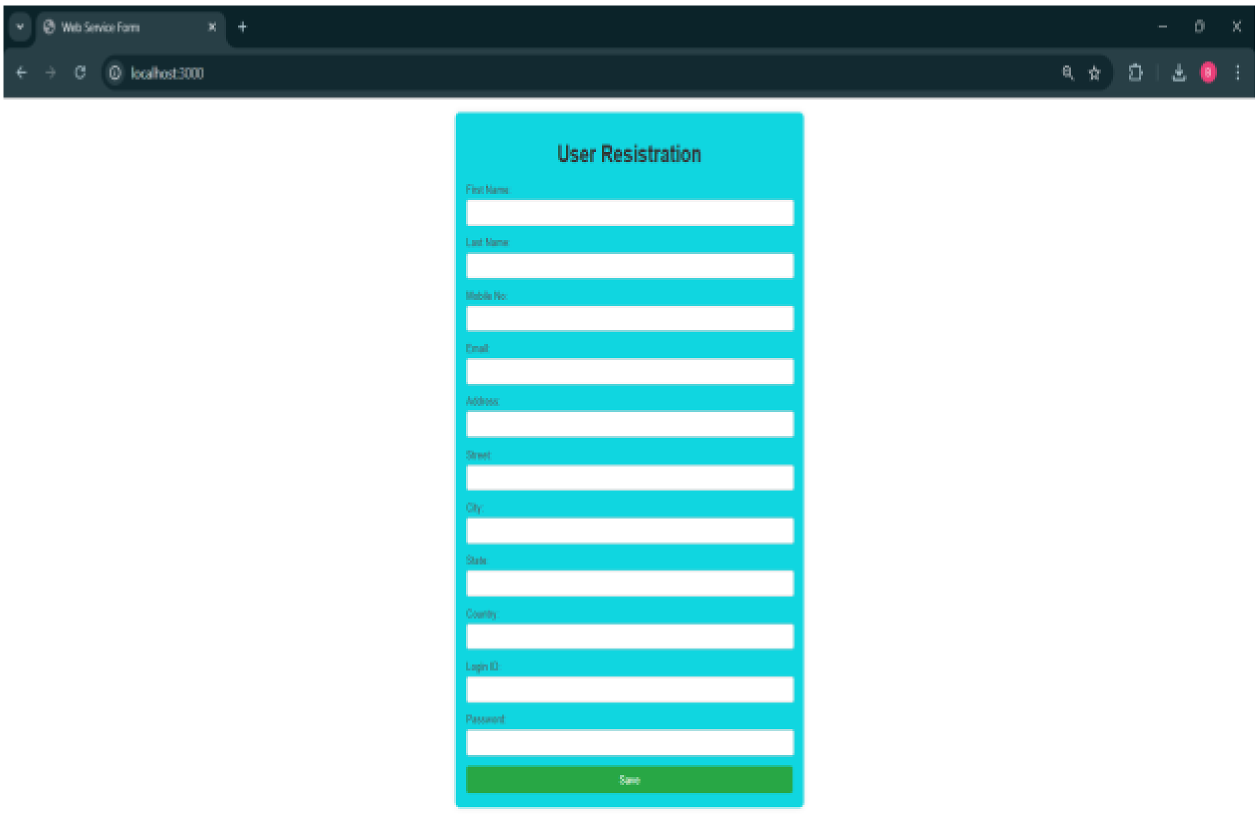
Handling New Connections:



```
io.on('connection', socket => {
    console.log('New client connected');

    socket.on('join', room => {
        if (!rooms[room]) {
            rooms[room] = [];
```

Chapter 4.2

User Management Web Application



User Resistration

First Name:

Last Name:

Mobile No:

Email:

Address:

Street:

City:

State:

Country:

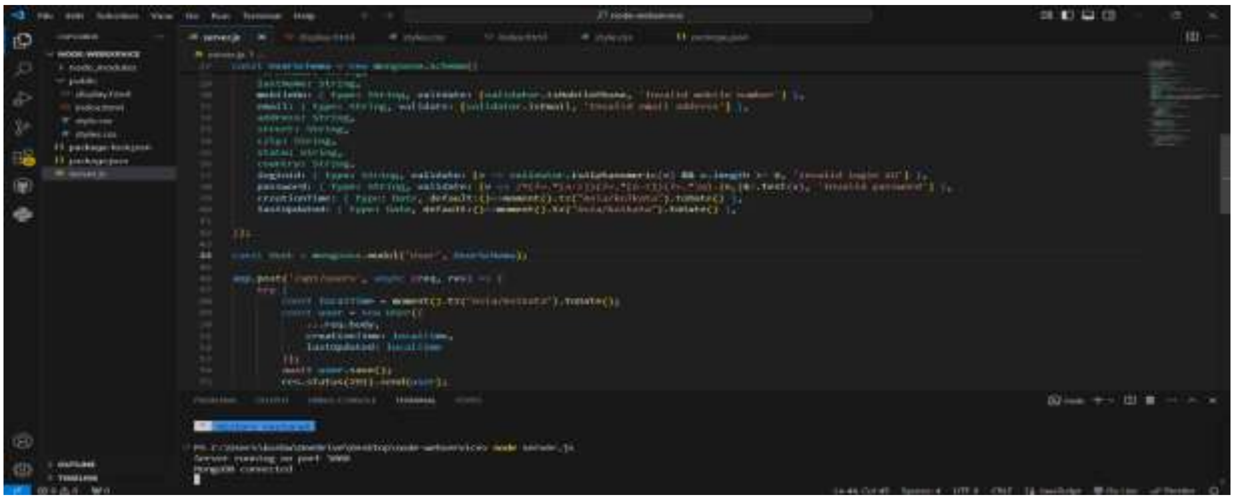
Login ID:

Password:

Save

Chapter 4.2

Source Code



```
const UserSchema = new mongoose.Schema({
    lastName: String,
    mobileNo: { type: String, validate: [validator.isMobilePhone, 'Invalid mobile number'] },
    email: { type: String, validate: [validator.isEmail, 'Invalid email address'] },
    address: String,
    street: String,
    city: String,
    state: String,
    country: String,
    loginId: { type: String, validate: [v => validator.isAlphanumeric(v) && v.length >= 8, 'Invalid login ID'] },
    password: { type: String, validate: [v => /[illegible]/.test(v), 'Invalid password'] },
    creationTime: { type: Date, default:()=>moment().tz('Asia/Kolkata').toDate() },
    lastUpdated: { type: Date, default:()=>moment().tz('Asia/Kolkata').toDate() },

});

const User = mongoose.model('User', UserSchema);

app.post('/api/users', async (req, res) => {
    try {
        const localTime = moment().tz('Asia/Kolkata').toDate();
        const user = new User({
            ...req.body,
            creationTime: localTime,
            lastUpdated: localTime
        });
        await user.save();
        res.status(201).send(user);
```

```
PS C:\Users\[illegible]\OneDrive\Desktop\node-webservices> node server.js
Server running on port 3000
MongoDB connected
```

Overview:

It is a dynamic web application that handles the registration and management of users. CRUD (Create, Read, Update, Delete) functionalities have been implemented in this application. A user-friendly interface has been provided for users to add, view, and delete.

The key components of the application are:

1. User Registration Form (index.html)
2. User Display Page (display.html)
3. Backend Server (server.js) – which interacts with MongoDB.

This project has been built using modern web technologies like HTML, CSS, JavaScript, jQuery, Node.js, Express.js, and MongoDB.

Code Explanation

1. `index.html` (User Registration Page)

Purpose: This page provides a form for users to register.

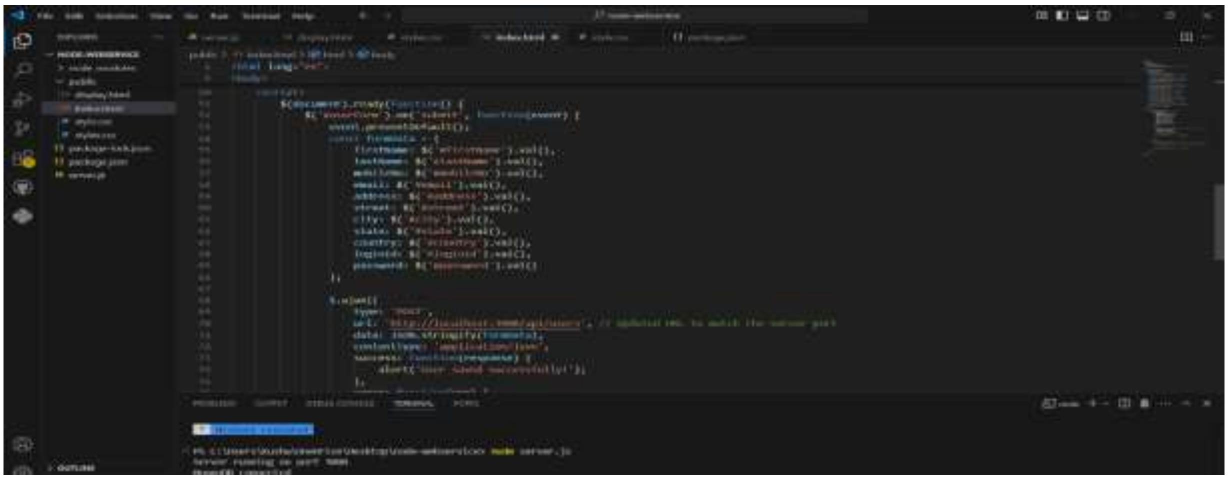
Key Features:

Form Fields: There are form fields to collect user details like:

First Name, Last Name, Email, Mobile No, Address, Login ID, and Password.

Form Submission: When submitting a form using jQuery, the data is sent to the backend API (http://localhost:3000/api/users) through a POST request.

Validation: Required fields ensure that all fields have been filled.



```
<html lang="en">
<head>
    <script>
        $(document).ready(function() {
            $('#userForm').on('submit', function(event) {
                event.preventDefault();
                const formData = {
                    firstName: $('#firstName').val(),
                    lastName: $('#lastName').val(),
                    mobileNo: $('#mobileNo').val(),
                    email: $('#email').val(),
                    address: $('#address').val(),
                    street: $('#street').val(),
                    city: $('#city').val(),
                    state: $('#state').val(),
                    country: $('#country').val(),
                    loginId: $('#loginId').val(),
                    password: $('#password').val()
                };

                $.ajax({
                    type: 'POST',
                    url: 'http://localhost:3000/api/users', // Updated URL to match the server port
                    data: JSON.stringify(formData),
                    contentType: 'application/json',
                    success: function(response) {
                        alert('User saved successfully!');
                    },
```

```
PS C:\Users\[illegible]\Desktop\node-webservice> node server.js
Server running on port 3000
MongoDB connected
```

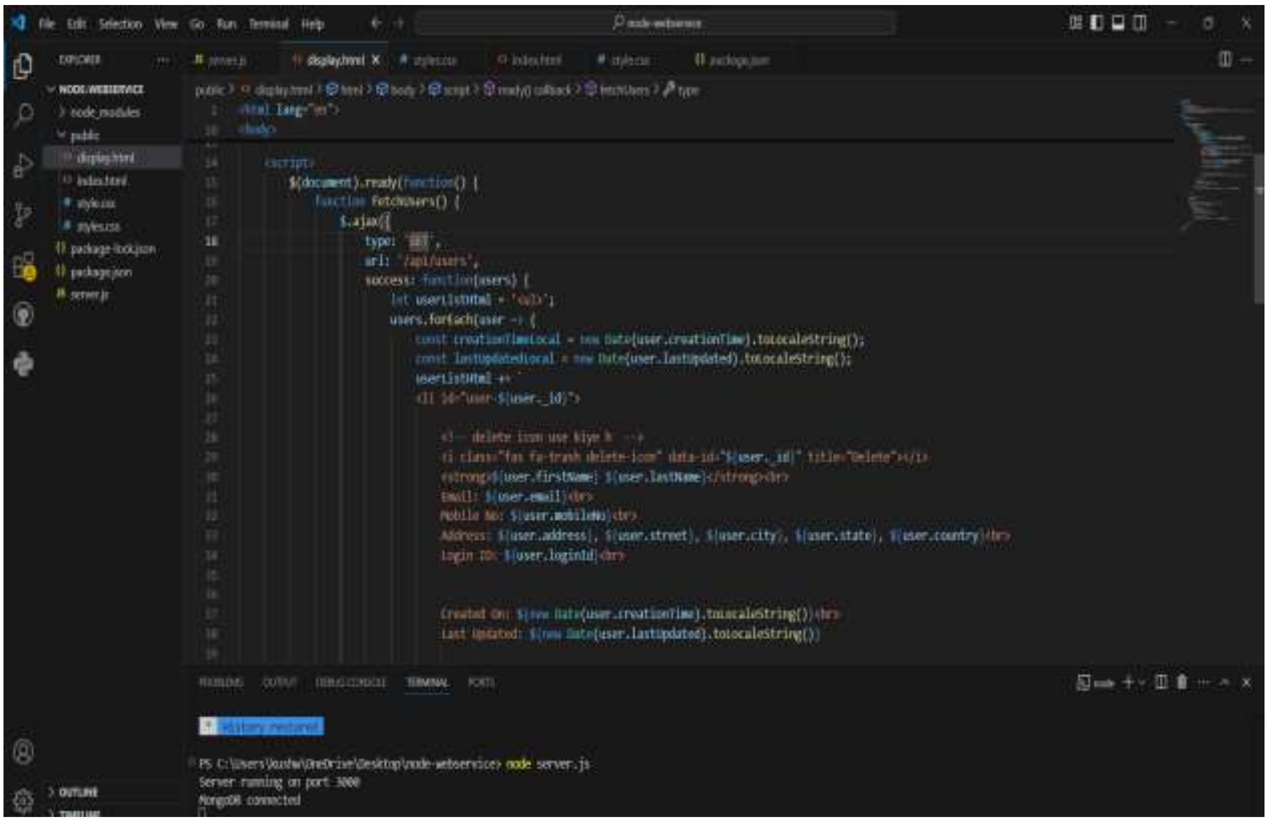
2. `display.html` (User Display Page)

Purpose: This page is used to list registered users and delete them.

Key Features:

Dynamic User List: Users are fetched from /api/users endpoint using jQuery and AJAX and displayed.

Delete Functionality: Delete icon is given to users to delete. When a delete request is sent, the entry is removed from the user list.


```
<html lang="en">
<body>

    <script>
        $(document).ready(function() {
            function fetchUsers() {
                $.ajax({
                    type: 'GET',
                    url: '/api/users',
                    success: function(users) {
                        let userListHtml = '<ul>';
                        users.forEach(user => {
                            const creationTimeLocal = new Date(user.creationTime).toLocaleString();
                            const lastUpdatedLocal = new Date(user.lastUpdated).toLocaleString();
                            userListHtml += `
                            <li id="user-${user._id}">

                                <!-- delete icon use kiye h  -->
                                <i class="fas fa-trash delete-icon" data-id="${user._id}" title="Delete"></i>
                                <strong>${user.firstName} ${user.lastName}</strong><br>
                                Email: ${user.email}<br>
                                Mobile No: ${user.mobileNo}<br>
                                Address: ${user.address}, ${user.street}, ${user.city}, ${user.state}, ${user.country}<br>
                                Login ID: ${user.loginId}<br>


                                Created On: ${new Date(user.creationTime).toLocaleString()}<br>
                                Last Updated: ${new Date(user.lastUpdated).toLocaleString()}
```

```
History restored

PS C:\Users\kushw\OneDrive\Desktop\node-webservice> node server.js
Server running on port 3000
MongoDB connected
```

2. server.js (Backend Server)

Purpose: This server-side file saves, fetches, and deletes user data in the MongoDB database.

Key Features:

API Endpoints:

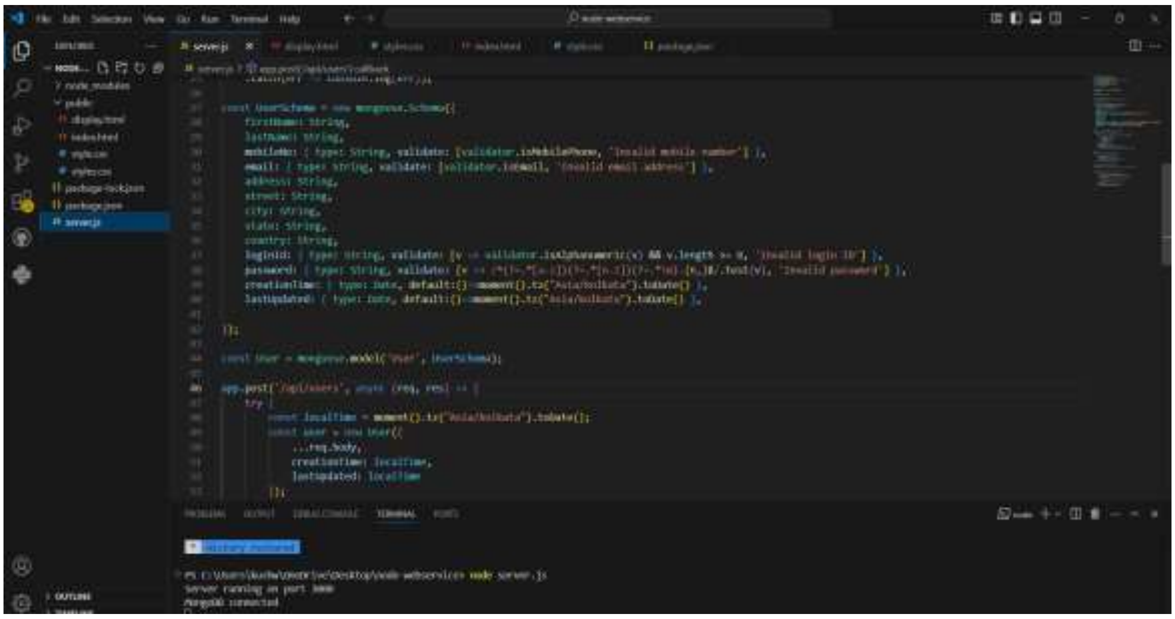
POST /api/users: To register a new user.

GET /api/users: To fetch all users.

DELETE /api/users/:id: To delete specific user.

Data Validation: Validation has been applied for Email, Mobile Number, and Password.

Timezone Handling: User creation and update times are stored in Indian Standard Time (IST) format.

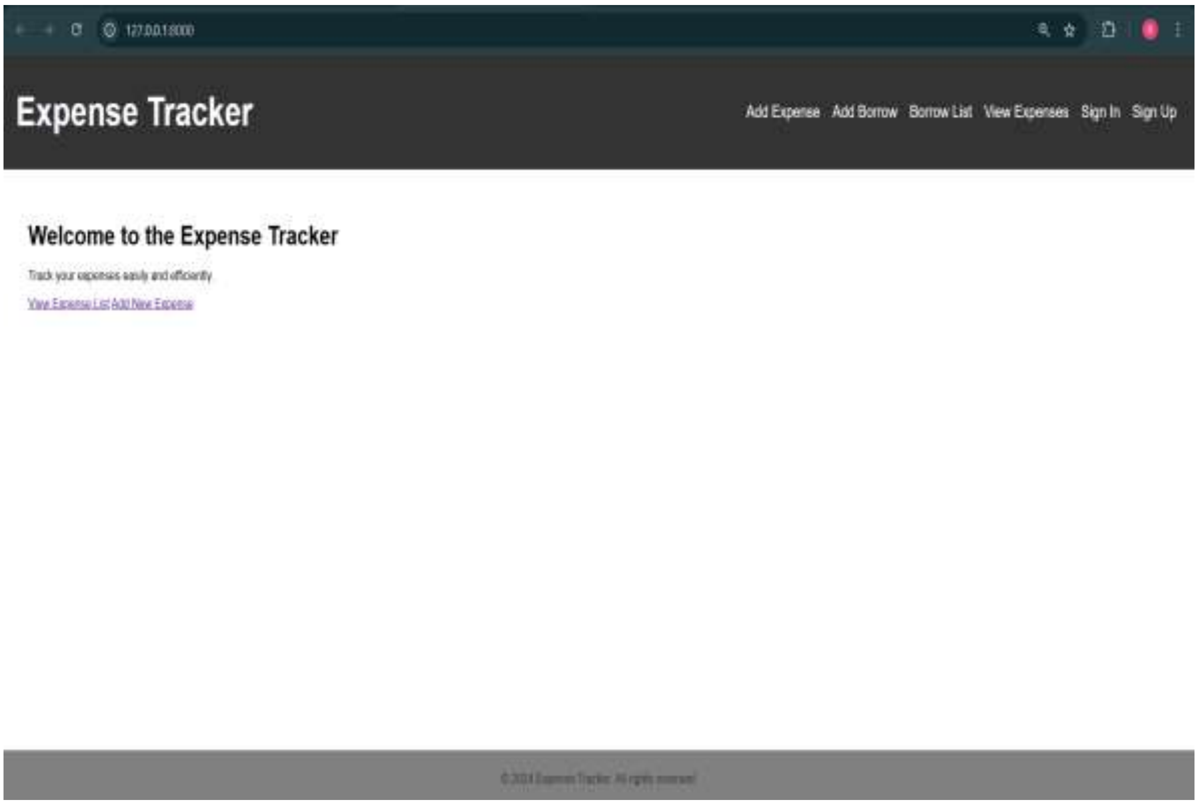


```
const UserSchema = new mongoose.Schema({
    firstName: String,
    lastName: String,
    mobileNo: { type: String, validate: [validator.isMobilePhone, 'Invalid mobile number'] },
    email: { type: String, validate: [validator.isEmail, 'Invalid email address'] },
    address: String,
    street: String,
    city: String,
    state: String,
    country: String,
    loginId: { type: String, validate: [v => validator.isAlphanumeric(v) && v.length >= 8, 'Invalid login ID'] },
    password: { type: String, validate: [v => /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d).{6,}$/.test(v), 'Invalid password'] },
    creationTime: { type: Date, default:()=>moment().tz("Asia/Kolkata").toDate() },
    lastUpdated: { type: Date, default:()=>moment().tz("Asia/Kolkata").toDate() },

});

const User = mongoose.model('User', UserSchema);

app.post('/api/users', async (req, res) => {
    try {
        const localTime = moment().tz("Asia/Kolkata").toDate();
        const user = new User({
            ...req.body,
            creationTime: localTime,
            lastUpdated: localTime
        });
```

```
PS C:\Users\[illegible]\OneDrive\Desktop\node-webservice> node server.js
Server running on port 3000
MongoDB connected
```

Chapter 4.3

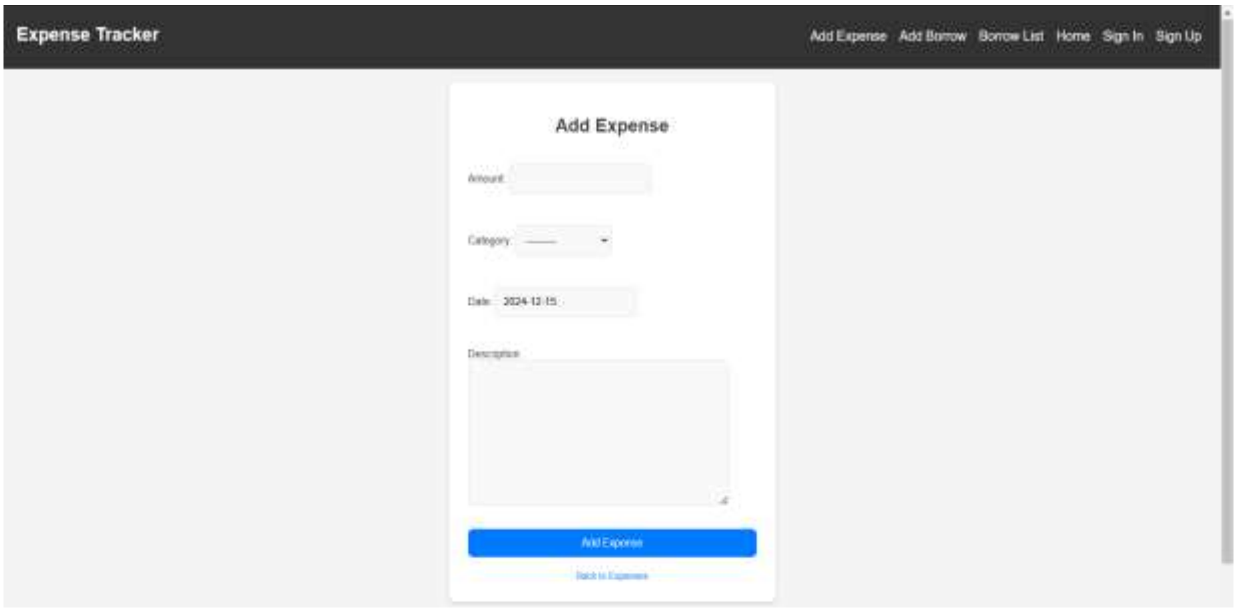
Expence Tracker.

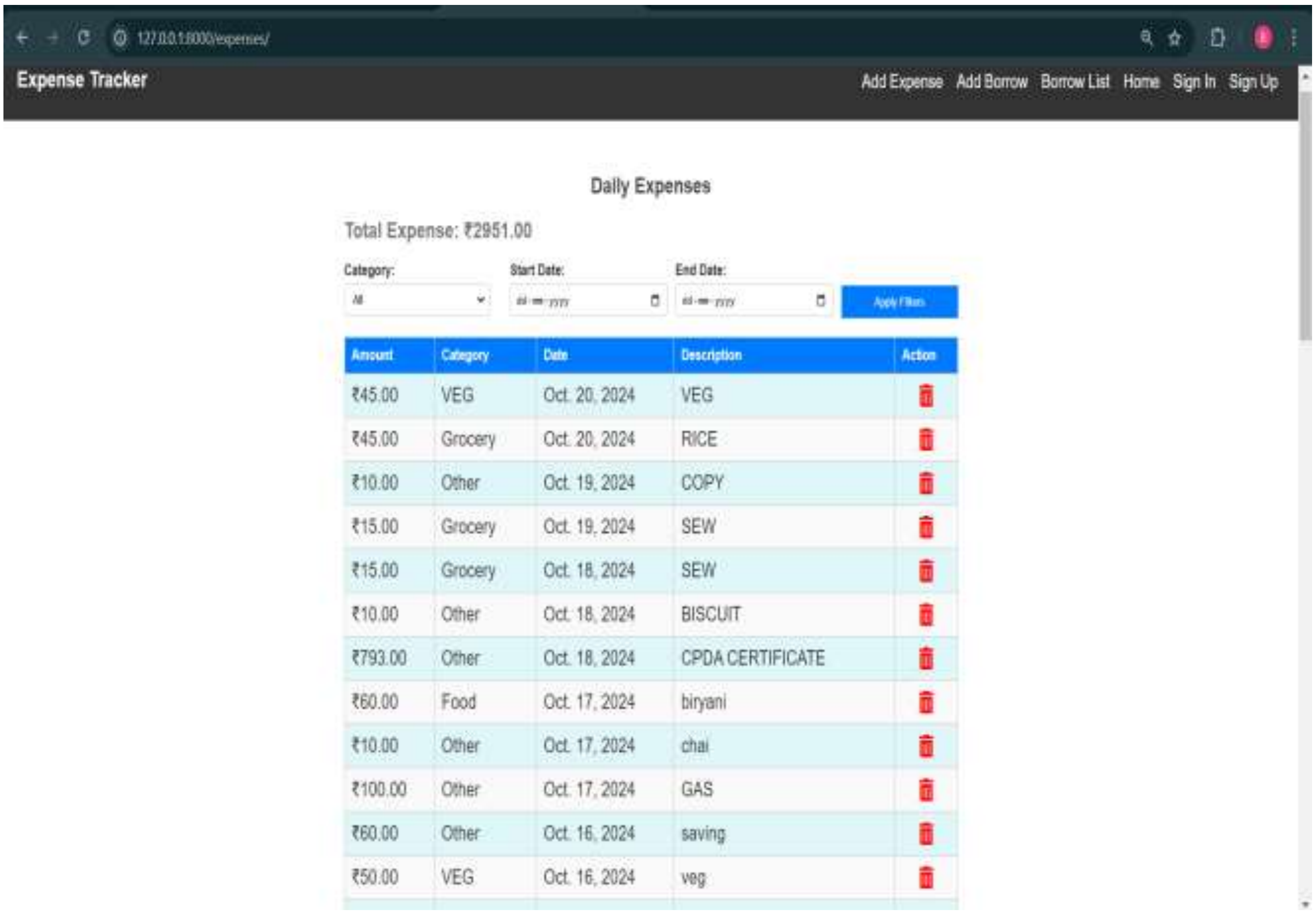


Welcome to the Expense Tracker

Track your expenses easily and efficiently

© 2024 Expense Tracker. All rights reserved.





127.0.0.1:8000/expenses/

Expense Tracker

Add Expense Add Borrow Borrow List Home Sign In Sign Up

Daily Expenses

Total Expense: ₹2951.00

Category: All

Start Date: dd-mm-yyyy

End Date: dd-mm-yyyy

Apply Filters

Amount	Category	Date	Description	Action
₹45.00	VEG	Oct. 20, 2024	VEG	
₹45.00	Grocery	Oct. 20, 2024	RICE	
₹10.00	Other	Oct. 19, 2024	COPY	
₹15.00	Grocery	Oct. 19, 2024	SEW	
₹15.00	Grocery	Oct. 18, 2024	SEW	
₹10.00	Other	Oct. 18, 2024	BISCUIT	
₹793.00	Other	Oct. 18, 2024	CPDA CERTIFICATE	
₹60.00	Food	Oct. 17, 2024	biryani	
₹10.00	Other	Oct. 17, 2024	chai	
₹100.00	Other	Oct. 17, 2024	GAS	
₹60.00	Other	Oct. 16, 2024	saving	
₹50.00	VEG	Oct. 16, 2024	veg	

Expense Tracker Web Application

Overview:

This Django-based **Expense Tracker** application helps users manage their daily expenses. The project allows users to **add, view, and filter expenses** by various categories. It features an intuitive interface with forms for adding expenses and a detailed view to see total expenses and filter them by category.

Application Features:

Home Page:

- A simple homepage with a **navigation bar**.
- Navigation options:
  Add Expense
  View Expenses

Add Expense Page:

- A form to input expense details:
- Amount: The expense amount.
- Category: The type of expense (e.g., food, travel).
- Date: The date of the expense.
- Description: Additional details about the expense

View Expenses Page:

- Displays all recorded expenses.
- Shows the total amount spent.
- Provides a filtering option to view expenses by specific categories.

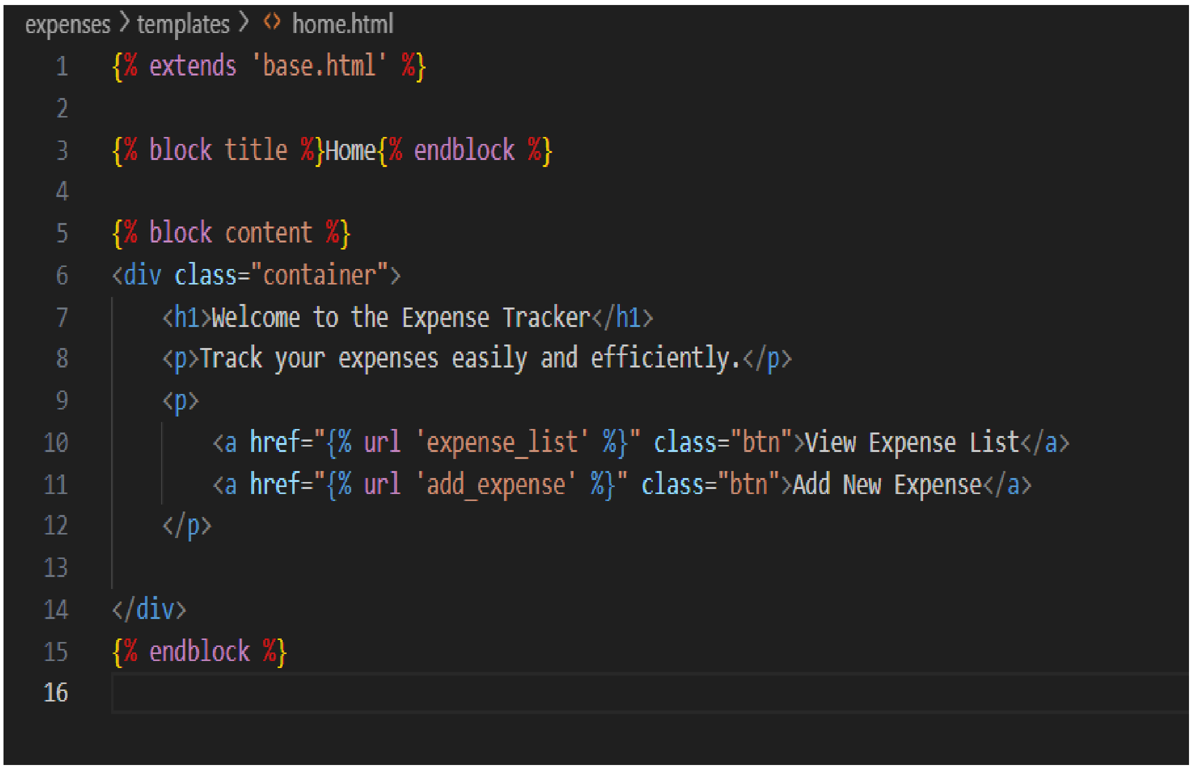
Categories:

Predefined categories include:

- Food
- Travel
- Shopping
- Grocery
- Vegetables
- Milk
- Juice

Code

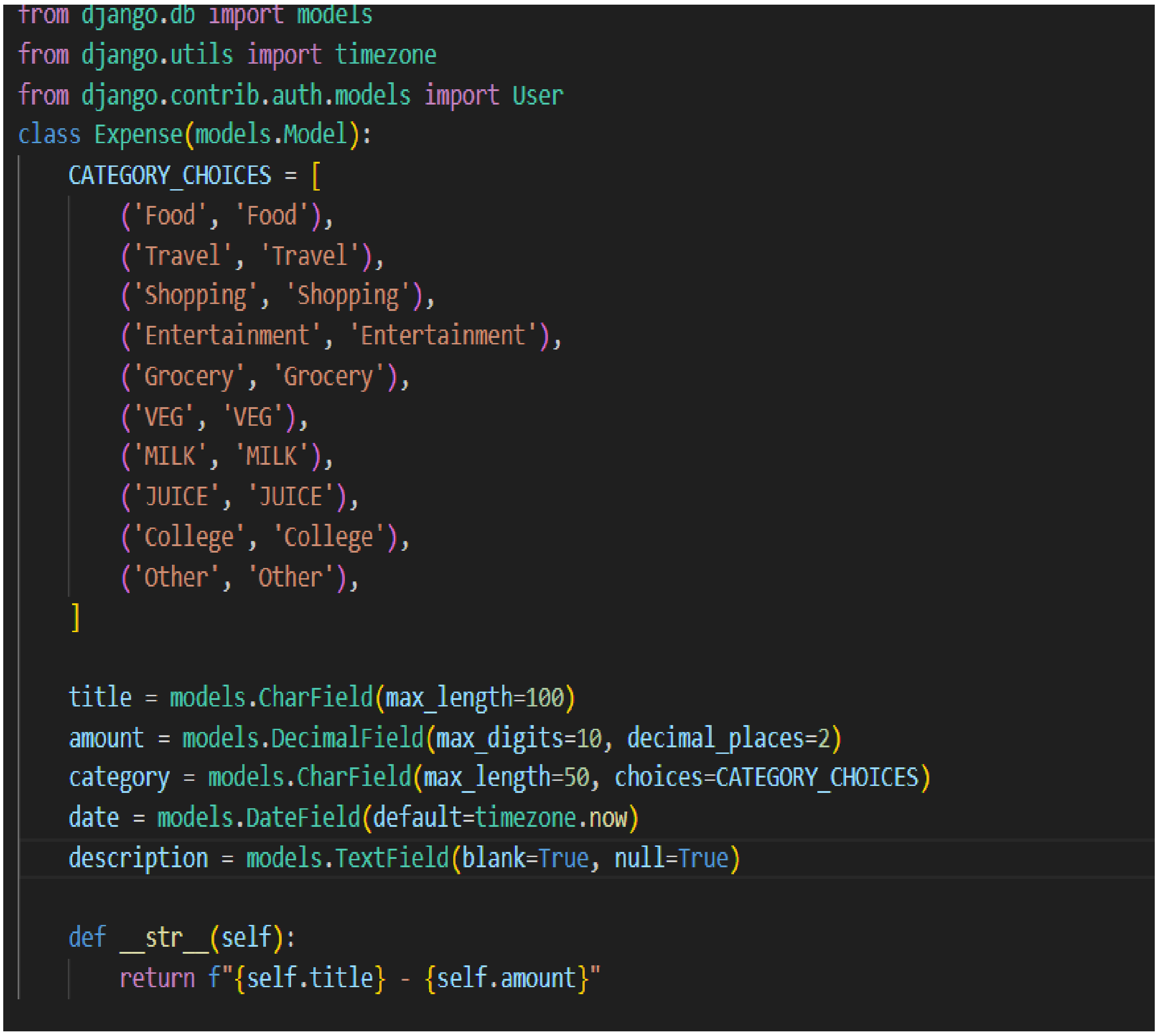
Home.html



expenses > templates > home.html

```
{% extends 'base.html' %}

{% block title %}Home{% endblock %}

{% block content %}
<div class="container">
    <h1>Welcome to the Expense Tracker</h1>
    <p>Track your expenses easily and efficiently.</p>
    <p>
        <a href="{% url 'expense_list' %}" class="btn">View Expense List</a>
        <a href="{% url 'add_expense' %}" class="btn">Add New Expense</a>
    </p>

</div>
{% endblock %}

```

Models



```python
from django.db import models
from django.utils import timezone
from django.contrib.auth.models import User
class Expense(models.Model):
    CATEGORY_CHOICES = [
        ('Food', 'Food'),
        ('Travel', 'Travel'),
        ('Shopping', 'Shopping'),
        ('Entertainment', 'Entertainment'),
        ('Grocery', 'Grocery'),
        ('VEG', 'VEG'),
        ('MILK', 'MILK'),
        ('JUICE', 'JUICE'),
        ('College', 'College'),
        ('Other', 'Other'),
    ]

    title = models.CharField(max_length=100)
    amount = models.DecimalField(max_digits=10, decimal_places=2)
    category = models.CharField(max_length=50, choices=CATEGORY_CHOICES)
    date = models.DateField(default=timezone.now)
    description = models.TextField(blank=True, null=True)

    def __str__(self):
        return f"{self.title} - {self.amount}"
```

Chapter 5
Conclusion and Future Scope

4.1 Conclusion

This chapter summarizes the main success of this research work and discusses an about future research work to achieve the ultimate goal in the field of performance of web accessibility, web security, load balancing and collective intelligence. An in depth literature survey was carried out and the critical analysis of the same raised the following major shortcomings and challenges in the web-centric query optimization techniques.

1. Network is congested due to heterogeneous data (i.e., text, images, videos etc), heavy weight of web applications and repetition of queries. Due to theseproblems the access time of web applications is very high, which reduces the overall performance of the web.

2. Web-centric queries are neither efficient nor secure.

3. Huge information is available on the servers but the load on servers is not still balanced. In industry the developers distribute the huge information of servers by introducing more servers which produce requirement of the collective intelligence. Further, to search the efficient and relevant server is also a big challenge. The diagnostic thought to above challenges guides towards the design of following efficient approach, model and frameworks:

Portable Extended Cache Memory to Reduce Web Traffic (PECA)

In this approach, it is desired to conserve the heavy data at the client side. The experiments were performed on few dummy web sites of different sizes while saving 96 heavy data at client side. Difference in the access times of different websites via traditional method and with the proposed approach was compared. A major improvement in the access time was observed in contrast to that by using traditional methods. Also, an attempt was made to reduce server load and network traffic congestion and it actually resulted into reduction of response time and hence an improved web performance could be observed.

. **Secure Web Access Model for sensitive data (SWAM)**

SWAM in the context of biometric recognition is being proposed. The proposed security model SWAM provide an interface to the authorized user's and reduce the threats regarding their sensitive areas. Online web services will be more secure using the online SWAM.

. **Collective Intelligence based Framework for Load Balancing of Web Servers**

Collective Intelligence based Framework for Load Balancing of Web Servers is being proposed. The aim of this work is to find the overall best server with shortest path and hence online balancing of web servers could be achieved with the help of collective intelligence based framework for online load balancing. The proposed concept is an extension of, rather than a replacement for, traditional exploration process.

4.2 Future Scope

The work covered in the thesis tries to solve various issues, which emerged as a result of literature survey. Still there are many unopened questions left and are of interest were identified and are mentioned below: 97

Web Security At Client Side

In this research work we proposed PECA, but it still has shortcomings. For example, when web document is required to be save in the portable extended memory it reduces the security due to decentralization of data. During updates at client side malicious codes may transfer to the client machine. So, a web security framework is required at the client side to make PECA more secure and better performer. So in near future, PECA and SWAM may be merged.

Server Side Load Balancing

Load balancing is a concept which is still under research. Everyday new frameworks, algorithms and models are being developed and existing models are updated. There is a vast scope for future enhancement. For example, the usersare sending arbitrary data as a query on the web, and hence web-centric queries can be optimized at server level to reduce server load to improve the server performance. Further, implementation of our work is pending and hence an improvement may be recommended in the same.

Ant Based Technology for Collective Intelligence At Server Side

The developers are distributing server's data to reduce the server's load. This mechanism is increasing the requirement of collective intelligence. In this research work, ant technology for collective intelligence is used, but this is framework is based on client's query. There are still many unopened questions. For example, what and how much data is available in near servers? Are these servers are reliable? What is the credibility of these servers? In particular the server must have all the relevant information of their relative servers.

References

1. https:// www. w 3 schools. com
2. https://www.geeksforgeeks.org
3. https://www.wikipedia.org/