Task Manager

Website Development Finals

Group Members:

Sampang, Michael Angelo

Santos, Nathaniel

Hernandez Liam

Domingo Eilice Eindyel

Objective:

The objective of the code is to create a collaborative to-do list application. Users can add tasks with deadlines, view a list of tasks, mark tasks as complete, and delete tasks. The application uses Web Sockets to keep the task list synchronized across all connected users in real-time.

Functionalities:

- **Task Management:** Add, complete, delete tasks.
- **Real-time Collaboration:** Synchronize task list updates between users.
- **Deadline Reminders:** Schedule notifications for expiring tasks.
- **Date & Time Display:** Show the current date and time.
- **Motivational Quotes:** Fetch and display a random motivational quote.
- **Verse of the Day:** Fetch and display a verse from the Bible.

Features:

- **User Interface:** Provides a web-based interface for users to interact with the to-do list.
- **WebSocket Integration:** Enables real-time communication and synchronization between clients.
- **Task Deadlines:** Allows users to set deadlines for tasks.
- **Task Completion:** Users can mark tasks as completed.
- **Task Expiration:** Notifies users when tasks are approaching or past their deadlines.
- **Motivational Content:** Provides inspirational quotes to users.
- **Religious Content:** Integrates with an API to display a daily verse.

Overall, this code demonstrates a well-structured web application with both front-end and back-end components working together to achieve a collaborative to-do list experience.

Structure:

The code consists of two main parts:

1. **HTML/JavaScript Frontend:** This part is responsible for the user interface and interaction. It defines the HTML structure of the web page using HTML tags and styles it uses CSS (linked in the `<head>` section). It also includes JavaScript code to:

```
<!DOCTYPE html>
<html>
<head>
    <title>To-Do List Application</title>
    <link rel="stylesheet" type="text/css" href="style.css">
    <link
href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=swap"
rel="stylesheet">
    <script src="script.js" defer></script>
</head>
<body>
    <div class="container">
        <h1>To-Do List</h1>
        <div class="datetime-container">
            <p id="date"></p>
            <p id="time"></p>
        </div>
        <div class="verse-of-the-day-container">
            <div class="verse-container">
                <p id="verse"></p>
            </div>
        </div>
        <div class="input-container">
            <input type="text" id="nameInput" placeholder="Your
Name">             <input type="text" id="taskInput" placeholder="Add new
task...">
            <input type="datetime-local" id="taskDeadline">
            <button onclick="addTask()">Add</button>
        </div>
        <ul id="taskList"></ul>
    </div>
    <div class="motivational-quote-container">
        <h2>Here's a Quote for You!</h2>
        <p id="motivationalQuote">Loading quote...</p>
    </div>
</body>
</html>
```

2.WebSocket Server (Node.js): This part runs on the server and manages the communication between multiple clients (web browsers). It uses the WebSocket protocol to establish real-time, two-way communication. Here's what it does:

- Listens for new client connections on port 8080.
- Broadcasts the list of existing tasks to new clients when they connect.
- Receives messages from clients regarding adding, completing, or deleting tasks.
- Updates the internal task list based on the received messages.
- Broadcasts updates about task changes (add, complete, delete, expire) to all connected clients.
- Scheduled task expiration by setting a timeout based on the deadline. When the deadline arrives, it broadcasts the expiration and removes the task from the server-side list.

```
const WebSocket = require('ws');

const wss = new WebSocket.Server({ port: 8080 });

let tasks = [];

wss.on('connection', (ws) => {
    console.log('Client connected');

    ws.send(JSON.stringify({ type: 'init', tasks }));

    ws.on('message', (message) => {
        const data = JSON.parse(message);

        if (data.type === 'add') {
            const { name, task } = data;
            tasks.push({ name, task });
            broadcast({ type: 'add', name, task });
            scheduleTaskExpiration(task);
        } else if (data.type === 'complete') {
            const taskIndex = tasks.findIndex(entry => entry.task.id ===
data.id);
            if (taskIndex !== -1) {
                tasks[taskIndex].task.completed = true;
                broadcast({ type: 'complete', id: data.id });
            }
        } else if (data.type === 'delete') {
            tasks = tasks.filter(entry => entry.task.id !== data.id);
            broadcast({ type: 'delete', id: data.id });
        } else if (data.type === 'expire') {
            tasks = tasks.filter(entry => entry.task.id !== data.id);
            broadcast({ type: 'expire', id: data.id });
        }
```

```
    });

    ws.on('close', () => {
        console.log('Client disconnected');
    });
});

function broadcast(data) {
    wss.clients.forEach((client) => {
        if (client.readyState === WebSocket.OPEN) {
            client.send(JSON.stringify(data));
        }
    });
}

function scheduleTaskExpiration(task) {
    const now = Date.now();
    if (task.deadline <= now) {
        broadcast({ type: 'expire', id: task.id });
    } else {
        setTimeout(() => {
            tasks = tasks.filter(entry => entry.task.id !== task.id);
            broadcast({ type: 'expire', id: task.id });
        }, task.deadline - now);
    }
}

console.log('WebSocket server running on ws://localhost:8080');
```

3. **JavaScript:** The JavaScript code handles various functionalities of the to-do list application:

- **WebSocket Connection:** Establishes a WebSocket connection with the server at `ws://localhost:8080`.
- **Task Management:**
  - **Initialization:** Fetches existing tasks from local storage and displays them on the page.
  - **Adding Tasks:** Validates user input, creates a task object with name, text, deadline, completion status, and notification status, sends the task data to the server via WebSocket.
  - **Completing Tasks:** Updates the task UI (strikethrough, color change), sends a "complete" message to the server with the task ID.
  - **Deleting Tasks:** Removes the task element from the UI, sends a "delete" message with the task ID to the server.
- **Task Expiration:**
  - **Checking Expiration:** Verifies if a task's deadline has passed during initialization and periodically.
  - **Notification:** Displays a desktop notification (if supported) and updates the task UI with a red background color for expired tasks.
  - **Marking Notified:** Prevents redundant notifications by marking a task as notified on the server after the first expiration notification.
- **Date and Time Display:** Updates the date and time elements on the page every second.
- **Motivational Quote:** Fetches a random motivational quote from an API and displays it on the page.
- **Verse of the Day:** Fetches the verse of the day from an API and displays it on the page.
- **Local Storage:** Saves and retrieves task data from the browser's local storage for persistence across page refreshes.

```
const ws = new WebSocket('ws://localhost:8080');

let tasks = JSON.parse(localStorage.getItem('tasks')) || [];

ws.onopen = function() {
    console.log('Connected to WebSocket server');
    if (tasks.length > 0) {
        tasks.forEach(task => {
            if (isValidTask(task)) {
                createTaskElement(task);
                checkTaskExpiration(task);
            }
        });
    }
};
```

```
ws.onmessage = function(event) {
    const data = JSON.parse(event.data);
    if (data.type === 'init') {
        tasks = data.tasks.filter(isValidTask);
        tasks.forEach(task => {
            createTaskElement(task);
            checkTaskExpiration(task);
        });
        saveTasksToLocalStorage();
    } else if (data.type === 'add') {
        if (isValidTask(data.task)) {
            createTaskElement(data.task);
            tasks.push(data.task);
            saveTasksToLocalStorage();
            checkTaskExpiration(data.task);
        }
    } else if (data.type === 'complete') {
        completeTaskById(data.id);
    } else if (data.type === 'delete') {
        deleteTaskById(data.id);
    } else if (data.type === 'expire') {
        expireTaskById(data.id);
    }
};

function isValidTask(task) {
    return task && task.name && task.text && !isNaN(task.deadline) &&
task.deadline > 0;
}

function addTask() {
    var nameInput = document.getElementById("nameInput");
    var taskInput = document.getElementById("taskInput");
    var taskDeadline = document.getElementById("taskDeadline").value;

    var name = nameInput.value.trim();
    var taskText = taskInput.value.trim();
    nameInput.value = "";
    taskInput.value = "";
    document.getElementById("taskDeadline").value = "";

    if (name === "" || taskText === "" || taskDeadline === "") {
        alert("Please enter your name, a task, and a deadline!");
        return;
    }

    const deadlineTimestamp = new Date(taskDeadline).getTime();
    if (isNaN(deadlineTimestamp) || deadlineTimestamp <= Date.now()) {
```

```
        alert("Please enter a valid future deadline!");
        return;
    }

    const task = { id: Date.now(), name: name, text: taskText, deadline:
deadlineTimestamp, completed: false, notified: false };
    ws.send(JSON.stringify({ type: 'add', task }));
}

function createTaskElement(task) {
    if (!isValidTask(task)) {
        return;
    }

    var taskList = document.getElementById("taskList");
    var li = document.createElement("li");
    li.id = task.id;

    var taskText = document.createElement("span");
    var taskDescription = `(${task.name}): ${task.text}`;
    taskText.textContent = taskDescription;

    var taskDeadline = document.createElement("span");
    taskDeadline.textContent = ` (Deadline: ${new
Date(task.deadline).toLocaleString()})`;
    taskDeadline.classList.add("deadline");

    var completeButton = document.createElement("button");
    completeButton.textContent = "Complete";
    completeButton.onclick = function() {
        ws.send(JSON.stringify({ type: 'complete', id: task.id }));
    };

    var deleteButton = document.createElement("button");
    deleteButton.textContent = "Delete";
    deleteButton.onclick = function() {
        ws.send(JSON.stringify({ type: 'delete', id: task.id }));
    };

    li.appendChild(taskText);
    li.appendChild(taskDeadline);
    li.appendChild(completeButton);
    li.appendChild(deleteButton);
    taskList.appendChild(li);

    if (task.completed) {
        completeTask(li);
    }
```

```
    checkTaskExpiration(task);
}

function completeTask(taskItem) {
    var taskText = taskItem.querySelector("span");
    taskText.style.textDecoration = "line-through";
    taskText.style.color = "gray";
    taskItem.classList.add("completed");
    taskItem.style.backgroundColor = "lightblue";

    const taskId = parseInt(taskItem.id, 10);
    const task = tasks.find(task => task.id === taskId);
    if (task) {
        task.completed = true;
        saveTasksToLocalStorage();
    }
}

function completeTaskById(id) {
    const taskItem = document.getElementById(id);
    if (taskItem) {
        completeTask(taskItem);
    }
}

function deleteTask(taskItem) {
    taskItem.remove();
    saveTasksToLocalStorage();
}

function deleteTaskById(id) {
    const taskItem = document.getElementById(id);
    if (taskItem) {
        taskItem.remove();
        tasks = tasks.filter(task => task.id !== id);
        saveTasksToLocalStorage();
    }
}

function expireTask(taskItem) {
    taskItem.classList.add("expired");
    taskItem.style.backgroundColor = "lightcoral";
}

function expireTaskById(id) {
    const taskItem = document.getElementById(id);
    if (taskItem) {
```

```
        const now = Date.now();
        const task = tasks.find(task => task.id === id);
        if (task && !task.completed && !task.notified) {
            notifyTaskExpiration(task.text);
            task.notified = true;
            saveTasksToLocalStorage();
        }
        if (task && !task.completed) {
            taskItem.classList.add("expired");
            taskItem.style.backgroundColor = "lightcoral";
        }
    }
}

function notifyTaskExpiration(taskText) {
    if (!("Notification" in window)) {
        console.error("This browser does not support desktop notification");
        return;
    }

    const taskDescription = taskText || "Task Expired";

    if (Notification.permission === "granted") {
        new Notification("Task Expired", { body: taskDescription });
    } else if (Notification.permission !== "denied") {
        Notification.requestPermission().then(permission => {
            if (permission === "granted") {
                new Notification("Task Expired", { body: taskDescription });
            }
        });
    }
}

function saveTasksToLocalStorage() {
    localStorage.setItem('tasks', JSON.stringify(tasks));
}

function checkTaskExpiration(task) {
    const now = Date.now();
    const timeRemaining = task.deadline - now;

    if (timeRemaining <= 0) {
        expireTaskById(task.id);
    } else {
        setTimeout(() => {
            expireTaskById(task.id);
        }, timeRemaining);
    }
```

```
}

document.addEventListener("DOMContentLoaded", function() {
    fetchAndDisplayMotivationalQuote();
    updateTime();
    updateVerse();
    setInterval(updateTime, 1000);
    setInterval(updateVerse, 24 * 60 * 60 * 1000);
});

function fetchAndDisplayMotivationalQuote() {
    fetch('https://api.quotable.io/random')
        .then(response => response.json())
        .then(data => {
            const motivationalQuoteElement =
document.getElementById('motivationalQuote');
            motivationalQuoteElement.textContent = data.content;
        })
        .catch(error => {
            console.error('Error fetching quote:', error);
        });
}

function updateTime() {
    const now = new Date();
    const dateElement = document.getElementById("date");
    const timeElement = document.getElementById("time");

    const options = { weekday: 'long', year: 'numeric', month: 'long', day:
'numeric' };
    const formattedDate = now.toLocaleDateString('en-US', options);

    const formattedTime = now.toLocaleTimeString('en-US', { hour: 'numeric',
minute: 'numeric', second: 'numeric', hour12: true });

    dateElement.textContent = "Date: " + formattedDate;
    timeElement.textContent = "Time: " + formattedTime;
}

function updateVerse() {
    fetch('https://labs.bible.org/api/?passage=votd&type=json')
        .then(response => response.json())
        .then(data => {
            const verse = data[0].text;
            document.getElementById("verse").textContent = verse;
        })
        .catch(error => {
            console.error('Error fetching verse:', error);
```

```
            document.getElementById("verse").textContent = "Failed to load
verse.";
        });
}
```

4. **CSS Styles:**

The CSS styles use a variety of selectors to target different elements on the webpage and define their appearance.

- Styles the overall layout (flexbox), font family, and background image.
- Defines a container with rounded corners, shadows, and padding.
- Styles headings, buttons, and list items with visual enhancements (rounded corners, shadows, hover effects).
- Creates a visually distinct style for completed tasks (fading out).
- Highlights expired tasks with a red background and text color.

```
body {
    font-family: 'Roboto', sans-serif;
    background: url('pic\ pic.jpg') no-repeat center center fixed;
    background-size: cover;
    display: flex;
    flex-direction: column;
    align-items: center;
}

.container {
    max-width: 600px;
    margin-top: 20px;
    padding: 20px;
    background-color: rgba(249, 249, 249, 0.9);
    border-radius: 10px;
    box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
}

h1 {
    text-align: center;
    color: #333;
    margin-bottom: 20px;
}

.input-container {
    display: flex;
    margin-bottom: 20px;
```

```
}

input[type="text"] {
    flex-grow: 1;
    padding: 10px;
    border-radius: 5px 0 0 5px;
    border: none;
    background-color: #fff;
    box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
}

button {
    padding: 10px 20px;
    background-color: #ff5851;
    color: #fff;
    border: none;
    border-radius: 0 5px 5px 0;
    cursor: pointer;
    transition: background-color 0.3s ease;
}

button:hover {
    background-color: #ff403a;
}

ul {
    list-style-type: none;
    padding: 0;
}

li {
    display: flex;
    align-items: center;
    padding: 10px;
    margin-bottom: 10px;
    background-color: #fff;
    border-radius: 5px;
    box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
    transition: opacity 1s ease;
}

li:hover {
    background-color: #f2f2f2;
}

li span {
    flex-grow: 1;
    margin-right: 10px;
```

```
}

li button {
    background-color: transparent;
    border: none;
    color: #888;
    cursor: pointer;
    transition: color 0.3s ease;
}

li button:hover {
    color: #333;
}

.motivational-quote-container {
    position: fixed;
    bottom: 10px;
    left: 10px;
    max-width: 300px;
    padding: 10px;
    background-color: #fff;
    border-radius: 5px;
    box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
}

.completed {
    transition: opacity 1s ease;
}

.fade-out {
    opacity: 0;
}

.datetime-container {
    text-align: center;
    margin-bottom: 20px;
}

#date, #time {
    margin: 5px;
}

.verse-of-the-day-container {
    text-align: center;
    margin-bottom: 20px;
}

.verse-container {
```

```
    position: absolute;
    top: 10px;
    right: 10px;
    max-width: 300px;
    padding: 10px;
    background-color: #fff;
    border-radius: 5px;
    box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
    text-align: center;
}

#verse {
    font-size: 16px;
    font-style: italic;
    color: #333;
}

.completed {
    text-decoration: line-through;
    color: gray;
}

.fade-out {
    opacity: 0;
    transition: opacity 3000ms ease-out;
}

.deadline {
    margin-left: 10px;
    font-size: 0.9em;
    color: #888;
}

.expired {
    background-color: red;
    color: red;
}
```