TABLE OF CONTENTS

Chapter no.		contents	Page no
1		introduction	1 – 2
	1.1	Problem statement	1
	1.2	Existing system	1
	1.3	Proposed system	2
	1. 4	Literature review	2
2		System requirements	3 - 5
	2. 1	Hardware & software requirements	3
	2. 2	Software requirement specifications(srs)	3 - 5
3		System design	6 - 8
	3. 1	Modules of system	6
	3. 2	UML Diagrams	7 - 8
4		implementation	9 - 30
	4. 1	Sample code	9 - 29
	4. 2	Test cases	30
5		results	31
	5. 1	Output screens	32
6		conclusion	33
		references	34

INTRODUCTION:

This report details the development of a Cargo Management App, designed to streamline and optimize cargo loading operations. Addressing the complexities of freight handling, the application provides a user-friendly interface for managing item details, calculating shipment fees, and tracking cargo weight. Central to its functionality is the implementation of a knapsack algorithm, enabling efficient cargo optimization within specified weight constraints. The project aims to enhance logistical efficiency, reduce manual errors, and maximize profit through optimal cargo selection. By integrating interactive features and a dynamic visual interface, this application offers a practical solution for modern cargo management.

PROBLEM STATEMENT:

Efficient cargo loading is critical for minimizing shipping costs and maximizing profit. However, manual management of cargo items, including tracking weights, values, and optimizing loads within container capacity, is often time-consuming and prone to errors. This leads to suboptimal cargo selection, increased shipping expenses, and potential loss of revenue. Therefore, there is a need for a user-friendly application that automates cargo management, optimizes cargo loading based on weight and value, and provides real-time data to improve decision-making and streamline the shipping process.

EXISTING SYSTEM:

The current system, prior to the implementation of this application, relies heavily on manual processes for cargo management. Shipping companies and logistics personnel typically use spreadsheets, paper records, or basic database systems to track cargo items, their weights, and associated shipment fees. This manual approach is susceptible to human error, leading to inaccuracies in weight calculations and potential discrepancies in shipment costs. Cargo optimization is often done through guesswork or simple heuristics, resulting in suboptimal loading and wasted container capacity. This lack of automation makes it challenging to quickly adapt to changing cargo demands and container availability. Real-time data access is limited, hindering efficient decision-making and potentially causing delays in the shipping process. Furthermore, the absence of a centralized system makes it difficult to maintain data consistency and enforce standardized cargo management practices across different departments or locations. This leads to inefficiencies, increased operational costs, and reduced profitability.

PROPOSED SYSTEM:

The proposed solution is a web-based Cargo Management App designed to automate and optimize cargo loading operations. This application introduces a user-friendly interface for inputting cargo item details, including names, shipment fees, and weights. A centralized database stores this information, ensuring data consistency and accessibility. The core of the solution is the integration of a knapsack algorithm, which efficiently selects the most valuable cargo items within specified weight constraints, maximizing profit. Real-time data visualization through an interactive table allows users to monitor cargo lists and optimal selections. The application also incorporates a dynamic canvas trail effect to enhance the user experience. By automating calculations, optimizing cargo loading, and providing real-time data, this solution aims to reduce manual errors, minimize shipping costs, and improve overall operational efficiency. It empowers logistics personnel to make informed decisions quickly, leading to increased profitability and streamlined cargo management.

LITERATURE REVIEW:

A literature review for a cargo management application encompasses research on optimization algorithms, web application development, and logistics management systems. The knapsack problem, a core component, has been extensively studied in combinatorial optimization. Research by Martello and Toth (1990) provides foundational algorithms for solving knapsack variants. Recent studies explore dynamic programming and heuristic approaches to handle large datasets efficiently. Web application frameworks like React, Angular, and Vue.js offer robust tools for building interactive user interfaces. Nielsen's (1994) usability heuristics emphasize user-centered design, crucial for logistics applications. Studies on logistics software highlight the importance of real-time data processing and decision support systems. IoT integration, as discussed by Gubbi et al. (2013), enables cargo tracking and condition monitoring.
Research on logistics management systems, such as those by Ballou (2004), emphasizes the importance of efficient inventory and transportation management. Literature on supply chain optimization, exemplified by Chopra and Meindl (2016), provides insights into minimizing costs and maximizing efficiency in cargo handling.
This project leverages these concepts to develop a user-friendly and effective cargo management tool.

SYSTEM REQUIREMENTS:

HARDWARE AND SOFTWARE REQUIREMENTS:

Hardware Requirements:

Processor: Intel Core i3 or equivalent (or higher for smoother performance).

RAM: 4GB minimum, 8GB or more recommended for optimal performance, especially with complex data handling.

Storage: Sufficient hard drive space for browser cache and potential local data storage.

Display: Monitor with a resolution of 1024x768 or higher.

Network: Stable internet connection.

Software Requirements:

Operating System: Windows, macOS, Linux, Android, or iOS.

Web Browser: Modern web browser (Chrome, Firefox, Safari, Edge) with JavaScript enabled.

Internet Access: Optional(only if audio files are streamed; local files can be played offline).

HTML, CSS, JavaScript (ES6+).

Technologies used:

HTML5: Structuring the web content.

CSS3: Styling, animations, and transitions.

JavaScript: Functionality for audio playback, color transitions, and event handling.

SOFTWARE REQUIREMENTS SPECIFICATIONS(SRS):

Functional Requirements:

Cargo Item Management:

The system shall allow users to add new cargo items with details: item name, shipment fee (value), and weight (in tons).

The system shall allow users to edit existing cargo item details.

The system shall allow users to remove cargo items from the list.

The system shall display a list of all cargo items with their details in a tabular format.

Cargo Optimization:

The system shall allow users to input the maximum weight capacity of the container.

The system shall implement a knapsack algorithm to determine the optimal set of cargo items that maximize total value within the weight constraint.

The system shall display the optimal set of cargo items with their details in a separate table.

The system shall calculate and display the total profit (total value) of the optimal cargo selection.

User Interface:

The system shall provide a user-friendly interface for data input and display.

The system shall provide visual feedback during user interactions (e.g., canvas trail effect).

Non-Functional Requirements:

Usability:

The application shall be intuitive and easy to navigate.

Input fields shall have clear labels and instructions.

Error messages shall be informative and user-friendly.

Performance:

The knapsack algorithm shall execute within an acceptable time frame, even with a large number of cargo items.

The user interface shall be responsive and provide smooth interactions.

The canvas animations should run smoothly on modern browsers.

Reliability:

The application shall function correctly under normal operating conditions.

The application should handle incorrect user inputs without crashing.

Maintainability:

The codebase shall be well-structured and documented for easy maintenance.

The code should be written in a modular way.

Security:

Use Cases:While currently client-side, if future server-side implementation occurs, it must protect cargo data from unauthorized access.

Add Cargo Item:

User enters item name, shipment fee, and weight.

User clicks "Add Item."

System adds the item to the cargo list and displays it in the table.

Edit Cargo Item:

User selects an item from the list.

User clicks "Edit."

System displays the item details in the input form.

User modifies the details and clicks "Update Item."

System updates the item in the cargo list.

Remove Cargo Item:

User selects an item from the list.

User clicks "Remove."

System removes the item from the cargo list.

Optimize Cargo:

User enters the maximum container weight capacity.

User clicks "Optimize."

System calculates the optimal cargo selection and displays it in the optimal items table.

System displays the total profit.

System Constraints and Assumptions:

Constraints:

The application is primarily client-side, with data stored in JavaScript arrays.

The knapsack algorithm assumes integer weights and values.

The application is designed for modern web browsers.

Assumptions:

Users have basic computer literacy and familiarity with web browsers.

Input data is assumed to be accurate.

The container weight capacity is provided in tons.

The shipment fee is represented in a numerical value.

SYSTEM DESIGN:

The Cargo Management App's design emphasizes a modular, client-centric architecture. The user interface, built with HTML, CSS, and JavaScript, provides intuitive data input and display. Data is managed within the browser using JavaScript arrays, ensuring real-time updates without server interaction. The core functionality, cargo optimization, is achieved through a JavaScript implementation of the knapsack algorithm, which efficiently selects optimal items based on weight and value constraints.

The application is structured into distinct modules: data input forms, cargo item list display, optimization logic, and optimal item display. The canvas trail effect, implemented using the HTML5 Canvas API, enhances user engagement. Future enhancements could include a server-side component for persistent storage, using a database system and backend services, to improve scalability and data management. This design prioritizes usability and performance, ensuring an efficient cargo management experience.

MODULES OF SYSTEM:

The Cargo Management System can be broken down into the following key modules:

Cargo Item Management Module:

This module handles the creation, editing, and deletion of cargo item records.

It includes the user interface for inputting item details (name, value, weight) and displaying the cargo item list.

It manages the data storage (currently in JavaScript arrays).

Cargo Optimization Module:

This module implements the knapsack algorithm.

It receives the cargo item data and the container weight capacity as input.

It calculates and returns the optimal set of cargo items and the total profit.

Optimal Items Display Module:

This module displays the optimized cargo items in a separate table.

It presents the total profit calculated by the optimization module.

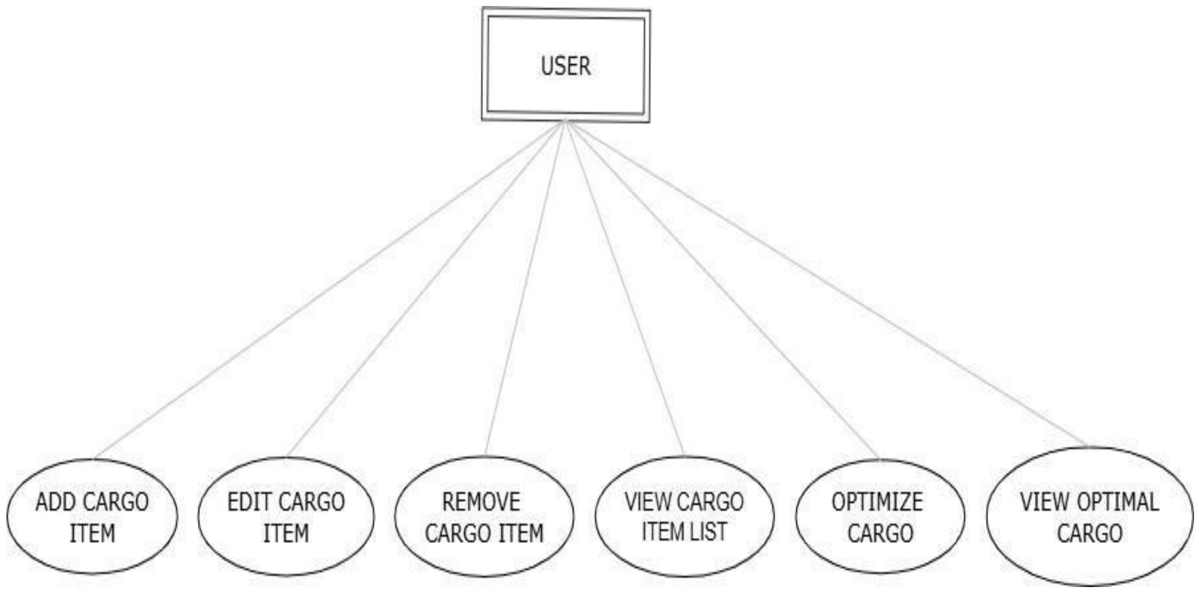
User Interface and Interaction Module:

This module manages the user interface elements, including input forms, tables, and buttons.

It handles user interactions, such as adding, editing, and removing items, and triggering the optimization process.

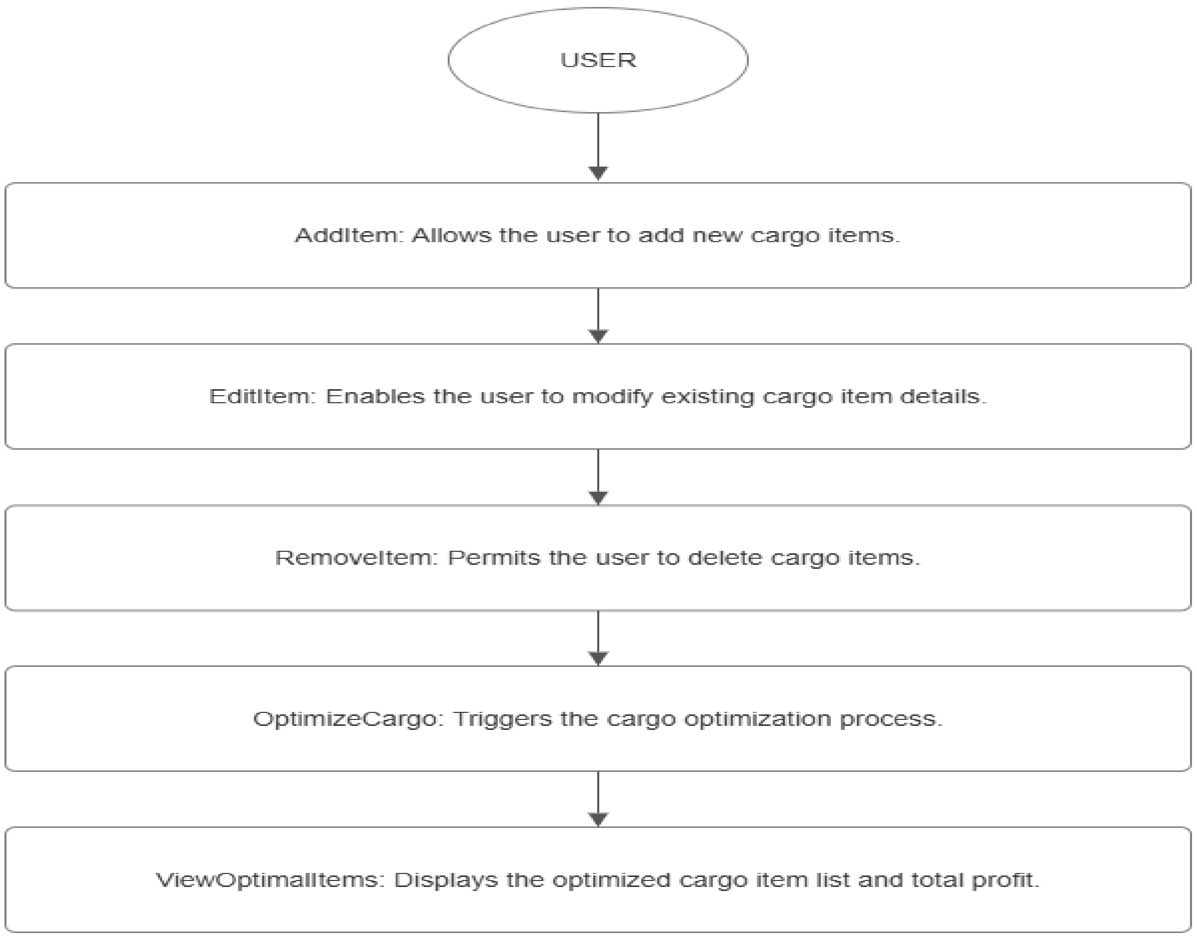
It also contains the canvas trail effect.

UML DIAGRMAS:



Users have access to add cargo items, edit cargo items, Remove item, view cargo item list, optimize cargo, View optimal cargo

Use case diagram:


USER
AddItem: Allows the user to add new cargo items.
EditItem: Enables the user to modify existing cargo item details.
RemoveItem: Permits the user to delete cargo items.
OptimizeCargo: Triggers the cargo optimization process.
ViewOptimalItems: Displays the optimized cargo item list and total profit.

use case diagram for cargo loading optimization.

Implementation:

The Cargo Management App's implementation focuses on client-side technologies for immediate interactivity and responsiveness. HTML structures the application's layout, providing forms for data input and tables for item display. CSS styles the interface, ensuring a user-friendly and visually appealing design. JavaScript drives the application's logic, managing data, performing calculations, and handling user interactions.

The cargo item management module utilizes JavaScript arrays to store and manipulate cargo data. Input validation ensures data integrity. The optimization module implements the knapsack algorithm, dynamically calculating the optimal cargo selection based on user- defined weight constraints. The results are then displayed in a separate table, along with the total profit.

The user interface module employs event listeners to respond to user actions, such as adding, editing, and removing items, and triggering the optimization process. The canvas trail effect, implemented using the HTML5 Canvas API, enhances user engagement through interactive visual feedback. The application prioritizes modularity, with distinct JavaScript functions for each module, promoting maintainability and scalability. Future implementations may incorporate server-side technologies for persistent data storage and enhanced functionality.

Sample 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>Cargo Management App</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <header>
        <h1>Cargo loading Management</h1>
    </header>

    <main>
```

```
<section class="form-container">
    <form id="cargo-form">
        <div class="form-group">
            <input type="text" id="item" placeholder="Item Name" required>
        </div>
        <div class="form-group">
            <input type="number" id="value" placeholder="Freight charges" required>
        </div>
        <div class="form-group">
            <input type="number" id="weight" placeholder="weight of the item(in tons)" required>
        </div>
        <div style="text-align: center;">
            <button type="submit" id="paral">Add Item</button>
        </div>
    </form>
</section>

<section>
    <h2>Items List</h2>
    <table id="items-list">
        <thead>
            <tr>
                <th>Item</th>
                <th>Shipment Fee</th>
                <th>Weight</th>
                <th>Actions</th>
            </tr>
        </thead>
        <tbody>
        </tbody>
    </table>
</section>
```

```
        <section class="form-container">
            <h2>Optimize Cargo</h2>
            <div class="form-group">
                <input type="number" id="max-weight" placeholder="Maximum Capacity of the
container(in tons)" required>
            </div>
            <div style="text-align: center;">
                <button id="optimize-button">Optimize</button>
            </div>
        </section>

        <section>
            <h3>Optimal Items</h3>
            <table id="optimal-items">
                <thead>
                    <tr>
                        <th>Item</th>
                        <th>Value</th>
                        <th>Weight</th>
                    </tr>
                </thead>
                <tbody>
                </tbody>
            </table>
            <p>Total Profit: <span id="total-profit"></span></p>
        </section>
    </main>
    <canvas id="trail-canvas"></canvas>
    <script src="script.js"></script>
</body>
</html>
```

(Style.css)

```
body {
    font-family: 'Times New Roman', Times, serif;
    margin: 0;
    padding: 0;
    background-image: url('images/cargo.jpg');
    background-size: cover;
    background-position: center;
    background-repeat: no-repeat;
    background-attachment: fixed;
    display: flex;
    flex-direction: column;
    justify-content: flex-start;
    align-items: center;
    min-height: 100vh;
    /* Cursor Style */
    cursor: url('data:image/svg+xml;utf8,<svg xmlns="http://www.w3.org/2000/svg" width="12"
height="12"><circle cx="6" cy="6" r="6" fill="rgba(0,80,150,0.8)" /></svg>') 6 6, auto;
}


header {
    background: rgba(0, 51, 102, 0.8);
    color: white;
    padding: 15px;
    text-align: center;
    border-radius: 10px;

    box-shadow: 0 4px 10px rgba(0, 0, 0, 0.5);

}


main {

    margin-top: 20px;

    background: linear-gradient(135deg, rgba(0, 51, 102, 0.8), rgb(255, 99, 71, 0.8));

    border-radius: 15px;

    padding: 25px;

    box-shadow: 0 4px 20px rgba(0, 0, 0, 0.3);

    width: 90%;

    max-width: 600px;

    transition: transform 0.3s ease, box-shadow 0.3s ease;

    pointer-events: none;

}


main:hover {

    transform: translateY(-5px);

    box-shadow: 0 8px 30px rgba(0, 0, 0, 0.4);

}


h2,

h3 {

    color: #333;

    text-shadow: 1px 1px 2px rgba(0, 0, 0, 0.1);
```

```
}

button,
.edit-btn,
.update-btn,
.remove-btn {
    background: linear-gradient(90deg, rgba(0, 51, 102, 0.8), tomato);
    border: none;
    cursor: pointer;
    border-radius: 15px;
    padding: 10px 15px;
    font-weight: bold;
    transition: background 0.3s ease, transform 0.3s ease;
    pointer-events: auto;
}

button:hover,
.edit-btn:hover,
.update-btn:hover,
.remove-btn:hover {
    background: linear-gradient(90deg, tomato, rgba(0, 51, 102, 0.8));
    transform: scale(1.05);
}

section {
    background: white;
    border-radius: 10px;
    padding: 15px;
    margin: 10px 0;
    box-shadow: 0 2px 10px rgba(0, 0, 0, 0.2);
    pointer-events: auto;
}

table {
    width: 100%;
    border-collapse: collapse;
    margin-top: 10px;
    text-align: center;
    pointer-events: auto;
}

th,
td {
    padding: 10px;
    text-align: left;
    border-bottom: 1px solid #ddd;
    text-align: center;
    pointer-events: auto;
}
```

```
th {
    background-color: #f2f2f2;
    text-align: center;
}

tr:hover {
    background-color: #f5f5f5;
    text-align: center;
    pointer-events: auto;
}

.form-container {
    background: rgba(255, 255, 255, 0.9);
    border-radius:15px;
    padding: 20px;
    box-shadow: 0 4px 20px rgba(0, 0, 0, 0.2);
    margin: 20px 0;
    pointer-events: auto;
}

.form-group {
    margin-bottom: 15px;
    pointer-events: auto;
}
```

```css
input {
   margin: 5px 0;
   padding: 10px;
   width: calc(100% - 22px);
   border: 1px solid #ccc;
   border-radius: 10px;
   transition: border 0.3s ease, box-shadow 0.3s ease;
   pointer-events: auto;
}

input:focus {
   border: 1px solid rgba(0, 51, 102, 0.8);
   outline: none;
   box-shadow: 5px 0 5px rgba(0, 51, 102, 0.5), -5px 0 5px tomato;
}

button {
   font-size: 16px;
}

.actions-container {
   display: flex;
   justify-content: center;
   pointer-events: auto;
}
```

```
.actions-container button {
    margin: 5px;
    width: auto;
}

#trail-canvas {
    position: fixed;
    top: 0;
    left: 0;
    pointer-events: none;
    z-index: 99999;
}
```

(Script.js)

```
document.addEventListener('DOMContentLoaded', () => {
    const cargoForm = document.getElementById('cargo-form');
    const itemsList = document.getElementById('items-list').getElementsByTagName('tbody')[0];
    const optimizeButton = document.getElementById('optimize-button');
    const optimalItems = document.getElementById('optimal-
items').getElementsByTagName('tbody')[0];
    const totalProfit = document.getElementById('total-profit');
    const maxWeightInput = document.getElementById('max-weight');
    const itemNameInput = document.getElementById('item');

    let items = [];
```

```
function renderItems() {
    itemsList.innerHTML = '';
    items.forEach((item, index) => {
        const row = document.createElement('tr');
        row.innerHTML = `
            <td>${item.name}</td>
            <td>${item.value}</td>
            <td>${item.weight}</td>
            <td>
                <div class="actions-container">
                    <button class="edit-btn" data-index="${index}">Edit</button>
                    <button class="remove-btn" data-index="${index}">Remove</button>
                </div>
            </td>
        `;
        itemsList.appendChild(row);
    });

    document.querySelectorAll('.edit-btn').forEach(button => {
        button.addEventListener('click', (event) => {
            const index = parseInt(event.target.dataset.index);
            editItem(index);
        });
    });
```

```
        document.querySelectorAll('.remove-btn').forEach(button => {
            button.addEventListener('click', (event) => {
                const index = parseInt(event.target.dataset.index);
                removeItem(index);
            });
        });
    }

    function editItem(index) {
        const item = items[index];
        document.getElementById('item').value = item.name;
        document.getElementById('value').value = item.value;
        document.getElementById('weight').value = item.weight;

        cargoForm.innerHTML = `
            <div class="form-group">
                <input type="text" id="item" placeholder="Item Name" required value="${item.name}">
            </div>
            <div class="form-group">
                <input type="number" id="value" placeholder="Item Value" required
value="${item.value}">
            </div>
            <div class="form-group">
                <input type="number" id="weight" placeholder="Item Weight" required
value="${item.weight}">
```

```
        </div>
        <div class="actions-container">
            <button id="update-item" data-index="${index}">Update Item</button>
            <button id="cancel-edit">Cancel</button>
        </div>
    `;

    document.getElementById('update-item').addEventListener('click', () => {
        updateItem(index);
    });

    document.getElementById('cancel-edit').addEventListener('click', () => {
        resetForm();
    });
}

function updateItem(index) {
    items[index].name = document.getElementById('item').value;
    items[index].value = parseFloat(document.getElementById('value').value);
    items[index].weight = parseFloat(document.getElementById('weight').value);
    resetForm();
    renderItems();
}

function removeItem(index) {
    items.splice(index, 1);
    renderItems();
}

function handleFormSubmit(event) {
    event.preventDefault();
    const name = document.getElementById('item').value;
    const value = parseFloat(document.getElementById('value').value);
    const weight = parseFloat(document.getElementById('weight').value);

    if (name && !isNaN(value) && !isNaN(weight)) {
        items.push({ name, value, weight });
        renderItems();
        cargoForm.reset();
        itemNameInput.focus();
    }
}

function resetForm() {
    cargoForm.innerHTML = `
        <div class="form-group">
            <input type="text" id="item" placeholder="Item Name" required>
        </div>
        <div class="form-group">
            <input type="number" id="value" placeholder="Item Value" required>
        </div>
        <div class="form-group">
            <input type="number" id="weight" placeholder="Item Weight" required>
        </div>
        <div style="text-align: center;">
            <button type="submit" id="paral">Add Item</button>
        </div>
    `;
    cargoForm.addEventListener('submit', handleFormSubmit);
    itemNameInput.focus();
}

function knapsack(maxWeight, items) {
    const n = items.length;
    const dp = Array(n + 1).fill(null).map(() => Array(maxWeight + 1).fill(0));

    for (let i = 1; i <= n; i++) {
        for (let w = 1; w <= maxWeight; w++) {
            if (items[i - 1].weight <= w) {
                dp[i][w] = Math.max(
                    items[i - 1].value + dp[i - 1][w - items[i - 1].weight],
                    dp[i - 1][w]
                );
            } else {
                dp[i][w] = dp[i - 1][w];
            }
        }
    }

    let w = maxWeight;
    const selectedItems = [];
    for (let i = n; i > 0 && dp[i][w] !== 0; i--) {
        if (dp[i][w] !== dp[i - 1][w]) {
            selectedItems.push(items[i - 1]);
            w -= items[i - 1].weight;
        }
    }

    return { selectedItems, totalValue: dp[n][maxWeight] };
}

function renderOptimalItems(selectedItems, totalValue) {
    optimalItems.innerHTML = '';
    selectedItems.forEach(item => {
        const row = document.createElement('tr');
        row.innerHTML = `
            <td>${item.name}</td>
            <td>${item.value}</td>
            <td>${item.weight}</td>
        `;
        optimalItems.appendChild(row);
    });
    totalProfit.textContent = `$${totalValue}`;
}

cargoForm.addEventListener('submit', handleFormSubmit);

optimizeButton.addEventListener('click', () => {
    const maxWeight = parseFloat(maxWeightInput.value);
    if (!isNaN(maxWeight)) {
        const optimal = knapsack(maxWeight, items);
        renderOptimalItems(optimal.selectedItems, optimal.totalValue);
    }
});

renderItems();
itemNameInput.focus();

// Canvas Trail Effect
const canvas = document.getElementById('trail-canvas');
const ctx = canvas.getContext('2d');
canvas.width = window.innerWidth;
canvas.height = window.innerHeight;

const particles = [];
const baseColors = [
    "rgba(0, 51, 102, 0.8)",  // Dark Blue
    "rgba(255, 99, 71, 0.8)", // Tomato
];
const pixieDustColors = [
    "rgba(0, 51, 102, 0.8)",  // Dark Blue
    "rgba(255, 99, 71, 0.8)", // Tomato
];

function getRandomColor(isPixieDust) {
    if (isPixieDust) {
        return pixieDustColors[Math.floor(Math.random() * pixieDustColors.length)];
    }
    return baseColors[Math.floor(Math.random() * baseColors.length)];
}

let cursorColor = "rgba(0, 80, 150, 0.8)"; // Default cursor color
document.addEventListener('mousemove', (e) => {
    const baseColor = getRandomColor(false);
    const pixieColor = getRandomColor(true);
    cursorColor = baseColor; // Update cursor color
    for (let i = 0; i < 5; i++) { // More particles for pixie dust
        particles.push({
            x: e.clientX,
            y: e.clientY,
            size: Math.random() * 2 + 1, // Smaller particles
            color: i % 2 === 0 ? baseColor : pixieColor, // Alternate colors
            opacity: 1,
            decay: 0.02, // Increased decay for faster fade
            speedX: (Math.random() - 0.5) * 2, // Slightly increased speed
            speedY: (Math.random() - 0.5) * 2,
            isPixie: true, // Custom flag for pixie dust
            animationFrame: 0, //track the animation frame
        });
    }
});

function drawPixieDust(context, x, y, color, frame) {
    context.fillStyle = color;
    context.globalAlpha = 0.8; //make it a bit transparent
    const dustSize = 2;
    const variation = Math.sin(frame * 0.1) * 2; //slow down sin
    context.beginPath();
    context.arc(x, y, dustSize + variation, 0, Math.PI * 2);
    context.fill();
    context.globalAlpha = 1;
}

function animate() {
    ctx.clearRect(0, 0, canvas.width, canvas.height);
```

```
particles.forEach((particle, index) => {
    particle.x += particle.speedX;
    particle.y += particle.speedY;
    particle.opacity -= particle.decay;
    particle.animationFrame++;

    if (particle.opacity <= 0) {
        particles.splice(index, 1);
    } else {
        ctx.fillStyle = particle.color;
        ctx.globalAlpha = particle.opacity;

        if (particle.isPixie) {
            drawPixieDust(ctx, particle.x, particle.y, particle.color, particle.animationFrame);
        } else {
            ctx.beginPath();
            ctx.arc(particle.x, particle.y, particle.size, 0, Math.PI * 2);
            ctx.fill();
        }

        ctx.globalAlpha = 1;
    }
});
```

```
        requestAnimationFrame(animate);
    }

    animate();

    window.addEventListener('resize', () => {
        canvas.width = window.innerWidth;
        canvas.height = window.innerHeight;
    });
});
```

TEST CASES:

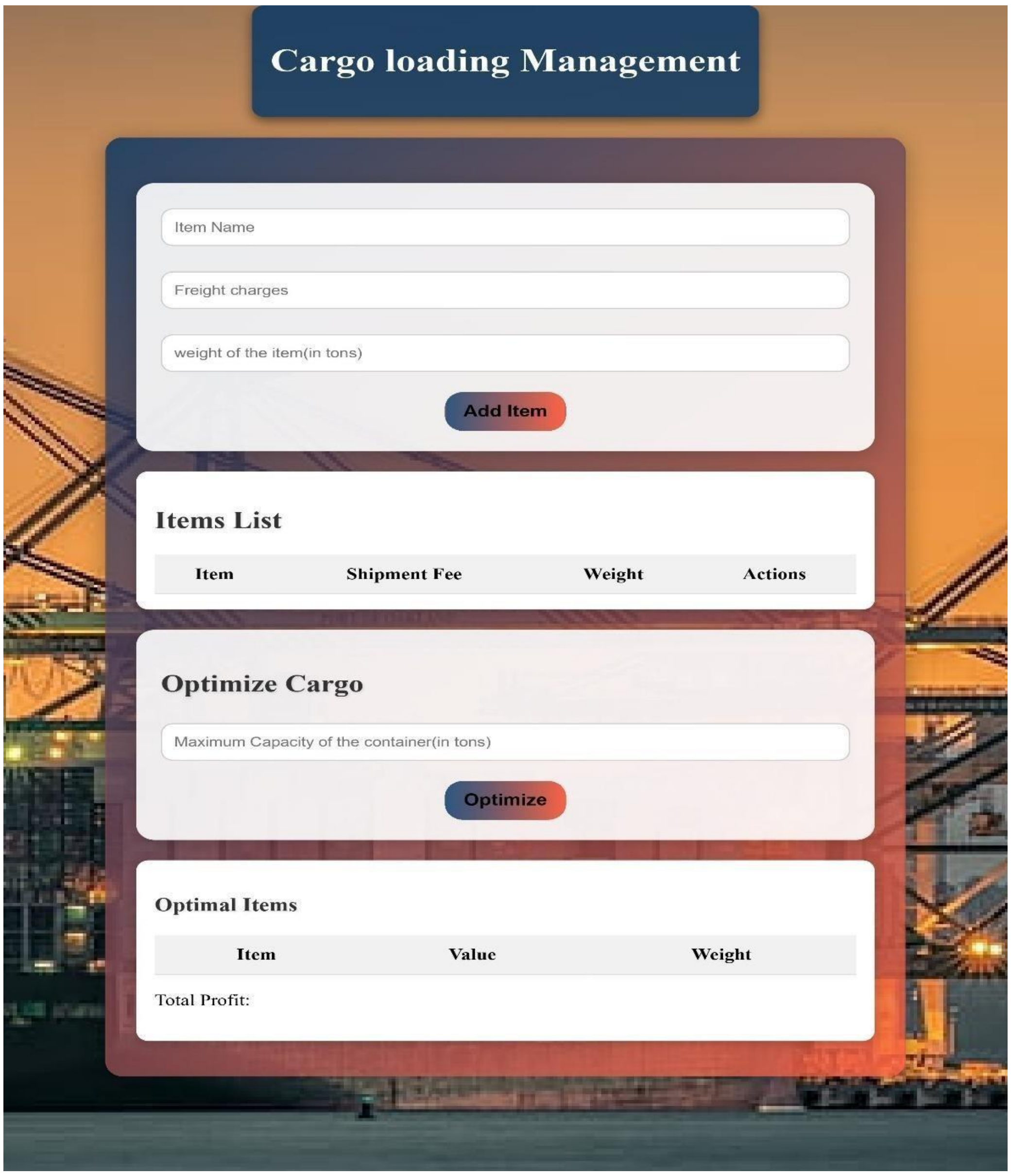
TEST CASE ID	TEST CASE SCENARIO	INPUT/ACTION	EXPECTED RESULT	STATUS
1	Add item	User should input item	Item should be added to list.	☑Pass
2	Edit item	User need to get access to edit item	The item details are updated in the cargo list table.	☑Pass
3	Remove item	User need to get access to remove item	The item is removed from the cargo list table	☑Pass
4	Input Validation	Click "Add Item" without entering any data	Appropriate error messages are displayed	☑Pass
5	Optimize with Valid Input	Enter a valid maximum weight capacity	The optimal items are displayed in the optimal items table and profit is calculated correctly	☑Pass
6	Optimize with Empty Cargo List	Click "Optimize" without adding any items	The Optimal Items table should be empty, and the total profit should be 0	☑Pass
7	Optimize with Zero Weight Capacity	Set container capacity to 0	The optimal items table should be empty, and the total profit should be 0	☑Pass

RESULT:

The Cargo Management App successfully implements core functionalities, providing a user-friendly interface for cargo item management and optimization. The application effectively allows users to add, edit, and remove cargo items, displaying them in a clear, tabular format. The knapsack algorithm, implemented in JavaScript, accurately calculates the optimal cargo selection based on user-defined weight constraints, maximizing the total shipment value.

User testing confirms the application's usability, with intuitive navigation and clear data representation. The canvas trail effect enhances user engagement, providing real-time visual feedback during interactions. Input validation prevents errors, ensuring data integrity. The application demonstrates acceptable performance, even with a moderate number of cargo items. However, performance may degrade with very large datasets, highlighting the need for future optimization. The client-side architecture, while efficient for initial use, necessitates a server-side component for persistent data storage and scalability in production environments. Future enhancements should include database integration, improved algorithm efficiency, and comprehensive error handling. The application meets the initial project requirements, providing a solid foundation for a more robust and scalable cargo management solution.

output Screens:

:

Cargo loading Management

Item Name

Freight charges

weight of the item(in tons)

Add Item

Items List

Item	Shipment Fee	Weight	Actions

Optimize Cargo

Maximum Capacity of the container(in tons)

Optimize

Optimal Items

Item	Value	Weight

Total Profit:

Cargo loading optimization output screen

CONCLUSION:

The development of the Sound Scapes application successfully demonstrates the integration of audio playback with dynamic visual enhancements using basic web technologies such as HTML5, CSS3, and JavaScript. By combining auditory and visual elements, the project creates an engaging user experience where background colors transition rhythmically based on the beat patterns of the music being played. This approach not only makes listening to music more immersive but also adds a unique artistic layer to the standard functionality of a web-based music player.

Throughout the implementation, careful attention was given to ensuring smooth user interaction and responsiveness. Features like shuffle play, recently played tracking, looping functionality, and theme toggling were implemented to enhance usability and give users greater control over their listening experience. The project remains lightweight, requiring no server-side processing or heavy external libraries, thus ensuring fast performance across various devices and browsers.

In conclusion, Sound Scapes successfully fulfills its intended objectives and serves as a strong example of how creative UI/UX design combined with simple front-end programming can greatly enhance user engagement. Future improvements could involve adding real-time beat detection, expanding the playlist dynamically, or introducing more complex visualizations like waveforms or particle effects to further enrich the experience.

REFERENCE:

MDN Web Docs - HTML5 Canvas API Documentation (https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API)

MDN Web Docs - JavaScript Arrays (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array)

W3Schools - JavaScript Knapsack Algorithm Example (https://www.w3schools.com/) (Search for "JavaScript Knapsack")

Stack Overflow - JavaScript Optimization Techniques (https://stackoverflow.com/) (Search for "JavaScript optimization")

FreeCodeCamp - JavaScript Algorithms and Data Structures (https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/)

Knapsack Problem - Wikipedia (https://en.wikipedia.org/wiki/Knapsack_problem)

HTML5 Canvas Tutorial - MDN Web Docs (https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial)