

# **SECURE FILE TRANSFER WITH NETWORK MONITORING**

**A MINI PROJECT REPORT**

*Submitted by*

<b>RITHUPRIYA S</b>	<b>231901043</b>
<b>SHAMRUDHA</b>	<b>231901048</b>
<b>VARSHINI P</b>	
<b>KALAIYARASI M</b>	<b>231901503</b>

*in partial fulfillment of the award of the degree of*

**BACHELOR OF ENGINEERING  
IN  
COMPUTER SCIENCE AND ENGINEERING**



**RAJALAKSHMI ENGINEERING COLLEGE, CHENNAI**

**An Autonomous Institute**

**CHENNAI**

**MAY 2025**

## **BONAFIDE CERTIFICATE**

Certified that this project “**SECURE FILE TRANSFER WITH NETWORK MONITORING**” is the bonafide work of “**RITHUPRIYA S, SHAMRUDHAVARSHINI P, KALAIYARASI M**” who carried out the project work under my supervision.

### **SIGNATURE**

**Mrs. JANANEE V**

**ASSISTANT PROFESSOR**

Dept. of Computer Science and Engg,  
Rajalakshmi Engineering College  
Chennai

This mini project report is submitted for the viva voce examination to be held on \_\_\_\_\_

**INTERNAL EXAMINER**

**EXTERNAL EXAMINER**

## ABSTRACT

The **Secure File Transfer Dashboard** is a real-time, interactive web application that simulates the behavior of a secure file transmission system. It combines backend simulation with frontend visual feedback to demonstrate the complete life cycle of a file transfer between a client and a server over a network. The dashboard serves as both a monitoring tool and an educational model, illustrating various aspects of data transfer such as chunk-wise speed updates, visual progress indicators, and animated movement of file icons from client to server.

This project integrates multiple technologies including **Flask (Python)** for the server-side logic, **JavaScript** for dynamic behavior, **Chart.js** for live graph visualization, and **Leaflet.js** for displaying the geographic locations of clients and servers. It features a robust simulation of network conditions by implementing a cyclic success-failure logic (e.g., 2 successful transfers followed by 1 failure), reflecting real-world uncertainties in data transmission.

Visual cues such as a moving file icon, a glowing animation for active transfer, confetti for successful completions, and shaking or red alerts for failures enhance the user experience and clearly convey the transfer status. Additionally, a responsive progress bar keeps users informed about the real-time state of the file delivery.

The dashboard is ideal for students, developers, and researchers to study file transmission concepts visually and interactively. It can also be adapted for use in training scenarios, system demos, or as a foundational tool in building more complex network monitoring solutions.

## **ACKNOWLEDGEMENT**

We express our sincere thanks to our honorable chairman **MR. S. MEGANATHAN** and the chairperson **DR. M.THANGAM MEGANATHAN** for their timely support and encouragement.

We are greatly indebted to our respected and honorable principal **Dr. S.N. MURUGESAN** for his able support and guidance.

No words of gratitude will suffice for the unquestioning support extended to us by our Head Of the Department **Mr. BENIDICT JAYAPRAKASH NICHOLAS M.E Ph.D.**, for being ever supporting force during our project work.

We also extend our sincere and hearty thanks to our internal guide **Mrs.V JANANEE**, for her valuable guidance and motivation during the completion of this project.

Our sincere thanks to our family members, friends and other staff members of computer science engineering.

**RITHUPRIYA S**

**SHAMRUDHA VARSHINI P**

**KALAIYARASI M**

## **TABLE OF CONTENTS**

<b>CHAPTER NO</b>	<b>TITLE</b>	<b>PAGE NO</b>
<b>1</b>	<b>INTRODUCTION</b>	<b>7</b>
1.1	Introduction	
1.2	Scope of the Work	
1.3	Problem Statement	
1.4	Aim and objective	
<b>2</b>	<b>SYSTEM SPECIFICATIONS</b>	
2.1	Hardware	<b>8</b>
2.2	Software	
<b>3</b>	<b>MODULE DESCRIPTION</b>	
3.1	Animation Module	
3.2	Graph Module	
3.3	Progress Module	<b>10</b>
3.4	Map Module	
3.5	Transfer Logic Module	
<b>4</b>	<b>CODING</b>	<b>12</b>
<b>5</b>	<b>SCREENSHOTS</b>	<b>22</b>
<b>6</b>	<b>CONCLUSION AND FUTURE ENHANCEMENT</b>	<b>25</b>
<b>7</b>	<b>REFERENCES</b>	<b>26</b>

## **List of figures**

<b>Figure no.</b>	<b>Title</b>	<b>Page no.</b>
<b>Fig 5.1</b>	<b>Dashboard with graph and location</b>	<b>22</b>
<b>Fig 5.2</b>	<b>File transferring animation</b>	<b>23</b>
<b>Fig 5.3</b>	<b>File transfer failed state with red alert</b>	<b>24</b>

# **CHAPTER 1**

## **INTRODUCTION**

### **1.1 Introduction**

The Secure File Transfer Dashboard simulates the process of transferring files over a network. It visualizes the progress using an animated dashboard, progress bar, and real-time graph, providing a hands-on understanding of transfer behavior including interruptions and completion success.

### **1.2 Scope of the Work**

The scope includes:

- Frontend visualization using HTML, CSS, JavaScript.
- Real-time graph using Chart.js.
- Simulated client-server location on a map.
- Randomized success/failure logic to emulate real network conditions.

### **1.3 Problem Statement**

Traditional dashboards lack visualization for transfer animations or real-time transfer feedback. This project addresses the gap by visually representing the file movement from client to server with transfer status.

### **1.4 Aim and Objectives**

- Animate file transfers in real time.
- Track and show data chunk transfer speed.
- Simulate failures and provide visual alerts.
- Integrate geolocation of server and client for realism.

## **CHAPTER 2**

### **SYSTEM SPECIFICATIONS**

The development and smooth functioning of the Secure File Transfer Dashboard require a well-balanced combination of hardware and software components. These specifications have been determined based on the tools and technologies used, as well as the need to support real-time simulation and animation rendering in a responsive and efficient manner.

#### **2.1 Hardware**

To ensure the project runs seamlessly during development and testing, the following minimum hardware configuration is recommended:

- Processor: Intel Core i3 / i5 or equivalent

The processor is central to the application's performance. An Intel Core i3 or i5 is sufficient to run local servers using Flask and to handle background data simulation threads. Higher-end processors (i7/i9 or Ryzen equivalents) will offer improved responsiveness, especially when handling multiple tasks such as live chart updates, concurrent file simulation, and browser rendering.

- RAM: 4 GB (Minimum)

The dashboard uses multiple scripts and libraries, including Chart.js and Leaflet.js, which are executed within the browser. A minimum of 4 GB RAM allows the system to handle both the Flask backend and a modern browser environment like Chrome or Firefox simultaneously. However, 8 GB or more is recommended for enhanced multitasking and faster load times.

- Storage: 128 GB SSD

An SSD ensures quicker read/write speeds, faster boot times, and faster loading of the Flask server and static assets. The entire project footprint is small (less than 100 MB), but having 128 GB SSD ensures a smooth overall development experience.



## 2.2 Software

A combination of frontend and backend technologies, open-source libraries, and a lightweight code editor are used to develop and demonstrate the project effectively:

- **Operating System:** Windows 10 or any modern Linux distribution (Ubuntu, Fedora)  
The dashboard is built using cross-platform technologies and runs inside a browser, making it OS-independent. However, Python and Flask must be properly configured on the system. Linux distributions may provide better performance for developers familiar with Unix-like environments.
- **Frontend Technologies:**
  - **HTML (HyperText Markup Language):** Used to structure the dashboard layout and UI components.
  - **CSS (Cascading Style Sheets):** Responsible for styling elements including buttons, progress bars, animations, and the layout responsiveness.
  - **JavaScript:** Powers the interactivity, real-time chart updates, progress tracking, and animation logic on both the visual file transfer and map.
- **Backend Technology:**
  - **Python (with Flask framework):** Flask is a lightweight and easy-to-use Python web framework used to handle HTTP requests, simulate file transfers, and provide API endpoints for chart and map data.
- **Libraries and Tools:**
  - **Chart.js:** A powerful JavaScript library used to render the live line graph representing file transfer speed over time. It is highly customizable and integrates smoothly with real-time data.
  - **Leaflet.js:** An open-source JavaScript library used to render an interactive world map that displays the dynamic locations of the client and server. It visually simulates the file route on a geographic scale.
- **Editor:**
  - **Visual Studio Code (VS Code):** A modern and lightweight code editor used for writing HTML, CSS, JavaScript, and Python code. It supports live preview extensions, integrated terminal for running Flask, and Git version control.

## **CHAPTER 3**

### **MODULE DESCRIPTION**

The Secure File Transfer Dashboard is composed of multiple functional modules, each responsible for a specific feature of the system. These modules work in coordination to create a realistic, interactive simulation of file transfer between a client and server. Below are detailed descriptions of the core modules involved:

#### **3.1 Animation Module**

The Animation Module is responsible for visually representing the movement of a file icon from the client to the server on the dashboard. It uses JavaScript to animate the file icon across a linear path in the UI, simulating an actual file being transferred. This module reacts to the result of the simulated transfer:

- If the transfer is successful, the file icon reaches the server, and a glowing effect (via CSS animation) is removed.
- If the transfer fails, the animation halts midway, and a red-colored “shake” animation is triggered to indicate failure.

This module enhances the visual appeal and makes the abstract concept of data transfer more tangible for users.

#### **3.2 Graph Module**

The Graph Module uses Chart.js to plot live data speeds in the form of a line chart. As the simulated file transfer progresses, this module receives backend data updates (every 500ms) via the /get-graph-data endpoint. The chart is updated in real-time with each new chunk of data, showing fluctuating speeds (in KB/s) to emulate real-world network behavior. It provides:

- X-axis: Chunk number or sequence.
- Y-axis: Transfer speed.

This graph is not only visually informative but also helps analyze the quality and performance of the simulated transfer.

### 3.3 Progress Module

The Progress Module displays a progress bar beneath the graph, which fills up in synchronization with the animation and graph updates. It reflects the real-time status of the ongoing file transfer operation by incrementally updating based on the number of data chunks sent.

In case of a failed transfer (every third cycle), the progress bar stops halfway (typically at 50%) to reflect an incomplete transfer. This visual cue helps users quickly identify whether the transfer reached completion.

### 3.4 Map Module

The Map Module leverages Leaflet.js, an open-source JavaScript mapping library, to simulate the client and server locations geographically. Every time a new transfer is initiated:

- Random coordinates are generated for the client and server.
- Both points are marked on the map using icon markers.
- A dotted polyline is drawn to visually connect them.

Additionally, a file icon moves along the path on the map from the client marker to the server marker. This movement is synchronized with the main animation, showing whether the file reaches its destination or fails mid-route.

### 3.5 Transfer Logic Module

This core logic module governs the success or failure of a file transfer operation. The design implements a deterministic pattern where:

- Every three transfers follow the pattern: Success → Success → Failure.
- This is handled via a transfer counter, incremented with each operation.
- The logic ( $\text{transferCounter} \% 3 \neq 2$ ) determines whether the current attempt is a success or failure

## CHAPTER 4

### CODING

#### **script.js**

```
const startBtn = document.getElementById("startBtn");
const fileIcon = document.getElementById("fileIcon");
const progressBar = document.getElementById("progressBar");
const statusMessage = document.getElementById("statusMessage");

let chart;
let chunkIndex = 0;
let map, clientMarker, serverMarker, fileMarker, line;
let transferCounter = 0; // 2 success, 1 failure cycle

function initChart() {
  const ctx = document.getElementById("transferChart").getContext("2d");
  const gradient = ctx.createLinearGradient(0, 0, 0, 200);
  gradient.addColorStop(0, 'rgba(54, 162, 235, 0.5)');
  gradient.addColorStop(1, 'rgba(54, 162, 235, 0)');

  chart = new Chart(ctx, {
    type: 'line',
    data: {
      labels: [],
      datasets: [{
        label: 'Transfer Speed (KB/s)',
        data: [],
        borderColor: 'blue',
        backgroundColor: gradient,
        borderWidth: 2,
        fill: true,
        tension: 0.3
      }]
    },
  },
```

```

options: {
  responsive: true,
  scales: {
    x: { title: { display: true, text: 'Chunks' } },
    y: { title: { display: true, text: 'Speed (KB/s)' } }
  }
}
});
}

```

```

function moveFileLocally(isSuccess = true) {
  const client = document.querySelector(".client");
  const server = document.querySelector(".server");

  const clientRect = client.getBoundingClientRect();
  const serverRect = server.getBoundingClientRect();

```

```

const totalSteps = 20;
let step = 0;
const distance = serverRect.left - clientRect.left;

```

```

fileIcon.classList.add("glow");
updateStatus("Sending...");
fileIcon.classList.remove("file-fail");

```

```

const failAt = isSuccess ? totalSteps : Math.floor(totalSteps / 2);

```

```

const interval = setInterval(() => {
  const left = (distance / totalSteps) * step + 40; // +40 to align starting near client
  fileIcon.style.left = `${left}px`;
  step++;

```

```

updateProgressBar(step, totalSteps);

if (step >= failAt) {
  clearInterval(interval);
  fileIcon.classList.remove("glow");
  if (!isSuccess) {
    fileIcon.classList.add("file-fail");
  }
}
}, 500);
}

function updateProgressBar(step, totalSteps) {
  const percent = Math.min(100, Math.floor((step / totalSteps) * 100));
  progressBar.style.width = percent + "%";
  progressBar.setAttribute("aria-valuenow", percent);
  progressBar.textContent = percent + "%";
}

function fetchGraphData() {
  fetch("/get-graph-data")
    .then(res => res.json())

    .then(data => {
      if (chart) {
        chart.data.labels = data.chunks;
        chart.data.datasets[0].data = data.speeds;
        chart.update();
      }
    });
}

```

```
}
```

```
function updateStatus(text, type = 'info') {
```

```
  if (statusMessage) {
```

```
    statusMessage.textContent = "Status: " + text;
```

```
    statusMessage.className = "alert alert-" + type;
```

```
  }
```

```
const fullScreenAlert = document.getElementById("fullScreenAlert");
```

```
if (fullScreenAlert) {
```

```
  if (type === "danger") {
```

```
    fullScreenAlert.style.display = "block"; // Show on failure
```

```
  } else {
```

```
    fullScreenAlert.style.display = "none"; // Hide on success/info
```

```
  }
```

```
}
```

```
}
```

```
function launchConfetti() {
```

```
  if (window.confetti) {
```

```
    confetti({
```

```
      particleCount: 100,
```


```
      spread: 70,
```

```
      origin: { y: 0.6 }  
    });
```

```
  }
```

```
}
```

```
}
```

```
//  Updated Map Animation with Accurate Fail Support

function animateFileOnMap(from, to, callback, isSuccess = true) {
  if (fileMarker) map.removeLayer(fileMarker);


  const duration = 10000;
  const steps = 100;
  const stepTime = duration / steps;
  let currentStep = 0;

  function interpolate(a, b, t) {
    return a + (b - a) * t;
  }

  const failStep = isSuccess ? steps : steps / 2;

  const interval = setInterval(() => {
    const lat = interpolate(from.lat, to.lat, currentStep / steps);
    const lng = interpolate(from.lng, to.lng, currentStep / steps);

    if (fileMarker) map.removeLayer(fileMarker);

    const iconHTML = isSuccess || currentStep < failStep ? "

```



```

function initMap(onMapReady = null) {
  if (!map) {
    map = L.map('map').setView([20, 0], 2);

    L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
      attribution: '© OpenStreetMap contributors'
    }).addTo(map);
  }

  fetch('/get-locations')
    .then(res => res.json())
    .then(data => {
      const client = data.client;
      const server = data.server;

      if (clientMarker) map.removeLayer(clientMarker);
      if (serverMarker) map.removeLayer(serverMarker);
      if (line) map.removeLayer(line);

      clientMarker = L.marker([client.lat, client.lng]).addTo(map).bindPopup("<br/>Client").openPopup();
      serverMarker = L.marker([server.lat, server.lng]).addTo(map).bindPopup("<br/>Server");

      line = L.polyline([[client.lat, client.lng], [server.lat, server.lng]], {
        color: 'blue',
        dashArray: '8, 10'
      }).addTo(map);



      const bounds = L.latLngBounds([client.lat, client.lng], [server.lat, server.lng]);
      map.fitBounds(bounds, { padding: [50, 50] });

      if (onMapReady) onMapReady(client, server);
    });
}

```

```

    });
}

//  Transfer Trigger
startBtn.addEventListener("click", () => {
  updateStatus("  Connecting...");
  fetch("/start-transfer")
    .then(res => res.json())
    .then(() => {
      chunkIndex = 0;


      const isSuccess = transferCounter % 3 !== 2;
      transferCounter++;

      moveFileLocally(isSuccess);

      const graphInterval = setInterval(() => {
        if (!isSuccess && chunkIndex >= 10) {
          clearInterval(graphInterval);
          return;
        }

        fetchGraphData();
        chunkIndex++;
        updateProgressBar(chunkIndex, 20);

        if (chunkIndex >= 20) clearInterval(graphInterval);
      }, 500);

      initMap((client, server) => {
        animateFileOnMap(client, server, () => {
          if (isSuccess) {
            updateStatus("  Transfer Completed!", "success");
          }
        });
      });
    });
}

```

```

        launchConfetti();
    } else {
        updateStatus(" + Transfer Failed!", "danger");
    }
}, isSuccess);
});
});
});

```

```

// 🚀 Init
window.onload = () => {
    initChart();
    initMap();
};

```

## App.py

```

from flask import Flask, render_template, jsonify
from threading import Thread
import time
import random

app = Flask(__name__)

# Global data structure to simulate graph values
graph_data = {
    "chunks": [],
    "speeds": []
}

# Background thread to simulate the file transfer
def simulate_transfer():
    graph_data["chunks"].clear()
    graph_data["speeds"].clear()

```

```

for i in range(1, 21):
    speed = random.randint(100, 500) # KB/s
    graph_data["chunks"].append(i)
    graph_data["speeds"].append(speed)
    time.sleep(0.5)

print("[✓] Transfer simulation complete.")

# Web Routes
@app.route("/")
def index():
    return render_template("index.html")

@app.route("/start-transfer")
def start_transfer():
    Thread(target=simulate_transfer).start()
    return jsonify({"status": "started"})

@app.route("/get-graph-data")
def get_graph_data():
    return jsonify(graph_data)

@app.route('/get-locations')
def get_locations():
    def random_location():
        return {
            'lat': round(random.uniform(-90, 90), 4),
            'lng': round(random.uniform(-180, 180), 4)
        }

    return jsonify({
        'client': random_location(),

```

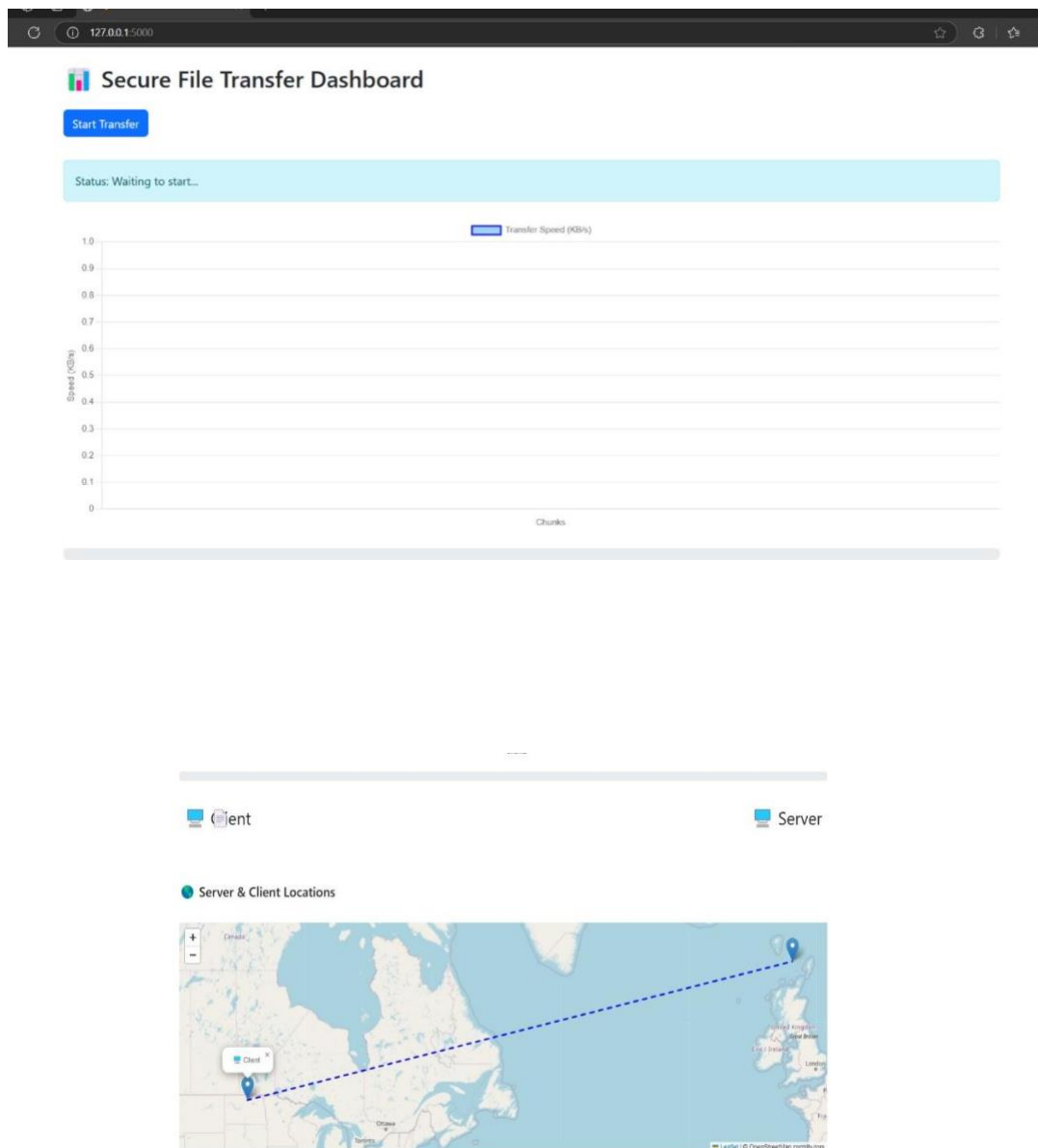
```
        'server': random_location()
    })

# Entry point
if __name__ == "__main__":
    app.run(debug=True)
```

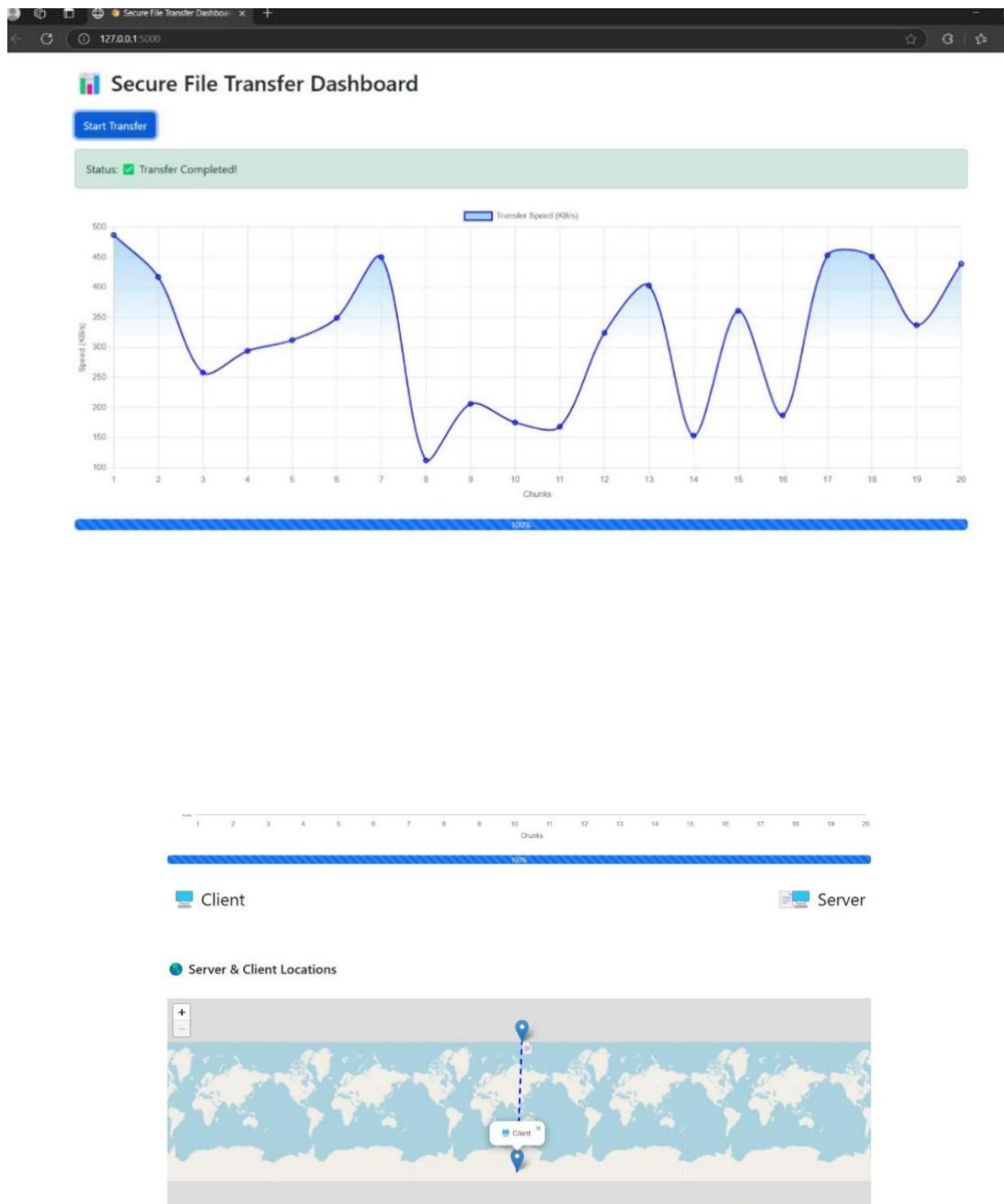
## CHAPTER 5

### SCREENSHOTS

- **Fig 5.1 Dashboard with graph and location**



- **Fig 5.2 File transferring animation**



- **Fig 5.3 File transfer failed state with red alert**





## **CHAPTER 6**

### **CONCLUSION**

The dashboard developed in this project offers an engaging and interactive way to visualize and understand secure file transfers. Through the integration of animated file transfer visuals, users can easily observe the movement of files between the client and server, simulating real-world file transfer scenarios. The incorporation of simulated failures, with visual cues indicating transfer success or failure, adds an educational element that allows users to better understand the dynamics of secure file transfers and how various network conditions can affect them.

The live graphs and progress bars complement the animation, providing real-time updates on the status of the file transfer. These features make the dashboard not only a functional tool for monitoring file transfers but also an effective resource for demonstrating network protocols, failure handling, and secure transfer techniques in a visually appealing manner.

This project serves as an excellent demo or educational tool, helping users and students grasp the intricacies of network security, file transfer protocols, and error recovery methods. By creating an intuitive interface with real-time network monitoring and diagnostics, the dashboard ensures that users can analyze, troubleshoot, and learn in an interactive environment, making it a valuable asset for both training and demonstrations.

Future improvements could include integrating more advanced security measures, adding real-world network traffic analysis, and further enhancing the interactivity of the dashboard for a more comprehensive educational experience.

## **CHAPTER 7**

### **REFERENCES**

- "Network Security Essentials: Applications and Standards" - William Stallings
- "Cryptography and Network Security: Principles and Practice" - William Stallings
- "Wireshark 101: Essential Skills for Network Analysis" - Laura Chappell
- "The Network Security Test Lab: A Step-by-Step Guide" - Michael Greg

