Case Study

SmartTransport Ticketing System with Weather-Aware Fare Adjustment**

Background

SmartTransport is a small transport company that runs buses between major cities in Ghana. They want to upgrade their ticketing backend. The company wants the system to automatically adjust ticket prices based on weather conditions to improve safety and maintain operational costs.

They have contacted your team to build a **backend-only ticket generation system**. There is no frontend. Everything happens inside Node.js.

Your task is to build an API endpoint that

1. Receives passenger travel details.
2. Fetches live weather data for the destination.
3. Calculates the final fare based on weather rules.
4. Generates a **PDF ticket**.
5. Saves the ticket into a folder.
6. Returns a link to download the ticket.

Scenario

Ama Boateng travels from **Accra** to **Kumasi** on 18th December 2025. She books her ticket using the SmartTransport mobile app. The mobile app sends the booking details to your backend.

Your backend must

- Pull the weather for **Kumasi** from the weather API.
- Adjust the fare.
- Generate a professional PDF ticket.
- Store the PDF in `/tickets`.
- Return a download link.

Guide for Students

How to Render HTML and Convert it Into a PDF in Node.js**

Your task is to create a system that first builds an HTML page with your ticket information, then converts that HTML into a PDF file stored in your backend. You will also create an endpoint that allows someone to download the PDF.

Follow the steps below.

1. Create an HTML template

Start by preparing a simple HTML file that represents how the ticket should look. This file will contain placeholders for values such as the passenger name, route, weather, and final fare.

For example, the template may contain text like
`{{name}}`
`{{from}}`
`{{weather}}`

Your backend will replace these placeholders with real data before turning the HTML into a PDF.

2. Load the template and insert the real data

Write a small function that reads the HTML file and replaces the placeholders with actual ticket details.

For example, if your HTML file has `{{name}}`, your function will replace it with the passenger's real name. After all replacements are made, you now have a complete HTML document containing real ticket information.

3. Convert the HTML into a PDF

Use a library like Puppeteer. Puppeteer acts like a browser inside the backend. It can open your completed HTML and print it as a PDF.

The process is simple:

1. Launch Puppeteer.
2. Load the completed HTML page.

3. Tell Puppeteer to save it as a PDF file.
4. Store the PDF in a folder in your backend, for example `/tickets`.

This gives you a finished ticket document.

4. Save the PDF with a unique name

Every ticket needs a unique file name.
A simple method is to use the current timestamp, such as `ticket_1734001234.pdf`.

Store the file inside the `/tickets` directory so your backend can find it later.

5. Build a download endpoint

Create an endpoint that accepts a file name in the URL.
This endpoint must check if the file exists, and if it does, return it to the user.

If the file does not exist, return a 404 message.

This allows anyone to download the ticket by clicking or opening a link like `/download-ticket/ticket_1734001234.pdf`.