Notice PDF Generation System - Design Documentation

System Architecture

The Notice PDF Generation System is designed to efficiently convert HTML templates into PDFs, store them in AWS S3, and provide secure download links using pre-signed URLs. It is built with a focus on scalability, performance, and resource efficiency.

Chosen Technologies

1. Node.js with TypeScript

- Its asynchronous, non-blocking nature makes it ideal for handling multiple PDF generation requests.
- TypeScript provides type safety, making the codebase more maintainable.
- It is a rich ecosystem with libraries

2. MongoDB

- Schema-less nature allows flexible storage of templates and dynamic data.

3. AWS S3

- Cost-effective and highly durable object storage solution.
- Provides seamless integration with pre-signed URLs for secure, time-limited access to stored PDFs.
- Reduces server storage burden, improving overall system performance.

4. Puppeteer

- Headless Chromium allows precise formatting and rendering and PDF generation
- Although the requirement needed to generate high-quality pdf for which the puppeteer seems to be great options.
- Provides complete control over HTML rendering, including styles and images.

5. PQueue

- Manages the task queue efficiently to control the number of concurrent Puppeteer page instances.
- **Limiting Concurrent Jobs** by controlling the number of PDFs being generated concurrently while queuing additional requests which prevents resource exhaustion by ensuring controlled execution of PDF generation tasks.

6. Docker

- Ensures consistent deployment across different environments.
- Isolates dependencies and simplifies scaling in containerized environments.

Implementation Details

1. Page Pooling in Puppeteer

- Instead of opening a new Puppeteer page for every request, we maintain a **pool of pages**.
- The pool is dynamically adjusted based on the number of active requests.
- When a request arrives:
    - It checks if an idle page is available in the pool.
    - If available, it reuses the page; otherwise, it creates a new one.
- Once the task is complete, the page is **returned to the pool** for reuse instead of being closed.

2. Efficient PDF Generation Workflow

1. **Template Creation**: Users define templates with dynamic data as well as the required fields.
2. **Notice Creation**: A notice is created by populating a template with dynamic data & the required fields.
3. **PDF Generation**:
    - The system fetches the HTML template and replaces placeholders with actual values.
    - A Puppeteer page is allocated from the pool.
    - The HTML content is loaded onto the page.
    - A PDF is generated and stored in S3.
4. **Pre-Signed URL Generation**: A signed URL is created for secure access to the stored PDF.

3. PQueue for Task Management

- Prevents overload by limiting the number of concurrent PDF generation tasks.
- Tasks exceeding the limit are queued and processed as soon as resources free up.

4. Memory Optimization

- Puppeteer processes are kept minimal to avoid excessive memory consumption.
- Page pooling ensures efficient resource usage.
- PDFs are streamed directly to S3 instead of being stored in memory.

System Flow Diagram

- **Step 1**: Create template → Store in MongoDB.
- **Step 2**: Create notice using template and data → Store in MongoDB.
- **Step 3**: Fetch notice and template → Replace placeholders with dynamic data.
- **Step 4**: Add PDF generation task to **PQueue**. **(Async)**
- **Step 5**: Generate PDF using **Puppeteer Page Pool**. **(Async)**
- **Step 6**: Store PDF in **AWS S3**. **(Async)**
- **Step 7**: Generate a pre-signed URL for downloading the PDF.

Performance Optimizations

- **Page Pooling**: Reduces browser instance creation overhead.
- **Task Queue (PQueue)**: Prevents overload and ensures smooth task execution.
- **Dockerization**: Allows running the service efficiently across environments.

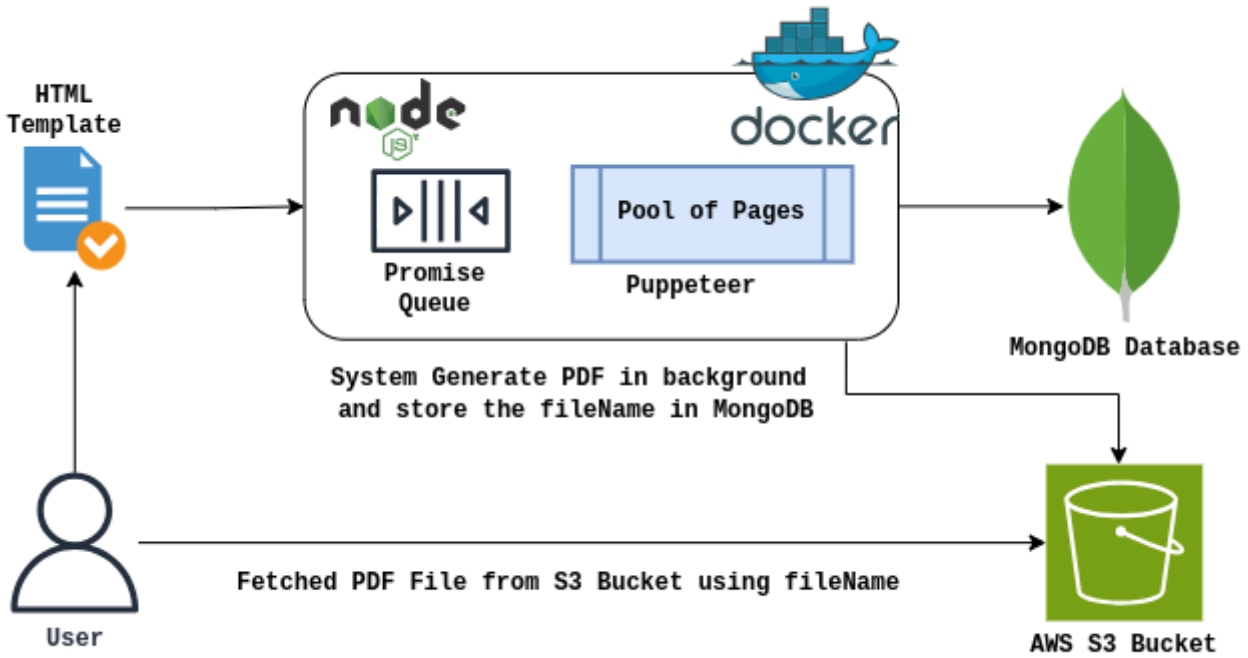
Performance Considerations

- The system can **generate 500-600 PDFs per minute** by optimizing task scheduling and Puppeteer instance management.
- Page reuse and pooling significantly reduce Puppeteer startup time.
- AWS S3 handles storage efficiently, reducing the load on the backend server.

Assumption Made

- The generated PDF will be used to create 1 notice PDF at a time
- A notice ID is unique, consisting of a template of a user (e.g., A notice PDF of a user with a Debt recovery HTML template)
- The user does not require PDF at the exact moment (Asynchronous) and can check the status and download the same in some time

Architecture Diagram



Performance Benchmarking:

1. As the requirement of the project is creating 500/600 pdf per minutes, we can consider that at least the system should provide a TPS of 10
2. We sent 20 requests simultaneously, and the system was capable of handling the response.