rocketseat BLOG

NODEJS BACKEND PDF

Avaliando estratégias de construção de PDF em NodeJS

por **João Paulo de Magalhães** há 2 anos

3 MIN DE LEITURA

Acredito que a maioria já teve alguma experiência com PDFs, seja ao receber a fatura do cartão de crédito por e-mail, ou até mesmo algum cliente passar alguma coisa, mas você já se perguntou como gerar um PDF com programação?

Recentemente aqui na Rocketseat precisei me fazer essa pergunta, isso por que existia a necessidade de automatizar a geração de certificados para nossos alunos, que até então eram feitos na mão.

E foi neste momento que eu tive a minha primeira escolha a ser realizada: "Onde que eu devo gerar estes PDFs?", basicamente eu tinha duas opções, gerar eles no front ou no back end. Só que antes de tomar essa decisão, eu precisava primeiro pensar nos casos de uso deste certificado, assim iria conseguir colocar os prós e contras na mesa. E no nosso caso foram:

- Consulta externa para validação do certificado
- As informações contidas no certificado são imutáveis
- Possibilidade de no futuro enviar o certificado por e-mail

Colocando isso em análise, se tornou bastante claro que era extremamente inviável gerar isso no front-end, pura e simplesmente por uma questão de logística. Perfeito, primeira

decisão tomada, porém meu alívio durou alguns segundos, pois precisava agora escolher qual **lib** eu iria utilizar.

Escolhendo a lib

Se tratando de javascript, já imaginamos que deva existir infinitas libs disponíveis para nossa felicidade. E no aqui no nosso caso, as libs que pesquisamos eram:

- node-html-pdf
- PhantomJS
- pdfkit
- pdfmake

Após a pesquisa inicial das libs, basta agora decidir se queremos gerar um PDF baseado em um HTML, ou se iremos utilizar uma API mais abstrata.

Uma das coisas que fiquei curioso foi pensar como que é feito essa transformação mágica de HTML pra PDF a partir do node e logo descobri que na verdade a maioria das soluções envolvem utilizar um headless browser para realizar tal ato. Foi aí que fiquei preocupado e pensei "Será que é interessante importar essa lib com 20kg de dependências?".

E cheguei a conclusão de que se o seu projeto já tem a necessidade de um *headless browser* maravilha, não tem por que não aproveitar e também gerar o PDF, mas no nosso caso,

não temos essa necessidade, então eu acabei descartando as libs **node-html-pdf** e **PhantomJS** pois ambas trabalhavam desta forma.

Isso nos deixa apenas com **pdfkit** e **pdfmake** nas mãos. Ao olhar um pouco mais em detalhes a **pdfmake** conseguimos observar que ela é baseada na **pdfkit**. Ambas as libs parecem receber atualizações regulares e ter uma boa comunidade em volta. Porém confesso que fiquei um pouco com o pé atrás na **pdfmake** pois eles estão aparentemente trabalhando em uma próxima versão e não separaram isso de uma maneira muito legal no repositório de código.

Mas o que me levou a bater realmente o martelo foi o fato de que a documentação das duas versões disponíveis do **pdfmake** não tinha a possibilidade de visualização separada, ou seja, não é possível saber o que estava sendo alterado de uma versão para outra. E como o **pdfkit** é a base, resolvi então seguir com ele.

Utilizando o pdfkit

A sua utilização acaba sendo bem simples e direta. Com uma api relativamente bem pequena, fica muito fácil saber o que deve ser feito para tirar o resultado da sua cabeça e colocá-lo em um PDF.

```
const PDFKit = require('pdfkit');
const fs = require('fs');

const pdf = new PDFKit();

pdf.text('Hello Rocketseat PDF');

pdf.pipe(fs.createWriteStream('output.pdf'));
pdf.end();
```

Com o código acima, estamos colocando um texto no PDF e também já salvando o resultado em um arquivo **output.pdf**.

A lib também conta com possibilidade de formatação de texto para deixar do jeito que você precisa.

```
pdf
  .font('SourceSansPro-Regular')
  .fontSize('13')
  .fillColor('#6155a4')
  .text('Texto formatado', {
    align: 'center'
  })
```

Como também não podia faltar, a inserção de imagens também é tranquilo de ser realizado:

```
// Adiciona uma imagem na posição X: 300 e Y: 300
pdf.image('images/Rocketseat.png', 300, 300)
```

Finalizando

Resolvi compartilhar essa experiência de como foi se aventurar no mundo da geração de PDFs com javascript pois muitas vezes não imaginamos a quantidade de escolhas que temos que realizar para uma simples tarefa ou propor uma solução a um problema.

Em um mundo onde existem + de 15 libs que fazem exatamente a mesma coisa e muitas vezes até do mesmo jeito é bem comum esquecermos de dar a devida atenção na escolha da que melhor encaixa para as nossas necessidades, quando na verdade, apenas o fato de dedicarmos um tempo para isso pode nos salvar de muita dor de cabeça futura 😉

Por enquanto é isso! Caso tenha qualquer dúvida, pode deixar nos comentários abaixo!

Abraços, e até a próxima!

PRÓXIMO POST

React Native em 2019, nova arquitetura e comparações com Flutter

POST ANTERIOR

Gerenciando variáveis ambiente no NodeJS

Deixe sua reação...

10 Responses

Gostei! Amei! Uau!

Não gostei!

Comentários **Comunidade** **Entrar**

Recomendar 8 Tweet Compartilhar

Ordenar por Mais votados

Participe da discussão...

FAZER LOGIN COM

OU REGISTRE-SE NO DISQUS ?

Nome

Victor Paiva Torres • 2 anos atrás

Parabéns pela publicação. Obrigado por tirar um tempo para compartilhar sua experiência gerando documentos pdf.

Um benefício muito bom em utilizar libs que convertem html para pdf é poder tornar mais fácil o processo de criação e atualização desses documentos. Se a empresa possui uma equipe de design e desenvolvedores frontend, dá pra trabalhar com os modelos de documentos pdf sem precisar envolver a parte da equipe mais focada no backend e outras demandas estratégicas, escrevendo html puro e simples.

Também é possível reaproveitar a base de código utilizada para gerar templates dinâmicos, se for o caso.

5 | 1 • Responder • Compartilhar ›

Rodrigo Campos • 2 anos atrás

como ficaria a geração de tabelas?

1 | • Responder • Compartilhar ›

João Paulo de Magalhães
Rocketseat ➚ Rodrigo Campos
• 2 anos atrás • edited

Fala Rodrigo, tudo bem?

A geração de tabelas com a lib escolhida (pdfkit) infelizmente não é possível no momento, até existem algumas issues abertas no Github, mas são bem antigas, então a chance disso acabar saindo é meio baixa.

O que você pode fazer é utilizar a lib **pdfmake** que inclusive é baseada na pdfkit, esta sim tem a funcionalidade de geração de tabelas.

O link da documentação para gerar tabelas com pdfmake é esse https://pdfmake.github.io/d...

1 | • Responder • Compartilhar ›

Bruno Aderaldo • 2 anos atrás

Utilizei o pdfmake para construir pdfs no front. Era pra gerar aqueles cartazes de super mercado. A dor de cabeça mesmo é só com questões de espaçamento e tal. Quando se quer um pdf simples é bem fácil e rápido. Mas quando tem que montariam layout mais específico e que seja dinâmico, torna um pouco mais complicado. Mas deu tudo certo, experiência bacana. Ótimo artigo.

1 | • Responder • Compartilhar ›

Bruno Sousa • um ano atrás

Olá pessoal, alguém poderia me tirar uma dúvida é porque to usando essa **pdfkit** Só que eu tava querendo gerar pdf no backend e

enviar para o mobile , porém não to conseguindo decodificar no lado do aplicativo

segue o exemplo que to usando:

```
let doc = new PDFDocument();
doc.info['Title'] = 'test Document';

doc.pipe(res);
doc.pipe(fs.createWriteStream('output.pdf')
doc.text('Hello Bruno Sousa PDF');
doc.end();

res.status(200).send();
```

minha dúvida é o que o doc.pipe(res) armazena no res como resposta, se é um blob ou algo desse tipo.

Responder • Compartilhar ›

Carlos Alberto Junior • um ano atrás

Primeiro parabéns pelo post.
Utilizo o Adonisjs como backend e estou tentando utilizá-lo com o Pdfkit, mas estou com problemas ao retornar o arquivo para o frontend em Angular 6.

Segue o código abaixo:

```
testPdf({ response }) {
var doc = new PDFDocument();

doc.fontSize(25).text("Here is some vector
graphics...", 100, 80);

doc
.save()
.moveTo(100, 150)
.lineTo(100, 250)
```

ver mais

Responder • Compartilhar ›

Raphael • um ano atrás

Poderia postar o resultado final do PDF gerado?

Responder • Compartilhar ›

Valdir Mendes • um ano atrás

Poha! Ajudou para caramba!

• Responder • Compartilhar ›

Rodrigo Andrade • 2 anos atrás

Fala João!

Alguns meses atrás também fiz os estudos das
ferramentas que você cita no artigo. Porém, na empresa aonde trabalho,
utilizamos os serviços da AWS. E um dos problemas que sempre
encontravamos é que o PhantomJS e similares, baixa um executável do
chromium/Chrome. E com isso, as políticas da AWS não permitem que