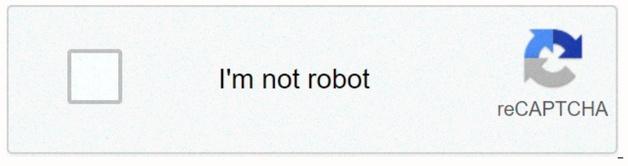

Continue

Dynamic html to pdf nodejs

I'm pretty new at NodeJs. I'm trying to parse it and download some html from the website to present some information for debugging. I try to succeed in the http module (see this article), but when printing chunks in this way, var req = http.request (options, functions (res) {res.setEncoding (utf8), res.on (data, functions (chunks){.log (chunks);} Print the < a0>. </a0> I don't get all the html that is dynamically loaded in ajax for example: <div class=container> ::before <div class=row> :::before.. </div> Is there any other module that can help me with this goal?thanks! I want to share my success with you (thanks @oKonyk) Note that npm install phantomjs can @oKonyk script, but if you want the script to run locally, you must set the following options: Phantom.create({parameters: options}, functions (){});This tutorial describes how to convert static Websites using HTML, CSS, and JavaScript (JS) to dynamic Websites using MongoDB, Express, Static HTML, CSS, JS, and Nodes .js. Similar to a typical MEAN/MERN stack (MongoDB, Express, Angle or Reaction, and Node .js), but instead of using Angular or React, use a template engine called Embedded JavaScript (EJS). Other popular template engines include handlebars, pugs and nunjuks. Learning a template language is easier than the JS framework because you can simply write HTML to insert the same code into multiple locations (called partials) or pass server-side variables (such as user names) that appear on the front end. First, .js the Node server, make sure you have node access installed on your local .js or VPS hosting provider. If you do not have it installed.js the web site on the node. Node .js allows you to write server-side code using a special form of JavaScript, so you can use a language you're already familiar with. The node .js installer is bundled with the package manager NPM. NPM is a repository for node modules and a reusable part of the code that can extend the functionality of the server. This is similar to a plug-in repository, where node modules can be thought of as code snippets or libraries (depending on their size). Windows users: Nodes and NPMs must be added to path for easy call at the command line. For detailed instructions, see the guide on how to install MongoDB. To test that the installation test installation is functioning properly, open the terminal window, and if the messages for nodes -v and npm -v results start with v, followed by some number (indicating the version), the installation succeeded. You are now ready to create your first server. After you create a static website, the first step is to .js express web server when you create a node-based app. First, move all static files (HTML, CSS, JS, images, etc.) of your Web site to a folder called public and create a file named server.js at the root. </div>of your website folder. The server .js file type: // Load var express = require('express') Express var app=express() in the node module. App.use (express.static('public')) that renders static files. The port's website runs on app.listen (8080). Then, in the terminal, type: Press ENTER to accept the default parameters for all of the following options, but verify that the entry point .js the server: Finally, enter npm start and move (the IP address of the VPS host, or localhost:8080/index.html (or the name of the web page) in the browser. Your Express server should provide a static file for your Web site. The next step forward is to learn how to use the EJS template engine to convert static files to dynamic files, and how to use parts to copy code repeatedly and insert server-side variables into the front end. Simple npm installation ejs --save does the trick. The –save parameter stores the module in a package.json file so that all users who clone the git repository (or download files for the site) can use the npm install command to install all node modules (called dependencies) required for the project. Convert static pages to EJS files Next, you need to convert static HTML files to dynamic EJS files and set up folder structures in the way that EJS expects. Create a folder named views in the root directory of the Web site, and then create two sub folders in that folder: pages and partials. Move all HTML files to the page's subfolder .html the file extension to .ejs. The folder structure should look like the image below. Code reuse - Create the first EJS part When you create a static site, you often repeat the code on all pages, such as the head (where the meta tags are located), headers, and footer sections. If you need to make changes, it's inconvenient to change them on every page (especially large sites), but you don't need to change them if you use EJS partials. When you edit a template (partial) file, the code on all pages that contain the file is updated. Use headers as an example to take up a common part of the Web site that you want to template. Creates a new file named header.ejs in a partial folder. Copy and paste all the code between tags on the EJS page <header></header> Finally, delete the code between <header></header> and tag (the same code you copied to the header.ejs partial file) on every page with a header, and then click on it <% include ('.. /partials/header') replace it with %>. You have now created the first EJS part. Repeat this procedure for other repeating code, such as head and footer sections. Small tip: If you have difficulty distinguishing between pages and partials, follow these steps because the .ejs file extensions are the same. With an underscore _ before the partial name, _ header.ejs. This is a name convention used by some developers and is useful. Rendering EJS pages Now we reach the exciting parts and render EJS pages and partials so that the server can see them on the front end. Initializes .js example // loadnode module var express = require('express') constant ejs = required ('ejs'); express var app=express(). App.use (express.static('public')) that renders static files. Set the view engine to ejs app.set ('view engine', 'ejs'). The port's website runs on app.listen (8080). GET root - display page ***// root root app.get(',function (req, res) { res.render('page/index') } } . First, you need to add the EJS node module to the server. Therefore, .js add const ejs = require('ejs') to the server's file (see example above). Then you need to tell the express server to use EJS, so add app.set ('view engine', 'ejs'). Now you need to configure the route. The root tells the server what to do if the user moves to a specific URL in the website, such as . There are usually two types of GET routes: the GET route and the POST route, which upload data from the front end to the server (usually via a form) before the page is rendered and the uploaded data is used in any way. You only need to view the EJS page, so you just need to use the GET route. Add .js after the app.listen(8080) line of the server. For index pages, the root looks like this: // *** GET root - display page ***// root root app.get(', function (req, res) { res.render('page/index') } ;' /' specifies the URL of the Web site that the code activates, and req represents the request and requests a response. Therefore, the response .com when you go to is rendering the page/index.ejs page (displayed in the browser). Add a similar route to other EJS pages. Aside from code reuse, the main attraction of templates is that server-side variables can be passed to the front end. A single variable, such as the user name of the current user, or an array like the details of all registered users. However, when using an API or database, the real strength of passing server-side variables becomes apparent. As a basic example, the following code displays Louise on the h2 tag on the index page.js. The first name is the name of the EJS variable (the name to display on the front end), and the second name is the variable that contains the data to send. (They don't have to be the same.) Index .ejs <h2>Name is <%= Name %></h2> For simple arrays, you can use this example instead to create a p-tag for all names in the listnames variable.js Render index page res.render ('page/index', {//EJS variable and server-side variable list name: list name }). });index.ejs <% list name.forEach(function (name) { %> <p><%= name %></p> <% }); Congratulations on %> converting a static .js to a node-based web app. You also created your first Express Web server and learned the basics of EJS templates. Are you .js to deploy your first Node-free web app.js and show it off? If you want to learn the skills learned in this tutorial, we recommend that you use EJS templates to work with apIs and databases. If you need help in any of these areas, use the database to hold my hand and check get a grip in the API guide. Start.