

# A Static Node.js and Express Site

---

## DESCRIPTION

### SHORT DESCRIPTION

Node.js and Express make a powerful pair when used together on the server side, and are used everyday to make fast, modular and dynamic web applications. For this project, you'll use Node.js, Express and Pug templates to create a portfolio site to show off the projects you've built.

### LONG DESCRIPTION

So far you've worked with a lot of "client-side" JavaScript. Which is just an industry way of saying JavaScript in the browser. All of the JavaScript you've written for the first five projects in this Techdegree is run right in the browser. Now it's time to continue your journey out of the browser and start writing some powerful "server-side" JavaScript. In this project, you'll create a gorgeous portfolio site to showcase the great projects you've built. The site will contain a modern landing page, an about page where you'll have a chance to share contact info, practice your elevator pitch and talk a little about yourself, and a series of project pages to show off and detail at least your first five projects from this Techdegree.

You'll create a JSON file to store all the data about the projects you've created.

You'll use Pug to complete provided templates that utilize the JSON to generate the markup that is ultimately displayed in the browser.

You'll use Node.js and Express to:

- Import the required dependencies
- Link the JSON with the Pug templates
- Set up routes to handle requests
- Set up the middleware to utilize static files like CSS
- Handle errors

- Set up a server to serve the project

After building this project, you should have a comfortable working knowledge of Node.js, Express and Pug, setting up a server, handling requests, working with server-side JavaScript, and building a powerful and modern back end project. And you'll have an important new skill to make you more marketable as a Full Stack JavaScript Developer.

---

## 25.0 HOURS TO COMPLETE

---

## PREREQUISITES

- Review the policy on Reusing Code in a Techdegree project.
- GitHub
  - Create a new repo for this project.
  - Create a .gitignore and use it to make sure your `node_modules` folder is not stored in or tracked by your repo.
  - Create a README.md file for your repo that explains what the project is and anything your user or fellow developers might need to know to use the project.
  - Create a license for your project. Learn more about that here.
- Understand what you are working with
  - Have a basic understanding of Node.js, Express, and Pug. See the Resources links on this page and the material in this unit for more help understanding these concepts.
- Download the project files. We've supplied several files for you to use in four folders:
  - `example-html` - contains three HTML files that you can open in the browser to see an example of what the final project should look like.
  - `mockups` - contains three .png files that demonstrate what the final project should look like.
  - `public` - contains the necessary CSS and client side JS for the project. For the basic requirements of this project, you won't need to do anything with these files,

except to reference the folder when you set up your middleware.

- `views` - contains the four Pug files you will need for this project. You won't need to create any new Pug files, but you will need to add info to each Pug file to make it work with your project. We've provided comments in each file to help guide you through what needs to be done in each file.
- Follow the Instructions Below
  - Be sure to reach out on Slack if you get stuck or run into difficulties

---

## SKILLS

- Node.js
- Express
- Pug
- Creating a server
- Setting up routes
- Middleware
- Handling server-side errors
- Creating and managing JSON data, connecting it to a templating engine with Node, and utilizing it in Pug templates
- Portfolio practice
- Creating and optimizing images for the web

---

## INSTRUCTIONS

- Initialize your project
  - Open the command line, navigate to your project, and run the `npm init` command to set up your package.json file.
- Add your dependencies

- At a minimum, your project will need Express and Pug installed via the command line.
- Don't forget to use the `--save` flag if you're using a version of npm prior to 5.0, to ensure that references to the dependencies are stored in your package.json file.
- Handle files and folders that shouldn't be stored in your repo
  - Create a .gitignore file in your directory and save a reference to the `node_modules` folder so that your repo doesn't store or track that folder.
- Images
  - Create an images folder in your directory to store your images.
  - Add a profile pic of yourself that you would feel comfortable sharing with potential employers. It should present well at 550px by 350px.
  - Take screenshots of your projects. You will need at least two screenshots for each project.
    - A main shot for the landing page which should be a square image that can display well at 550px by 550px.
    - Between one and three additional images that can be any dimensions you want, but work well in this project as landscape images that present well at 1200px by 550px.
- Add your project data to your directory
  - Create a data.json file at the root of your directory
  - The recommended structure for your JSON is to create an object literal that contains a single property called `projects`. The value of projects is an array containing an object for each project you wish to include in your portfolio.
  - Each project object should contain the following properties:
    - `id` - give each project a unique `id`, which can just be a single digit number starting at `0` for the first project, `1` for the second project, etc.
    - `project_name`
    - `description`
    - `technologies` - an array of strings containing a list of the technologies used in the project
    - `live_link` - link to the live version of the project
    - `github_link` - link to the GitHub repo

- `image_urls` - an array of strings containing file paths from the views folder to the images themselves. You'll need a main image to be shown on the landing page, and three images to be shown on the project page.
  - **Note:** Feel free to add extra projects to this portfolio if you have them to show off.
- Setup your server, routes and middleware
  - Create an `app.js` file at the root of your directory.
  - Add variables to require the necessary dependencies. You'll need to require:
    - Express
    - The `path` module
    - Your `data.json` file
  - Set up your middleware:
    - set your "view engine" to "pug"
    - use a static route and the `express.static` method to serve the static files located in the `public` folder
  - Set your routes. You'll need:
    - An "index" route (`/`) to `render` the "Home" page with the locals set to `data.projects`
    - An "about" route (`/about`) to `render` the "About" page
    - Dynamic "project" routes (`/project` or `/projects`) based on the `id` of the project that `render` a customized version of the Pug project template to show off each project. Which means adding data, or "locals", as an object that contains data to be passed to the Pug template.
  - Finally, start your server. Your app should `listen` on port 3000, and log a string to the console that says which port the app is listening to.
- Handle errors
  - If a user navigates to a non-existent route, or if a request for a resource fails for whatever reason, your app should handle the error in a user friendly way.
  - Add an error handler to `app.js` that sets the error message to a user friendly message, and sets the status code.
  - Log out a user friendly message to the console when the app is pointed at a URL that doesn't exist as a route in the app, such as `/error/error`.
  - Refer to the video on Error handling Middleware, which is linked in the project resources list.

- Complete your Pug files
  - Go through each of the four Pug templates to inject your data. The Pug files contain comments that detail each change you will need to make. You can and should delete these comments when you are finished with this step. But you should wait to do so until everything is working as it should, in case you need to refer to these notes during development.
  - Leave the example HTML files in your project so your reviewer can reference them.
- Layout, CSS and styles
  - The layout of the finished project should match the provided mockups.
  - Without affecting the layout, consider making adjustments to things like the color, background color, fonts, shadows, transitions and animations to make this project your own.
  - You can even add additional assets like more images or a even a customized logo to the header. If you do so, just make sure your changes work on the mobile layout before dialing them in on the desktop layout.
- Add good code comments
- Check for cross-browser consistency
- Review the "How you'll be graded" section
- Quality Assurance and Project Submission Checklist
  - Perform QA testing on your project, checking for bugs, user experience and edge cases.
  - Check off all of the items on the Student Project Submission Checklist.
- **NOTE: Seeking assistance**
  - If you're feeling stuck or having trouble with this project:
    - Review material in the unit.
    - Practice your Google skills by finding different ways to ask the questions you have, paying close attention to the sort of results you get back depending on how your questions are worded.
    - Reach out to the team on Slack.
- **NOTE: What you submit is what will get reviewed.**

- When you submit your project, a snapshot is taken of your repository, and that is what the reviewer will see. Consequently, any changes you make to your repo after you submit will not be seen by the reviewer. So before you submit, it's a smart idea to do a final check to make sure everything in your repo is exactly what you want to submit.

---

## EXTRA CREDIT

- Customize the package.json file
  - Set up your package.json file so that running `npm start` will run the app.
- Use error handling middleware to render a Pug template
  - Create a new Pug template in the `views` folder and name it `error.pug`. This Pug file should extend the layout, be set to block content, and display the `error.message`, `error.status`, and `error.stack` properties.
  - When the request URL is for a non-existent route, the `error.pug` template should be displayed in the browser along with the following properties:
    - `error.message`
    - `error.status`
    - `error.stack`
- Customize the style
  - Change or add at least three of the following to make this project your own:
    - `color`
    - `background color`
    - `font`
    - `box or text shadows`
    - `transitions or animations`
    - `add a logo`
  - You can either add your changes to the end of the CSS file or add your own and link it in the head of the `layout.pug` file, below the other CSS links.

- Document your style changes in your README.md file and the project submission notes.
- Do not alter the layout or position of the important elements on the page.

- **NOTE:**

- To get an "Exceeds Expectations" grade for this project, you'll need to complete **each** of the items in this section. See the rubric in the "**How You'll Be Graded**" tab above for details on how you'll be graded.
- If you're shooting for the "Exceeds Expectations" grade, it is recommended that you mention so in your submission notes.
- Passing grades are final. If you try for the "Exceeds Expectations" grade, but miss an item and receive a "Meets Expectations" grade, you won't get a second chance. Exceptions can be made for items that have been misgraded in review.