Project Instructions

To complete this project, follow the instructions below. If you get stuck, ask a question on Slack or in the Treehouse Community.

^ 13 steps

1 **Initialize your project**

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

2 **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.

3 **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.

4 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.

5 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, which can be hosted for free on GitHub pages. Check the project resources list for a helpful reference link.
    - `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 one to 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.

6 **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
    - Your `data.json` file
    - **Optionally** - the `path` module which can be used when setting the absolute path in the express.static function.
- 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/:id` or `/projects/:id`) 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.

7 **Handle errors**

- If a user navigates to a non-existent or undefined route, such as `/noroute` or `/project/noroute`, or if a request for a resource fails for whatever reason, your app should handle the problem in a user friendly way.
- Since incoming requests to undefined routes aren't technically "server errors", Express doesn't throw an error when this happens. So you need to create you're own way of catching this event when it happens and responding appropriately. You do this with a 404 handler, which you'll add near the bottom of `app.js` to catch any requests for undefined routes.
    - The 404 handler should create a custom new `Error()`, set its `status` property to `404` and set its `message` property to a user friendly message. Then the 404 handler should log out the new error's message and status properties.
- After the 404 handler in `app.js` add a global error handler that will deal with any server errors the app encounters. This handler should ensure that there is an `err.status` property and an `err.message` property if they don't already exist, and then log out the `err` object's `message` and `status`.
- Test your error handling by pointing the browser at a few undefined routes, like `/noroute` and `/project/noroute`, as well as temporarily throwing an intentional 500 error in one of your routes and then navigating to the page for that route.

For more information on handling errors, check out this **Practice Error Handling in Express** practice session.

8 **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.

Note: Consider adding a target attribute set to `_blank` on the a tags for the live links to your projects so that they open in a new window.

---

9 **Layout, CSS and styles**

- The layout of the finished project should match the provided mockups.
- To really make this project your own, you should customize the CSS following the suggestions in the Extra Credit section at the bottom of this page.

---

10 **Add good code comments**

Extra Credit

To get an "exceeds" rating, complete all of the steps below:

^ 3 steps

1 **Customize the package.json file**

- Set up your package.json file so that running `npm start` will run the app.

2 **Render helpful Pug templates in your error handling middleware**

- Create two new Pug templates in the `views` folder and name them `error.pug` and `page-not-found.pug`. These Pug files should extend the layout, be set to block content, and display the error's `message`, `status`, and optionally, the `stack` properties.
- When the status property is a 404, the `page-not-found.pug` template should be rendered. For any other error status codes, the `error.pug` template should be rendered. Be sure that that you're passing the `{error}` or `{err}` that you're creating or updating as the second parameter in the render method.
- These templates should include a link back to the home page.

3 **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
  - a custom logo
- Your can either add your changes to the end of the CSS file or add your own CSS file 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.
- Do not alter the layout or position of the important elements on the page. The basic layout and style should generally match the mockups.