Prerequisites

- A GitHub account: get one at https://github.com/join (free plan)
- A git client installed
- Jekyll installed: https://jekyllrb.com/docs/installation/
- A good text editor (eg Visual Studio Code): https://code.visualstudio.com/download

If you have successfully installed the software, all the following commands will work and output version numbers:

- `git --version`
- `bundle --version`
- `jekyll --version`

Getting Started

- Fork the starter repo from Suze's GitHub account into your own GitHub account: https://github.com/SuzeShardlow/github-pages-workshop
- Using the terminal / command line:
  - Clone the repo (`git clone <url of your forked repo>`).
  - Go to the folder that you cloned the repo to (`cd github-pages-workshop`).
  - Install dependencies: `bundle install`
  - Start Jekyll: `bundle exec jekyll serve --watch`
- Use a browser to navigate to: http://127.0.0.1:4000/github-pages-workshop/
- You should see the start point which is a completely unstyled basic HTML site.
- Open the folder that you cloned the repo to with your editor (VS Code for example). This is your project folder.

Using the Template Styling: Copy CSS, Web Fonts, JavaScript

Each template comes with some CSS, JavaScript and web font assets. In this task, we'll copy these into our Jekyll project.

- Copy all files and subfolders from your chosen template's `assets` folder into the project's `assets` folder. (This is the folder named `assets` in the root of the project, that currently contains one sub-folder named `images`).
- For example if you chose dopetrope, the `assets` folder for that template is `html-templates/dopetrope/assets` and everything in there (including sub-folders) should be copied to your project folder's `assets` folder:
  - `cp -r html-templates/<template name>/assets/* assets`
  - e.g. `cp -r html-templates/dopetrope/assets/* assets`

Updating Site Configuration

We need to update some sitewide configuration values that Jekyll will use on all of our site's pages.

- Open the file `_config.yml` in your editor.
- Configure your site's name, and your name as the site owner:
    - Change the value of `name` from `My Site` to the name you'd like to use for your site.
    - Change the value of `owner` from `My Name` to your own name.
- Save your changes.

Building the Home Page

Copy Your Chosen Template's Home Page HTML

- Open your chosen template's `homepage.html` file in your editor.
    - For example if you chose dopetrope, the home page is in `html-templates/dopetrope/homepage.html`
- Copy all of the HTML from this file.
- Paste it into the home page layout: `_layouts/home.html`, replacing everything in that file.
- Save your changes to `_layouts/home.html`.

Update Header and Footer Areas with Site and Owner Name

In this task, we'll replace some placeholder values with the ones that you set in `_config.yml` and make sure our template assets were copied correctly.

- Fix the site name in the Header area:
    - Find and replace the text `TODO Site Name` with `{{site.name}}`
- Fix the copyright message in the Footer area:
    - Find and replace the text `TODO Site Owner` with `{{site.owner}}`
- Save your changes.
- On the command line, stop Jekyll with `Ctrl-C` and restart it with `bundle exec jekyll serve --watch` (to pick up configuration changes).
- Refresh http://127.0.0.1:4000/github-pages-workshop/ and check that this now shows the home page with the template you chose (dopetrope / verti / hyperspace).
- On the page and in the title bar of the browser tab you are using, it should also show your chosen site name that you set in `_config.yml`.
- Check that your name appears in the copyright statement at the bottom of the page.

Create Reusable Header and Footer Includes

Now we'll make the page header / navigation reusable across multiple pages using Includes.

- Cut all the markup between `<!-- start header include -->` and `<!-- end header include -->` and paste it into `_includes/header.html`
- In `_layouts/home.html`, place the following code between `<!-- start header include -->` and `<!-- end header include -->`
    - `{% include header.html %}`
- Save your changes.
- Refresh http://127.0.0.1:4000/github-pages-workshop/ and check that the page looks the same as before.

Next, we'll repeat this process to make a reusable footer include:

- Cut all the markup between `<!-- start footer include -->` and `<!-- end footer include -->` and paste it into `_includes/footer.html`
- In `_layouts/home.html`, place the following code between `<!-- start footer include -->` and `<!-- end footer include -->`
    - `{% include footer.html %}`
- Save your changes then refresh the page in the browser, making sure that it looks the same as before.

Set Up Home Page Content and Image

Here, we'll add the home page content and image which is defined in the file `index.md`.

- Make sure you are still editing `_layouts/home.html`. Replace all of the Lorem Ipsum placeholder text and associated `<p>` tags with `{{content}}`
- Replace the text `https://via.placeholder.com/1600x1200` with `{{site.baseurl}}{{page.main_image}}`
- Save your changes.
- Refresh http://127.0.0.1:4000/github-pages-workshop/ and check that the main content text now matches the content that's in `index.md`, and that you have a real image on the home page.

Building the Projects Page

Now we are going to build a page that showcases our project portfolio, with links to individual detailed project pages.

Copy Your Chosen Template's Projects Page HTML

- Go to http://127.0.0.1:4000/github-pages-workshop/projects/ and ensure that you see the basic no design project list page.
- Open your chosen template's `projects.html` file in your editor.
    - For example if you chose dopetrope, the projects page is in `html-templates/dopetrope/projects.html`
- Copy all of the HTML from this file.
- Paste it into the projects page layout: `_layouts/projects.html`, replacing everything in that file.
- Save your changes to `_layouts/projects.html`.
- Refresh http://127.0.0.1:4000/github-pages-workshop/projects/ and check that this now shows the template projects page.

Configure the Projects Page to Use the Projects Collection

Now we need to replace the six example projects with a loop to dynamically render one project "box" per project. Later, when you add new projects, the page will automatically pick them up when Jekyll builds it.

Each project's details are stored in its own markdown file in the `_projects` folder. All the projects need the same keys in their front matter so we can write logic that works for the whole Collection.

- Build a loop that adds one "project box" for each project in the collection:
    - Remove 5 of the 6 project div elements from `_layouts/projects.html`. These divs begin at the line below the comment: `<!-- project box starts here -->` and end at the comment `<!-- project box ends here -->`
    - Save your changes and refresh the projects page to make sure only one project appears now.
    - We now need to add a loop around that div, so that Jekyll adds one project div per project from our projects collection in the `_projects` folder.
    - Underneath the comment `<!-- project for loop starts here -->`, add `{% for project in site.projects %}`
    - On the line above the comment `<!-- project for loop ends here -->` add `{% endfor %}`
    - Save your changes and refresh the projects page - you should now have 6 project boxes again.

- Replace the placeholder content with real project content.
    - Replace `TODO Project` with `{{project.title}}`
    - Replace `TODO Project subtitle` with `{{project.subtitle}}`
    - Replace `https://via.placeholder.com/1600x1200` with `{{site.baseurl}}{{project.main_image}}`
    - Replace all instances of `href="project.html"` with `href="{{site.baseurl}}{{project.url}}"`
    - To show the project categories, we need to loop over the data in the front matter:
        - Replace `CATEGORY, CATEGORY, CATEGORY` with `{% for category in project.categories %}{{category}} {% endfor %}`
    - Save your changes and refresh the projects page - you should now have 6 project boxes each having different project content.

Clicking on any project should take you to its project page, which we'll build out properly in the next step…

Building the Project Page

Copy Your Chosen Template's Project Page HTML

- Go to http://127.0.0.1:4000/github-pages-workshop/projects/project1
- You should see the placeholder project page. Let's replace that with the page from your chosen template…
- Open your chosen template's `project.html` file in your editor.
    - For example if you chose dopetrope, the project page is in `html-templates/dopetrope/project.html`
- Copy all of the HTML from this file.
- Paste it into the project page layout: `_layouts/project.html`, replacing everything in that file.
- Save your changes. Refresh the browser to check that the styles and layout from your template are working.

Configure the Project Page to Use Data from Each Project

Now we'll add the project page content, title and image.

- Make sure you are still editing `_layouts/project.html`, and replace all of the Lorem Ipsum text and associated `<p>` tags with `{{content}}`
- Replace the text `https://via.placeholder.com/1600x1200` with `{{site.baseurl}}{{page.main_image}}`
- To show the project categories, we need to loop over the data in the front matter like we did before:
    - Replace `CATEGORY, CATEGORY, CATEGORY` with `{% for category in page.categories %}{{category}} {% endfor %}`

- Save your changes.
- Go to http://127.0.0.1:4000/github-pages-workshop/projects/ and verify that you can click through to any of the 6 projects and see a completed project detail page.

Building the Blog Page

Copy Your Chosen Template's Blog Page HTML

The template site includes a blog - here, we'll create the page that lists all of the blog entries.

- Open http://127.0.0.1:4000/github-pages-workshop/blog/
    - You should see the placeholder blog page. Let's replace that with the page from your chosen template.
- Open your chosen template's `blog.html` file in your editor.
    - For example if you chose dopetrope, the blog page is in `html-templates/dopetrope/blog.html`
- Copy all of the HTML from this file.
- Paste it into the blog page layout: `_layouts/blog.html`, replacing everything in that file.
- Save your changes to `_layouts/blog.html`.
- Refresh http://127.0.0.1:4000/github-pages-workshop/blog/ and check that this now shows the template blog page.

Configure the Blog Page to Use the Posts Collection

Now we need to replace the six example blog posts with a loop to dynamically render one "box" per blog post. Notice that, unlike the projects page, the blog page has a wider box for the most recent post, then half width boxes for the next two and finally ⅓-width boxes for subsequent blog posts. We will need to add a `for` loop with some extra logic to build out this design.

- Build a loop that adds one "project box" for each blog post in the collection:
    - Remove any 5 of the 6 blog post div elements from `_layouts/blog.html`.
        - These divs begin at the line below the comment: `<!-- blog box starts here -->` and end at the comment `<!-- blog box ends here -->`
    - Save your changes and refresh the blog page to make sure only one blog post appears now.
    - We now need to add a loop around that div, so that Jekyll adds one blog post div per blog post from our posts in the `_posts` folder.
        - Underneath the line `<!-- blog for loop starts here -->` add `{% for post in site.posts %}`
        - Before the comment `<!-- blog for loop ends here -->` , add `{% endfor %}`

    - Save your changes and refresh the blog page - you should now have 6 blog post boxes again. Note that the layout of the 6 boxes no longer matches the design from your template -- we'll fix that shortly…
- Next, replace the placeholder content with real project content. Most of this has been done for you already, but we still need to pull the post dates in from the front matter:
    - Replace `Dec 01 2019` with `{{ post.date | date: '%b %e, %Y' }}`
    - Save your changes and refresh the blog page - you should now have 6 blog post boxes each having different content. Clicking on any blog post should take you to its blog post page, which we'll build out properly in the next step…

- Finally, let's go back and fix the layout on the blog template to match the original design with one big box at the top, then a row of two half width boxes and a row of ⅓ width boxes.
    - Inside your for loop in `_layouts/blog.html`, replace the line starting `<div class="col-4 col-12-small">` (the div that contains your blog post box - underneath the comment `<!-- blog box starts here -->`) with all of the following:

```
{% case forloop.index %}
{% when 1 %}
<div class="col-12 col-12-small">
{% when 2 or 3 %}
<div class="col-6 col-12-small">
{% else %}
<div class="col-4 col-12-small">
{% endcase %}
```

    - Save your changes, then refresh the page in the browser and you should see the desired layout where the latest blog post takes up the whole screen width, the next two are half screen width and any subsequent posts are ⅓ screen width.

Building the Blog Post Page

Copy Your Chosen Template's Blog Post Page HTML

- Go to http://127.0.0.1:4000/github-pages-workshop/blog/example-blog-post
- You should see the placeholder blog post page. Let's replace that with the page from your chosen template.
- Open your chosen template's `blogpost.html` file in your editor.
    - For example if you chose dopetrope, the blog post page is in `html-templates/dopetrope/blogpost.html`
- Copy all of the HTML from this file.
- Paste it into the project page layout: `_layouts/blogpost.html`, replacing everything in that file.
- Save your changes. Refresh the browser to check that the styles and layout from your template are working.

Configure the Blog Post Page to Use Data from Each Blog Post

- Replace the text `Dec 01 2019` with `{{ page.date | date: '%A %b %e, %Y' }}`
- Save your changes.
- Refresh http://127.0.0.1:4000/github-pages-workshop/blog/example-blog-post and you should see your blog posts displayed with a longer date format that shows the day of the week.
- Go to http://127.0.0.1:4000/github-pages-workshop/blog and verify that you can click through to any of the 6 projects and see a completed blog post page.

Building the About Page

This will be very similar to building the home page, and, unlike with the previous pages, you'll need to add all of Jekyll's Liquid templating logic yourself.

Copy Your Chosen Template's About Page HTML

- Go to http://127.0.0.1:4000/github-pages-workshop/about/
- You should see the placeholder about page. Let's replace that with the page from your chosen template…
- Open your chosen template's `about.html` file in your editor.
    - For example if you chose dopetrope, the about page is in `html-templates/dopetrope/about.html`
- Copy all of the HTML from this file.
- Paste it into the about page layout: `_layouts/about.html`, replacing everything in that file.
- Save your changes.
- Refresh http://127.0.0.1:4000/github-pages-workshop/about/ and check that this now shows the template about page.
- As supplied, the about page markup doesn't use our head, header or footer include files. Introduce these as follows:
    - Replace everything between `<!-- start head include -->` and `<!-- end head include -->` with `{% include head.html %}`
    - Repeat this process to do the same for the `header.html` and `footer.html` includes.

Set Up About Page Content and Image

- Now we'll add the about page content, title and image. Make sure you are still editing `_layouts/about.html`, and replace all of the Lorem Ipsum text and associated `<p>` tags with `{{content}}`
- Replace the text `TODO Page Title` with `{{page.title}}`
- Replace the text `https://via.placeholder.com/1600x1200` with `{{site.baseurl}}{{page.main_image}}`

- Save your changes.
- Refresh http://127.0.0.1:4000/github-pages-workshop/about/ and check that the main content text now matches the content that's in `about.md`, and that you have a real image on the home page.

Pushing the New Site to GitHub

- On the command line, add all newly created files to the repo: `git add .`
- Commit all changes: `git commit -m "Workshop progress." .`
- Push changes to GitHub: `git push origin master`

Configuring GitHub Pages to Launch Your Website

We have to configure the repo on GitHub so that GitHub knows to serve this repo as a GitHub Pages website.

- Go to your repo's page on GitHub.
- Click the "Settings" tab.
- Scroll down to "GitHub Pages" settings.
- Change "Source" from "None" to "master branch".
- Check the "Enforce HTTPs" check box to force all page requests to be served with SSL.
- The page will reload and you will need to scroll down to "GitHub Pages" settings again.
- You should see a URL that your site will be published at.
- If it says "Your site is ready to be published at…", wait a moment and refresh the page.
- Keep refreshing the page until it says "Your site is published at…"
- Click the URL and you should see your site live!

Adding a Custom Domain

GitHub allows you to serve a GitHub Pages site on a domain that you own. This is achieved by setting up a CNAME alias with your domain registrar that points to GitHub. The details of this process vary depending on who you use for your domain registrar.

GitHub provides documentation for this process here: https://help.github.com/en/github/working-with-github-pages/configuring-a-custom-domain-for-your-github-pages-site

Bonus:

Configuring Social Media Links ("dopetrope" template only)

- The "dopetrope" template has social media links in the footer, the others do not.
- If you chose a template with social media links, on each page, find the text `<ul class="social">`
- Delete the `<li>...</li>` for any social networks that you don't use or don't want on the site.
- Change each remaining `href="#"` to contain the link to your social media page on the relevant network.
- Save your changes, and reload the page you just edited to test them.