Hippo Showcase Site

Aesculeae domus vincemur et Veneris adsuetus lapsum

Hippo Digital

29/09/2023

Contents

Styling

Setting up a dev environment

To be able to visualise the changes you make to the theme and content there are a few steps that you need to take before you start:

- Create an `.env.local`file in the root of your local repository to store the `env variables` that you will need to connect to other aspects of this service.
- Install the packages required to start your local development environment

Create an env file

- Create a new file in the root of your local project and name it `.env.local`.
- Copy the content below and paste it into the file.

```
NEXTAUTH_SECRET=frltbb9LZIesEjb3nTsjSO/WBbguJsvzfTa5D8H984Q=
SECRET_PASSWORD=secretpassword
NEXTAUTH_URL=http://localhost:3000
```

> **N.B.** Optional - generate a new secret for the NEXTAUTH_SECRET using `openssl`
>
> ```
> openssl rand -base64 32
> ```
>
> **Install the packages and start the service**
>
> From your command line prompt:

- Make sure that you are in the root of your project if not run `cd ~/my_local_project_directory`
- Run `yarn` or `npm install`
- Once the package installation has completed run `yarn dev` or `npm run dev`
- A local instance of the service should now be available in your browser window at http://localhost:3000

Theme directory and structure

All theme configuration is stored in one directory and spans three files and one directory. Examples of each file and its formatting can be found in the `public/theme` directory and can be used as a starting point for your site.

```
public/theme
├── font.config.js
├── fonts
│   ├── Roboto-Bold.ttf
│   ├── Roboto-BoldItalic.ttf
│   ├── Roboto-Italic.ttf
│   └── Roboto-Regular.ttf
├── images
│   └── logo.svg
├── theme.config.js
└── theme.scss
```

Logo

- File type - .svg (recommended), .jpg, .png, .webp
- Size - max-height 70px, max-width 300px
- File location - `public/theme/images`
- Settings location`public/theme/theme.config.js`.
- Width and height - numeric value only e.g. `140` not ~~`140px`~~
- Image src - relative file path e.g. `/theme/images/logo.svg`
- Alt text - essential for accessibility

Example configuration

```
logo: {
        src: "/theme/images/logo.svg",
        height: 70,
        width: 140,
        alt: "Hippo Data logo."
    }
```

If you don't wish to use a logo you can set the value of the logos `src` attribute to `null` as in the example below. This will make the theme use the name value for the heading on the site.

```
const theme = {
    pageTitlePostfix: 'Hippo Data',
    name: 'Showcase site',
    serviceName: '',
    logo: {
    src: null,
    height: 100,
    width: 282,
```

```
    alt: 'Hippo Data logo.',
    }
...
}
```

Fonts

By default, the theme only makes use of two font weights (400, 700) and two font styles (normal, italic).

The theme allows for two font families, one for the body text and one for headings. The example configurations below are an examples of optimised font set up.

Changing the default fonts requires only 1-3 easy steps: 1. Configure the fonts for use in `public/theme/font.config.js`. 2. **Optional** Reference the fonts for use in `public/theme/theme.scss`. 3. **Optional** - Upload the fonts and all the variations you want to use to the `public/theme/fonts` directory.

Google Fonts

Using google fonts is the easiest way of adding or changing the fonts for your showcase project site. #### Overview of font.config.js

```
import { Epilogue, Anybody } from 'next/font/google';

export const bodyFont = Anybody({
weight: ['400', '700'],
subsets: ['latin'],
display: 'swap',
variable: '--font-body',
});

export const headingFont = Epilogue({
weight: ['400', '700'],
subsets: ['latin'],
display: 'swap',
variable: '--font-header',
});
```

The above configuration utilises two fonts from Google - Epilogue and Anybody. 1. Both fonts have two references each in the config file: one in the import statement and the other in export const ... = . 2.

Both fonts have a variable with a different value `variable: '--font-body'` and `variable: '--font-header'`

Changing the fonts

1. Choose the fonts you want to use on Google fonts. We recommend using variable fonts for the best performance and flexibility.
2. Open `public/theme/font.config.js` in there you will see the current default set of Google shown above.
3. Change the import statement and replace the name of the font with the one you want to use. In this example we are going to change the body font to `Gabarito`. Make sure to delete the name of the font that you no longer want to use.
4. Change the `export const .... =` to use the newly named import that you have chosen.

Your refactored `font.config.js` should now look something like this:

```
import { Epilogue, Gabarito } from 'next/font/google';

export const bodyFont = Gabarito({
weight: ['400', '700'],
subsets: ['latin'],
display: 'swap',
variable: '--font-body',
});

export const headingFont = Epilogue({
weight: ['400', '700'],
subsets: ['latin'],
display: 'swap',
variable: '--font-header',
});
```

N.B. Not all fonts have both 400 and 700 weights please check before making your changes as this will cause an error and prevent the site from building

Local fonts

If you want to use a local font you can simply add the font files to the repository. Here are the steps you need to complete (there is a complete working example using Roboto that is included in `/public/theme/font.config.js`):

1. Upload your font files to the `/public/theme/fonts` directory. N.B. you can safely delete the Roboto fonts in this directory if you are not going to use them. They are there for the purposes of this example only.
2. Add an import statement for `localFont` in `public/theme/font.config.js`. If you don't intend to

```
import { localFont } from 'next/font/local';
```

3. Follow the example below to configure your local font. N.B. The route to the fonts directory is important and must start `fonts/` followed by the name and extension of your file.
4.

```
import localFont from 'next/font/local';

export const roboto = localFont({
  variable: '--font-body',
  src: [
    {
      path: 'fonts/Roboto-Regular.ttf',
      weight: '400',
      style: 'normal',
    },
    {
      path: 'fonts/Roboto-Italic.ttf',
      weight: '400',
      style: 'italic',
    },
    {
      path: 'fonts/Roboto-Bold.ttf',
      weight: '700',
      style: 'normal',
    },
    {
      path: 'fonts/Roboto-Bolditalic.ttf',
      weight: '700',
      style: 'italic',
    },
  ],
});
```

Using system fonts

If you intend or prefer to use system fonts only and don't want to include custom fonts: 1. Delete the contents of `font.config.js` but leave the file in place. This is important for the site to function. 2. Enter the values of the system fonts against the relevant SASS variable in `/public/theme/theme.scss` (see below)

```
/*FONTS*/
$govuk-font-family: HelveticaNeue,Helvetica,Arial,sans-serif;
$govuk-font-family-headings: HelveticaNeue,Helvetica,Arial,sans-serif;
```

Using the same font for heading and paragraph text

To configure the site to use just one font for paragraph and header text edit `public/theme/theme.scss` and reference the font variable name that you want to use across the site. In the example below both heading and paragraph text will use the font with the variable name `--font-body`, which is set in `font.config.js`

```
$govuk-font-family: var(--font-body);
$govuk-font-family-headings: var(--font-body);
```

Colours

Accessibility

The theme utilises the GOV.UK Design System to ensure that all aspect of the users journey meet accessibility standards. Before you apply colours to your theme it is recommended to check the colour contrast using WebAIM's contrast checker.

Editing colours

There are 33 colour variables that can changed to match your brand, each are grouped to make editing easier. Each variable will accept any CSS colour value.

N.B. Do not rename or delete any of the variable names as this will result in errors that will prevent the site from building

```
$govuk-page-width: 1200px;
/*FONTS*/
$govuk-font-family: var(--font-body);
$govuk-font-family-headings: var(--font-body);

/*COLOURS*/
$govuk-brand-colour: #1d70b8;
$govuk-canvas-background-colour: #fcfcfc;
$govuk-body-background-colour: #FFFFFF;
$govuk-text-colour: #0b0c0c;
$govuk-secondary-text-colour: #505a5f;
$govuk-border-colour: #b1b4b6;

/*LINKS*/
$govuk-link-colour: #1d70b8;
$govuk-hover-colour: #003078;
$govuk-link-visited-colour: #4c2c92;
$govuk-link-hover-colour: #003078;
$govuk-link-active-colour: #0b0c0c;

/*BUTTONS*/
//$govuk-button-background-colour: ;
//$govuk-button-text-colour: white;

/*HEADER*/
$govuk-header-background:  white;
$govuk-header-text-colour: $govuk-text-colour;
$govuk-header-link-active: #0b0c0c;
//$govuk-header-nav-item-border-color: ;
$govuk-header-border-color: #b1b4b6;
$govuk-header-border-width: 1px;

/*FOCUS STATE*/
$govuk-focus-text-colour: #0b0c0c;
```

```
$govuk-focus-colour: #fd0;

/*FORMS*/
$govuk-error-colour: #d4351c;
$govuk-success-colour: #00703c;
$govuk-input-border-colour: $govuk-text-colour;

/*MISC*/
$govuk-details-summary-title-colour: #1d70b8;

/*BANNERS and CALLOUTS*/
$govuk-inset-text-border-colour: #5694ca;
$govuk-inset-text-background-colour: #e9f2fd;
$govuk-warning-callout-colour: #FF611A;
$govuk-notification-banner-border: $govuk-brand-colour;

/*DASHBOARD*/
$govuk-summary-card-title-background: $govuk-canvas-background-colour;
//$govuk-tag-green: #00703c;
//$govuk-tag-yellow: #ffdd00;
//$govuk-tag-grey: #505a5f;
//$govuk-tag-blue: #1d70b8;

.govuk-header {
  position: relative;
  margin-bottom: 40px;
  border-bottom: 1px solid #e0e0e0;
}

@media (min-width: 40.0625em) {
  p, .govuk-body, .govuk-body-m {
    line-height: 1.5 !important;
  }
}
```

Menus

There are three menus that can be configured: 1. **Standard menu** - this appears in the header of the site and is only visible when users have not logged in to the service. 2. **Protected menu** - this menu replaces the standard menu once a user has authenticated to the service. It should contain a link to the

users dashboard. 3. **Footer menu** - always available to the user and should contain links to policies relating to your brand. ### Configuring menus All three menus are configured in exactly the same way: 1. **Content** - the text that appears in the link 2. **Link** - URL of the link destination 3. **Title** - longer text description of the links purpose for screen readers and assistive technologies. Not normally visible to the use.

A complete working example of this configuration can be found in `/public/theme/theme.config.js`

```
footerMenu: [
    {
        content: 'Help and support',
        link: 'https://example.com/help-and-support',
        title: 'Get help and support with this service'
    },
    {
        content: 'Terms of use',
        link: 'https://example.com/terms-of-use',
        title: 'Terms of use',
    }
  ]
```

Metadata

A brand or service name can be added to the end of all page titles on the site. The configuration for the `pageTitlePostfix` can be found in `/public/theme/theme.config.js` (see the example below).

```
const theme = {
   pageTitlePostfix: "Compassion in Dying",
   logo: {
      src: "/theme/images/logo.svg",
      height: 70,
      width: 140,
      alt: "Compassion in dying logo."
   },
...
```

Creating content

Using components

Working with data

Hosting

Hosting the service in an AWS account

Each component of the service is deployed via terraform. Each service has instructions in each README on how to deploy the service to AWS either using GitHub Actions or locally using terraform:

- Core infrastructure
- Frontend
- Authentication Layer
- API / Database Layer

How each component is related and connected can be discovered can be seen in the Logical Architecture Diagram

Each repository has a set of GitHub workflows, which take care of orchestrating the deployment. These workflows depend on having some GitHub environments setup. Each repository needs to have a `dev` and `production` environment created with the following secrets setup

```
AWS_ACCESS_KEY_ID
AWS_SECRET_ACCESS_KEY
```

For more information on getting these keys see the Managing access keys for IAM users AWS documentation

Using Docker

Each repository in this services has a README file that defines how to deploy the component either locally or as part of a build using terraform. However it is also possible to directly build the docker image from the `Dockerfile` in each of the projects (that is for the projects where there is a Dockerfile).

For a full walkthough of docker and containers see the docker getting started documentation

Building and publishing the docker image

To build a docker image, the following commands should be run in the same folder that the `Dockerfile` you wish to build. Once in the folder you can run

```
docker build -t {tag_name} .
```

where `{tag_name}` is the tag that you want attach to the built image. Once the image has been built you can now publish the image and the associated tags to any container repository.

- Docker hub
- AWS ECR
- Azure Container Registry

The published images can then be consumed in any service that supports running docker containers, for example,

- AWS ECS
- AWS EKS
- Azure AKS
- Azure Container Instances

Running the docker image

To run a docker container we need to do 2 things:

- Create a file with the appropriate environment variables to pass the connection strings and envrionment details to the container
- Expose the appropriate ports for the container on the host and run the container

Creating an environment variable file To create an environment variable file, discover what environment variables are required for the components, these are documented in the README file for the associated component. Once you know the environment variables and values you can create the file by running

```
touch env.list
```

and then

```
open env.list
```

and cut and paste the environment variables and values into that file an save.

Running the container and exposing the ports To run the docker continer we can simply run the following command (assumes `env.list` is in the same directory as the command that is running)

```
docker run -d -p {host-port}:{container-port} -env-file env.list {tag-name}
```

so to run a container from the `choices_api:latest` tag and expost the container port `7126` to `8000` we can run:

```
docker run -d -p 8000:7126 -env-file env.list choices_api:latest
```