PaletteFlow

PaletteFlow is a tool developed by students at OSU-Cascades in collaboration with fare*well, designed to streamline the client onboarding process for customers building a website. Its primary objective is to provide an intuitive platform that assists clients in establishing a preliminary brand identity, enabling a faster start to site development. PaletteFlow guides users through the selection of an initial color and brand archetype, leveraging this information to generate an accessible color palette and site theme. Users can further customize their theme, and PaletteFlow produces a branding document that can be exported as a PDF.

Setup Instructions

Prerequisites

- Node.js (v18+)
- npm or yarn

Steps

1. **Clone the repo**
    - `git clone https://github.com/natitack/codespaces-nextjs.git`
    - `cd codespaces-nextjs.git`
2. **Install dependencies**
    - `npm install`
        - or
    - `yarn install`
3. **Run locally**
    - `npm run dev`
        - or
    - `yarn dev`

Visit http://localhost:3000

VS Code

```
- Make sure the Dev Containers extension is installed.
- Open the project folder in VS Code.
- Click "Reopen in Container" when prompted.
```

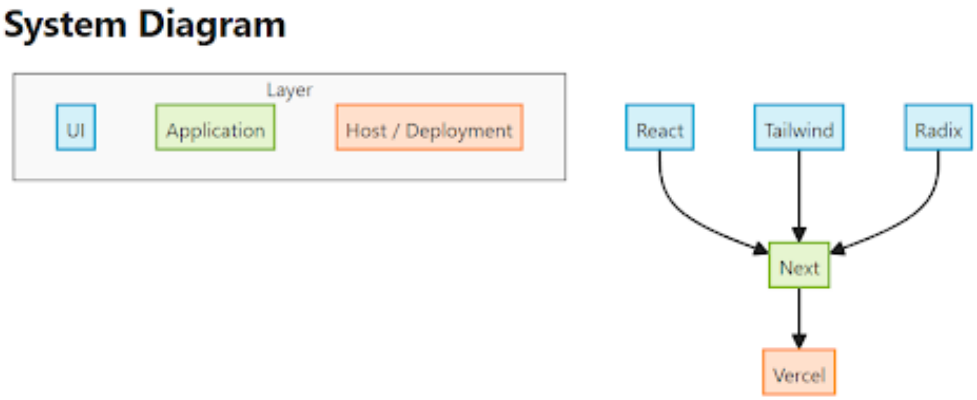
Codespaces

```
- Start your code space and the deployment should start automatically
```

How to Deploy

This project is built on Next.JS 14, and can be easily deployed to Vercel. To deploy PaletteFlow, refer to the Vercel deployment guide (https://vercel.com/docs/getting-started-with-vercel/import). This resource provides step-by-step instructions for importing your project, configuring settings, and launching your application on Vercel.

System Architecture



How to Modify

Adding New Relume Components

PaletteFlow supports modular addition of new Relume hero and feature components. Code for these components can be copied directly from the Relume website. To add a new Relume component (such as a hero or feature layout), follow these steps:

1. **Create the Component**

- Copy the component code from the Relume website and save it as a `.tsx` file.
- Place your new component file in the appropriate directory:
  - Hero components: `/components/heroelements/`
  - Feature components: `/components/featureelelemts/`

2. **Register the Component**

   - For hero components, edit `/components/heroelements/RelumeHeroWrapper.tsx` and add your new component to the `heroLayouts` map.
   - For feature components, edit `/components/featureelelemts/RelumeFeatureWrapper.tsx` and add your new component to the `featureLayouts` map.

3. **Add Preview Images**

   - Remember to add a PNG preview image of your layout to the associated public directory:
     - Hero: `/public/images/previews/hero/`
     - Feature: `/public/images/previews/feature/`
   - Name the image file to match your layout key (e.g., `header99.png`).
   - Preview images are required for all new layouts to ensure they appear correctly in the deliverable export.

Summary:

- Create your component in the correct folder (copy from Relume if needed).
- Register it in the wrapper (`RelumeHeroWrapper` or `RelumeFeatureWrapper`).
- Add a preview image for deliverables (required).

Adding New Fonts

PaletteFlow allows you to easily add new Google Fonts for users to select. To add a new font option:

1. **Import the Font**

   - Open `/hooks/useFontOptions.ts`.
   - Import your desired font from `next/font/google`. For example:

```
import { Lato } from "next/font/google";
```

2. **Initialize the Font**

   - Create a font instance with the required options:

```
const lato = Lato({ subsets: ["latin"], weight: "400" });
```

3. **Add to Font Options**

- Add your new font to the `fontOptions` array:

```
{ value: lato.className, label: "Lato" },
```

4. **Use in the Font Picker**

   - The new font will automatically appear in the font selection dropdown in the UI (e.g., in the FontStep component).

Summary:

- Import the font in `/hooks/useFontOptions.ts`.
- Initialize it and add it to the `fontOptions` array.
- The font will be available for users to select in the app.

Updating the Mood Step

The Mood Step in PaletteFlow allows users to select a brand personality, which automatically sets related style options such as font, button style, and color mood. To update or add new moods, follow these steps:

1. **Open the MoodStep Component**

   - Edit the file: `/components/steps/MoodStep.tsx`.

2. **Modify the MOOD_CONFIG Object**

   - Locate the `MOOD_CONFIG` constant at the top of the file.

   - To **add a new mood**, add a new entry to the object with a unique key. Each mood should include:

     - `label`: Display name
     - `emoji`: Emoji icon
     - `description`: Short description
     - `styles`: An object with the following properties:
       - `font`: The label of a font from your font options (e.g., `"Nunito"`)
       - `buttonStyle`: Button style string (e.g., `"medium"`)
       - `heroLayout`: Hero layout key (e.g., `"header1"`)
       - `featureLayout`: Feature layout key (e.g., `"layout398"`)
       - `chroma`: Chroma value (number, e.g., `0.30`)
       - `lightness`: Lightness value (number, e.g., `0.90`)

   - Example:

```
newmood: {
  label: "New Mood",
  emoji: "□",
  description: "Energetic, fresh, unique",
  styles: {
    font: "Nunito",
    buttonStyle: "medium",
    heroLayout: "header1",
    featureLayout: "layout398",
    chroma: 0.50,
    lightness: 0.75
  }
}
```

- To **edit an existing mood**, update its properties as needed.

3. **Ensure Font Mapping**
    - The `font` property in each mood's `styles` should match the `label` of a font in your `/hooks/useFontOptions.ts` file.
4. **Save and Test**
    - Save your changes. The new or updated mood will appear in the Mood Step UI, and selecting it will update the brand's style settings accordingly.

Summary:

- Edit `MOOD_CONFIG` in `/components/steps/MoodStep.tsx` to add or update moods.
- Use font labels that exist in your font options.
- Save and test the Mood Step in the app.

Adding and Referencing New Fields in Choices Context

PaletteFlow uses a global Choices Context (`/context/ChoicesContext.tsx`) to store user selections such as color, font, mood, and more. If you want to add a new field (for example, `backgroundPattern` or `logoShape`) to the context and use it throughout the app, follow these steps:

1. Add the Field to the ChoicesState Interface

Open `/context/ChoicesContext.tsx` and update the `ChoicesState` interface to include your new field:

```
interface ChoicesState {
  // ...existing fields...
  backgroundPattern: string; // <-- Add your new field here
}
```

2. Add the Field to the Initial State

In the `getInitialChoices` function, add a default value for your new field:

```
return {
  // ...existing fields...
  backgroundPattern: "dots", // <-- Default value for new field
};
```

3. Reference or Update the Field in Components

To **read** the value in a component:

```
import { useChoices } from "@/context/ChoicesContext";

const { choices } = useChoices();
console.log(choices.backgroundPattern); // Access the value
```

To **update** the value:

```
const { updateChoice } = useChoices();
updateChoice("backgroundPattern", "stripes");
```

4. Add to setAllChoices

If you use `setAllChoices`, ensure your new field is included in any objects passed to it.

Summary:

- Add your field to `ChoicesState` and the initial state.
- Migrate old data if needed.
- Use `choices.<field>` to read, and `updateChoice("<field>", value)` to update.
- Update any UI or logic to use the new field.

This ensures your new field is globally available and persists across sessions.

Adding a New Step to the Brand Builder

To add a new step (such as a new design choice or configuration) to the Brand Builder flow in PaletteFlow, follow these steps:

1. Create Your Step Component

- Create a new file for your step in `/components/steps/`, e.g. `NewStep.tsx`.
- Export a React component that accepts `value` and `onChange` props.
- Example:

```
// filepath: /components/steps/NewStep.tsx
import { Flex, Text } from "@radix-ui/themes";

export function NewStep({ value, onChange }) {
  return (
    <Flex direction="column" gap="4">
      <Text size="5" weight="bold">Choose your new option</Text>
      {/* Your UI here */}
    </Flex>
  );
}
```

2. Add the Field to Choices Context

- Add a new field to the `ChoicesState` interface in `/context/ChoicesContext.tsx`.
- Add a default value for your field in the `getInitialChoices` function.
- Example:

```
// filepath: /context/ChoicesContext.tsx
interface ChoicesState {
  // ...existing fields...
  newField: string; // <-- Add your new field
}

// In getInitialChoices:
return {
  // ...existing fields...
  newField: "", // <-- Default value
};
```

3. Register the Step in the Brand Builder

- Open `/pages/brand-builder.tsx`.
- Add your new step to the `STEPS` array, specifying a unique `id`, your component, and a title.
- Example:

```
// filepath: /pages/brand-builder.tsx
import { NewStep } from "../components/steps/NewStep";

const STEPS = [
  // ...existing steps...
  { id: "newField", component: NewStep, title: "New Step Title" },
];
```

4. (Optional) Update the Live Preview

- If your new step affects the live preview, update the `LivePreview` component to use the new field from `choices`.

5. (Optional) Update Deliverables

- If your new field should appear in exported documents or deliverables, update the relevant export logic to include it.

Known Issues, Future Improvements

- Moodstep options need more meaningful differentiation
- Additional Relume feature and hero components should be added
- Color selection does not handle colors with low color saturation and lightness values well because the mood step overrides those values. One way to resolve this issue is to have a low saturation option in the mood step
- The client is very interested in having an inspiration step component, incorporating a ranking process of existing websites to inform archtype selection
- Font Selection component does not show what a font would look like until after it is selected.
- Deliverable component needs aesthetic refinement
- Application is not mobile friendly