How to create the React Component

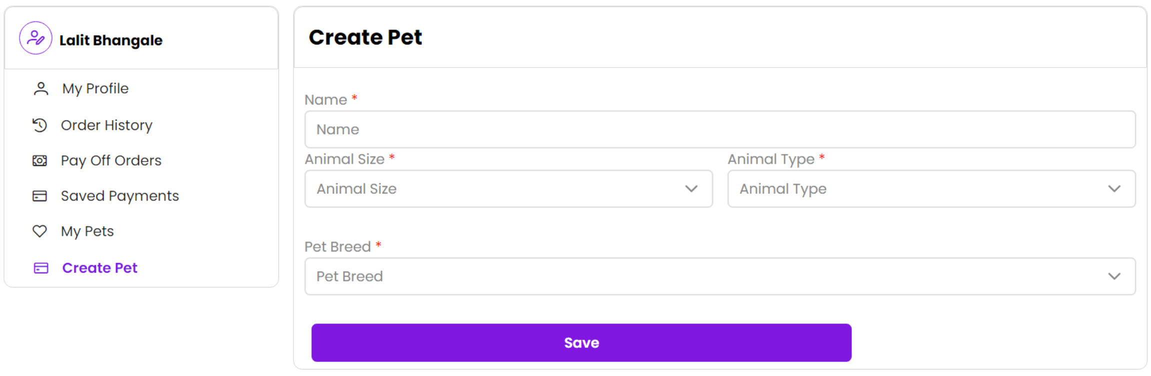
About Community Brands

Community Brands delivers purpose-built solutions to nearly 100,000 leading nonprofits, associations, K-12 private schools and faith-based organizations worldwide to help them thrive and succeed in today's fast-paced, evolving world. Our focus on accelerating innovation, fulfilling unmet needs and bringing to market modern technology solutions and engagement platforms helps power social impact, effect positive change and create opportunity. With Community Brands solutions and services, purpose-driven organizations better engage their members, donors, educators and volunteers; raise more money; effectively manage revenue; and provide professional development and insights to power their missions. To learn more, visit www.communitybrands.com.

Table of Contents

1. Creating new page in react.. 4
2. Define/create the Component.. 5
3. Import React into the Component.. 6
4. Import and Use the Component 7
5. How to create/generate forms 8
   5.1. Declare state & constant variables those are required to manage state. 8
   5.2. Initializes a form using the useForm hook... 9
   5.3. Define field by using useMemo hook that need to be render on page. 10
   5.4. Call API to render data by using useEffect hook. 11
   5.6 Add function which will trigger on submit form and saving data. 12
   5.7. Design form by using HTML & JSX and return it.. 13
6. Add newly created component to index.jsx... 14
7. Add pages to navigate/router.. 14
8. Add menu under the profile pages.. 15
9. Run Your Application 19
10. Details Sections:.. 19

We are planning to build page to create pet record, UI will look like,



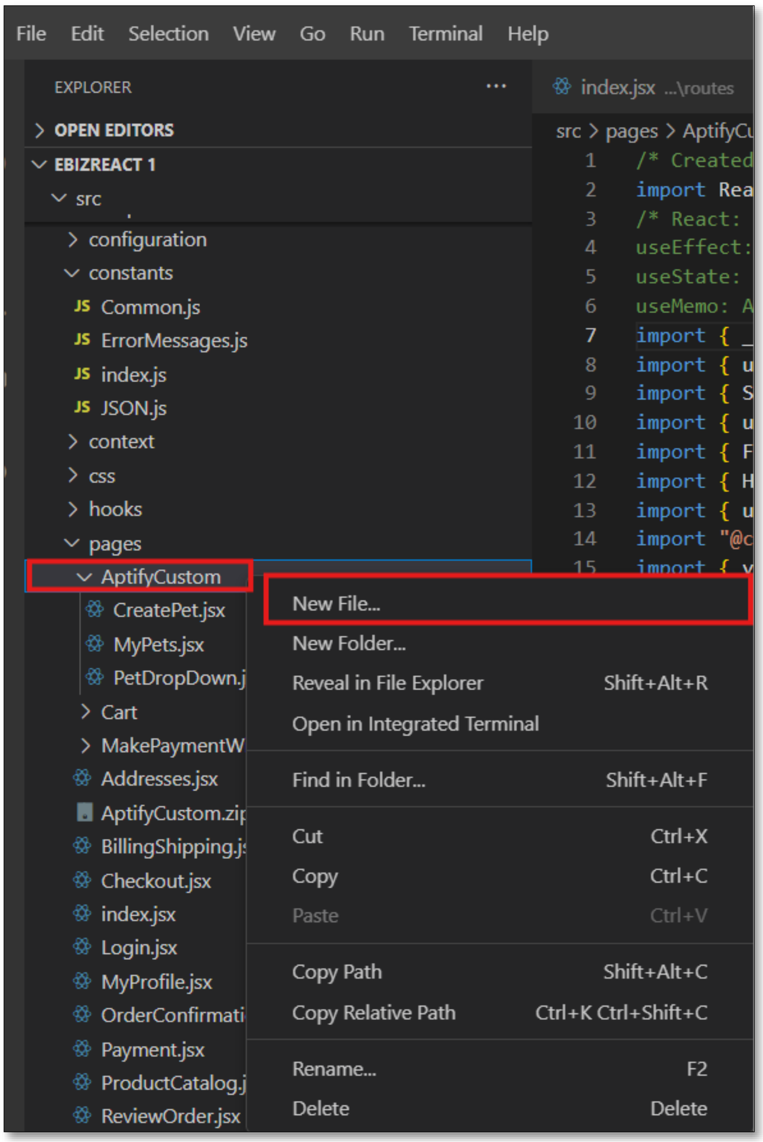
Note: Prerequisites files for this page are as follows, you will find code for this file at end of document.

a. Css file \src\css\Pet.scss

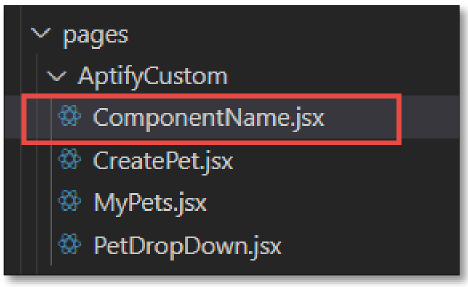
b. Validation file \src\validations\AptifyCustom\createMyPetValidation.js

1. Creating new page in react

- Create `jsx` file in `pages > AptifyCustom` folder. If `AptifyCustom` folder does not exist, create folder and right click on `AptifyCustom` Folder. Click New File and name the file with extension `jsx`.



```
/* Created
import Rea
/* React:
useEffect:
useState:
useMemo: A
import { _
import { u
import { S
import { u
import { F
import { H
import { u
import "@c
import { v
```



2. Define/create the Component

- Create a functional component named **`CreatePets`**, Match the file name with the component name, using PascalCase.

Example:

```
const CreatePet = () => {
```

```
return (
  <>
        //Design your UI
  </>
)}

export default CreatePet;
```

3. Import React into the Component.

- React: The core library for building user interfaces.

```
import React from "react";
```

Example Usage –

```
/* React: The core library for building user interfaces.
useEffect: A hook for performing side effects in function components (e.g.,
data fetching, subscriptions).
useState: A hook for adding state to function components.
useMemo: A hook for memoizing expensive calculations to optimize performance.
*/
import React, { useEffect, useState, useMemo } from "react";

/* _get and _post: Functions for making GET and POST requests to your API. */
import { _get, _post } from "@api/APIClient";

/* useStateUser: A custom hook, likely for managing user state. */
import { useStateUser } from "@hooks/useStateUser";

/* SimpleButton: A reusable button component. */
import { SimpleButton } from "@/components/atoms";
/* useForm: A hook for managing form state and validation. */
import { useForm } from "react-hook-form";
";/*  FormBuilder: A component for dynamically building forms. */
import { FormBuilder } from "@/components/molecules”

/* HTTP_STATUS_CODES: An object containing HTTP status codes for reference. */
import { HTTP_STATUS_CODES } from "@/constants";

/* useToast: A hook for displaying toast notifications. */
import { useToast } from "@/context/ToasterProvider";
/* A stylesheet for styling your components. */
import "@css/PetCatalog.scss";
```

```
/* yupResolver: A resolver for integrating Yup validation with react-hook-
form. */
import { yupResolver } from "@hookform/resolvers/yup";

/* createMyPetValidationSchema: A Yup schema for validating form data. */
import { createMyPetValidationSchema } from
"@/validations/AptifyCustom/createMyPetValidation";

            const CreatePet = () => {

            return (
             <>
            //Design your UI
             </>
            )}

            export default CreatePet;
```

4. Import and Use the Component

- Import the Component. First, make sure you import the `SimpleButton` component into your jsx file/component.

```
import SimpleButton from './path/to/SimpleButton;
```

Example.

```
            import React from 'react';
            import { SimpleButton } from "@/components/atoms";


            const CreatePet = () => {

                    return (
                   <>
                   {/*  Design your UI */}
                   <SimpleButton label={"Save"} />
                   </>
            )}

            export default CreatePet;
```

5. How to create/generate forms

5.1. Declare state & constant variables those are required to manage state.

```
 /*  This  line  initializes  a  state  variable  user  using  a  custom  hook
useStateUser(). This hook likely manages the state related to the user.  */
     const user = useStateUser();

/*  These lines destructure functions from the useToast hook. showToastSuccess,
showToastError,  and  showToastInfo  are  likely  functions  to  display  different
types of toast notifications (success, error, info). */
    const { showToastSuccess } = useToast();
    const { showToastError } = useToast();
    const { showToastInfo } = useToast();

/*  This  line  initializes  a  state  variable  loading  with  a  default  value  of
false. setLoading is the function to update this state. */
    const [loading, setLoading] = useState(false);

/*  These lines initialize state variables sizeData, breedData, and typeData as
empty  arrays.  setAnimalSize,  setPetBreed,  and  setAnimalType  are  the  functions
to update these states. */
    const [sizeData, setAnimalSize] = useState([]);
    const [breedData, setPetBreed] = useState([]);
    const [typeData, setAnimalType] = useState([]);

/*  These  lines  transform  the  sizeData,  typeData,  and  breedData  arrays  into
options for a dropdown or select input. Each item is mapped to an object with
name and value properties. */
     const sizeOptions = sizeData.map(item => ({ name: item.Name, value: item.ID }));
    const typeOptions = typeData.map(item => ({ name: item.Name, value: item.ID }));
    const breedOptions = breedData.map(item => ({ name: item.Name, value: item.ID }));
```

5.2. Initializes a form using the useForm hook.

```
/*  This block initializes a form using the useForm hook. defaultValues sets the initial values for the form fields.
mode: "onBlur" specifies that validation should occur when an input loses focus.
resolver: yupResolver(createMyPetValidationSchema) integrates Yup validation schema for form validation. */

    const form = useForm({
        defaultValues: {
            Name: '',
            animalSize: null,
            animalType: null,
            petBreed: null
        },
        mode: "onBlur",
        resolver: yupResolver(createMyPetValidationSchema)
    });

/*  This line destructures reset, isSubmitting, and isValid from the form object.
     reset is a function to reset the form.
     isSubmitting indicates if the form is currently being submitted.
     isValid indicates if the form is valid according to the validation schema.
*/
    const { reset,  formState: { isSubmitting, isValid } } = form;
```

5.3. Define field by using useMemo hook that need to be render on page.

```
/* This line uses the useMemo hook to memoize the fields array.
   This means fields will only be recalculated when sizeOptions, typeOptions,
or breedOptions change, improving performance.  */
    const fields = useMemo(() => [
            {
                /* This object defines a text input field for the form. */
                type: "text",
                name: "Name",
                label: "Name",
                isRequired: true
            },
            {
                /* This object defines a group of fields.
                type: "group" specifies that this is a group of inputs.
                group: [...] contains the fields within the group. */
                type: "group",
                group:
                [
                    {
                     /* This object defines a dropdown input for selecting animal size. */
                        type: "dropdown",
                        name: "animalSize",
                        label: "Animal Size",
                        options: sizeOptions,  /* provides the options for the dropdown. */
                        isRequired: true /* makes this field mandatory. */
                    }, {
                     /* This object defines a dropdown input for selecting animal type. */
                        type: "dropdown",
                        name: "animalType",
                        label: "Animal Type",
                        options: typeOptions, /* provides the options for the dropdown. */
                        isRequired: true /* makes this field mandatory. */
                    }
                ]
            },
            {
                /* This object defines a dropdown input for selecting pet breed. */
                type: "dropdown",
                name: "petBreed",
                label: "Pet Breed",
                options: breedOptions, /* provides the options for the dropdown. */
                isRequired: true  /* makes this field mandatory. */
            }
        ],[sizeOptions, typeOptions, breedOptions]);
```

5.4. Call API to render data by using useEffect hook.

- This is a React Hook that runs the provided function after the component mounts. The empty dependency array [] means it runs only once, similar to componentDidMount.
- Use the `_get` method for API call for the GET Request. To import the `_get`, `_post` etc. method writes below code

```
import { _get,_post,_patch,_delete } from "@api/APIClient";
```

```
/* This is a React Hook that runs the provided function after the component mounts.
   The empty dependency array [] means it runs only once, like componentDidMount. */
    useEffect(() => {
        const getPetSize = async () =>
            const response = await _get("/v1/PetSize", {
                withCredentials: true
            });
            setAnimalSize(response.data);
        };
        const getPetType = async () => {
            const response = await _get("/v1/PetType", {
                withCredentials: true
            });
            setAnimalType(response.data);
        };
        const getPetBreed = async () => {
            const response = await _get("/v1/PetBreed", {
                withCredentials: true
            });
            setPetBreed(response.data);

        };
        getPetSize();
        getPetType();
        getPetBreed();

    }, []);
```

5.6 Add function which will trigger on submit form and saving data.

- Write a function name **`savePetDetails`** use camelCase for the naming convention.
- Below line defines an asynchronous function named **`savePetDetails`** that takes data as an argument which is a form data and this trigger on submit button click on the FormBuilder.

```
/* This line defines an asynchronous function named savePetDetails that takes
data as an argument which is a form data and this trigger on submit button
click on the FormBuilder. */
    const savePetDetails = async (data) => {
        try {
            setLoading(true);
            if (data?.Name != null && data?.animalSize != null && data?.petBreed != null &&
data?.animalType != null) {
                const payLoad = {
                    "Name": data?.Name,
                    "SizeID": data?.animalSize,
                    "BreedID": data?.petBreed,
                    "TypeID": data?.animalType
                }
                const createPetRecords = await _post("/v1/Pets/" +
user?.AuthenticatedPersonId, payLoad, {
                    withCredentials: true,
                    headers: {
                        "Content-Type": "application/json"
                    }
                });

                if (createPetRecords.status === HTTP_STATUS_CODES.OK) {
                    setLoading(false);
                    showToastSuccess({
                        summary: "Success",
                        detail: createPetRecords.data?.Message + " #" +
createPetRecords.data?.PetID
                    });
                    reset({
                        Name: '',
                        animalSize: '',
                        petBreed: '',
                        animalType: ''
                    })
                }
                return true;
            } else {
                showToastInfo({
                    summary: "info",
                    detail: "Please fill the mandatory details."

                });
                setLoading(false);
            }
        } catch (error) {
            setLoading(false);
            showToastError({
                summary: "error",
                detail: error

            });
            console.log(error);
        }
    };
```

5.7. Design form by using HTML & JSX and return it

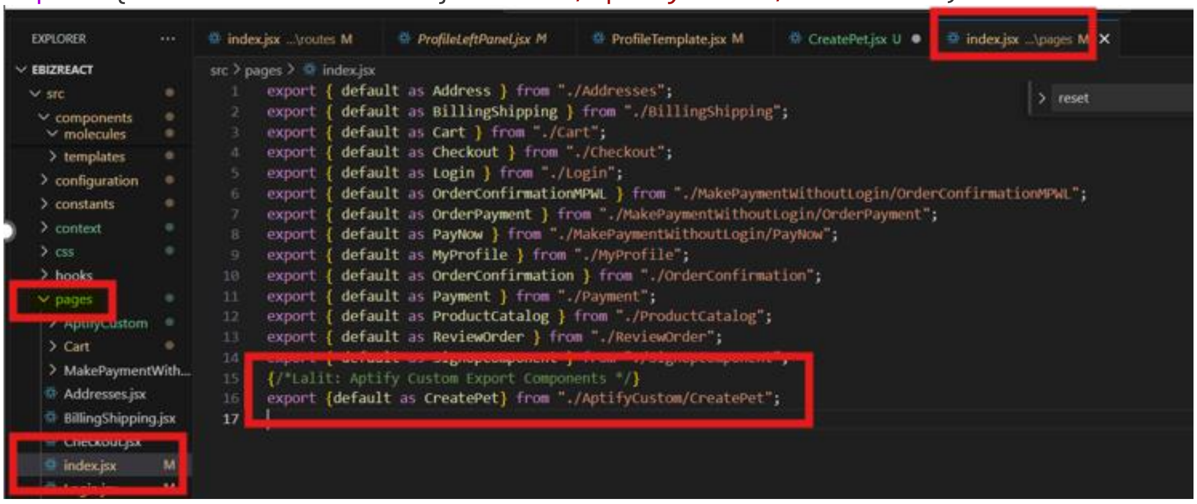
```
return (
        <>
            <main>
                <div>
                    <div className="eb-border-gray">
                        <div className="eb-profile-text-color font-bold text-2xl px-3 pt-3 
pb-3 eb-border-gray border-noround-bottom">
                            Create Pet
                        </div>
                        <br />
                        <section>
                            <div className="eb-container">
                    {/* This component renders a form using the FormBuilder component. */}
                                <FormBuilder
                                fields={fields
                                form={form}
                                onSubmit={savePetDetails}>
                                    <div className="col-8 mt-4">
                                    {/* This component renders a button. */}
                                        <SimpleButton
                                            type="submit"
                                            navigatelink={"false"}
                                            label={"Save"}
                                            className={"simpleButtonStyle"}
                                            loading={loading}
                                    />
                                    </div>
                                </FormBuilder>
                            </div>
                        </section>
                    </div>
                </div>
            </main>
        </>
    );
```

6. Add newly created component to index.jsx.

- After creating the component, you need re-export to the pages/index.jsx so that it can be easily referenced/imported elsewhere.

`src/pages/index.jsx`

```
export {default as CreatePet} from "./AptifyCustom/CreatePet";
```

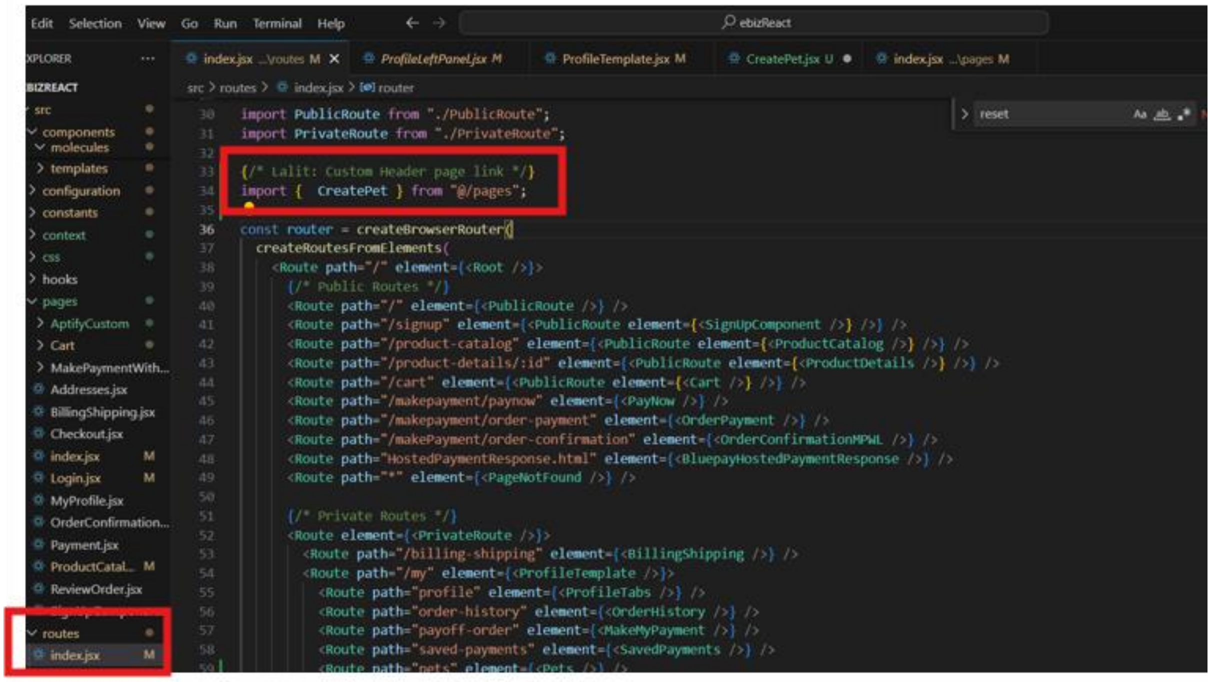


```
export { default as Address } from "./Addresses";
export { default as BillingShipping } from "./BillingShipping";
export { default as Cart } from "./Cart";
export { default as Checkout } from "./Checkout";
export { default as Login } from "./Login";
export { default as OrderConfirmationMPWL } from "./MakePaymentWithoutLogin/OrderConfirmationMPWL";
export { default as OrderPayment } from "./MakePaymentWithoutLogin/OrderPayment";
export { default as PayNow } from "./MakePaymentWithoutLogin/PayNow";
export { default as MyProfile } from "./MyProfile";
export { default as OrderConfirmation } from "./OrderConfirmation";
export { default as Payment } from "./Payment";
export { default as ProductCatalog } from "./ProductCatalog";
export { default as ReviewOrder } from "./ReviewOrder";
export { default as SignupComponent } from "./SignupComponent";
{/*Lalit: Aptify Custom Export Components */}
export {default as CreatePet} from "./AptifyCustom/CreatePet";
```

7. Add pages to navigate/router

- Open routes/Index.jsx file
- To add pages to the router, we need to import our pages into the component and add their references.

```
{/* Lalit: Custom Header page link */}
import {  CreatePet } from "@/pages";
```



```
import PublicRoute from "./PublicRoute";
import PrivateRoute from "./PrivateRoute";

{/* Lalit: Custom Header page link */}
import {  CreatePet } from "@/pages";

const router = createBrowserRouter(
  createRoutesFromElements(
    <Route path="/" element={<Root />}>
      {/* Public Routes */}
      <Route path="/" element={<PublicRoute />} />
      <Route path="/signup" element={<PublicRoute element={<SignUpComponent />} />} />
      <Route path="/product-catalog" element={<PublicRoute element={<ProductCatalog />} />} />
      <Route path="/product-details/:id" element={<PublicRoute element={<ProductDetails />} />} />
      <Route path="/cart" element={<PublicRoute element={<Cart />} />} />
      <Route path="/makepayment/paynow" element={<PayNow />} />
      <Route path="/makepayment/order-payment" element={<OrderPayment />} />
      <Route path="/makePayment/order-confirmation" element={<OrderConfirmationMPWL />} />
      <Route path="HostedPaymentResponse.html" element={<BluepayHostedPaymentResponse />} />
      <Route path="*" element={<PageNotFound />} />

      {/* Private Routes */}
      <Route element={<PrivateRoute />}>
        <Route path="/billing-shipping" element={<BillingShipping />} />
        <Route path="/my" element={<ProfileTemplate />}>
          <Route path="profile" element={<ProfileTabs />} />
          <Route path="order-history" element={<OrderHistory />} />
          <Route path="payoff-order" element={<MakeMyPayment />} />
          <Route path="saved-payments" element={<SavedPayments />} />
          <Route path="pets" element={<Pets />} />
```

-
- If the route is public (i.e., the user does not need to be logged in), it can be added directly to the Public Routes section
- Public routes are defined using the PublicRoute component. Give alias to your page to show in URL like create-pet.
  For example:

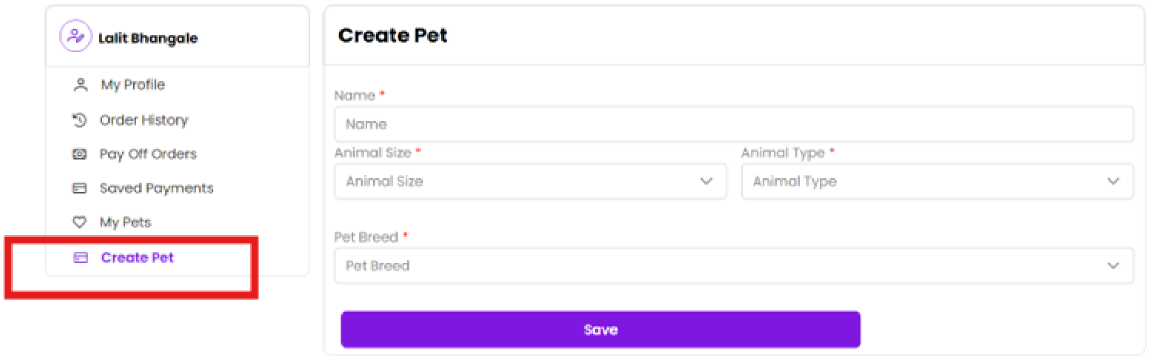
```
 {/* Private Routes */}
<Route element={< PublicRoute />}>
      <Route path="create-pet" element={<CreatePet />} />
</Route
```

- For routes where the user needs to be logged in, they can be added to the Private Routes section, as shown within the Private Routes element.
- Private routes are wrapped within the PrivateRoute component to ensure they are only accessible to authenticated users.

```
 For example:

 {/* Private Routes */}
<Route element={<PrivateRoute />}>
      <Route path="create-pet" element={<CreatePet />} />
</Route>
```

8. Add menu under the profile pages

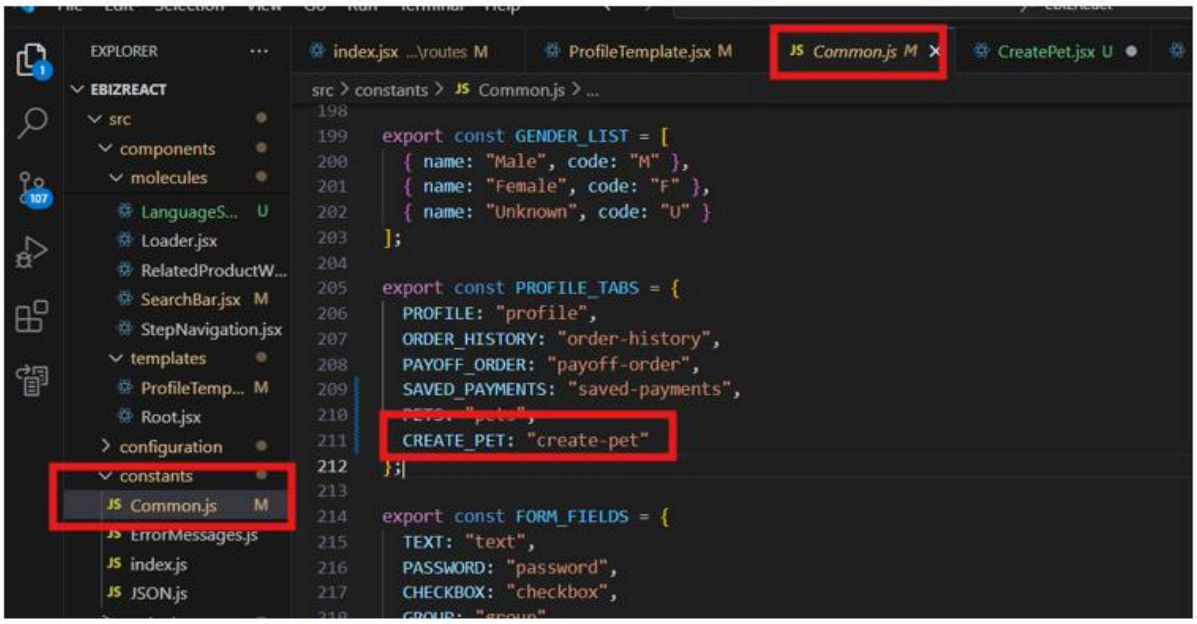
To show menu under the profile tab, we need to follow the couple step.



a. Add constant for new tab i.e. create-pet

i. Open \src\constants\Common.js and add below code

```
,
 CREATE_PET: "create-pet"
```



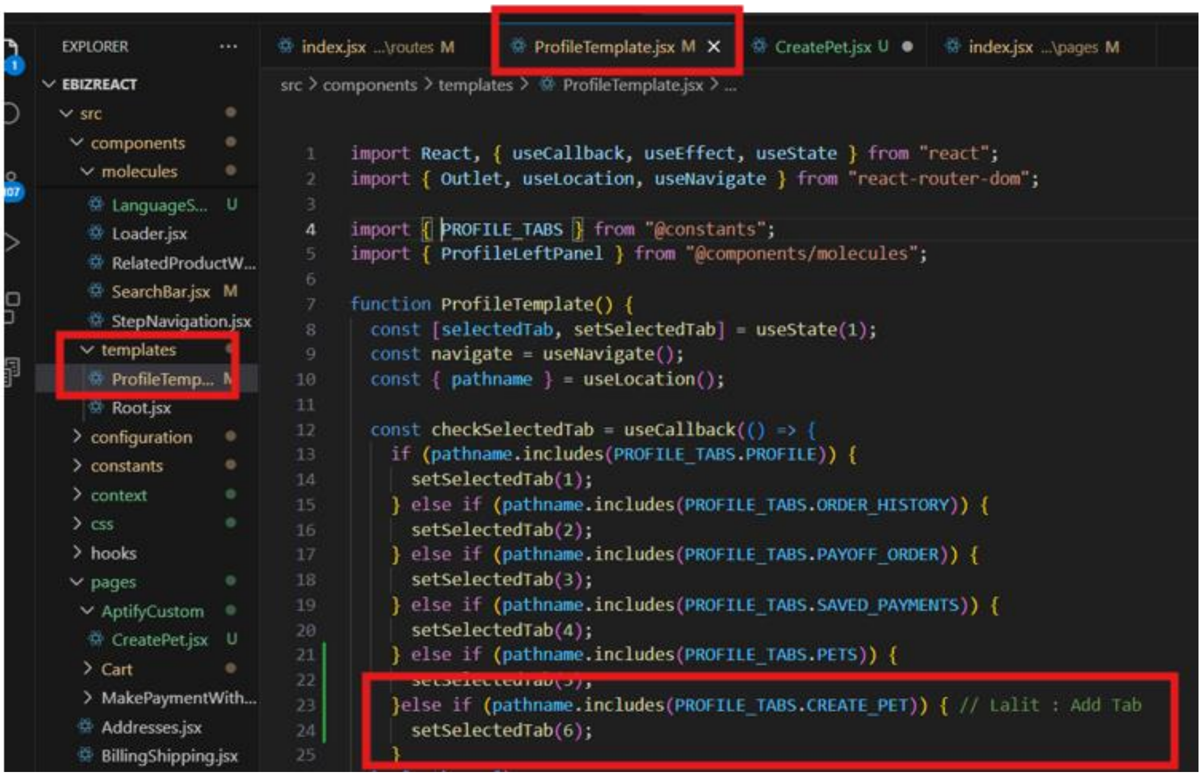
```
export const GENDER_LIST = [
  { name: "Male", code: "M" },
  { name: "Female", code: "F" },
  { name: "Unknown", code: "U" }
];

export const PROFILE_TABS = {
  PROFILE: "profile",
  ORDER_HISTORY: "order-history",
  PAYOFF_ORDER: "payoff-order",
  SAVED_PAYMENTS: "saved-payments",
  PETS: "pets",
  CREATE_PET: "create-pet"
};

export const FORM_FIELDS = {
  TEXT: "text",
  PASSWORD: "password",
  CHECKBOX: "checkbox",
  GROUP: "group"
```

b. Add tab into profile template file.

I. Open \src\components\templates\ProfileTemplate.jsx

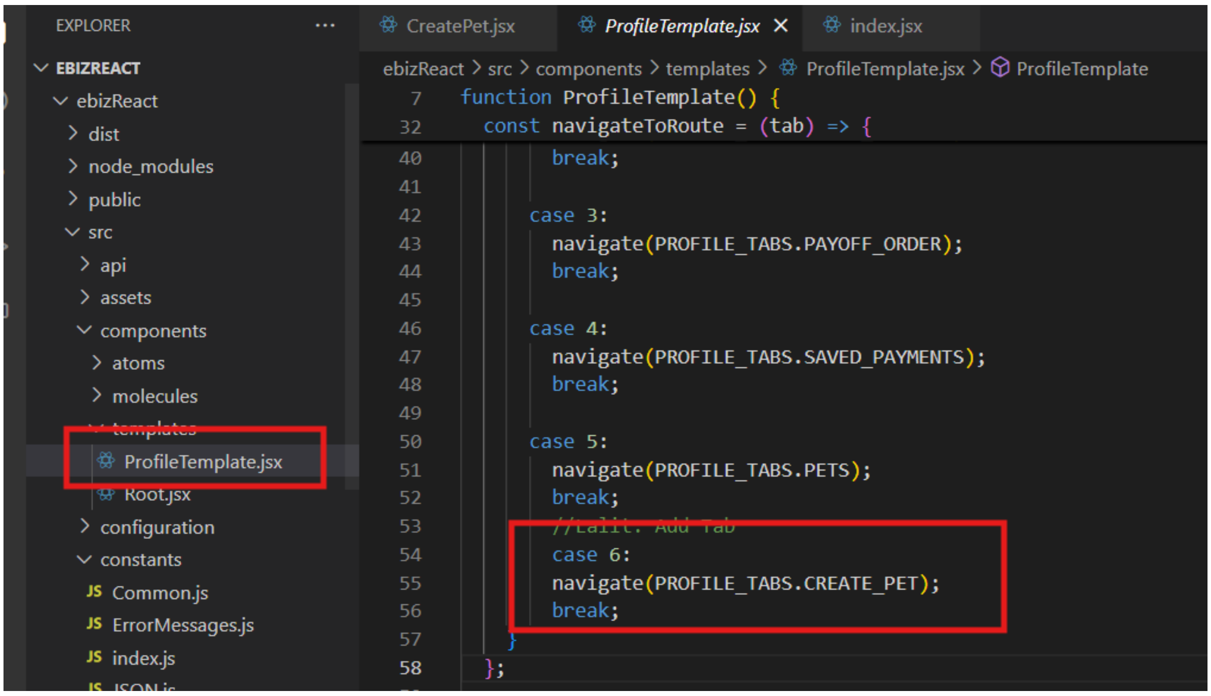
II. Add below code, refer below screen for more detail about where to add code

```
}else if (pathname.includes(PROFILE_TABS.CREATE_PET)) { // Lalit : Add Tab
      setSelectedTab(6);
    }
```



```
import React, { useCallback, useEffect, useState } from "react";
import { Outlet, useLocation, useNavigate } from "react-router-dom";

import { PROFILE_TABS } from "@constants";
import { ProfileLeftPanel } from "@components/molecules";

function ProfileTemplate() {
  const [selectedTab, setSelectedTab] = useState(1);
  const navigate = useNavigate();
  const { pathname } = useLocation();

  const checkSelectedTab = useCallback(() => {
    if (pathname.includes(PROFILE_TABS.PROFILE)) {
      setSelectedTab(1);
    } else if (pathname.includes(PROFILE_TABS.ORDER_HISTORY)) {
      setSelectedTab(2);
    } else if (pathname.includes(PROFILE_TABS.PAYOFF_ORDER)) {
      setSelectedTab(3);
    } else if (pathname.includes(PROFILE_TABS.SAVED_PAYMENTS)) {
      setSelectedTab(4);
    } else if (pathname.includes(PROFILE_TABS.PETS)) {
      setSelectedTab(5);
    }else if (pathname.includes(PROFILE_TABS.CREATE_PET)) { // Lalit : Add Tab
      setSelectedTab(6);
    }
```

```
//Lalit: Add Tab
        case 6:
        navigate(PROFILE_TABS.CREATE_PET);
        break;
```



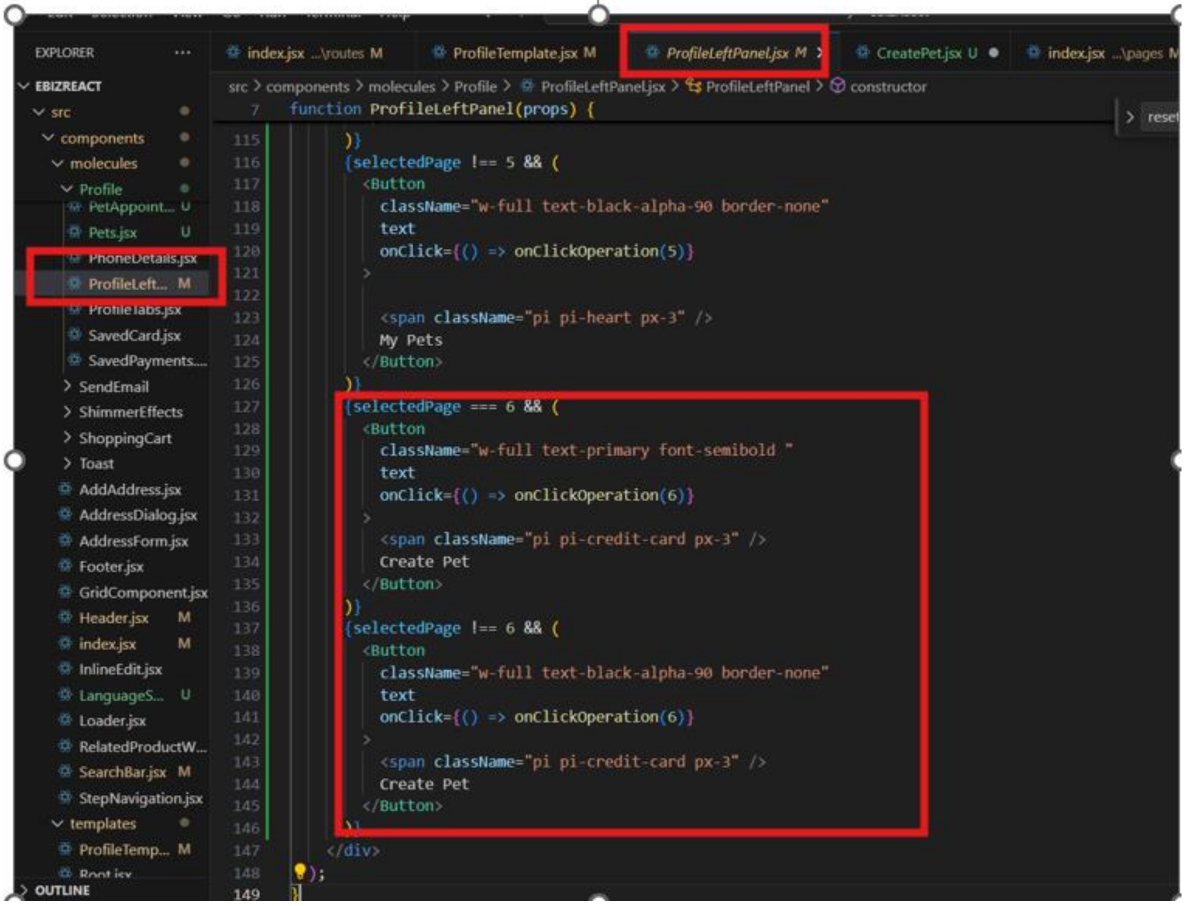
```
function ProfileTemplate() {
  const navigateToRoute = (tab) => {
        break;

      case 3:
        navigate(PROFILE_TABS.PAYOFF_ORDER);
        break;

      case 4:
        navigate(PROFILE_TABS.SAVED_PAYMENTS);
        break;

      case 5:
        navigate(PROFILE_TABS.PETS);
        break;
        //Lalit: Add Tab
        case 6:
        navigate(PROFILE_TABS.CREATE_PET);
        break;
    }
  };
```

c. Add HMTL for menu into ProfileLeftPanel

I. Open src\components\molecules\Profile\ProfileLeftPanel.jsx file

II. Add below code, refer below screen for more detail about where to add code.

```
{selectedPage === 6 && (
        <Button
          className="w-full text-primary font-semibold "
          text
          onClick={() => onClickOperation(6)}
        >
          <span className="pi pi-credit-card px-3" />
          Create Pet
        </Button>
      )}
      {selectedPage !== 6 && (
        <Button
          className="w-full text-black-alpha-90 border-none"
          text
          onClick={() => onClickOperation(6)}
        >
          <span className="pi pi-credit-card px-3" />
          Create Pet
        </Button>
      )}
```



```
  function ProfileLeftPanel(props) {
          )}
          {selectedPage !== 5 && (
            <Button
              className="w-full text-black-alpha-90 border-none"
              text
              onClick={() => onClickOperation(5)}
            >

              <span className="pi pi-heart px-3" />
              My Pets
            </Button>
          )}
          {selectedPage === 6 && (
            <Button
              className="w-full text-primary font-semibold "
              text
              onClick={() => onClickOperation(6)}
            >
              <span className="pi pi-credit-card px-3" />
              Create Pet
            </Button>
          )}
          {selectedPage !== 6 && (
            <Button
              className="w-full text-black-alpha-90 border-none"
              text
              onClick={() => onClickOperation(6)}
            >
              <span className="pi pi-credit-card px-3" />
              Create Pet
            </Button>
          )}
        </div>
  );
}
```

9. Run Your Application

- Start the Development Server: `In your terminal, run below command,`

```
npm dev run
```

- View Your Component: Open your browser and go to `http://localhost:3000`. You should see your component displayed on the page.

10. Refer full code for Createpet component:

Add code details for each step at below

```
/* Date. 05 Sep 2024 - Create the Aptify custom components */
import React, { useEffect, useState, useMemo } from "react";
/* React: The core library for building user interfaces.
useEffect: A hook for performing side effects in function components (e.g., data
fetching, subscriptions).
useState: A hook for adding state to function components.
useMemo: A hook for memoizing expensive calculations to optimize performance. */
import { _get, _post } from "@api/APIClient"; /* _get and _post: Functions for making
GET and POST requests to your API. */
import { useStateUser } from "@hooks/useStateUser"; /* useStateUser: A custom hook,
likely for managing user state. */
import { SimpleButton } from "@/components/atoms"; /* SimpleButton: A reusable button
component. */
import { useForm } from "react-hook-form";   /* useForm: A hook for managing form state
and validation. */
import { FormBuilder } from "@/components/molecules";/*  FormBuilder: A component for
dynamically building forms. */
import { HTTP_STATUS_CODES } from "@/constants"; /* HTTP_STATUS_CODES: An object
containing HTTP status codes for reference. */
import { useToast } from "@/context/ToasterProvider"; /* useToast: A hook for displaying
toast notifications. */
import "@css/PetCatalog.scss";  /* A stylesheet for styling your components. */
import { yupResolver } from "@hookform/resolvers/yup"; /* yupResolver: A resolver for
integrating Yup validation with react-hook-form. */
import { createMyPetValidationSchema } from
"@/validations/AptifyCustom/createMyPetValidation"; /* createMyPetValidationSchema: A
Yup schema for validating form data. */

/* Create the Component named "CreatePet" */
const CreatePet = () => {
 /* This line initializes a state variable user using a custom hook useStateUser().
    This hook likely manages the state related to the user.  */
    const user = useStateUser();

/*  These lines destructure functions from the useToast hook. showToastSuccess,
showToastError,
    and showToastInfo are likely functions to display different types of toast
notifications (success, error, info). */
    const { showToastSuccess } = useToast();
    const { showToastError } = useToast();
    const { showToastInfo } = useToast();

/*  This line initializes a state variable loading with a default value of false.
setLoading is the function to update this state. */
    const [loading, setLoading] = useState(false);

/*  These lines initialize state variables sizeData, breedData, and typeData as empty
arrays. setAnimalSize, setPetBreed,
    and setAnimalType are the functions to update these states. */
    const [sizeData, setAnimalSize] = useState([]);
    const [breedData, setPetBreed] = useState([]);
    const [typeData, setAnimalType] = useState([]);

/*  These lines transform the sizeData, typeData, and breedData arrays into options for
a dropdown or select input.
    Each item is mapped to an object with name and value properties. */
    const sizeOptions = sizeData.map(item => ({ name: item.Name, value: item.ID }));
    const typeOptions = typeData.map(item => ({ name: item.Name, value: item.ID }));
    const breedOptions = breedData.map(item => ({ name: item.Name, value: item.ID }));

/*  This block initializes a form using the useForm hook.
    defaultValues sets the initial values for the form fields.
    mode: "onBlur" specifies that validation should occur when an input loses focus.
    resolver: yupResolver(createMyPetValidationSchema) integrates Yup validation schema
for form validation. */
    const form = useForm({
        defaultValues: {
            Name: '',
            animalSize: null,
            animalType: null,
            petBreed: null
        },
        mode: "onBlur",
        resolver: yupResolver(createMyPetValidationSchema)
    });

/*  This line destructures reset, isSubmitting, and isValid from the form object.
    reset is a function to reset the form.
    isSubmitting indicates if the form is currently being submitted.
    isValid indicates if the form is valid according to the validation schema. */
    const { reset,  formState: { isSubmitting, isValid } } = form;


    /* const fields = useMemo(() => [...], [sizeOptions, typeOptions, breedOptions]);
    This line uses the useMemo hook to memoize the fields array.
    This means fields will only be recalculated when sizeOptions, typeOptions, or
breedOptions change, improving performance.
     */
    const fields = useMemo(() => [
            {
                /* This object defines a text input field for the form. */
                type: "text",
                name: "Name",
                label: "Name",
                isRequired: true
            },
            {
                /* This object defines a group of fields.
                type: "group" specifies that this is a group of inputs.
                group: [...] contains the fields within the group. */

                type: "group",
                group:
                [
                    {
                        /* This object defines a dropdown input for selecting animal
size. */
                        type: "dropdown",
                        name: "animalSize",
                        label: "Animal Size",
                        options: sizeOptions,  /* provides the options for the dropdown.
*/
                        isRequired: true /* makes this field mandatory. */
                    }, {
                        /* This object defines a dropdown input for selecting animal
type. */
                        type: "dropdown",
                        name: "animalType",
                        label: "Animal Type",
                        options: typeOptions, /* provides the options for the dropdown.
*/
                        isRequired: true /* makes this field mandatory. */
                    }
                ]
            },
            {
                /* This object defines a dropdown input for selecting pet breed. */
                type: "dropdown",
                name: "petBreed",
                label: "Pet Breed",
                options: breedOptions, /* provides the options for the dropdown. */
                isRequired: true  /* makes this field mandatory. */
            }
        ],[sizeOptions, typeOptions, breedOptions]);

    /* This is a React Hook that runs the provided function after the component mounts.
    The empty dependency array [] means it runs only once, similar to componentDidMount.
*/
    useEffect(() => {

        /* This defines an asynchronous function getPetSize that fetches pet size data
from the server. */
        const getPetSize = async () => {
            /* Inside getPetSize, this line sends a GET request to the endpoint
/v1/PetSize with credentials included.
            The await keyword ensures the function waits for the response before
proceeding. */
            const response = await _get("/v1/PetSize", {
                withCredentials: true
            });
            /* Once the response is received, this line updates the state with the
fetched data using the setAnimalSize function. */
            setAnimalSize(response.data);

        };
        /* Similar to getPetSize, this defines an asynchronous function getPetType that
fetches pet type data from the server. */
        const getPetType = async () => {
            /* Sends a GET request to the endpoint /v1/PetType with credentials
included. */
            const response = await _get("/v1/PetType", {
                withCredentials: true
            });
            /* Updates the state with the fetched pet type data. */
            setAnimalType(response.data);
        };
        /* Defines an asynchronous function getPetBreed that fetches pet breed data from
the server. */
        const getPetBreed = async () => {
            /* Sends a GET request to the endpoint /v1/PetBreed with credentials
included. */
            const response = await _get("/v1/PetBreed", {
                withCredentials: true
            });
            /* Updates the state with the fetched pet breed data. */
            setPetBreed(response.data);

        };
        /* Calls the getPetSize function to fetch and set pet size data. */
        getPetSize();
        /* Calls the getPetType function to fetch and set pet type data. */
        getPetType();
        /* Calls the getPetBreed function to fetch and set pet breed data. */
        getPetBreed();

    }, []);


    /* This line defines an asynchronous function named savePetDetails that takes data
as an argument which is a form data
    and this trigger on submit button click on the FormBuilder. */
    const savePetDetails = async (data) => {
        /* The try block is used to handle any errors that might occur during the
execution of the code inside it. */
        try {
            /* setLoading(true); sets a loading state to true, likely to show a loading
indicator in the UI. */
            setLoading(true);

            /* This if statement checks if the data object has non-null values for Name,
animalSize, petBreed, and animalType. */
            if (data?.Name != null && data?.animalSize != null && data?.petBreed != null
&& data?.animalType != null) {
            /* If the condition is true, a payLoad object is created with properties
Name, SizeID, BreedID, and TypeID from
            the data object. */
             const payLoad = {
                    "Name": data?.Name,
                    "SizeID": data?.animalSize,
                    "BreedID": data?.petBreed,
                    "TypeID": data?.animalType
                }
                /*  An asynchronous POST request is made to the endpoint /v1/Pets/
followed by the authenticated user's ID.
```

```
                    The payLoad object is sent as the request body.
                    The request includes credentials and sets the Content-Type header to
application/json. */
                const createPetRecords = await _post("/v1/Pets/" +
user?.AuthenticatedPersonId, payLoad, {
                    withCredentials: true,
                    headers: {
                        "Content-Type": "application/json"
                    }
                });

                if (createPetRecords.status === HTTP_STATUS_CODES.OK) {
                    /* if the response status is OK, the loading state is set to false.
*/
                    setLoading(false);
                    /* A success toast message is shown with a summary of “Success” and
details from the response data. */
                    showToastSuccess({
                        summary: "Success",
                        detail: createPetRecords.data?.Message + " #" +
createPetRecords.data?.PetID
                    });
                    /* The form fields are reset to empty strings.*/
                    reset({
                        Name: '',
                        animalSize: '',
                        petBreed: '',
                        animalType: ''
                    })
                }
                return true;
            } else {
                /* If any of the required fields are missing, an info toast message is
shown and the loading state is set to false. */
                showToastInfo({
                    summary: "info",
                    detail: "Please fill the mandatory details."

                });
                setLoading(false);
            }

        } catch (error) {
            /* If an error occurs, the loading state is set to false, an error toast
message is shown,
            and the error is logged to the console. */
            setLoading(false);
            showToastError({
                 summary: "error",
                 detail: error

             });
             console.log(error);
         }
     };

     return (
         <>
             <main>
                 <div>
                     <div className="eb-border-gray">
                         <div className="eb-profile-text-color font-bold text-2xl px-3
pt-3 pb-3 eb-border-gray border-noround-bottom">
                             Create Pet
                         </div>
                         <br />
                         <section>
                             <div className="eb-container">
                             {/* This component renders a form using the FormBuilder
component. */}
                                 <FormBuilder
                                 fields={fields}  /*  passes the fields array to the
FormBuilder. */
                                 form={form}  /* passes the form object to the
FormBuilder. */
                                 onSubmit={savePetDetails} /* sets the function to call
when the form is submitted. */
                                 >
                                     {/* This div contains the submit button */}
                                     <div className="col-8 mt-4">
                                     {/* This component renders a button. */}
                                         <SimpleButton
                                             type="submit"
                                             navigatelink={"false"} /* likely disables
navigation on click. */
                                             label={"Save"}
                                             className={"simpleButtonStyle"}
                                             loading={loading} /* binds the button’s
loading state to the loading state variable. */
                                             /* disabled={isSubmitting || !isValid} This
is commented out. If uncommented, it would disable the button when the form is
submitting or invalid. */
                                         />
                                     </div>
                                 </FormBuilder>
                             </div>
                         </section>
                     </div>
                 </div>
             </main>
         </>
     );
};
/* Export this component named CreatePet, This promotes code reusability and
maintainability by
encapsulating related functionality in separate files. */
export default CreatePet;
```

11. Prerequisite files

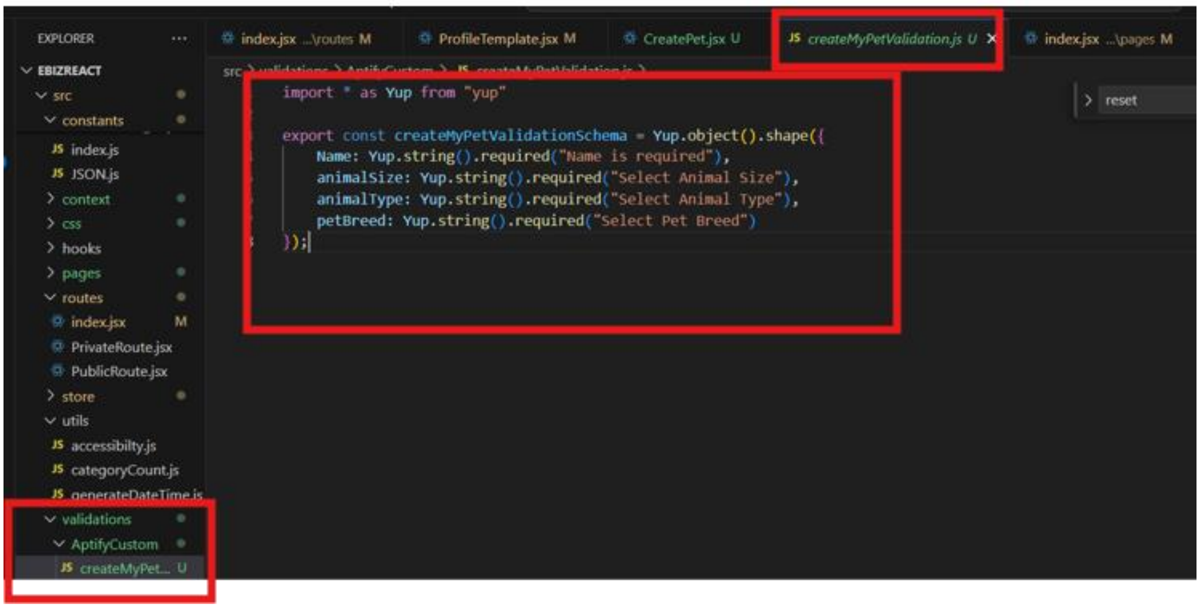
a. Validation file:

- Open Validation folder and create new folder "AptifyCustom" and create new file createMyPetValidation.js

    Path: \src\validations\AptifyCustom\createMyPetValidation.js

- Add below code

```
import * as Yup from "yup"

export const createMyPetValidationSchema = Yup.object().shape({
    Name: Yup.string().required("Name is required"),
    animalSize: Yup.string().required("Select Animal Size"),
    animalType: Yup.string().required("Select Animal Type"),
    petBreed: Yup.string().required("Select Pet Breed")
});
```

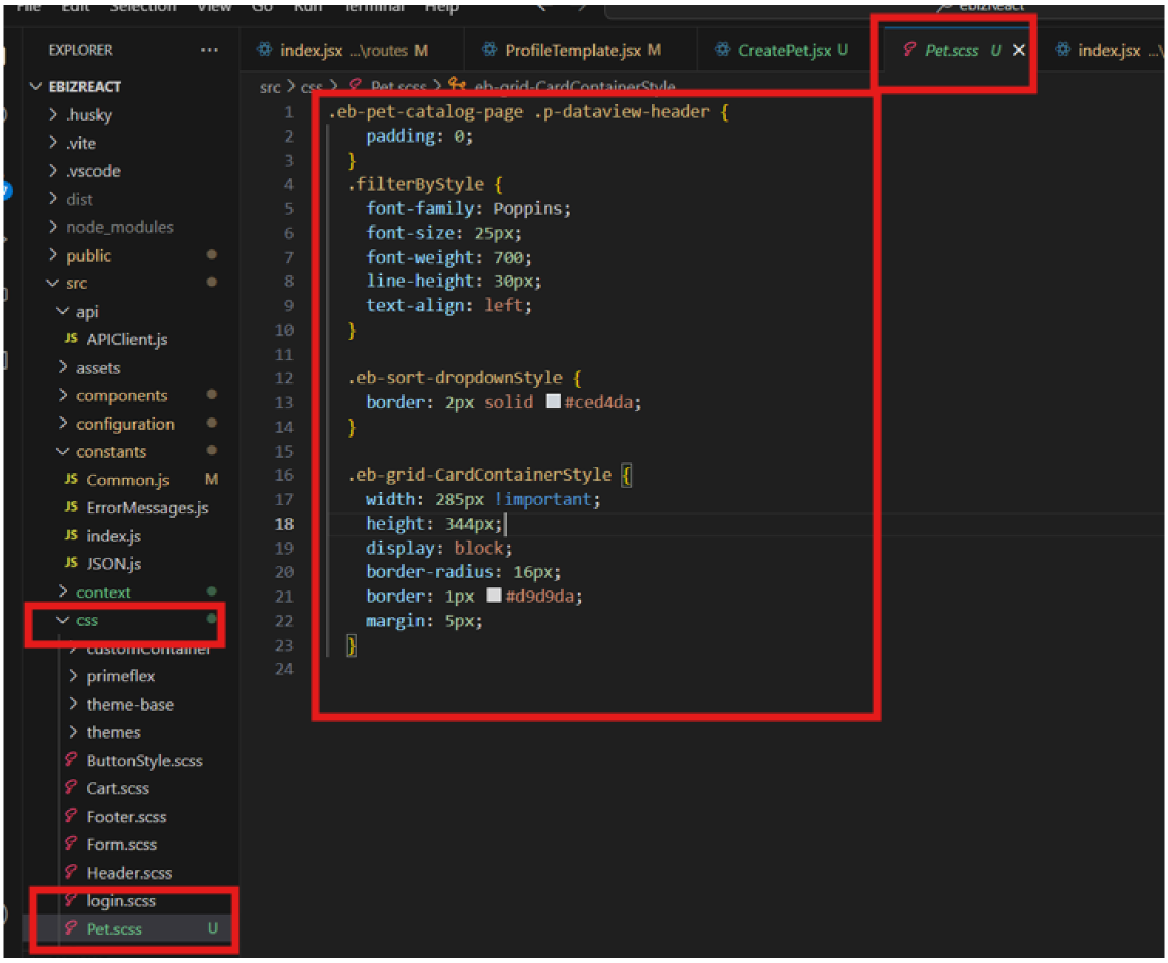


```
import * as Yup from "yup"

export const createMyPetValidationSchema = Yup.object().shape({
    Name: Yup.string().required("Name is required"),
    animalSize: Yup.string().required("Select Animal Size"),
    animalType: Yup.string().required("Select Animal Type"),
    petBreed: Yup.string().required("Select Pet Breed")
});
```

b. Css

- Open CSS folder and create new file Pet.scss
  Path: \src\css\Pet.scss
- Add below code

```
.eb-pet-catalog-page .p-dataview-header {
    padding: 0;
  }
  .filterByStyle {
    font-family: Poppins;
    font-size: 25px;
    font-weight: 700;
    line-height: 30px;
    text-align: left;
  }

  .eb-sort-dropdownStyle {
    border: 2px solid #ced4da;
  }

  .eb-grid-CardContainerStyle {
    width: 285px !important;
    height: 344px;
    display: block;
    border-radius: 16px;
    border: 1px #d9d9da;
    margin: 5px;
  }
```



```
.eb-pet-catalog-page .p-dataview-header {
    padding: 0;
  }
  .filterByStyle {
    font-family: Poppins;
    font-size: 25px;
    font-weight: 700;
    line-height: 30px;
    text-align: left;
  }

  .eb-sort-dropdownStyle {
    border: 2px solid #ced4da;
  }

  .eb-grid-CardContainerStyle {
    width: 285px !important;
    height: 344px;
    display: block;
    border-radius: 16px;
    border: 1px #d9d9da;
    margin: 5px;
  }
```