```
import LoadingSpinner from '@/shared/components/loading-spinner';
import Modal from '@/shared/components/modal';
import { Button } from '@/shared/components/ui/Button';
import Input from '@/shared/components/ui/Input';
import { useToast } from '@/shared/components/ui/Toasts/use-toast';
import { debounce } from '@/shared/utils/helpers';
import { trpc } from '@/shared/utils/trpc/trpc';
import { ArrowLeft, Plus } from 'lucide-react';
import { useMemo, useState } from 'react';
import { Renderer, RichTextarea } from 'rich-textarea';
import { Highlight, themes } from 'prism-react-renderer';
import Prism from 'prismjs';
import 'prismjs/components/prism-python';
import { useTheme } from 'next-themes';
import { useAuthContext } from '@/shared/components/ui/auth.provider';

interface Props {
  isOpen: boolean;
  onClose: () => void;
  onAddSuccessfuly: () => void;
}

const AddSwarmModal = ({ isOpen, onClose, onAddSuccessfuly }: Props) => {
  const { user } = useAuthContext();
  const theme = useTheme();
```

```
const [step, setStep] = useState<'info' | 'code'>('info');
const [swarmName, setSwarmName] = useState('');
const [description, setDescription] = useState('');
const [tags, setTags] = useState('');
const [useCases, setUseCases] = useState<
  {
    title: string;
    description: string;
  }[]
>([
  {
    title: '',
    description: '',
  },
]);

const [code, setCode] = useState('');

const validateSwarmName = trpc.explorer.validateSwarmName.useMutation();

const debouncedCheckName = useMemo(() => {
  const debouncedFn = debounce((value: string) => {
    validateSwarmName.mutateAsync(value);
  }, 400);
  return debouncedFn;
}, []);
```

```
const toast = useToast();
const nextStep = () => {
  // validate
  // name empty
  // validatino is not loading
  if (validateSwarmName.isPending) {
    toast.toast({
      title: 'validating swarm name',
      variant: 'destructive',
    });
    return;
  }
  if (swarmName.length === 0) {
    toast.toast({
      title: 'Swarm Name is required',
      variant: 'destructive',
    });
    return;
  }
  // check name validation
  if (validateSwarmName.data && !validateSwarmName.data.valid) {
    toast.toast({
      title: 'Invalid Swarm Name',
      description: validateSwarmName.data.error,
      variant: 'destructive',
```

```
    });
    return;
  }
  // at least 1 use case
  // usecases should not be empty ,title and description
  for (const useCase of useCases) {
    if (
      useCase.title.trim().length === 0 ||
      useCase.description.trim().length === 0
    ) {
      toast.toast({
        title: `Use case ${useCase.title.trim().length === 0 ? 'title' : 'description'} is required`,
        variant: 'destructive',
      });
      return;
    }
  }

  setStep('code');
};
const addSwarm = trpc.explorer.addSwarm.useMutation();
const submit = () => {
  // code should'nt empty
  if (code.trim().length === 0) {
    toast.toast({
      title: 'Code is required',
```

```
    variant: 'destructive',
  });
  return;
}
addSwarm
  .mutateAsync({
    name: swarmName,
    description,
    useCases,
    tags,
    code,
  })
  .then(() => {
    toast.toast({
      title: 'Swarm added successfully ',
    });
    onClose();
    onAddSuccessfuly();
    // reset form
    setStep('info');
    setSwarmName('');
    setDescription('');
    setTags('');
    setUseCases([{ title: '', description: '' }]);
    setCode('');
  });
```

```
};

if (!user) return null;

const pythonRenderer: Renderer = (value) => {
  return (
    <Highlight
      prism={Prism}
      theme={theme.theme === 'dark' ? themes.oneDark : themes.oneLight}
      code={value}
      language="python"
    >
      {({ className, style, tokens, getLineProps, getTokenProps }) => (
        <div className={className} style={style}>
          {tokens.map((line, i) => (
            <div {...getLineProps({ line, key: i })}>
              {line.map((token, key) => (
                <span {...getTokenProps({ token, key })} />
              ))}
            </div>
          ))}
        </div>
      )}
    </Highlight>
  );
};
```

```
return (
  <Modal
    className="max-w-2xl"
    isOpen={isOpen}
    onClose={onClose}
    title="Add Swarm"
  >
    <div className="flex flex-col gap-2 overflow-y-auto h-[75vh] relative px-4">
      {step === 'info' && (
        <>
          <div className="flex flex-col gap-1">
            <span>Swarm Name</span>
            <div className="relative">
              <Input
                value={swarmName}
                onChange={(v) => {
                  setSwarmName(v);
                  debouncedCheckName(v);
                }}
                placeholder="Enter Swarm Name"
              />
              {validateSwarmName.isPending ? (
                <div className="absolute right-2 top-2">
                  <LoadingSpinner />
                </div>
              ) : (
```

```
      <div className="absolute right-2.5 top-2.5">
        {swarmName.length > 0 && validateSwarmName.data && (
          <span
            className={
              validateSwarmName.data.valid
                ? 'text-green-500'
                : 'text-red-500'
            }
          >
            {validateSwarmName.data.valid ? '' : ''}
          </span>
        )}
      </div>
    )}
  </div>
  {swarmName.length > 0 &&
    !validateSwarmName.isPending &&
    validateSwarmName.data &&
    !validateSwarmName.data.valid && (
      <span className="text-red-500 text-sm">
        {validateSwarmName.data.error}
      </span>
    )}
</div>
<div className="flex flex-col gap-1">
  <span>Description</span>
```

```
  <textarea
    value={description}
    onChange={(e) => setDescription(e.target.value)}
    placeholder="Enter description"
    className="w-full h-20 p-2 border rounded-md bg-transparent outline-0 resize-none"
  />
</div>
<div className="flex flex-col gap-1">
  <span>Tags</span>
  <Input
    value={tags}
    onChange={setTags}
    placeholder="Tools, Search, etc."
  />
</div>
<div className="mt-2 flex flex-col gap-1">
  <span>Use Cases</span>
  <div className="flex flex-col gap-2">
    {useCases.map((useCase, i) => (
      <div key={i} className="flex gap-4 items-center">
        <span className="w-8">
          <span># {i + 1}</span>
        </span>
        <div className="w-full flex flex-col gap-1 py-2">
          <Input
            value={useCase.title}
```

```
                onChange={(v) => {
                  const newUseCases = [...useCases];
                  newUseCases[i].title = v;
                  setUseCases(newUseCases);
                }}
                placeholder="Title"
              />
              <textarea
                value={useCase.description}
                onChange={(e) => {
                  const newUseCases = [...useCases];
                  newUseCases[i].description = e.target.value;
                  setUseCases(newUseCases);
                }}
                placeholder="Description"
                            className="w-full h-20 p-2 border rounded-md bg-transparent outline-0
resize-none"
              />
            </div>
            <div className="w-4">
              {i > 0 && (
                <button
                  onClick={() => {
                    const newUseCases = [...useCases];
                    newUseCases.splice(i, 1);
                    setUseCases(newUseCases);
```

```
            }}
            className="text-red-500"
          >

          </button>
        )}
      </div>
    </div>
  ))}
  <div className="flex justify-center">
    <button
      onClick={() =>
        setUseCases([...useCases, { title: '', description: '' }])
      }
      className="text-blue-500"
    >
      <Plus />
    </button>
  </div>
 </div>
</div>
<div className="flex justify-end mt-4">
 <Button className="w-32" onClick={nextStep}>
  Next
 </Button>
</div>
```

```
      </>
    )}

    {
      // code editor
      step === 'code' && (
        <>
          <div className="w-full flex gap-2">
            <ArrowLeft onClick={() => setStep('info')} />
            <span>Back</span>
          </div>
          <div className="flex flex-col mt-2 h-full">
            <span>Swarm Code ( paste here )</span>
            <style>
              {`.prism-code{
                  background-color: transparent !important;
                }`}
            </style>
            <RichTextarea
              spellCheck="false"
              autoCorrect="off"
                        className="dark all-initial h-full bg-transparent p-2 outline-none text-black
overflow-y-auto resize-none !caret-white"
              style={{
                width: '100%',
                height: '100%',
```

```
            }}
            value={code}
            onChange={(e) => {
              setCode(e.target.value);
            }}
          >
            {pythonRenderer}
          </RichTextarea>
          <div className="flex justify-end">
            <Button
              disabled={addSwarm.isPending}
              onClick={submit}
              className="w-32 mt-4"
            >
              Submit
            </Button>
          </div>
        </div>
      </>
    )
  }
</div>
</Modal>
);
};
```

```
export default AddSwarmModal;
```