```
import LoadingSpinner from '@/shared/components/loading-spinner';
import Modal from '@/shared/components/modal';
import { useAuthContext } from '@/shared/components/ui/auth.provider';
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, launchConfetti } from '@/shared/utils/helpers';
import { trpc } from '@/shared/utils/trpc/trpc';
import { Plus } from 'lucide-react';
import { useMemo, useState } from 'react';

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

const AddPromptModal = ({ isOpen, onClose, onAddSuccessfully }: Props) => {
  const { user } = useAuthContext();

  const [promptName, setPromptName] = useState('');
  const [description, setDescription] = useState('');
  const [prompt, setPrompt] = useState('');
  const [tags, setTags] = useState('');
  const [useCases, setUseCases] = useState<
    {
```

```
    title: string;
    description: string;
  }[]
>([
  {
    title: '',
    description: '',
  },
]);

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

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

const toast = useToast();

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

const submit = () => {
  // Validate prompt
  if (validatePrompt.isPending) {
```

```
  toast.toast({
    title: 'Validating prompt',
  });
  return;
}

if (promptName.trim().length < 2) {
  toast.toast({
    title: 'Name should be at least 2 characters long',
    variant: 'destructive',
  });
  return;
}

if (prompt.trim().length === 0) {
  toast.toast({
    title: 'Prompt is required',
    variant: 'destructive',
  });
  return;
}

if (validatePrompt.data && !validatePrompt.data.valid) {
  toast.toast({
    title: 'Invalid Prompt',
    description: validatePrompt.data.error,
```

```
    variant: 'destructive',
  });
  return;
}

// Validate use cases
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;
  }
}

const trimTags = tags
  .split(',')
  .map((tag) => tag.trim())
  .filter(Boolean)
  .join(',');

// Add prompt
```

```
  addPrompt
    .mutateAsync({
      name: promptName,
      prompt,
      description,
      useCases,
      tags: trimTags,
    })
    .then(() => {
      toast.toast({
        title: 'Prompt added successfully ',
      });
      onClose();

      //celeberate the confetti
      launchConfetti();

      onAddSuccessfully();
      // Reset form
      setPromptName('');
      setDescription('');
      setPrompt('');
      setTags('');
      setUseCases([{ title: '', description: '' }]);
    });
};
```

```
if (!user) return null;


return (
  <Modal
    className="max-w-2xl"
    isOpen={isOpen}
    onClose={onClose}
    title="Add Prompt"
  >
    <div className="flex flex-col gap-2 overflow-y-auto h-[75vh] relative px-4">
      <div className="flex flex-col gap-1">
        <span>Name</span>
        <div className="relative">
          <Input
            value={promptName}
            onChange={setPromptName}
            placeholder="Enter name"
          />
        </div>
      </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>Prompt</span>
  <div className="relatvie">
    <textarea
      value={prompt}
      onChange={(v) => {
        setPrompt(v.target.value);
        debouncedCheckPrompt(v.target.value);
      }}
      required
      placeholder="Enter prompt here..."
      className="w-full h-20 p-2 border rounded-md bg-transparent outline-0 resize-none"
    />
    {validatePrompt.isPending ? (
      <div className="absolute right-2 top-2">
        <LoadingSpinner />
      </div>
    ) : (
      <div className="absolute right-2.5 top-2.5">
        {prompt.length > 0 && validatePrompt.data && (
          <span
            className={
```

```
            validatePrompt.data.valid
              ? 'text-green-500'
              : 'text-red-500'
          }
        >
          {validatePrompt.data.valid ? '' : ''}
        </span>
      )}
    </div>
  )}
 </div>
 {prompt.length > 0 &&
  !validatePrompt.isPending &&
  validatePrompt.data &&
  !validatePrompt.data.valid && (
    <span className="text-red-500 text-sm">
      {validatePrompt.data.error}
    </span>
  )}
</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
          disabled={addPrompt.isPending}
          onClick={submit}
          className="w-32"
        >
          Submit
        </Button>
      </div>
    </div>
  </Modal>
 );
};

export default AddPromptModal;
```