```
'use client';

import { createContext, useContext, useState } from 'react';
import { CodeEditorProps, EditorType, LanguageType } from '../code-editor/type';

export enum THEMES {
  LIGHT = 'light',
  DARK = 'dark',
  SYSTEM = 'system',
}

export const themes: THEMES[] = [THEMES.LIGHT, THEMES.DARK];

export const SINGLE = 'single';

export const themeOptions = [
  { label: 'Single theme', value: SINGLE },
  { label: 'Sync with your system', value: THEMES.SYSTEM },
] as const;

export const StrokeColor = {
  DARK: '#00000068',
  LIGHT: '#ffffff',
};

export const EditorContext = createContext<CodeEditorProps>({
```

```
  theme: '',
  language: { name: '', icon: '' },
  padding: '',
  model: '',
  codeValue: '',
});

const languages: LanguageType[] = [
  { name: 'JavaScript', icon: 'js-icon' },
  { name: 'TypeScript', icon: 'ts-icon' },
  { name: 'Python', icon: 'py-icon' },
  // Add other languages as needed
];



export default function EditorProvider({
  children,
  model,
}: {
  children: React.ReactNode;
  model: string;
}) {
  const [language, setLanguage] = useState<LanguageType>(languages[0]);
  const [theme, setTheme] = useState<string>(themes[0]);
  const paddings = ['0', '4px', '8px', '16px']; // Define paddings array
  const [padding, setPadding] = useState(paddings[2]);
```

```
const [codeValue, setCodeValue] = useState<string>('');

function handleCodeValueChange(newCode: string) {
  setCodeValue(newCode);
}

async function handleLanguageChange(newLanguage: LanguageType) {
  setLanguage(newLanguage);
}

function handleChange(type: EditorType, newContent: LanguageType | string) {
  switch (type) {
    case EditorType.language: {
      return handleLanguageChange(newContent as LanguageType);
    }
    case EditorType.theme: {
      return setTheme(newContent as string);
    }
    case EditorType.padding: {
      return setPadding(newContent as string);
    }
    default:
      throw new Error(`${type} is invalid`);
  }
}
return (
```

```
    <EditorContext.Provider
      value={{
        theme,
        model,
        language,
        padding,
        codeValue,
        setCodeValue,
        setLanguage,
        handleChange,
        handleCodeValueChange,
      }}
    >
      {children}
    </EditorContext.Provider>
  );
}

export const useEditorContext = () => useContext(EditorContext);
```