Домашняя работа 3: CSS-переменные, темизация сайта

Студент: Даньшин Семён
Группа: K3440

Описание задания

Задание: выполнить темизацию ранее реализованного сайта. Добавить к текущему варианту сайта дополнительную тему, в итоге должно получиться либо: светлая и тёмная с ориентиром на пользовательские настройки. Либо две кастомные темы с переключателем через JS.

Теоретическая часть

CSS-переменные (Custom Properties)

CSS-переменные (CSS Custom Properties) - это механизм, позволяющий определять переиспользуемые значения в CSS. Они объявляются с префиксом `--` и используются через функцию `var()`.

Синтаксис:

```
:root {
  --primary-color: #3b82f6;
  --text-color: #1f2937;
  --spacing: 1rem;
}

.element {
  color: var(--text-color);
  padding: var(--spacing);
  background: var(--primary-color);
}
```

Преимущества CSS-переменных:

1. **Переиспользование** - определяем значение один раз, используем многократно
2. **Динамическое изменение** - можно менять через JavaScript
3. **Каскадирование** - переменные наследуются по DOM-дереву
4. **Темизация** - легко реализовать несколько тем
5. **Адаптивность** - можно переопределять для разных media queries

Темизация через CSS-переменные

Темизация - это возможность изменять внешний вид приложения, переключаясь между различными цветовыми схемами (темами). Классический пример - светлая и тёмная темы.

Основные подходы:

1. **Переключение класса на корневом элементе**

```
:root {
  --background: white;
  --text: black;
}

.dark {
  --background: black;
  --text: white;
}
```

2. **Media query для автоматического определения**

```
@media (prefers-color-scheme: dark) {
  :root {
    --background: black;
    --text: white;
  }
}
```

Практическая часть: Темизация в проекте BankingThing

1. Архитектура темизации

Проект BankingThing использует библиотеку `next-themes` для управления темами и `HeroUI` как UI-фреймворк с встроенной поддержкой темизации.

Providers.tsx - настройка темизации

```
"use client";

import type { ThemeProviderProps } from "next-themes";
import * as React from "react";
import { HeroUIProvider } from "@heroui/system";
import { ToastProvider } from "@heroui/toast";
import { useRouter } from "next/navigation";
import { ThemeProvider as NextThemesProvider } from "next-themes";

export interface ProvidersProps {
  children: React.ReactNode;
  themeProps?: ThemeProviderProps;
}

export function Providers({ children, themeProps }: ProvidersProps) {
```

```
  const router = useRouter();

  return (
    <HeroUIProvider navigate={router.push}>
        <ToastProvider placement="top-right"/>
        <NextThemesProvider {...themeProps}>{children}
</NextThemesProvider>
    </HeroUIProvider>
  );
}
```

Ключевые моменты:

- `NextThemesProvider` - обеспечивает контекст темы для всего приложения
- `HeroUIProvider` - интегрирует UI-библиотеку с темизацией
- `themeProps` - позволяет настраивать параметры темы

Layout.tsx - конфигурация темы

```
export const viewport: Viewport = {
  themeColor: [
    { media: "(prefers-color-scheme: light)", color: "white" },
    { media: "(prefers-color-scheme: dark)", color: "black" },
  ],
  userScalable: false,
};

export default function RootLayout({
  children,
}: {
  children: React.ReactNode;
}) {
  return (
    <html suppressHydrationWarning lang="en">
      <head />
      <body
        className={clsx(
          "h-screen text-foreground bg-background font-sans antialiased",
          fontSans.variable,
        )}
      >
        <Providers themeProps={{ attribute: "class", defaultTheme: "dark"
}}>
          {/* ... */}
        </Providers>
      </body>
    </html>
  );
}
```

Конфигурация темы:

- `attribute: "class"` - тема переключается через класс на элементе `<html>`
- `defaultTheme: "dark"` - темная тема по умолчанию
- `suppressHydrationWarning` - предотвращает предупреждения при SSR
- `themeColor` - цвет для браузера в зависимости от темы

2. Переключатель темы

ThemeSwitch.tsx

```
"use client";

import { FC } from "react";
import { VisuallyHidden } from "@react-aria/visually-hidden";
import { SwitchProps, useSwitch } from "@heroui/switch";
import { useTheme } from "next-themes";
import { useIsSSR } from "@react-aria/ssr";
import clsx from "clsx";
import { SunFilledIcon, MoonFilledIcon } from "@/components/icons";

export const ThemeSwitch: FC<ThemeSwitchProps> = ({
  className,
  classNames,
}) => {
  const { theme, setTheme } = useTheme();
  const isSSR = useIsSSR();

  const onChange = () => {
    theme === "light" ? setTheme("dark") : setTheme("light");
  };

  const {
    Component,
    slots,
    isSelected,
    getBaseProps,
    getInputProps,
    getWrapperProps,
  } = useSwitch({
    isSelected: theme === "light" || isSSR,
    "aria-label": `Switch to ${theme === "light" || isSSR ? "dark" :
"light"} mode`,
    onChange,
  });

  return (
    <Component {...getBaseProps({...})}>
      <VisuallyHidden>
        <input {...getInputProps()} />
      </VisuallyHidden>
      <div {...getWrapperProps()}>
```

```
          {!isSelected || isSSR ? (
            <SunFilledIcon size={22} />
          ) : (
            <MoonFilledIcon size={22} />
          )}
        </div>
      </Component>
    );
  };
```

Функциональность:

- `useTheme()` - хук из next-themes для работы с текущей темой
- `setTheme()` - функция переключения темы
- `isSSR` - проверка серверного рендеринга для корректной гидратации
- Иконки солнца и луны для визуального представления темы
- `VisuallyHidden` - делает input доступным для screen readers

3. Использование CSS-переменных в компонентах

colorUtils.ts - работа с цветами темы

```
// HSL color strings for the recharts Pie component, which requires direct
color values.
export const CHART_COLORS = [
  "hsl(var(--heroui-primary-400))",
  "hsl(var(--heroui-success-400))",
  "hsl(var(--heroui-warning-400))",
  "hsl(var(--heroui-secondary-400))",
  "hsl(var(--heroui-danger-400))",
  "hsl(var(--heroui-primary-300))",
  "hsl(var(--heroui-success-300))",
  "hsl(var(--heroui-warning-300))",
  "hsl(var(--heroui-secondary-300))",
  "hsl(var(--heroui-danger-300))",
];

// Special color for the "Other" category
const OTHER_CATEGORY_CHART_COLOR = "hsl(var(--heroui-default-400))";
const OTHER_CATEGORY_SEMANTIC_COLOR = "default";
```

Использование CSS-переменных:

- `var(--heroui-primary-400)` - CSS-переменная для primary цвета
- `hsl()` - цвета определены в HSL формате для лучшей темизации
- Цвета автоматически меняются при переключении темы
- Переменные определены в HeroUI и адаптируются под светлую/темную тему

4. Tailwind CSS и темизация

tailwind.config.js

```
import {heroui} from "@heroui/theme"

const config = {
  content: [
    './components/**/*.{js,ts,jsx,tsx,mdx}',
    './app/**/*.{js,ts,jsx,tsx,mdx}',
    "./node_modules/@heroui/theme/dist/**/*.{js,ts,jsx,tsx}"
  ],
  theme: {
    extend: {
      fontFamily: {
        sans: ["var(--font-sans)"],
        mono: ["var(--font-mono)"],
      },
    },
  },
  darkMode: "class",
  plugins: [heroui()],
}
```

Конфигурация:

- `darkMode: "class"` - темная тема активируется через класс
- `heroui()` - плагин, добавляющий CSS-переменные для темизации
- `var(--font-sans)`, `var(--font-mono)` - CSS-переменные для шрифтов

5. Примеры использования в компонентах

Пример 1: Классы Tailwind с автоматической темизацией

```
<body
  className={clsx(
    "h-screen text-foreground bg-background font-sans antialiased",
    fontSans.variable,
  )}
>
```

Утилитные классы Tailwind:

- `text-foreground` - цвет текста, автоматически адаптируется под тему
- `bg-background` - фон, меняется в зависимости от темы
- Эти классы используют CSS-переменные под капотом

Пример 2: Семантические цвета

```
<Chip color={isAllowed ? "success" : "danger"} variant="flat">
  {isAllowed ? "предоставлен" : "не предоставлен"}
</Chip>
```

Семантические цвета HeroUI:

- `success`, `danger`, `primary`, `secondary`, `warning`
- Каждый цвет имеет свою палитру (100-900)
- Автоматически адаптируются под светлую/темную тему

Пример 3: Динамические цвета в графиках

```
<div className="flex flex-col items-center">
  {Icon && <Icon className={'w-6 h-6'} />}
  <span className={'text-[10px]'}>{item.label}</span>
</div>
```

Цвета иконок и текста автоматически меняются благодаря наследованию CSS-переменных.

Преимущества реализованной темизации

1. Автоматическое определение темы

Приложение может автоматически определять предпочтения пользователя через media query:

```
themeColor: [
  { media: "(prefers-color-scheme: light)", color: "white" },
  { media: "(prefers-color-scheme: dark)", color: "black" },
]
```

2. Сохранение выбора пользователя

`next-themes` автоматически сохраняет выбор темы в localStorage:

- При следующем посещении применяется сохранённая тема
- Избегает "вспышки" неправильной темы при загрузке

3. SSR-совместимость

Правильная обработка серверного рендеринга:

- `suppressHydrationWarning` - предотвращает ошибки гидратации
- `useIsSSR` - корректная работа на сервере и клиенте

4. Полная интеграция с UI-компонентами

Все компоненты HeroUI и Tailwind автоматически поддерживают темизацию без дополнительной настройки.

5. Масштабируемость

Легко добавить дополнительные темы:

```
<NextThemesProvider
  themes={['light', 'dark', 'ocean', 'forest']}
  defaultTheme="dark"
>
  {children}
</NextThemesProvider>
```

Технические детали реализации

CSS-переменные HeroUI

HeroUI определяет следующие основные переменные:

```
:root {
  /* Основные цвета */
  --heroui-primary: ...;
  --heroui-secondary: ...;
  --heroui-success: ...;
  --heroui-warning: ...;
  --heroui-danger: ...;

  /* Фоновые цвета */
  --heroui-background: ...;
  --heroui-foreground: ...;
  --heroui-content1: ...;
  --heroui-content2: ...;

  /* Семантические цвета */
  --heroui-default: ...;
  --heroui-divider: ...;
  --heroui-focus: ...;
}

.dark {
  /* Переопределение для темной темы */
  --heroui-background: ...;
  --heroui-foreground: ...;
  /* ... */
}
```

Цветовая палитра

Каждый цвет имеет градации от 50 до 900:

```
--heroui-primary-50: ...;
--heroui-primary-100: ...;
/* ... */
--heroui-primary-900: ...;
```

Это позволяет создавать тонкие цветовые вариации в интерфейсе.

Заключение

Проект BankingThing демонстрирует современный подход к темизации веб-приложений:

1. **CSS-переменные** используются для всех цветов и основных значений
2. **next-themes** обеспечивает удобное управление темами
3. **HeroUI** предоставляет готовую систему CSS-переменных
4. **Tailwind CSS** интегрируется с системой темизации
5. **Автоматическое определение** предпочтений пользователя
6. **SSR-совместимость** для корректной работы с Next.js

Темизация через CSS-переменные - это мощный и гибкий инструмент, который позволяет:

- Легко поддерживать множество тем
- Обеспечивать консистентность дизайна
- Улучшать пользовательский опыт
- Снижать количество повторяющегося кода

Реализация в проекте является best practice для современных React-приложений и может служить примером для других проектов.