```
// @ts-nocheck

// MIT License

// Copyright (c) 2021 Pine Wu


// Permission is hereby granted, free of charge, to any person obtaining a copy

// of this software and associated documentation files (the "Software"), to dea

// in the Software without restriction, including without limitation the rights
// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
// copies of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:


// The above copyright notice and this permission notice shall be included in a
// copies or substantial portions of the Software.

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM

// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN TH
// SOFTWARE.

// @filename: renderer.ts
// source: https://github.com/shikijs/shiki/blob/ca5a6c60a255fb390776ada849e01
4a916fc/packages/shiki/src/renderer.ts

import { FontStyle } from './stackElementMetadata'
import { IThemedToken } from './themedTokenizer'

export interface HtmlRendererOptions {
  langId?: string
  fg?: string
  bg?: string
}

export function renderToHtml(
  lines: IThemedToken[][],
  options: HtmlRendererOptions = {}
) {
  const bg = options.bg || '#fff'

  let html = ''

  html += `<pre class="shiki" style="background-color: ${bg}">`
  if (options.langId) {
    html += `<div class="language-id">${options.langId}</div>`
  }
  html += `<code>`

  lines.forEach((l: IThemedToken[]) => {
    html += `<span class="line">`

    l.forEach((token) => {
      const cssDeclarations = [`color: ${token.color || options.fg}`]
      if (token.fontStyle & FontStyle.Italic) {
        cssDeclarations.push('font-style: italic')
      }
      if (token.fontStyle & FontStyle.Bold) {
        cssDeclarations.push('font-weight: bold')
      }
      if (token.fontStyle & FontStyle.Underline) {
        cssDeclarations.push('text-decoration: underline')
```

```ts
      }
      html += `<span style="${cssDeclarations.join('; ')}">${escapeHtml(
        token.content
      )}</span>`
    })
    html += `</span>\n`
  })
  html = html.replace(/\n*$/, '') // Get rid of final new lines
  html += `</code></pre>`

  return html
}

const htmlEscapes = {
  '&': '&amp;',
  '<': '&lt;',
  '>': '&gt;',
  '"': '&quot;',
  "'": '&#39;',
}

function escapeHtml(html: string) {
  return html.replace(/[&<>"']/g, (chr) => htmlEscapes[chr])
}

// @filename: highlighter.ts
// source: https://github.com/shikijs/shiki/blob/ca5a6c60a255fb390776ada849e01
4a916fc/packages/shiki/src/highlighter.ts

import type {
  Highlighter,
  HighlighterOptions,
  ILanguageRegistration,
  IShikiTheme,
  IThemeRegistration,
  StringLiteralUnion,
} from './types'
import { Resolver } from './resolver'
import { tokenizeWithTheme } from './themedTokenizer'
import { renderToHtml } from './renderer'

import { getOnigasm } from './loader'
import { Lang, languages as BUNDLED_LANGUAGES } from './languages'
import { Registry } from './registry'
import { Theme } from './themes'

function resolveLang(lang: ILanguageRegistration | Lang) {
  return typeof lang === 'string'
    ? BUNDLED_LANGUAGES.find((l) => l.id === lang || l.aliases?.includes(lang))
    : lang
}

function resolveOptions(options: HighlighterOptions) {
  let _languages: ILanguageRegistration[] = BUNDLED_LANGUAGES
  let _themes: IThemeRegistration[] = options.themes || []

  if (options.langs?.length) {
    _languages = options.langs.map(resolveLang)
  }
  if (options.theme) {
    _themes.unshift(options.theme)
  }
  if (!_themes.length) {
    _themes = ['nord']
  }

  return { _languages, _themes }
}

```

```
export async function getHighlighter(
  options: HighlighterOptions
): Promise<Highlighter> {
  const { _languages, _themes } = resolveOptions(options)
  const _resolver = new Resolver(getOnigasm(), 'onigasm')
  const _registry = new Registry(_resolver)

  if (options.paths?.themes) {
    _registry.themesPath = options.paths.themes
  }

  if (options.paths?.languages) {
    _resolver.languagesPath = options.paths.languages
  }

  const themes = await _registry.loadThemes(_themes)
  const _defaultTheme = themes[0]
  let _currentTheme: IShikiTheme | undefined
  await _registry.loadLanguages(_languages)

  /**

   * Shiki was designed for VS Code, so CSS variables are not currently suppor
   * See: https://github.com/shikijs/shiki/pull/212#issuecomment-906924986
   *

   * Instead, we work around this by using valid hex color codes as lookups in
   * final "repair" step which translates those codes to the correct CSS varia
.

   */
  const COLOR_REPLACEMENTS: Record<string, string> = {
    '#000001': 'var(--shiki-color-text)',
    '#000002': 'var(--shiki-color-background)',
    '#000004': 'var(--shiki-token-constant)',
    '#000005': 'var(--shiki-token-string)',
    '#000006': 'var(--shiki-token-comment)',
    '#000007': 'var(--shiki-token-keyword)',
    '#000008': 'var(--shiki-token-parameter)',
    '#000009': 'var(--shiki-token-function)',
    '#000010': 'var(--shiki-token-string-expression)',
    '#000011': 'var(--shiki-token-punctuation)',
    '#000012': 'var(--shiki-token-link)',
  }

  function fixCssVariablesTheme(theme: IShikiTheme, colorMap: string[]) {
    theme.bg = COLOR_REPLACEMENTS[theme.bg] || theme.bg
    theme.fg = COLOR_REPLACEMENTS[theme.fg] || theme.fg
    colorMap.forEach((val, i) => {
      colorMap[i] = COLOR_REPLACEMENTS[val] || val
    })
  }

  function getTheme(theme?: IThemeRegistration) {
    const _theme = theme ? _registry.getTheme(theme) : _defaultTheme
    if (!_theme) {
      throw Error(`No theme registration for ${theme}`)
    }
    if (!_currentTheme || _currentTheme.name !== _theme.name) {
      _registry.setTheme(_theme)
      _currentTheme = _theme
    }
    const _colorMap = _registry.getColorMap()
    if (_theme.name === 'css-variables') {
      fixCssVariablesTheme(_theme, _colorMap)
    }
    return { _theme, _colorMap }
  }

  function getGrammer(lang: string) {
    const _grammer = _registry.getGrammer(lang)
    if (!_grammer) {
      throw Error(`No language registration for ${lang}`)
    }
    return { _grammer }
  }

  function codeToThemedTokens(
    code: string,
    lang = 'text',
    theme?: StringLiteralUnion<Theme>,
    options = { includeExplanation: true }
  ) {
    if (isPlaintext(lang)) {
      return [[{ content: code }]]
    }
    const { _grammer } = getGrammer(lang)
    const { _theme, _colorMap } = getTheme(theme)
    return tokenizeWithTheme(_theme, _colorMap, code, _grammer, options)
  }

  function codeToHtml(
    code: string,
    lang = 'text',
    theme?: StringLiteralUnion<Theme>
  ) {
    const tokens = codeToThemedTokens(code, lang, theme, {
      includeExplanation: false,
    })
    const { _theme } = getTheme(theme)
    return renderToHtml(tokens, {
      fg: _theme.fg,
      bg: _theme.bg,
    })
  }

  async function loadTheme(theme: IShikiTheme | Theme) {
    await _registry.loadTheme(theme)
  }

  async function loadLanguage(lang: ILanguageRegistration | Lang) {
    const _lang = resolveLang(lang)
    _resolver.addLanguage(_lang)
    await _registry.loadLanguage(_lang)
  }

  function getLoadedThemes() {
    return _registry.getLoadedThemes()
  }

  function getLoadedLanguages() {
    return _registry.getLoadedLanguages()
  }

  function getBackgroundColor(theme?: StringLiteralUnion<Theme>) {
    const { _theme } = getTheme(theme)
    return _theme.bg
  }

  function getForegroundColor(theme?: StringLiteralUnion<Theme>) {
    const { _theme } = getTheme(theme)
    return _theme.fg
  }

  return {
    codeToThemedTokens,
    codeToHtml,
    getTheme: (theme: IThemeRegistration) => {
      return getTheme(theme)._theme
    },
    loadTheme,
    loadLanguage,
    getBackgroundColor,
    getForegroundColor,
    getLoadedThemes,
    getLoadedLanguages,
  }
}

function isPlaintext(lang: string | null | undefined) {
  return !lang || ['plaintext', 'txt', 'text'].includes(lang)
}

```