```
// @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 deal
// 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 all
// 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 THE
// SOFTWARE.

// @filename: renderer.ts
// source: https://github.com/shikijs/shiki/blob/ca5a6c60a255fb390776ada849e01e1744a916fc/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')
```

```
      }
      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/ca5a6c60a255fb390776ada849e01e1744a916fc/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 suppo
rted.
   * See: https://github.com/shikijs/shiki/pull/212#issuecomment-906924986
   *
   * Instead, we work around this by using valid hex color codes as lookups i
n a
   * final "repair" step which translates those codes to the correct CSS vari
ables.
   */
  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) {
```

```
198       throw Error(`No language registration for ${lang}`)
199     }
200     return { _grammer }
201   }
202
203   function codeToThemedTokens(
204     code: string,
205     lang = 'text',
206     theme?: StringLiteralUnion<Theme>,
207     options = { includeExplanation: true }
208   ) {
209     if (isPlaintext(lang)) {
210       return [[{ content: code }]]
211     }
212     const { _grammer } = getGrammer(lang)
213     const { _theme, _colorMap } = getTheme(theme)
214     return tokenizeWithTheme(_theme, _colorMap, code, _grammer, options)
215   }
216
217   function codeToHtml(
218     code: string,
219     lang = 'text',
220     theme?: StringLiteralUnion<Theme>
221   ) {
222     const tokens = codeToThemedTokens(code, lang, theme, {
223       includeExplanation: false,
224     })
225     const { _theme } = getTheme(theme)
226     return renderToHtml(tokens, {
227       fg: _theme.fg,
228       bg: _theme.bg,
229     })
230   }
231
232   async function loadTheme(theme: IShikiTheme | Theme) {
233     await _registry.loadTheme(theme)
234   }
235
236   async function loadLanguage(lang: ILanguageRegistration | Lang) {
237     const _lang = resolveLang(lang)
238     _resolver.addLanguage(_lang)
239     await _registry.loadLanguage(_lang)
240   }
241
242   function getLoadedThemes() {
243     return _registry.getLoadedThemes()
244   }
245
246   function getLoadedLanguages() {
247     return _registry.getLoadedLanguages()
248   }
249
250   function getBackgroundColor(theme?: StringLiteralUnion<Theme>) {
251     const { _theme } = getTheme(theme)
252     return _theme.bg
253   }
254
255   function getForegroundColor(theme?: StringLiteralUnion<Theme>) {
256     const { _theme } = getTheme(theme)
257     return _theme.fg
258   }
259
260   return {
261     codeToThemedTokens,
262     codeToHtml,
263     getTheme: (theme: IThemeRegistration) => {
264       return getTheme(theme)._theme
265     },
266     loadTheme,
267     loadLanguage,
```

```
    getBackgroundColor,
    getForegroundColor,
    getLoadedThemes,
    getLoadedLanguages,
  }
}

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

// @filename: random
const ReallyReallyLongFileName =
  // prettier-ignore
  'really-really-really-really-really-really-really-really-really-really-real
ly-really-really-really-really-really-really-really-really-really-really-real
ly-really-really-really-really-really-really-really-really-really-really-real
ly-really-really-really-really-really-really-really-really-really-really-real
ly-really-really-really-really-really-really-really-really-really-really-real
ly-really-really-really-really-really-really-really-really-really-really-real
ly-really-really-really-really-really-really-really-really-really-really-real
ly-really-really-really-rea' + 'lly-really-really-really-really-really-really
-really-really-really-really-really-really-really-really-really-really-really
-real' + 'ly-really-really-really-really-really-really-really-really-really-r
eally-really-really-really-really-really-really-really-really-really-really-r
eally-really-really-really-really-really-really-really-really-really-really-r
eally-really-really-really-really-really-really-really-really-really-really-r
eally-really-really-really-really-really-really-really-really-really-really-r
eally-really-really-really-really-really-really-really-really-really-really-r
eally-really-really-really-really-really-really-really-really-really-really-r
eally-really-really-really-really-really-really-really-really-really-really-r
eally-really-really-really-really-really-really-really-really-really-really-l
ong-file-name.pdf'
```