```
/*---------------------------------------------------------------------------------------------
 *  Copyright (c) Microsoft Corporation. All rights reserved.
 *  Licensed under the MIT License. See License.txt in the project root for license information.
 *--------------------------------------------------------------------------------------------*/

import 'vs/css!./linesDecorations';
import { DecorationToRender, DedupOverlay } from 'vs/editor/browser/viewParts/glyphMargin/glyphMargin';
import { RenderingContext } from 'vs/editor/browser/view/renderingContext';
import { ViewContext } from 'vs/editor/common/viewModel/viewContext';
import * as viewEvents from 'vs/editor/common/viewEvents';
import { EditorOption } from 'vs/editor/common/config/editorOptions';


export class LinesDecorationsOverlay extends DedupOverlay {

  private readonly _context: ViewContext;

  private _decorationsLeft: number;
  private _decorationsWidth: number;
  private _renderResult: string[] | null;

  constructor(context: ViewContext) {
    super();
    this._context = context;
    const options = this._context.configuration.options;
    const layoutInfo = options.get(EditorOption.layoutInfo);
    this._decorationsLeft = layoutInfo.decorationsLeft;
    this._decorationsWidth = layoutInfo.decorationsWidth;
    this._renderResult = null;
    this._context.addEventHandler(this);
  }

  public override dispose(): void {
    this._context.removeEventHandler(this);
    this._renderResult = null;
    super.dispose();
  }

  // --- begin event handlers

  public override onConfigurationChanged(e: viewEvents.ViewConfigurationChangedEvent): boolean {
    const options = this._context.configuration.options;
    const layoutInfo = options.get(EditorOption.layoutInfo);
    this._decorationsLeft = layoutInfo.decorationsLeft;
    this._decorationsWidth = layoutInfo.decorationsWidth;
    return true;
  }
  public override onDecorationsChanged(e: viewEvents.ViewDecorationsChangedEvent): boolean {
    return true;
  }
  public override onFlushed(e: viewEvents.ViewFlushedEvent): boolean {
    return true;
  }
  public override onLinesChanged(e: viewEvents.ViewLinesChangedEvent): boolean {
    return true;
  }
  public override onLinesDeleted(e: viewEvents.ViewLinesDeletedEvent): boolean {
    return true;
  }
  public override onLinesInserted(e: viewEvents.ViewLinesInsertedEvent): boolean {
    return true;
  }
  public override onScrollChanged(e: viewEvents.ViewScrollChangedEvent): boolean {
    return e.scrollTopChanged;
  }
  public override onZonesChanged(e: viewEvents.ViewZonesChangedEvent): boolean {
    return true;
```

```
  }

  // --- end event handlers

  protected _getDecorations(ctx: RenderingContext): DecorationToRender[] {
    const decorations = ctx.getDecorationsInViewport();
    const r: DecorationToRender[] = [];
    let rLen = 0;
    for (let i = 0, len = decorations.length; i < len; i++) {
      const d = decorations[i];
      const linesDecorationsClassName = d.options.linesDecorationsClassName;
      const zIndex = d.options.zIndex;
      if (linesDecorationsClassName) {
        r[rLen++] = new DecorationToRender(d.range.startLineNumber, d.range.endLineNumber, linesDecorationsClassName, zIndex);
      }
      const firstLineDecorationClassName = d.options.firstLineDecorationClassName;
      if (firstLineDecorationClassName) {
        r[rLen++] = new DecorationToRender(d.range.startLineNumber, d.range.startLineNumber, firstLineDecorationClassName, zIn
      }
    }
      return r;
  }

  public prepareRender(ctx: RenderingContext): void {
    const visibleStartLineNumber = ctx.visibleRange.startLineNumber;
    const visibleEndLineNumber = ctx.visibleRange.endLineNumber;
    const toRender = this._render(visibleStartLineNumber, visibleEndLineNumber, this._getDecorations(ctx));

    const left = this._decorationsLeft.toString();
    const width = this._decorationsWidth.toString();
    const common = '" style="left:' + left + 'px;width:' + width + 'px;"></div>';

    const output: string[] = [];
    for (let lineNumber = visibleStartLineNumber; lineNumber <= visibleEndLineNumber; lineNumber++) {
      const lineIndex = lineNumber - visibleStartLineNumber;
      const decorations = toRender[lineIndex].getDecorations();
      let lineOutput = '';
      for (const decoration of decorations) {
        lineOutput += '<div class="cldr ' + decoration.className + common;
      }
      output[lineIndex] = lineOutput;
    }

    this._renderResult = output;
  }

  public render(startLineNumber: number, lineNumber: number): string {
    if (!this._renderResult) {
      return '';
    }
      return this._renderResult[lineNumber - startLineNumber];
  }
}
```