readme.md for @uptime.link/webwidget

@uptimelink/webwidget

the webwidget for public use of uptimelink

Install

To install the `@uptimelink/webwidget` package, you need to have Node.js and npm (or yarn) installed. Once you have these prerequisites, you can install the package via npm by running the following command in your terminal:

```
npm install @uptimelink/webwidget
```

Alternatively, if you use yarn, you can run:

```
yarn add @uptimelink/webwidget
```

Usage

Introduction

The `@uptimelink/webwidget` package provides a web component that can be embedded into web pages to display uptime information for a given project on the UptimeLink platform. The component is implemented using TypeScript and leverages modern web standards including Web Components and LitElement.

Basic Setup

First, you will need to import the `UptimelinkWebwidget` class and define it in your project. To do this, create an HTML file and include a script to register the web component.

```
// index.ts
import { UptimelinkWebwidget } from '@uptimelink/webwidget';

// Append the webwidget component to the document body
document.body.appendChild(UptimelinkWebwidget.demo());
```

Setting Up in HTML

To use the widget in an HTML document, you will need to include the built JavaScript file. Here is an example:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Uptime Link Widget</title>
  </head>
  <body>
    <script type="module">
      import { UptimelinkWebwidget } from './path_to_your_built_index.js';
      customElements.define('uptimelink-webwidget', UptimelinkWebwidget);

      const widgetElement = document.createElement('uptimelink-webwidget');
      widgetElement.projectSlug = 'uptime.link';
      document.body.appendChild(widgetElement);
    </script>
  </body>
</html>
```

Creating a Custom Page with the Widget

You can integrate the widget into a custom page to display uptime status. First, create a new TypeScript file for the page.

```
// pages/customPage.ts
import { html } from '@design.estate/dees-element';

export const customPage = () => html`
  <style>
    .container {
      box-sizing: border-box;
      position: absolute;
      top: 20px;
      height: 120px;
      padding: 40px;
      width: 100%;
      background: rgba(0, 0, 0, 0.2);
    }
  </style>
  <div class="container">
    <uptimelink-webwidget
      projectSlug="custom-project-slug"
    ></uptimelink-webwidget>
  </div>
`;
```

Styling the Widget

The `UptimelinkWebwidget` component comes with default styles but you can override those styles to match your site's aesthetics. Below is an example of how this can be done:

```
// customStyles.ts
import { UptimelinkWebwidget } from '@uptimelink/webwidget';
import { cssManager } from '@design.estate/dees-element';

UptimelinkWebwidget.styles = [
  cssManager.defaultStyles,
  css`
    .mainbox {
      background-color: #f0f0f0;
```

```
      color: #333;
      border: 1px solid #ccc;
    }
    .statusindicator {
      background: #28a745;
    }
  `,
];

document.body.appendChild(UptimelinkWebwidget.demo());
```

Advanced Interactions

The `UptimelinkWebwidget` allows for advanced interactions such as hovering effects to show detailed information. This is handled within the component's lifecycle and event handlers. Below is a detailed code snippet demonstrating lifecycle hooks and event listeners:

```
// uppimelink-webwidget.ts
import {
  DeesElement,
  property,
  html,
  customElement,
  type TemplateResult,
  cssManager,
} from '@design.estate/dees-element';
import { DeesWindowLayer } from '@design.estate/dees-catalog';

@customElement('uptimelink-webwidget')
export class UptimelinkWebwidget extends DeesElement {
  @property({ type: Boolean }) isOnTop = false;
  @property() public projectSlug: string;
  @property() public isFocused = false;
  @property() public isElevated = false;
  @property() public showExpanded: boolean = false;

  constructor() {
    super();
    this.setupEventing();
```

```
}

public static styles = [cssManager.defaultStyles];

public render(): TemplateResult {
  return html`
    <style>
      /* Add your custom styles here */
    </style>
    <div class="mainbox ${this.isFocused ? 'focused' : null}">
      <div class="firstLine">
        <div class="statusindicator"></div>
        <div class="statustext">All systems are up!</div>
      </div>
      ${this.showExpanded
        ? html` <div class="expanded">/* Expanded view content */</div> `
        : null}
    </div>
  `;
}

private async setupEventing() {
  const domtools = await this.domtoolsPromise;
  await this.updateComplete;
  const mainbox = this.shadowRoot.querySelector('.mainbox') as HTMLDivElement;

  mainbox.onmouseenter = async () => {
    if (!this.isOnTop) {
      const rect = mainbox.getBoundingClientRect();
      const uptimelinkWidget = new UptimelinkWebwidget();
      uptimelinkWidget.isOnTop = true;
      uptimelinkWidget.style.position = 'fixed';
      uptimelinkWidget.style.top = `${rect.top}px`;
      uptimelinkWidget.style.left = `${rect.left}px`;
      document.body.append(uptimelinkWidget);
      return;
    }
    this.isElevated = true;
    this.isFocused = true;
```

```
      this.windowLayer = await DeesWindowLayer.createAndShow({ blur: true });
      await domtools.convenience.smartdelay.delayFor(200);
      if (!this.isFocused) return;
      this.showExpanded = true;
      await this.performUpdate();
      await domtools.convenience.smartdelay.delayFor(50);
      const expandedDiv = this.shadowRoot.querySelector(
        '.expanded',
      ) as HTMLElement;
      expandedDiv.style.opacity = '1';
    };

    mainbox.onmouseleave = async () => {
      if (!this.isOnTop) return;
      this.windowLayer.destroy();
      domtools.convenience.smartdelay.delayFor(200).then(() => {
        if (!this.isFocused) {
          this.isElevated = false;
          this.remove();
        }
      });
      if (!this.showExpanded) {
        this.isFocused = false;
        return;
      }
      this.showExpanded = false;
      await domtools.convenience.smartdelay.delayFor(50);
      this.isFocused = false;
    };
  }
}
```

Utilizing Multiple Widgets

You can also utilize multiple instances of the web widget on a single page. Below is an example of how you could do this:

```
// multipleWidgets.ts
import { UptimelinkWebwidget } from '@uptimelink/webwidget';
```

```
const widget1 = document.createElement('uptimelink-webwidget');
const widget2 = document.createElement('uptimelink-webwidget');

widget1.projectSlug = 'project1';
widget2.projectSlug = 'project2';

document.body.appendChild(widget1);
document.body.appendChild(widget2);
```

Building and Serving

To build the project, use the following npm scripts:

```
npm run build
```

To watch for changes and rebuild automatically:

```
npm run watch
```

Testing

Currently, the `test` script is identical to the `build` script. Running tests requires building the project:

```
npm run test
```

Contributing

Contributions to the `@uptimelink/webwidget` project are welcome! Please follow the guidelines provided in the repository. Reach out on the project issues page for discussions, questions, or follow the established process for submitting pull requests.

Conclusion

This guide covered the steps needed to install, set up, and use the `@uptimelink/webwidget` package in your TypeScript projects. We also explored various ways to customize and extend the widget, ensuring seamless integration into your existing web applications. For more detailed information,

refer to the documentation.

Feel free to check the links provided in the initial sections for more context and updates about the project's status and availability. undefined

---

Revision #3
Created 2026-03-28 11:15:33 UTC by foss.global Team
Updated 2026-03-28 12:22:15 UTC by foss.global Team