






Lexi Chat Widget

A self-contained, embeddable chat widget delivered via **iframe loader** for maximum compatibility and safety. Works across most sites and browsers without CSS/JS conflicts.

-  Isolated (Shadow DOM inside an iframe)
-  One-line embed for customers
-  No reliance on host fonts/styles
-  CSP & CORS friendly
-  Angular + standalone component

Quick start (for this repo)

```
# install\ nnp install

# develop (serves the iframe entry)
npm run serve:iframe

# build distributable
npm run build:iframe

# preview the built widget
npm run preview
```

Build output goes to `dist/widget/`.

Project layout

```
chat-widget/
├─ package.json
├─ angular.json
├─ tsconfig.json
├─ src/
│   └─ app/
│       └─ chat-widget/
│           ├─ chat-widget.component.ts      # widget logic
│           ├─ chat-widget.component.html    # widget template
│           └─ chat-widget.component.scss    # styles (Shadow DOM)
│   └─ assets/
│       └─ lexi-chat.png                    # header logo (used by HTML)
```

```

| | └─ lexi-logo.png           # round avatar (used by HTML)
| |   └─ lexi-loader.js       # tiny loader customers paste
| └─ iframe-index.html        # iframe host page
|   └─ iframe-main.ts         # iframe bootstrap & param wiring
└─ dist/
    └─ widget/                # build output for CDN

```

Note: In HTML/SCSS, image paths should be `assets/...` (not `/assets/...`) so they work from any CDN mount.

Build configuration

We ship a dedicated **iframe** build (stable filenames):

- `src/iframe-index.html` → `dist/widget/index.html`
- `src/iframe-main.ts` → `dist/widget/main.js`
- `src/assets/**` → `dist/widget/assets/**`

Scripts (in `package.json`):

```

{
  "scripts": {
    "build:iframe": "ng build --configuration iframe --output-path=dist/widget",
    "serve:iframe": "ng serve --configuration iframe",
    "preview": "npx http-server dist/widget -p 8080"
  }
}

```

Add this build configuration under your project's `architect.build.configurations` in `angular.json` (example):

```

"iframe": {
  "outputHashing": "none",
  "index": "src/iframe-index.html",
  "main": "src/iframe-main.ts",
  "polyfills": [],
  "assets": [
    "src/assets"
  ],
  "optimization": true
}

```

Deploy

Upload everything under `dist/widget/` to your CDN, e.g.:

```
https://cdn.yourdomain.com/widget/index.html
https://cdn.yourdomain.com/widget/main.js
https://cdn.yourdomain.com/widget/assets/lexi-loader.js
https://cdn.yourdomain.com/widget/assets/lexi-chat.png
https://cdn.yourdomain.com/widget/assets/lexi-logo.png
```

After deploying, update the default `data-src` (or hardcoded `src`) in `assets/lexi-loader.js` if needed.

Embedding (for your customers)

Minimal copy-paste

```
<script
  src="https://cdn.yourdomain.com/widget/assets/lexi-loader.js"
  defer
  data-src="https://cdn.yourdomain.com/widget/index.html"
  data-api-url="https://api.yourdomain.com"
  data-company-id="acme-123">
</script>
```

Full options

```
<script
  src="https://cdn.yourdomain.com/widget/assets/lexi-loader.js"
  defer
  data-src="https://cdn.yourdomain.com/widget/index.html"
  data-api-url="https://api.yourdomain.com"
  data-company-id="acme-123"
  data-theme="dark"           <!-- dark | light (default: dark) -->
  data-position="right"      <!-- right | left (default: right) -->
  data-width="380"           <!-- open width in px (default: 380) -->
  data-height="600"          <!-- open height in px (default: 600) -->
  data-title="Lexi Chat">    <!-- iframe title/ARIA label -->
</script>
```

Attributes reference

Attribute	Required	Default	Description
<code>data-src</code>	✓	—	URL to <code>index.html</code> (the iframe page)
<code>data-api-url</code>	✓	—	Backend API base URL
<code>data-company-id</code>	✓	—	Tenant/company identifier
<code>data-theme</code>	✗	<code>dark</code>	<code>dark</code> <code>light</code>
<code>data-position</code>	✗	<code>right</code>	<code>right</code> <code>left</code> (dock position)
<code>data-width</code>	✗	<code>380</code>	Open width in px
<code>data-height</code>	✗	<code>600</code>	Open height in px
<code>data-title</code>	✗	<code>Lexi Chat</code>	iFrame title/aria-label

Programmatic control (optional)

The loader exposes a tiny API:

```
<script>
  // open or close the widget programmatically
  window.lexi?.open();
  window.lexi?.close();
</script>
```

How the loader works

- Inserts a fixed `<iframe>` at bottom-right (or left) sized **64×64** (launcher).
- When the widget opens, the iframe resizes to your requested width/height.
- Communication uses `postMessage`:
- Widget → Parent: `{ type: 'lexi:ready' | 'lexi:open' | 'lexi:close' | 'lexi:size' }`
- Parent (loader) resizes using **iframe attributes** (works even with strict CSP).

In the component, `toggle()` should inform the parent:

```
window.parent?.postMessage({ type: opening ? 'lexi:open' : 'lexi:close' }, '*');
```

Fonts, icons, and assets

- **Fonts:** Use the **system UI stack** by default (no external fetch). If you prefer Inter, self-host it **inside the iframe** (e.g., `assets/fonts/...`) and link it in `iframe-index.html`. Avoid Google Fonts on customer sites (CSP/CORS).
- **Icons:** Inline SVG only (already implemented). No icon fonts.
- **Images:** Use `assets/...` paths in HTML/SCSS so they resolve relative to the deployed folder.

Security, CSP, and CORS

- **CSP (host site) must allow:**
- `script-src` for your loader domain (e.g., `https://cdn.yourdomain.com`)
- `frame-src` (or `child-src`) for the same CDN
- `connect-src` for your API (e.g., `https://api.yourdomain.com`)
- **CORS** on your API should allow the iframe origin:

```
Access-Control-Allow-Origin: https://cdn.yourdomain.com
Access-Control-Allow-Methods: POST, OPTIONS
Access-Control-Allow-Headers: content-type, authorization
```

- The loader sets a conservative `sandbox`:

```
allow-scripts allow-same-origin allow-forms
allow-popups allow-downloads allow-modals
allow-popups-to-escape-sandbox
```

Add/remove capabilities only if needed (e.g., microphone).

Browser support

- Chrome / Edge / Firefox / Safari (evergreen) 
- iOS Safari 13+ 
- Internet Explorer 

Troubleshooting

Nothing renders - Open DevTools console. If you see 404s, confirm the loader's `data-src` points to the deployed `index.html`. - Verify `main.js` loads (Network tab). CSP may be blocking `frame-src`; add your CDN to `frame-src`.

Widget opens but can't call the API - Check CORS on your API. `connect-src` of the host's CSP must include your API origin.

Images not showing - Ensure image paths are `assets/...` and deployed under `dist/widget/assets/`.

Loader inserted twice - Remove duplicate `<script>` includes; loader guards with `window.__LEXI_WIDGET__`.

No animation on open - Host may block inline styles via CSP. We resize via attributes as a fallback (works, just not animated).

Mixed content warning - All URLs (loader, iframe, API, images) must be **HTTPS**.

Versioning & CDN

- Publish to versioned paths:
 - `https://cdn.yourdomain.com/widget/v1/index.html`
 - `https://cdn.yourdomain.com/widget/v1/main.js`
 - `https://cdn.yourdomain.com/widget/v1/assets/lexi-loader.js`
 - Update customer snippets to a specific version (e.g., `/v1/`). For breaking changes, bump to `/v2/`.
-

Development tips

- Component encapsulation is `ViewEncapsulation.ShadowDom` (keeps styles isolated).
 - Keep images small and cacheable; set long CDN cache headers.
 - Prefer system fonts for zero-config embeds.
-

License

Add your license of choice here (MIT recommended for embeddable widgets).

Changelog

- **1.0.0**
- Initial public release
- iframe loader + Shadow DOM widget
- Timestamped messages, suggestion chips, modern UI