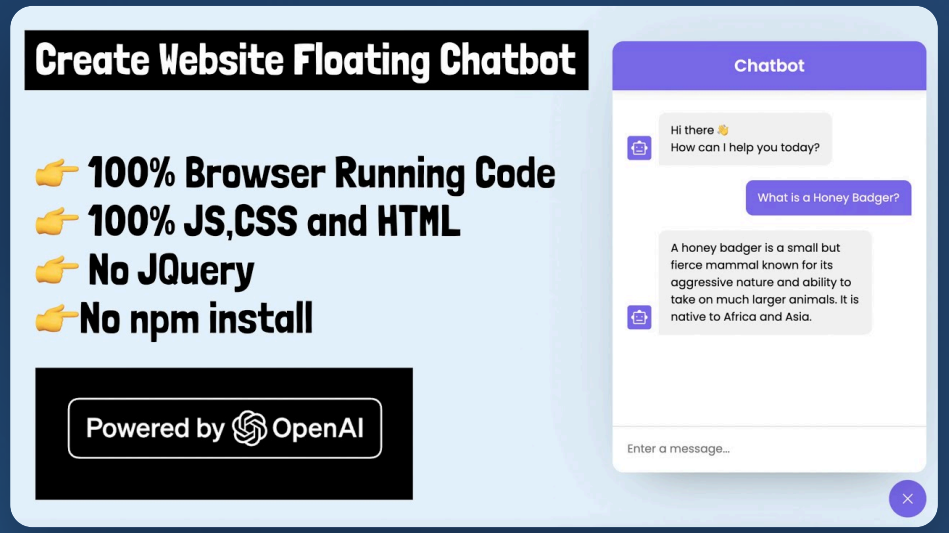
Embedding Vue 3 Chatbot as a Widget

A Comprehensive Guide to Seamless Integration



Problem & Requirements

Embedding an existing Vue 3 chatbot into external websites with seamless integration

The Challenge

You have an existing AI chatbot built with Vue 3 that serves as an agent interface. Now you need to embed this chatbot into external websites as a widget that integrates seamlessly with the host UI, similar to how tools like Intercom or Drift provide their chat widgets.

Key Requirements

Visually Compact & Expandable

Floating button or corner widget that expands when clicked

Dynamic Loading

Load on host website without impacting its performance

API Communication

Communicate with existing backend APIs for all interactions

Configuration Options

Support positioning, branding (colors/logo), and user identification (JWT/query params)

Embedding Approaches Overview

Three main approaches to embed your Vue 3 chatbot as a widget in external websites

1

Web Components

Use Vue 3's defineCustomElement API to create a custom HTML element that encapsulates your chatbot functionality with Shadow DOM for style isolation.

- Seamless integration
- Framework-agnostic
- Good encapsulation

2

UMD Bundle

Package your Vue 3 chatbot as a UMD library that can be included via script tag and initialized with a simple JavaScript function call.

- Simple implementation
- Easy to include
- Wide browser support

3

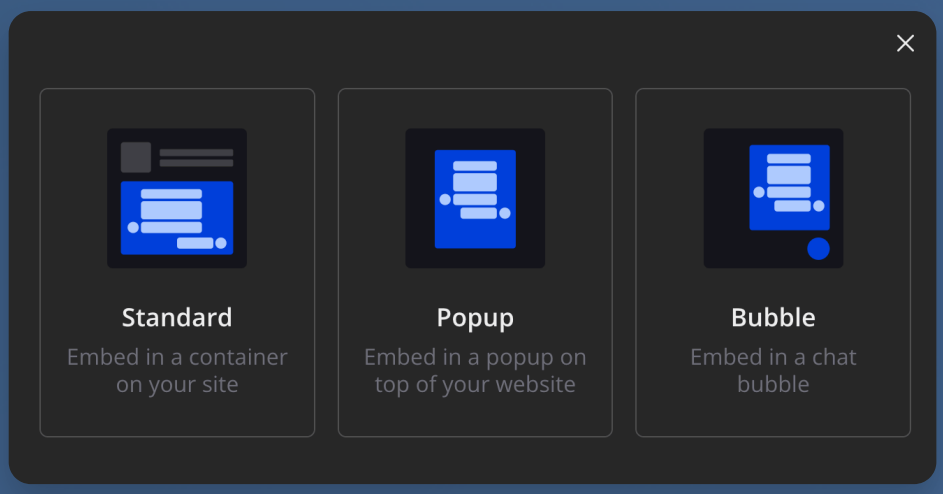
Iframe Approach

Embed your chatbot in an iframe that provides complete isolation from the host website, preventing any CSS or JavaScript conflicts.

- Complete isolation
- Security benefits
- No CSS conflicts

Visual Comparison

Each approach offers different levels of integration, encapsulation, and complexity. The right choice depends on your specific requirements for customization, security, and ease of implementation.



Web Components Approach

Using Vue 3's defineCustomElement API to create encapsulated chatbot widgets

How It Works

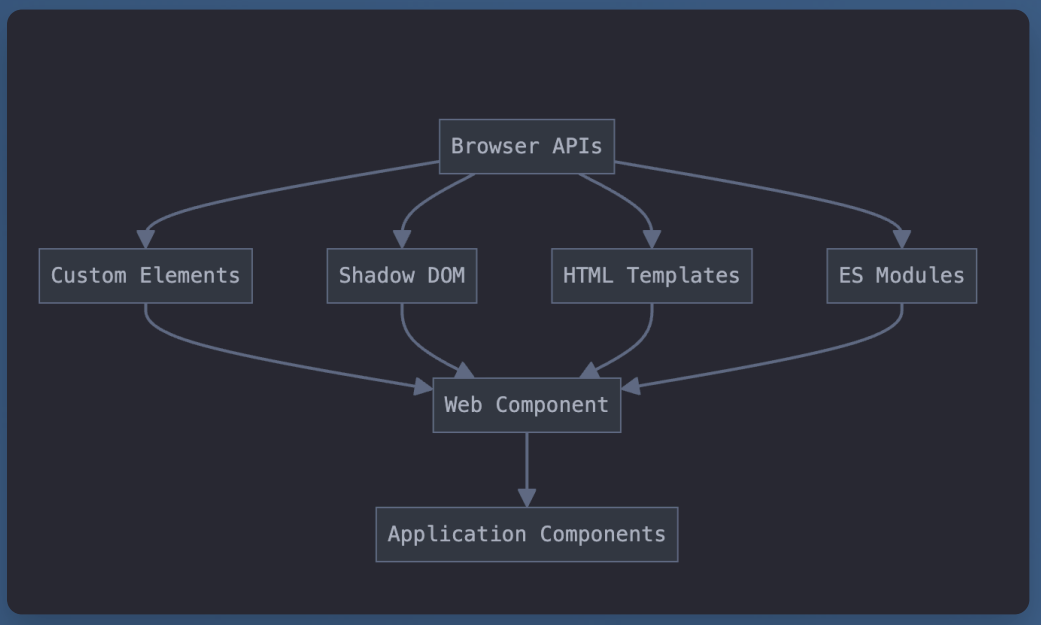
Vue 3's defineCustomElement API allows you to convert Vue components into native web components that can be used in any HTML page, regardless of the framework.

```
import { defineCustomElement } from 'vue'
import Chatbot from './Chatbot.vue'

const ChatbotElement =
defineCustomElement(Chatbot)

customElements.define('chatbot-widget',
ChatbotElement)
```

Web Components Technology Stack



Pros

- Seamless integration with host website
- Framework-agnostic compatibility
- Shadow DOM provides style encapsulation
- Native browser support without extra libraries

Cons

- More complex implementation
- State management requires adaptation
- Plugin-based solutions don't work in web components
- Limited browser support for older browsers

Key Implementation Details

Web components use Shadow DOM to encapsulate styles and prevent conflicts with the host website. Custom Elements API allows defining new HTML tags with custom behavior.

Vue 3 automatically handles props, events, and lifecycle hooks when converting components to web elements, making the integration process smoother.

UMD Bundle & Iframe Approaches

Alternative methods for embedding your Vue 3 chatbot widget

UMD Bundle

Package your Vue 3 chatbot as a UMD library that can be included via script tag and initialized with a JavaScript function.

Pros

- Simple implementation
- Easy to include with script tag
- Wide browser support

Cons

- Less encapsulation
- Potential CSS conflicts
- Global namespace pollution

```
// Build configuration
export default defineConfig({
  build: {
    lib: {
      entry: 'src/widget.js',
      name: 'ChatWidget',
```

Iframe Approach

Embed your chatbot in an iframe that provides complete isolation from the host website, preventing any conflicts.

Pros

- Complete isolation
- Security benefits
- No CSS or JS conflicts

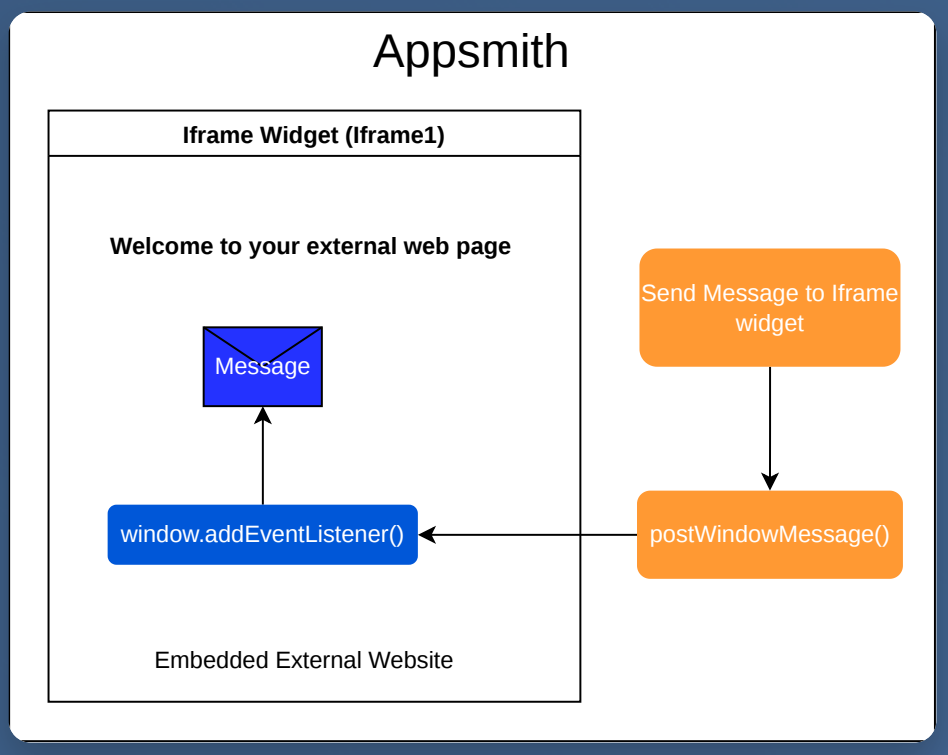
Cons

- Limited integration
- Navigation challenges
- Fixed height issues

```
// Communication from child to parent
function sendMessage(payload) {
  const content = window.parent
  const stringPayload = JSON.stringify(payload)
  content.postMessage(stringPayload, '*')
}
```

Implementation Comparison

The iframe approach provides complete isolation but requires postMessage for communication, while UMD bundle offers easier integration but with potential conflicts. Choose based on your specific needs for security vs. integration.



Recommended Solution

Web Components with Vue 3's defineCustomElement API offers the best balance for your chatbot widget

Web Components Approach

We recommend using Vue 3's defineCustomElement API to create a web component that encapsulates your chatbot functionality with Shadow DOM for style isolation.

- Seamless integration with any website while maintaining isolation
- Framework-agnostic compatibility across all modern browsers
- Shadow DOM prevents CSS conflicts with host website
- Easy configuration through HTML attributes and JavaScript API

Implementation Overview

1 Set Up Project

Create a new Vue 3 project with Vite and configure for web components

2 Create Chatbot Component

Build your chatbot UI with expandable/collapsible functionality

3 Define Custom Element

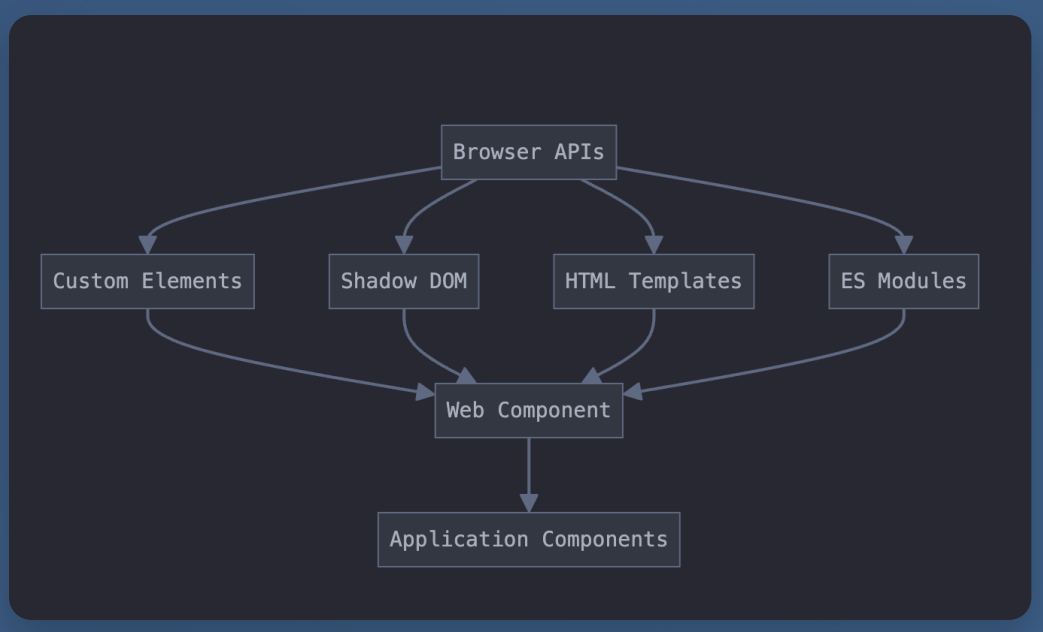
Use defineCustomElement to convert Vue component to web component

4 Build & Distribute

Package as UMD/ES module and provide simple embed code

Recommended Architecture

The web component approach creates a self-contained chatbot widget that can be embedded in any website while maintaining communication with your existing backend APIs.



Comparison with Alternatives

Unlike UMD bundles that risk CSS conflicts or iframes that limit integration, web components provide the ideal balance of isolation and seamless integration.

- Better than UMD: No CSS conflicts, proper encapsulation
- Better than iframe: Seamless integration, dynamic height
- Native browser support without additional dependencies

Implementation Steps

Step-by-step guide to implementing your Vue 3 chatbot as a web component widget

1 Set Up Project

Create a new Vue 3 project with Vite and organize your component structure for the chatbot widget.

```
# Create a new Vue project
npm create vite@latest chatbot-widget -- --template vue

# Navigate to project directory
cd chatbot-widget

# Install dependencies
npm install
```

2 Configure Vite

Configure Vite to build your Vue component as a web component with proper output formats.

```
// vite.config.js
import { defineConfig } from 'vite'
import vue from '@vitejs/plugin-vue'

export default defineConfig({
  plugins: [vue({ customElement: true })],
  build: {
    lib: {
      entry: 'src/widget.js',
      name: 'ChatWidget',
```

3 Create Chatbot Component

Build your chatbot UI with expandable/collapsible functionality and proper styling.

```
// src/components/ChatbotWidget.vue
<template>
  <div class="chatbot-container">
    <div class="chat-button" @click="toggleChat">
      {{ isOpen ? 'Close' : 'Chat' }}
    </div>
    <div v-if="isOpen" class="chat-window">
      // Chat interface here
    </div>
  </div>
```

4 Define Custom Element

Use Vue 3's defineCustomElement API to convert your component into a web component.

```
// src/widget.js
import { defineCustomElement } from 'vue'
import ChatbotWidget from
'./components/ChatbotWidget.vue'

// Convert Vue component to custom element
const ChatbotElement =
defineCustomElement(ChatbotWidget)

// Register the custom element
```

Implementation Tips

- Use CSS custom properties for dynamic theming and branding options
- Implement event system for communication between widget and host page
- Add authentication options via JWT or query parameters for user identification

Configuration & Communication

Customize your chatbot widget and enable seamless communication with backend APIs

Configuration Options

Positioning

Control widget placement (bottom-right, bottom-left, etc.)

Branding

Customize colors, logo, and visual elements

User Identification

Authenticate via JWT or query parameters

Visibility

Control when and where the widget appears

```
// HTML attributes for configuration
<chatbot-widget
  position="bottom-right"
  primary-color="#4fd1c5"
  logo-url="https://example.com/logo.png"
  user-token="eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
  api-endpoint="https://api.example.com/chat"
></chatbot-widget>
```

API Communication

Event System

Use CustomEvents for widget-host communication

Authentication

Pass JWT tokens or API keys for secure API access

Data Transfer

Handle bidirectional data flow between widget and backend

```
// Event communication example
// Widget to host
const event = new CustomEvent('chat-message', {
  detail: {
    type: 'user-message',
    content: 'Hello, I need help',
    timestamp: Date.now()
  }
});
```