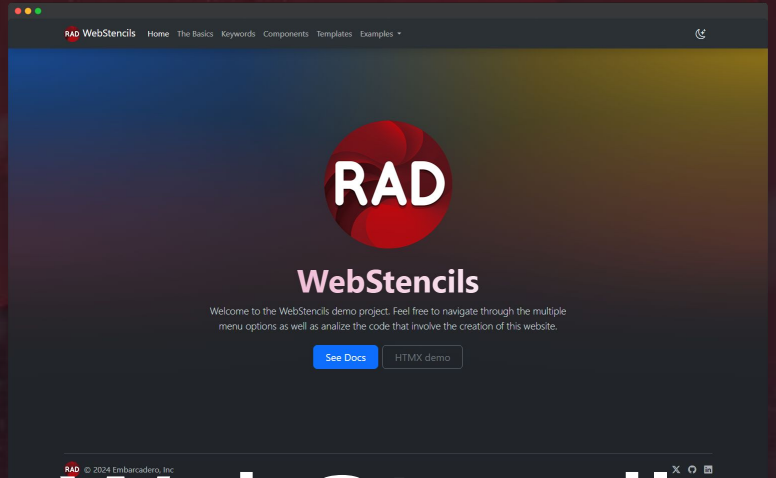


<WebStencils>

Reimagining web dev in RAD Studio

Antonio Zapater
Pre-sales consultant engineer
Embarcadero Technologies
antonio.zapater@embarcadero.com

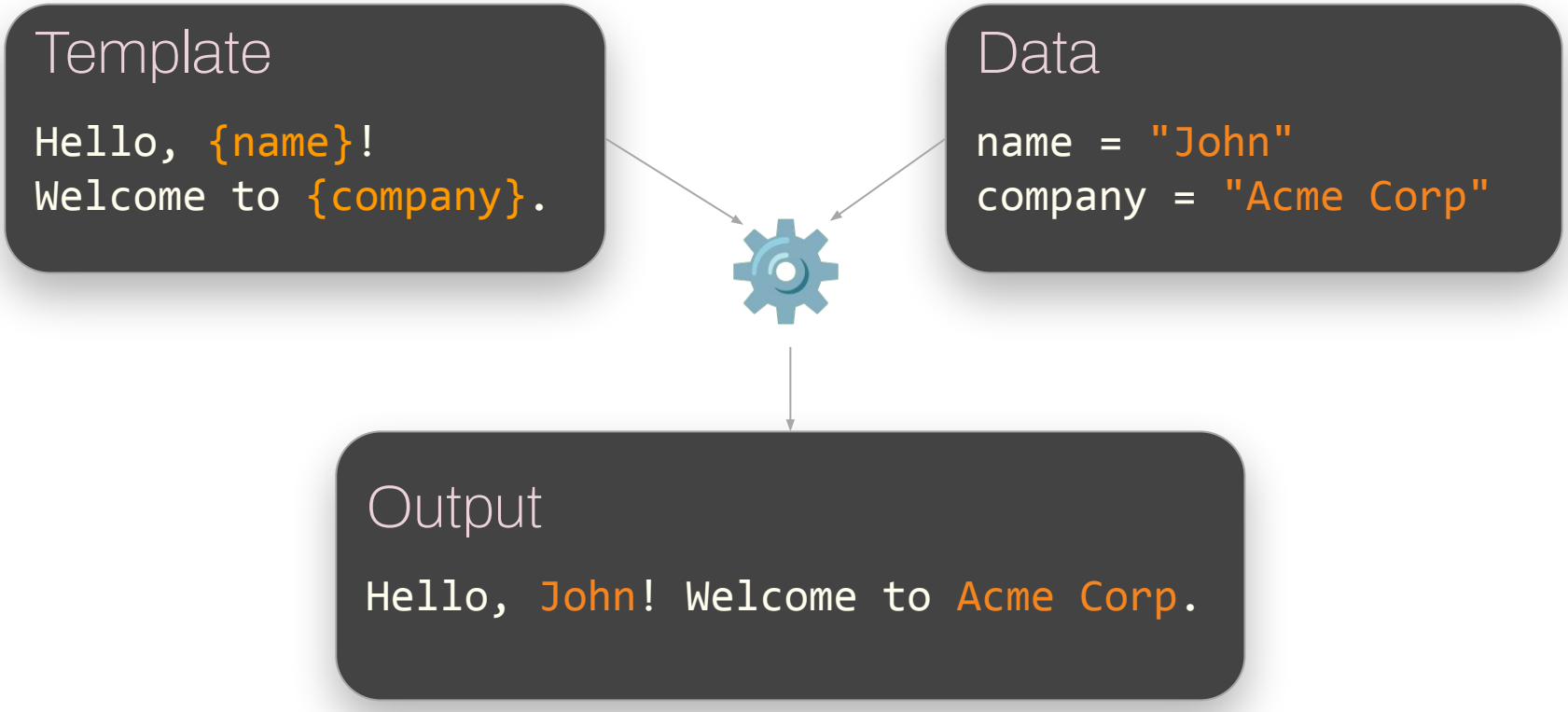
embarcadero®


What is a
Template Engine

What is a Template Engine?

- A tool that separates design from content
- Allows dynamic generation of output (e.g., HTML)
- Uses placeholders and special syntax in templates
- Replaces placeholders with actual data at runtime

Template Engine: Basic Concept



Why Use a Template Engine?

- Separates presentation logic from business logic
- Improves maintainability and readability
- Enables non-programmers to modify layouts
- Facilitates reuse of design elements

Common Template Engine Features

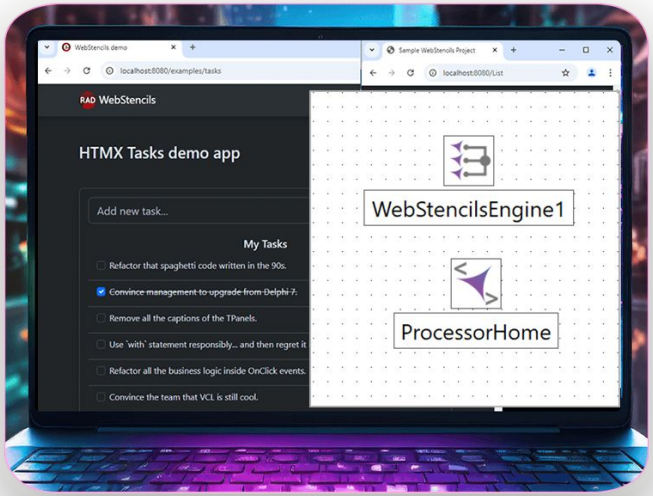
- Variable substitution
- Conditional statements
- Loops
- Includes (nested templates)
- Layout inheritance

Introducing

WebStencils

Introducing WebStencils

- A template engine designed for RAD Studio
- Similar approach to Razor for .NET
- Primarily focused on web development
  (but not limited to it)
- Can be used with RAD Server and WebBroker



WebStencils Syntax: Basics

- Uses `@` symbol for code expressions
- Curly braces `{}` for multi-line code blocks
- `@object.value` to read data from server objects
- `@keywords` for special processing
- Supports simple expressions

Example

```
<p>Hello, @user.name!</p>
@if user.isAdmin {
    <p>Welcome, admin!</p>
}
```

Demo

WebStencils Keywords

@query - access HTTP parameters

@* .. *@ - comment

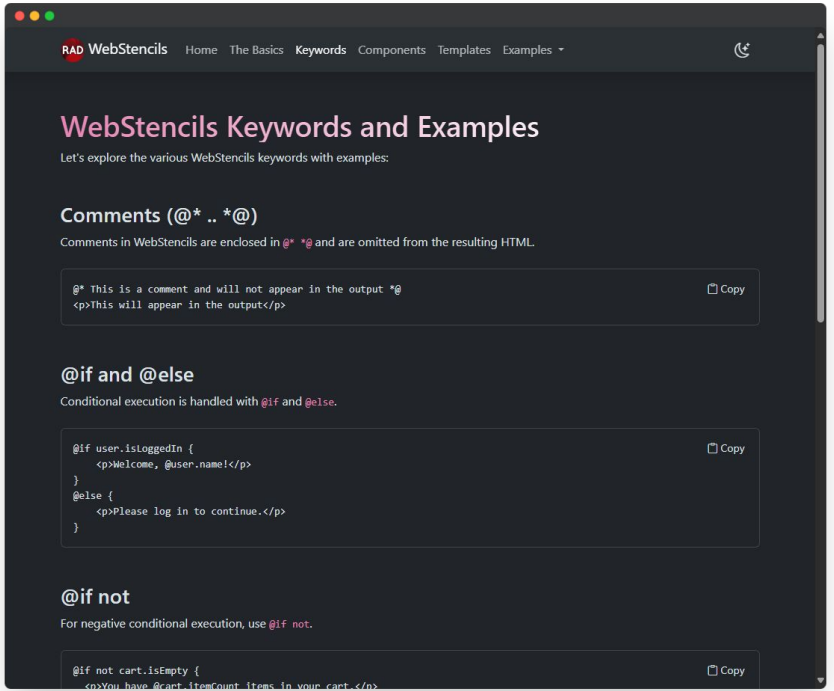
@if [not] object.value { ... } [**@else** { ... }]

@ForEach (var object in list) { ... }

@ForEach object { ... }

@Import

@Scaffolding



WebStencils: Syntax Examples

Loops

```
<ul>
@ForEach (var product in productList) {
    <li>@product.name - @product.price</li>
}
</ul>
```

Import

```
@if user.isLoggedIn {
    @Import userPanel.html
}
@else {
    <p>Please log in.</p>
}
```

WebStencils: Template structure

@LayoutPage

Indicates which template file to use

@RenderBody

Placeholder in a template file where to place actual content

@ExtraHeader { ... }

Optional block of code to be added as extra header information

@RenderHeader

Where to add the extra header of the page

WebStencils: Layouts

Main layout (layout.html)

```
<html>
<head>@RenderHeader</head>
<body>
    @RenderBody
</body>
</html>
```

Content page (index.html)

```
@LayoutPage layout.html
@ExtraHeader { <script src="MyJS.js"> }
<h1>Welcome to my page!</h1>
```

Output

```
<html>
<head>
  <script src="MyJS.js">
</head>
<body>
  <h1>Welcome to my page!</h1>
</body>
</html>
```

WebStencils: Components

- **WebStencils Engine**
  - Manages overall template processing
  - Handles file dispatching and request matching
  - Create `TWebStencilsProcessor` components, if needed
  - Provides shared settings to individual `TWebStencilsProcessor` comp.

- **WebStencils Processor**
  - Processes individual files/templates
  - Properties InputFilename, InputLines...
  - Can be used standalone or managed by the Engine

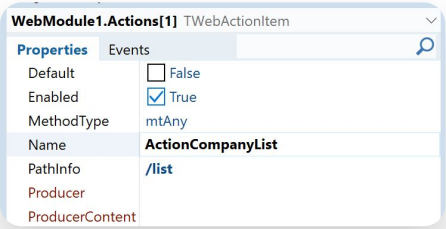
Integration with Delphi

- Use `AddVar` to pass Delphi objects to templates
  - Use objects, lists, or datasets
  - Registered as script variables
  - Define modules variables with attributes `[WebStencilsVar('varName')]`
- Handle events like `OnScaffolding` and `OnValue` for custom keywords

Example

```
WebStencilsProcessor.InputFileName := 'template.html';
WebStencilsProcessor.AddVar('user', UserObject);
Result := WebStencilsProcessor.Content;
```

WebStencils: Mapping



WebBroker Actions

- Connect action to WebStencilsProcessor
- Use OnAction handler to:
  - Initialize the processor script variables
  - Run the processor
  - Return the resulting HTML
- In the script, use the variables to generate the HTML

```
procedure TWebModule1.WebModule1WebActionItem1Action( Sender : TObject;
  Request : TWebRequest; Response : TWebResponse; var Handled : Boolean );
begin
  ClientDataSet1.Open;
  ProcessorCompanyList.AddVar( 'dataset', ClientDataSet1, False ); // do not destroy
  Response.Content := ProcessorCompanyList.Content;
  ClientDataSet1.Close;
  Handled := True;
end;
```

```
@foreach dataset {

  <a href="/company?id=@loop.custno" class="list-group
    <div class="d-flex w-100 justify-content-between">
      <h5 class="mb-1">@loop.company</h5>
      <small>@loop.country</small>
    </div>
    <p class="mb-1">@loop.city</p>
    <small>@loop.state</small>
  </a>

}
```

Both **WebBroker** and **RAD Server** offer a mapping layer to associate a URL with a file script

Demo

Introduction to
HTMX

What is HTMX?

HTMX allows you to access AJAX, CSS Transitions, WebSockets and Server Sent Events directly in HTML, using attributes, so you can build modern user interfaces with the simplicity and power of hypertext.

– *htmx.org*

What is HTMX?

Key features

- Uses standard HTTP requests (AJAX)
- Server-side rendering with dynamic updates
- Partial page updates without full reloads
- Easy integration with any backend technology

Benefits

- Very lightweight (~15kb minimized + gziped – Vue is ~40kb)
- Zero dependencies
- Reduces complexity of client-side JavaScript
- Maintains a familiar HTML-centric approach

HTMX vs Traditional AJAX

- Simplifies AJAX by embedding behavior directly in HTML.
- No need for complex JavaScript; everything is declarative in HTML.

HTMX Example

Button Example

```
<button hx-get="/endpoint" hx-target="#result">Click me!</button>
<div id="result"></div>
```

HTML & JavaScript Example

Button Example

```
<button id="myButton">Click me!</button>
<div id="result"></div>
```

```
<script>
    document.getElementById("myButton").addEventListener("click", function() {
        fetch('/endpoint')
            .then(response => response.text())
            .then(data => {
                document.getElementById("result").innerHTML = data;
            });
    });
</script>
```

HTMX Quick Reference

htmx.org/docs

Core Attributes

- **hx-get**/**hx-post**/**hx-put**/**hx-delete**: Triggers an HTTP request (GET, POST, PUT, DELETE) to an endpoint.
- **hx-target**: Specifies the element in the DOM that should be updated with the server's response.
- **hx-swap**: Controls how the server response is inserted into the **hx-target** (e.g., **innerHTML**, **outerHTML**, **beforeend**, **afterbegin**...).
- **hx-boost:** add progressive enhancement for links and forms.

Triggers

- **hx-trigger**: Specifies events that will trigger the request (e.g., **click**, **change**, **load**, or custom events).
- **hx-trigger="every 2s"**: Repeatedly triggers the request every specified interval.

Event Listeners

- **hx-on**: Binds additional events (e.g., **htmx:beforeRequest**, **htmx:afterSwap**, **htmx:error**) to run custom JavaScript.

Comparison with Other JS Frameworks

Most JS Frameworks (React, Vue, Angular, Svelte…)

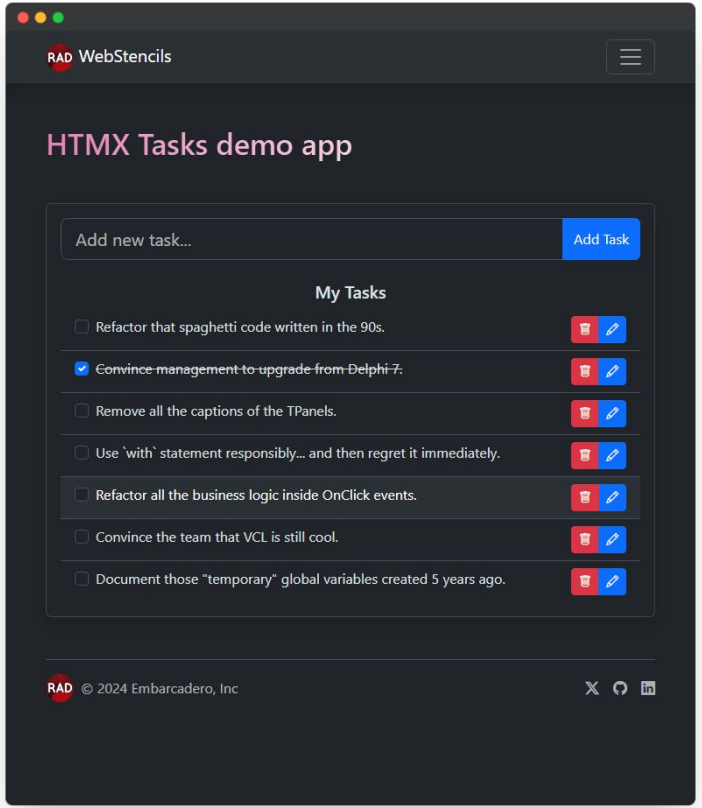
- **Source of Truth:** State management is complex, often requiring tools like Redux, Pinia etc.
- **Complexity:** High learning curve; often overkill for small projects.
- **Performance:** Re-rendering can be costly; needs virtual DOM optimization.

HTMX

- No complex state management - The backend is the only source of truth.
- Easier to reason about - no need for state syncing.
- Simple to use for small to medium-sized projects.
- Server-driven approach - offloads complexity to the server.

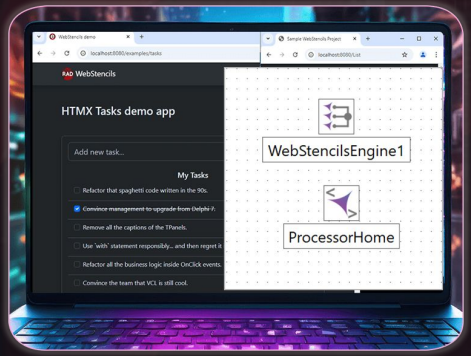
Using HTMX with WebStencils

- Only one <script> line
- Server-side rendering
- Granular HTML injection
- Backend agnostic
- Reusability
- No JavaScript (almost)



Demo

Questions ?



Free ebook

<WebStencils>

Antonio Zapater
Pre-sales consultant engineer
Embarcadero Technologies
antonio.zapater@embarcadero.com