# Demos Subdomain, TLS, and IIS Routing

Scope
Routing Strategy
Current Demos and Routing Map
IIS URL Rewrite Configuration
How to Add a New Demo Project

## Scope

This page documents:

- What was implemented to publish multiple FrameworX demo projects under `https://demo.tatsoft.com/<DemoName>`
- The current IIS URL Rewrite configuration (`web.config`) and its consolidated **RewriteMap-based** routing approach
- How to add new demo projects following the same pattern

High-Level Architecture

Public entry point

A dedicated subdomain was created: `demo.tatsoft.com`. End users access demos by path, for example:

```
https://demo.tatsoft.com/ProveIt
https://demo.tatsoft.com/Tupinix
https://demo.tatsoft.com/SolarPanel
https://demo.tatsoft.com/BottlingLine
```

TLS / SSL model

- `demo.tatsoft.com` is covered by the same TLS certificate used for `tatsoft.com` at the Cloudflare edge (browser  Cloudflare).
- In Cloudflare DNS, `demo.tatsoft.com` is configured as a **proxied CNAME** pointing to the origin. Proxied records route traffic through Cloudflare's network.
- Between Cloudflare and the origin Azure VM, the connection uses a **Cloudflare Origin Certificate** (Cloudflare  origin), which is the recommended approach when the origin only receives traffic through Cloudflare.

---

## Routing Strategy

Because all traffic arrives at a single host (`demo.tatsoft.com`), IIS URL Rewrite is used as a reverse-proxy router. The configuration uses RewriteMaps to centralize project-to-port resolution, eliminating per-project rule duplication and making it trivial to add new demos.

### RewriteMaps

Two RewriteMaps drive all routing decisions:

| Map name | Input | Output | Purpose |
|---|---|---|---|
| `ProjectNameToRoute` | Lowercase project name | Route code (`p1`–`p4`) | Resolves the cookie value from the URL segment |
| `ProjectRouteToPort` | Route code (`p1`–`p4`) | Local port (`3101`–`3401`) | Resolves the backend port from the cookie |

### Key mechanics

1. **Entry redirect to** `_setroute`

   Visiting the site root or `/<Project>` (with or without trailing slash) redirects to `/<Project>/_setroute`. The root redirects to `/ProveIt/_setroute` by default.

2. **Cookie-based route selection**

   The `_setroute` endpoint resolves the cookie value dynamically using `ProjectNameToRoute` and sets the routing cookie:

   *ProjectRoute=p1; Path=/; Secure; SameSite=None*

   The user is then redirected to the unified `/html5/` endpoint.

3. **Deep-link normalization**

   Direct access to `/<Project>/html5/...` (bookmarks, refreshes, deep links) is handled by a dedicated rule that re-sets the cookie and redirects to `/html5/<path>`, preserving the sub-path.

4. **Unified reverse proxy for** `/html5/`

   All requests under `/html5/` are forwarded to the correct local backend. The target port is resolved dynamically from the cookie using `ProjectRouteToPort`:

   `http://127.0.0.1:{ProjectRouteToPort:{C:2}}/html5/{R:1}`

5. **Unified reverse proxy for** `/rtServices/`

   WebSocket upgrade requests and all `/rtServices/` traffic are routed to the correct backend using the same cookie + RewriteMap mechanism:

   `http://127.0.0.1:{ProjectRouteToPort:{C:2}}/rtServices/{R:1}`

   After the WebSocket upgrade completes, subsequent client  backend traffic flows through that long-lived connection and routing decisions are locked in for the session.

---

## Current Demos and Routing Map

| Demo path | Cookie **`ProjectRoute`** | Local backend | Public URL |
|---|---|---|---|
| `/ProveIt` | **p1** | `http://127.0.0.1:3101/` | `https://demo.tatsoft.com/ProveIt` |
| `/Tupinix` | **p2** | `http://127.0.0.1:3201/` | `https://demo.tatsoft.com/Tupinix` |
| `/SolarPanel` | **p3** | `http://127.0.0.1:3301/` | `https://demo.tatsoft.com/SolarPanel` |
| `/BottlingLine` | **p4** | `http://127.0.0.1:3401/` | `https://demo.tatsoft.com/BottlingLine` |

---

## IIS URL Rewrite Configuration

### Rewrite Maps

```
<rewriteMaps>

  <!-- Project name (lowercase) -> route code (cookie value) -->
  <rewriteMap name="ProjectNameToRoute">
    <add key="proveit"      value="p1" />
    <add key="tupinix"      value="p2" />
    <add key="solarpanel"   value="p3" />
    <add key="bottlingline" value="p4" />
  </rewriteMap>

  <!-- Route code (cookie value) -> local backend port -->
  <rewriteMap name="ProjectRouteToPort">
    <add key="p1" value="3101" />
    <add key="p2" value="3201" />
    <add key="p3" value="3301" />
    <add key="p4" value="3401" />
  </rewriteMap>

</rewriteMaps>
```

### Rewrite Rules

**Rule 1: Site root  ProveIt** `_setroute` **-** Stop processing other rules

Visiting the bare hostname redirects to the default demo entry point.

**Rule 1**

```
<rule name="Default -> ProveIt _setroute" stopProcessing="true">
  <match url="^$" />
  <action type="Redirect" url="/ProveIt/_setroute" redirectType="Found" />
</rule>
```

**Rule 2:**`/<Project> /<Project>/_setroute` **-** Stop processing other rules

A single rule handles all project root redirects using a grouped match. Case-insensitive.

**Rule 2**

```
<rule name="Project root -> _setroute" stopProcessing="true">
  <match url="^(ProveIt|Tupinix|SolarPanel|BottlingLine)/?$" ignoreCase="true" />
  <action type="Redirect" url="/{R:1}/_setroute" redirectType="Found" />
</rule>
```

**Rule 3: Set routing cookie and redirect to** `/html5/`- Stop Processing Other rules

Resolves the project name to its cookie value via `ProjectNameToRoute`, sets the `ProjectRoute` cookie, then redirects to the unified `/html5/` endpoint.

**Rule 3**

```
<rule name="SetRoute by project" stopProcessing="true">
  <match url="^(ProveIt|Tupinix|SolarPanel|BottlingLine)/_setroute$" ignoreCase="true" />
  <serverVariables>
    <set name="RESPONSE_Set-Cookie"
         value="ProjectRoute={ProjectNameToRoute:{ToLower:{R:1}}}; Path=/; Secure; SameSite=None" />
  </serverVariables>
  <action type="Redirect" url="/html5/" redirectType="Found" />
</rule>
```

**Rule 4: Deep-link normalization —** `/<Project>/html5/... /html5/...` - Stop processing other rules

Handles bookmarks, browser refreshes, and direct deep links that include the project prefix. Re-sets the cookie and redirects to `/html5/`, preserving any sub-path.

**Rule 4**

```
<rule name="Normalize project html5 -> /html5" stopProcessing="true">
  <match url="^(ProveIt|Tupinix|SolarPanel|BottlingLine)/html5/?(.*)$" ignoreCase="true" />
  <serverVariables>
    <set name="RESPONSE_Set-Cookie"
         value="ProjectRoute={ProjectNameToRoute:{ToLower:{R:1}}}; Path=/; Secure; SameSite=None" />
  </serverVariables>
  <action type="Redirect" url="/html5/{R:2}" redirectType="Found" />
</rule>
```

**Rule 5: Reverse proxy —** `/html5/...` **local backend -** Stop processing other rules

Reads the `ProjectRoute` cookie, resolves the port via `ProjectRouteToPort`, and proxies the request to the correct local backend. The cookie capture group `{C:2}` extracts the route code.

**Rule 5**

```
<rule name="html5 -> backend by ProjectRoute cookie" stopProcessing="true">
  <match url="^html5/(.*)$" />
  <conditions>
    <add input="{HTTP_COOKIE}" pattern="(^|;\s*)ProjectRoute=(p[1-4])($|;)" />
  </conditions>
  <action type="Rewrite" url="http://127.0.0.1:{ProjectRouteToPort:{C:2}}/html5/{R:1}" />
</rule>
```

**Rule 6: Reverse proxy — `/rtServices/...` local backend -** Stop processing other rules

Same mechanism as Rule 5. Routes WebSocket upgrade requests and all `/rtServices/` traffic to the correct local backend using the `ProjectRoute` cookie.

**Rule 6**

```
<rule name="rtServices -> backend by ProjectRoute cookie" stopProcessing="true">
  <match url="^rtServices/(.*)$" />
  <conditions>
    <add input="{HTTP_COOKIE}" pattern="(^|;\s*)ProjectRoute=(p[1-4])($|;)" />
  </conditions>
  <action type="Rewrite" url="http://127.0.0.1:{ProjectRouteToPort:{C:2}}/rtServices/{R:1}" />
</rule>
```

---

# How to Add a New Demo Project

## Step 1 — Decide the new mapping

| Parameter | Example | Notes |
|---|---|---|
| Demo path | `/NewDemo` | Used in URLs and rule patterns |
| Cookie value | `p5` | Next available value after `p4` |
| Local backend port | `3501` | Must be reachable at `127.0.0.1:PORT` |
| Entry page | `/NewDemo/html5/` | Keep consistent with existing demos |

## Step 2 — Deploy and start the backend locally

The backend must be reachable from IIS on the Azure VM:

```
http://127.0.0.1:3501/
```

## Step 3 — Add entries to both RewriteMaps

```
<!-- In ProjectNameToRoute -->
<add key="newdemo" value="p5" />

<!-- In ProjectRouteToPort -->
<add key="p5" value="3501" />
```

## Step 4 — Add the project name to the rule patterns (Rules 2, 3, 4)

Three rules use an alternation pattern listing all project names. Add the new name to each:

```
<!-- Rule 2: Project root -> _setroute -->
<match url="^(ProveIt|Tupinix|SolarPanel|BottlingLine|NewDemo)/?$" ignoreCase="true" />

<!-- Rule 3: SetRoute by project -->
<match url="^(ProveIt|Tupinix|SolarPanel|BottlingLine|NewDemo)/_setroute$" ignoreCase="true" />

<!-- Rule 4: Normalize project html5 -->
<match url="^(ProveIt|Tupinix|SolarPanel|BottlingLine|NewDemo)/html5/?(.*)$" ignoreCase="true" />
```

### Step 5 — Expand the cookie pattern in Rules 5 and 6

Update the regex to include the new route code:

```
<!-- Before -->
<add input="{HTTP_COOKIE}" pattern="(^|;\s*)ProjectRoute=(p[1-4])($|;)" />

<!-- After (to include p5) -->
<add input="{HTTP_COOKIE}" pattern="(^|;\s*)ProjectRoute=(p[1-5])($|;)" />
```

### Step 6 — Testing checklist

| Test | Expected result |
|---|---|
| Open `https://demo.tatsoft.com/NewDemo` | Browser redirects through `_setroute` and lands on `/html5/` |
| Inspect cookies | `ProjectRoute=p5` is set with `Secure; SameSite=None` |
| Open a deep link, e.g. `/NewDemo/html5/somepage` | Cookie is set; browser lands on `/html5/somepage` |
| Monitor `/rtServices/` requests | Requests are routed to `127.0.0.1:3501` |
| Refresh the page | Cookie persists; page reloads correctly without redirect loop |

---

# Operational Notes and Pitfalls

**Rule order matters.**
The first matching rule with `stopProcessing="true"` wins. Do not reorder rules without understanding the full evaluation flow.

### Cookie attributes

The `ProjectRoute` cookie is set with `Secure; SameSite=None`. This is consistent with modern browser requirements when cross-site behaviour may occur. Do not change these attributes without a clear reason.

### WebSocket routing and rtServices

FrameworX creates a dedicated WebSocket connection per demo. The initial WebSocket upgrade request is made against the shared `/rtServices` endpoint — it is not namespaced under `/DemoName/`. Because WebSockets are long-lived, IIS must route the upgrade request to the correct backend from the start. That is why Rules 5 and 6 both rely on the `ProjectRoute` cookie: it is guaranteed to be present by the time any `/rtServices/` or `/html5/` request is made.

After the WebSocket upgrade completes, all subsequent client  backend traffic flows through that long-lived connection. Routing decisions are effectively locked in for the session and are no longer evaluated by IIS URL Rewrite.

### Cookie race condition (multiple tabs)

If a second demo is opened in a new tab while another demo's WebSocket upgrade is still being negotiated, the overwritten `ProjectRoute` cookie may cause the second demo's `/rtServices/` request to be routed to the wrong backend. Once both WebSocket connections are established, sessions are independent.