# **Embedding a Standalone Wix Dashboard via IFrame: Security & UI Implementation**

When integrating a custom dashboard into Wix using an IFrame, it's crucial to follow Wix’s security guidelines and use modern UI tools for a smooth user experience. Below we cover how to securely embed an external dashboard in Wix (authentication, IFrame parameters, CORS) and how to build a responsive UI with **shadcn/ui**, **Radix UI**, **Tailwind CSS**, and **React Hook Form**. Code examples and tips for developing on Replit are included for practical guidance.

## **Wix Dashboard IFrame Integration & Security**

**IFrame Dashboard Extensions in Wix:** Wix allows apps to include **dashboard pages** that are loaded in the Wix site owner’s dashboard as an IFrame. When a site owner opens your app’s dashboard via Wix, Wix will load your provided IFrame URL along with **query parameters** that give context about the site and user​

[dev.wix.com](https://dev.wix.com/docs/build-apps/develop-your-app/frameworks/self-hosting/supported-extensions/dashboard-extensions/iframe-query-parameters-for-dashboard-extensions#:~:text=Name%20Value%20Comments%20,hasn%27t%20seen%20this%20URL%20before)

. Important parameters include:

* **instance** – a signed token containing the **app instance ID** (instanceId) and user/site info. The instanceId uniquely identifies your app installation on that Wix site​  
  [dev.wix.com](https://dev.wix.com/docs/build-apps/develop-your-app/frameworks/self-hosting/supported-extensions/dashboard-extensions/iframe-query-parameters-for-dashboard-extensions#:~:text=Name%20Value%20Comments%20,hasn%27t%20seen%20this%20URL%20before).
* Other parameters like locale (user’s language), siteUrl, etc., are provided for context​  
  [dev.wix.com](https://dev.wix.com/docs/build-apps/develop-your-app/frameworks/self-hosting/supported-extensions/dashboard-extensions/iframe-query-parameters-for-dashboard-extensions#:~:text=Name%20Value%20Comments%20,hasn%27t%20seen%20this%20URL%20before).

**Signed Instance Token:** The instance query param is Wix’s mechanism for authenticating requests. It contains a **Base64-encoded JSON** with details (instanceId, user ID, site info, etc.) and a **signature**. These two parts are separated by a . in the token​

[dev.wix.com](https://dev.wix.com/docs/build-apps/develop-your-app/access/app-instances/about-app-instances#:~:text=The%20encoded%20instance%20parameter%20is,parts%20separated%20by%20a%20dot)

. The signature is an HMAC SHA-256 hash of the data part, signed with your app’s **secret key** (from your Wix Developers config)​

[dev.wix.com](https://dev.wix.com/docs/build-apps/develop-your-app/access/app-instances/about-app-instances#:~:text=%2A%20Signature%3A%20HMACSHA,data%20to%20see%20these%20properties)

. Wix generates this token on the client side for each request. Because the token is signed with your secret, you can use it to verify that requests truly come from Wix and have not been tampered with​

[dev.wix.com](https://dev.wix.com/docs/build-apps/develop-your-app/access/app-instances/about-app-instances#:~:text=When%20a%20user%20accesses%20your,ensure%20its%20integrity%20and%20authenticity)

​

[dev.wix.com](https://dev.wix.com/docs/build-apps/develop-your-app/access/app-instances/about-app-instances#:~:text=%2A%20Signature%3A%20HMACSHA,data%20to%20see%20these%20properties)

.

**How to Verify the Instance Token:** On your backend server, you should **parse and validate** the instance token on **every incoming request** from the IFrame. For example, in Node.js you can split the token by the . delimiter to get the signature and data. Compute an HMAC SHA-256 of the data portion using your Wix app secret and compare it to the signature (after Base64 decoding)​

[dev.wix.com](https://dev.wix.com/docs/build-apps/develop-your-app/access/app-instances/about-app-instances#:~:text=The%20encoded%20instance%20parameter%20is,parts%20separated%20by%20a%20dot)

. Only proceed if they match. Then decode the JSON data to get the details of the site and user. (Never expose your secret key in the frontend; do this validation on the server side only.)

**Example – Verifying Wix Instance (Node.js):** Below is a simplified example of verifying the instance token using Node’s crypto module:

const crypto = require('crypto');

const APP\_SECRET = process.env.WIX\_APP\_SECRET; // Your Wix app secret (keep safe!)

function verifyInstance(instanceToken) {

const [sigEncoded, dataEncoded] = instanceToken.split('.');

if (!sigEncoded || !dataEncoded) throw new Error("Invalid token format");

// Recompute HMAC SHA-256 on the data part

const hmac = crypto.createHmac('sha256', APP\_SECRET);

hmac.update(dataEncoded);

const expectedSig = hmac.digest('base64').replace(/=+$/, ''); // compute signature (strip padding)

if (expectedSig !== sigEncoded) {

throw new Error("Invalid signature – request not authenticated!");

}

// Decode the data part from Base64 to JSON

const jsonData = JSON.parse(Buffer.from(dataEncoded, 'base64').toString('utf-8'));

return jsonData; // contains instanceId, uid, etc.

}

This function splits the instance token, verifies the signature matches the data (using the shared secret), and returns the decoded info. You should call such a function for incoming API requests from your dashboard IFrame to ensure they are from an authorized Wix site/user.

**Using the Instance for Authentication:** Once verified, the instance data tells you **which site and user** is making the request. Key fields include the site’s instanceId (unique per installation) and the user’s uid (Wix user ID of the logged-in site owner or collaborator)​

[dev.wix.com](https://dev.wix.com/docs/build-apps/develop-your-app/access/app-instances/about-app-instances#:~:text=Field%20Description%20,The%20Plan)

. You can use this to identify the account in your system or auto-login the user to your app. For example, if you maintain accounts per Wix site, use the instanceId as a key to find that site’s data, and ensure the uid matches an authorized user for sensitive actions.

**Security tip:** Wix documentation notes that if the instance data contains an aid (anonymous ID) without a real user uid, then a non-logged-in person is trying to access the dashboard. In such cases, you should **restrict access** to protect sensitive data​

[dev.wix.com](https://dev.wix.com/docs/build-apps/develop-your-app/access/app-instances/about-app-instances#:~:text=Warning%3A%20For%20dashboard%20security%2C%20restrict,uid)

(only site owners or collaborators should reach the dashboard). Always verify the user’s role (e.g. compare uid to siteOwnerId in the instance data to confirm the site owner is the one accessing)​

[dev.wix.com](https://dev.wix.com/docs/build-apps/develop-your-app/access/app-instances/about-app-instances#:~:text=Note%3A%20To%20check%20if%20the,is%20the%20same%20as%20the)

​

[dev.wix.com](https://dev.wix.com/docs/build-apps/develop-your-app/access/app-instances/about-app-instances#:~:text=Warning%3A%20For%20dashboard%20security%2C%20restrict,uid)

.

**Passing the Instance Token to Your Backend:** In your IFrame’s frontend (the React app running in the IFrame), grab the instance param from the URL and include it with every API request to your backend. For example, you can read it using window.location.search:

On the server, you would extract this token (from headers or query) and run the verification as shown earlier. Wix specifically recommends using the instance token to authenticate requests from your app frontend to your backend​

[dev.wix.com](https://dev.wix.com/docs/build-apps/develop-your-app/frameworks/wix-cli/app-development/implement-a-backend-for-your-app#:~:text=Authenticate%20incoming%20requests)

. This ensures that your backend only serves data when the request originates from a valid Wix dashboard session.

**CORS Configuration for Wix App Requests:** If your frontend IFrame and backend are on different domains, you must configure **CORS** (Cross-Origin Resource Sharing) on your backend so that the Wix-served frontend can call it. In the Wix app context, your React frontend might be served on a Wix domain (if using the Wix Apps CDN) or on your own domain. Wix’s guidelines state that your server must allow requests from your app’s domain​

[dev.wix.com](https://dev.wix.com/docs/build-apps/develop-your-app/frameworks/wix-cli/app-development/implement-a-backend-for-your-app#:~:text=When%20making%20HTTP%20requests%20from,requests%20from%20your%20frontend%27s%20domain)

. For example, if Wix hosts your app’s static files on https://<your-app-id>.wix.run, you need to allow that origin in your backend responses​

[dev.wix.com](https://dev.wix.com/docs/build-apps/develop-your-app/frameworks/wix-cli/app-development/implement-a-backend-for-your-app#:~:text=When%20making%20HTTP%20requests%20from,requests%20from%20your%20frontend%27s%20domain)

.

In an Express.js server, you could enable CORS like so:

const express = require('express');

const app = express();

const allowedOrigin = "https://<your-app-id>.wix.run"; // Wix CDN domain serving your app

app.use((req, res, next) => {

res.setHeader("Access-Control-Allow-Origin", allowedOrigin);

res.setHeader("Access-Control-Allow-Methods", "GET,POST,PUT,OPTIONS");

res.setHeader("Access-Control-Allow-Headers", "Content-Type, Authorization");

if (req.method === "OPTIONS") {

return res.sendStatus(204); // quick response to preflight

}

next();

});

Here we allow the Wix frontend origin and necessary headers (e.g. Authorization for the instance token). Adjust the methods and headers to only what your app needs​

[dev.wix.com](https://dev.wix.com/docs/build-apps/develop-your-app/frameworks/wix-cli/app-development/implement-a-backend-for-your-app#:~:text=Add%20the%20following%20headers%20to,CORS%20requests%20from%20your%20app)

. If you are fully self-hosting the fronted (not on wix.run), then use your own domain as the allowed origin instead.

**Local/Dev CORS:** During development, your frontend might run at localhost (e.g. Vite or Replit preview URL). Be sure to also allow your dev origin. Wix notes that for local development you should allow localhost (or the specific dev host/port) as an origin while testing​

[dev.wix.com](https://dev.wix.com/docs/build-apps/develop-your-app/frameworks/wix-cli/app-development/implement-a-backend-for-your-app#:~:text=Copy)

.

**HTTPS and Embedding:** Wix **requires HTTPS** for any embedded content. Ensure your IFrame URL is served over HTTPS, otherwise Wix will refuse to display it​

[support.wix.com](https://support.wix.com/en/article/wix-editor-embedding-a-site-or-a-widget#:~:text=,please%20contact%20the%20provider%20directly)

. Additionally, your external site should not send headers that block framing. Some web servers send X-Frame-Options: SAMEORIGIN or a Content-Security-Policy that disallows framing by other domains. If such a policy is in place, Wix cannot embed your page​

[support.wix.com](https://support.wix.com/en/article/wix-editor-embedding-a-site-or-a-widget#:~:text=,please%20contact%20the%20provider%20directly)

. Make sure to either remove those headers or configure them to allow Wix’s domains as valid frame ancestors. For example, using Content-Security-Policy: frame-ancestors https://\*.wix.com https://editor.wix.com; (check Wix documentation for the exact domains). This will ensure the Wix dashboard can display your IFrame content.

## **Responsive UI with shadcn/ui, Radix UI, Tailwind, and React Hook Form**

Building your dashboard’s front-end with **shadcn/ui** components, Radix UI, Tailwind CSS, and React Hook Form will enable a modern, responsive, and accessible interface.

* **shadcn/ui Library:** *shadcn/ui* is a toolkit of pre-built React components that are *highly customizable* and follow best practices out-of-the-box. It is built on top of **Radix UI** (headless accessible components) and styled with **Tailwind CSS**​  
  [tailkits.com](https://tailkits.com/blog/shadcnui-for-beginner/#:~:text=interior%20pieces%E2%80%94customizable%20UI%20components%20that,instantly%20upgrade%20your%20website). You can think of shadcn/ui as a collection of “ready-made” UI pieces (buttons, forms, modals, tables, etc.) that you can drop into your app, saving development time while ensuring consistency. Each component can be tailored via Tailwind classes or modified since the source is in your project. This gives you a native look but with far less effort than building from scratch.
* **Radix UI:** Under the hood, shadcn components use Radix UI primitives. **Radix UI** provides unstyled, accessible building blocks for common UI patterns (dialogs, dropdowns, accordions, etc.)​  
  [refine.dev](https://refine.dev/blog/radix-ui/#:~:text=%60Radix%60%20is%20an%20open,to%20build%20components%20with%20Radix). Using Radix ensures your dashboard meets accessibility standards (proper ARIA roles, focus handling, keyboard navigation) without you writing that logic. Radix UI components are “headless,” meaning you have full control of styling – which is where Tailwind and shadcn/ui’s default styles come in.
* **Tailwind CSS:** Tailwind is a utility-first CSS framework that makes it easy to build responsive designs. Instead of writing custom CSS for each component, you apply utility classes (like p-4 for padding or text-center) directly in your JSX. Tailwind’s utilities and responsive breakpoints let you quickly adapt your layout for different screen sizes. For example, setting width classes or flexbox utilities can make your dashboard adjust to various dashboard container sizes. Tailwind **“makes styling quicker with utility classes”**​  
  [tailkits.com](https://tailkits.com/blog/shadcnui-for-beginner/#:~:text=Click%20me%20) and ensures consistency. shadcn/ui components come pre-styled with Tailwind, but you can override or extend them by modifying the classes. This means you get a great default look and can easily implement custom brand styles or responsive tweaks as needed.
* **React Hook Form:** Managing form state and validation in React can be repetitive. **React Hook Form (RHF)** is a lightweight library that simplifies form handling with minimal re-renders​  
  [react-hook-form.com](https://react-hook-form.com/#:~:text=Super%20Light). It works seamlessly with controlled or uncontrolled components and has good integration with component libraries. In fact, shadcn/ui provides form wrapper components that integrate with React Hook Form to build accessible forms easily​  
  [tailkits.com](https://tailkits.com/blog/shadcnui-for-beginner/#:~:text=Forms%20are%20another%20area%20where,libraries%20like%20React%20Hook%20Form). RHF’s API is intuitive (using hooks like useForm and registering inputs) and very performant. By using RHF in your Wix dashboard app, you ensure that complex forms (for example, settings forms or data entry in the dashboard) remain snappy and responsive even as the user types or interacts – important for a good UX.

**Using These Tools Together:** In practice, these technologies complement each other. For example, you might use a shadcn/ui **Form** component which internally uses Radix form field primitives and Tailwind styling, and hook it up with React Hook Form for state management and validation. The result is a form that is accessible, nicely styled, and requires very little code on your part to handle input state or errors.

**Example – Login Form with shadcn/ui and React Hook Form:** Below is a short example of a form using shadcn/ui components. This form collects a username and password and uses React Hook Form for handling submission:

import { useForm } from "react-hook-form";

import { Form, FormField, FormItem, FormLabel, FormControl, FormMessage } from "@/components/ui/form";

import { Input } from "@/components/ui/input";

import { Button } from "@/components/ui/button";

export default function LoginForm() {

const form = useForm(); // initialize React Hook Form

const onSubmit = data => {

console.log("Form submitted:", data);

// handle login logic (e.g., call backend API)

};

return (

<Form onSubmit={form.handleSubmit(onSubmit)}>

<FormField

name="username"

control={form.control}

render={({ field }) => (

<FormItem>

<FormLabel>Username</FormLabel>

<FormControl>

<Input placeholder="Enter username" {...field} />

</FormControl>

<FormMessage /> {/\* displays validation errors for this field \*/}

</FormItem>

)}

/>

<FormField

name="password"

control={form.control}

render={({ field }) => (

<FormItem>

<FormLabel>Password</FormLabel>

<FormControl>

<Input type="password" placeholder="Enter password" {...field} />

</FormControl>

<FormMessage />

</FormItem>

)}

/>

<Button type="submit" className="w-full mt-2">Login</Button>

</Form>

);

}

In this example, the shadcn/ui <Form> and related components handle the layout and accessibility of the form, while React Hook Form’s form.control manages the state of inputs. Each <FormField> binds a field name to an input (<Input> from shadcn/ui, which is a Tailwind-styled text input). <FormMessage> will automatically show an error message if validation fails for that field. This setup ensures you get instant feedback and solid accessibility with minimal code. You can add validation rules via React Hook Form (for instance, requiring the fields or using a schema validator like Zod, as suggested in shadcn’s docs​

[tailkits.com](https://tailkits.com/blog/shadcnui-for-beginner/#:~:text=Forms%20are%20another%20area%20where,libraries%20like%20React%20Hook%20Form)

). Thanks to Tailwind, this form is also mobile-responsive (the classes used by shadcn/ui are mobile-first and you can add responsive utility classes as needed).

Beyond forms, shadcn/ui includes many other components (tables, dialogs, navigation bars, etc.) that you can use to build a rich dashboard. All of them use Radix under the hood for accessibility and are styled with Tailwind, so they will maintain a consistent look. You can customize any component’s appearance by editing the Tailwind classes in the component file (since shadcn/ui encourages copying the needed component code into your project). This gives a great balance between using a library and owning your code.

## **Developing the Application on Replit – Tips & Recommendations**

Using Replit can speed up development and testing of your Wix app dashboard. Here are some practical recommendations for building this project on Replit:

* **Project Setup:** Replit supports Node.js and React out-of-the-box. You can create a new Node.js Replit and initialize a React project (for example with Vite or Create React App). Install your dependencies in the Replit workspace: e.g. npm install react react-dom shadcn-ui radix-ui tailwindcss react-hook-form cors express. Since Replit provides a persistent environment, all these can be installed via the Replit console.
* **Tailwind Configuration:** Set up Tailwind CSS in your React project (Tailwind requires a config file and PostCSS processing). On Replit, ensure you run the build or start script that processes Tailwind. For instance, if using Vite, add the Tailwind plugin and run the dev server; if using CRA, ensure npm start runs the Tailwind compilation. Replit will host the dev server and give you a preview URL (usually something like https://<project>.<user>.repl.co). This URL is accessible over HTTPS, which is great for embedding in Wix during development.
* **Environment Variables:** Keep secrets (like your Wix app secret for verifying the instance token) out of your code. Replit has a **Secrets** management where you can add environment variables. Use that to store values like WIX\_APP\_SECRET and any other API keys. In your server code on Replit, access it via process.env.WIX\_APP\_SECRET. Replit will not show these values in the public code, keeping them safe.
* **Running Frontend & Backend:** You can serve both the frontend and backend from one Replit if you configure it. For example, you might use a single Express server that serves the built React app and also provides API routes. Alternatively, run the React dev server separately from an API server. If you do the latter, note that you’ll have two ports (one for frontend, one for backend). Make sure to configure CORS between them for local testing (similar to how we allowed localhost). Replit’s preview URL by default maps to one port (usually the one your Run command starts). An easier approach is to build the React app and have Express serve the static files (so everything is on one domain).
* **Testing the IFrame in Wix:** While developing, you can use a **Wix test site** and add your app (in Dev mode from Wix Developer Center) to that site. Point the app’s dashboard URL to your Replit’s URL. Since it’s HTTPS, Wix can load it. You might need to update your app’s allowed origin or redirect URI settings in the Wix Dev Center to include the Replit URL. Once configured, attempt to open the app in the Wix site dashboard. Use the browser dev tools to ensure the instance param is present and that your requests are going through. Debug any CORS errors by checking response headers in the Network tab. On Replit, you can check the Logs to see incoming requests to your server and verify the instance verification is working.
* **Keepalive and Performance:** Note that free Replit instances may sleep after some period of inactivity. When testing with Wix, you might need to “wake” your Replit by opening it, or consider using Replit’s always-on feature (if available) or a ping service during development. Also, Replit is fine for development and demonstration, but for production use you’ll likely want a more robust hosting for your app’s backend. Still, Replit is very handy to quickly prototype the app and even collaborate in real-time.
* **Iterate and Secure:** As you build, frequently test that navigating the Wix dashboard triggers your Replit server calls as expected. Each time Wix loads the IFrame, a new instance token is generated – ensure your app handles a new token each time (the user might refresh or reopen the app). If you encounter any authentication issues, double-check that your Replit backend isn’t missing required headers. For example, Wix might send the instance token in the URL by default; if you prefer it in headers, you have to implement that as shown earlier. Also double-check that the Replit domain is allowed by your server CORS (you might use a wildcard Access-Control-Allow-Origin during development for simplicity, then tighten it for production).

By following Wix’s security guidelines (verifying instance tokens and setting CORS properly) and leveraging the UI libraries mentioned, you can create a **secure, robust Wix dashboard extension**. The combination of **shadcn/ui + Radix + Tailwind + React Hook Form** will significantly speed up frontend development, giving you a professional-looking, responsive UI with less effort. Developing on Replit is an efficient way to get started and test your integration quickly before deploying your app for all Wix users.

**Sources:**

1. Wix Developers Documentation – *Iframe Dashboard Query Parameters*​  
   [dev.wix.com](https://dev.wix.com/docs/build-apps/develop-your-app/frameworks/self-hosting/supported-extensions/dashboard-extensions/iframe-query-parameters-for-dashboard-extensions#:~:text=Name%20Value%20Comments%20,hasn%27t%20seen%20this%20URL%20before)
2. Wix Developers Documentation – *About App Instance (Instance Token Structure)*​  
   [dev.wix.com](https://dev.wix.com/docs/build-apps/develop-your-app/access/app-instances/about-app-instances#:~:text=The%20encoded%20instance%20parameter%20is,parts%20separated%20by%20a%20dot)​  
   [dev.wix.com](https://dev.wix.com/docs/build-apps/develop-your-app/access/app-instances/about-app-instances#:~:text=Field%20Description%20,The%20Plan)
3. Wix Developers Documentation – *Instance Token Security and Usage*​  
   [dev.wix.com](https://dev.wix.com/docs/build-apps/develop-your-app/access/app-instances/about-app-instances#:~:text=When%20a%20user%20accesses%20your,ensure%20its%20integrity%20and%20authenticity)​  
   [dev.wix.com](https://dev.wix.com/docs/build-apps/develop-your-app/access/app-instances/about-app-instances#:~:text=Note%3A%20To%20check%20if%20the,is%20the%20same%20as%20the)​  
   [dev.wix.com](https://dev.wix.com/docs/build-apps/develop-your-app/access/app-instances/about-app-instances#:~:text=Warning%3A%20For%20dashboard%20security%2C%20restrict,uid)
4. Wix Developers Documentation – *Implementing a Backend for Your Wix App (CORS & Auth)*​  
   [dev.wix.com](https://dev.wix.com/docs/build-apps/develop-your-app/frameworks/wix-cli/app-development/implement-a-backend-for-your-app#:~:text=When%20making%20HTTP%20requests%20from,requests%20from%20your%20frontend%27s%20domain)​  
   [dev.wix.com](https://dev.wix.com/docs/build-apps/develop-your-app/frameworks/wix-cli/app-development/implement-a-backend-for-your-app#:~:text=Authenticate%20incoming%20requests)
5. Wix Help Center – *Embedding External Sites (HTTPS and framing requirements)*​  
   [support.wix.com](https://support.wix.com/en/article/wix-editor-embedding-a-site-or-a-widget#:~:text=,please%20contact%20the%20provider%20directly)
6. Tailkits Blog – *Introduction to shadcn/ui (Radix + Tailwind UI library)*​  
   [tailkits.com](https://tailkits.com/blog/shadcnui-for-beginner/#:~:text=interior%20pieces%E2%80%94customizable%20UI%20components%20that,instantly%20upgrade%20your%20website)​  
   [tailkits.com](https://tailkits.com/blog/shadcnui-for-beginner/#:~:text=Click%20me%20)
7. Tailkits Blog – *Using shadcn/ui Forms with React Hook Form*​  
   [tailkits.com](https://tailkits.com/blog/shadcnui-for-beginner/#:~:text=Forms%20are%20another%20area%20where,libraries%20like%20React%20Hook%20Form)
8. Radix UI Overview – *Accessible Headless UI Components*​  
   [refine.dev](https://refine.dev/blog/radix-ui/#:~:text=%60Radix%60%20is%20an%20open,to%20build%20components%20with%20Radix)
9. React Hook Form Documentation – *Overview and Benefits*​  
   [react-hook-form.com](https://react-hook-form.com/#:~:text=Super%20Light)