ProBridge

Cross-Chain Bridge UI Kit

Developer Documentation • v1.0

Stack	Technology
Frontend Framework	Next.js 16 + React 19
Web3 Layer	wagmi v2 + viem v2 + RainbowKit v2
Smart Contracts	Solidity 0.8.24, Hyperlane Warp Routes
State Management	Zustand + Redux DevTools
Animations	Framer Motion 12
Styling	Tailwind CSS v4

1. Overview

ProBridge is a full-stack cross-chain token bridge built on top of Hyperlane — the permissionless interoperability protocol. It lets users transfer ERC20 tokens and native ETH across EVM chains with real-time fee quotes, delivery tracking, and a complete admin panel.

The bridge flow works as follows:

- User calls transferRemote() on the source chain warp route contract
- Tokens are locked (collateral) or burned (synthetic) on the source chain
- Hyperlane Mailbox dispatches a cross-chain message with the recipient and amount
- Hyperlane validators sign the checkpoint
- The relayer delivers the message to the destination chain Mailbox
- Destination warp route mints or releases tokens to the recipient

> 💡 *ProBridge uses Mailbox.quoteDispatch() for fee estimation — no IGP required. This avoids gas limit overflow issues on testnets.*

2. Feature Reference

2.1 Bridge

- ERC20 bridging: HypERC20Collateral locks tokens → HypERC20 mints synthetic on destination
- Native ETH bridging: HypNative locks ETH → HypERC20Synthetic mints wETH on destination
- Real-time fee quotes from Mailbox.quoteDispatch()
- USD fee display with live ETH price via CoinGecko API
- Balance validation — checks wallet balance including fee for native tokens
- "Use Max" button fills maximum bridgeable amount
- Approve + Bridge 2-step flow with step indicators

2.2 Transaction History

- Persisted in localStorage, keyed per wallet address
- Cross-tab sync via StorageEvent listener
- Real-time delivery polling using ERC20 balance change detection
- Age-based fallback: txs older than 10 min are auto-resolved
- Expandable rows with amount, chains, date, recipient, explorer link
- Clear history with confirmation

2.3 Admin Panel

- Pause / Unpause contract (emergency stop for all transfers)
- Enroll remote router: link destination contract as trusted router
- Set ISM: update Interchain Security Module per contract
- Transfer ownership: hand off to multisig (Gnosis Safe recommended)
- Owner detection: all write actions are disabled for non-owners

2.4 Deployment Wizard (/deploy)

- Step 1 — Chains: add/remove chains with ID, RPC, explorer, color, testnet flag
- Step 2 — Contracts: warpRoute + mailbox + optional IGP/ISM per chain
- Step 3 — Tokens: symbol, decimals, native toggle, address per chain
- Step 4 — Generate: download 3 ready-to-use config files
- Live address validation and form guards on every step

3. Quick Start

3.1 Requirements

Tool	Version	Purpose
Node.js	18+	Run the Next.js dev server
npm / yarn	any	Package manager
MetaMask	any	Browser wallet for testing
WalletConnect ID	free	cloud.walletconnect.com

3.2 Installation

```
git clone <your-repo-url> probridge-ui
cd probridge-ui
npm install
cp .env.example .env.local
```

Edit .env.local and add your WalletConnect Project ID:

```
NEXT_PUBLIC_WALLETCONNECT_PROJECT_ID=your_project_id_here
npm run dev
```

Open http://localhost:3000 in your browser.

4. Configuration System

All chain, contract, and token configuration lives in 3 files under /config. You never need to touch component code to add a new chain or token — just edit config files and restart.

4.1 chains.config.ts

Defines the list of EVM chains available in the bridge UI. Each entry follows the ChainConfig interface:

```
export interface ChainConfig {
  id:              number;        // EVM chain ID
  name:            string;        // Full display name
  shortName:       string;        // Badge name (e.g. "Base")
  label:           string;        // Subtitle (e.g. "Sepolia Testnet")
  nativeCurrency: { name, symbol, decimals };
  rpcUrls:         string[];      // First = primary, rest = fallbacks
  explorerUrl:     string;        // Block explorer root URL
  explorerTxPath: string;         // Usually "/tx/"
  color:           string;        // Hex color for chain badge
  iconSvg?:        string;        // Inline SVG for chain icon
  testnet?:        boolean;
}
```

4.2 contracts.config.ts

Maps chainId → deployed contract addresses. warpRoute and mailbox are required. igp and ism are optional.

```
export const CONTRACTS_CONFIG: Record<number, ContractAddresses> = {
  84532: {
    warpRoute: "0xYOUR_HYP_ERC20_COLLATERAL",
    mailbox:   "0xHYPERLANE_MAILBOX",
    igp:       "0xYOUR_IGP",       // optional
    ism:       "0xYOUR_ISM",       // optional
  },
};
```

4.3 tokens.config.ts

Defines bridgeable tokens with addresses per chain. For native gas tokens (ETH/BNB), set isNativeGas: true and provide warpRouteAddress to override the default warpRoute lookup.

```
export const TOKENS_CONFIG: TokenConfig[] = [
  {
    symbol:   "USDC",
    name:     "USD Coin",
    decimals: 6,
    addresses: { 84532: "0xCOLLATERAL", 97: "0xSYNTHETIC" },
    coingeckoId: "usd-coin",
  },
];
```

5. Smart Contract Deployment

5.1 Contract Architecture

ProBridge uses Hyperlane Warp Routes — a standard interface for cross-chain token transfers. There are two contract types:

Contract	Chain	Role
HypERC20Collateral	Source	Locks existing ERC20 tokens on transfer, releases on receipt
HypERC20 (Synthetic)	Destination	Mints new tokens on message delivery, burns on return
HypNative	Source	Locks native ETH on transfer, releases on receipt
HypERC20 (wETH)	Destination	Synthetic ERC20 representing bridged ETH

5.2 Deployment Steps (Remix IDE)

Recommended: use Remix IDE (remix.ethereum.org). No local setup required.

Compiler settings: Solidity 0.8.24, EVM Version: Cancun

- Step 1: Deploy BlockBridgeIGP on the source chain (no constructor args)
- Step 2: Deploy BlockBridgeMultisigISM on the source chain (no constructor args)
- Step 3: Deploy HypERC20Collateral with (_mailbox, _token)
    - _mailbox: Hyperlane Mailbox address on source chain
    - _token: Address of the ERC20 to bridge (e.g. USDC on Base Sepolia)
- Step 4: Deploy HypERC20 (synthetic) on destination chain with (_mailbox, _name, _symbol, _decimals)
- Step 5: Call enrollRemoteRouter() on both contracts to link them

> ⚠ *enrollRemoteRouter() expects a bytes32 router — pad the 20-byte address with 24 leading zeros. Example: 0x000000000000000000000000{address_without_0x}*

5.3 Hyperlane Mailbox Addresses

Chain	Chain ID	Mailbox Address
Ethereum	1	0xc005dc82818d67AF737725bD4bf75435d065D239
BNB Chain	56	0x2971b9Aec44bE4eb673DF1B88cDB57b96eefe8a4
Arbitrum	42161	0x979Ca5202784112f4738403dBec5D0F3B9daabB9
Optimism	10	0xd4C1905BB1D26BC93DAC913e13CaCC278CdCC892
Base	8453	0xeA87ae93Fa0019a82A727bfd3eBd1cFCa8f64f1D
Polygon	137	0x5d934f4e2f797775e53561bB72aca21ba36B96BB
Base Sepolia (testnet)	84532	0x6966b0E55883d49BFB0b8E4AcEA2855b4C9Bf4e9
BNB Testnet	97	0xF9F6F5646F478d5ab4e20B0F910C92F1CCC9Cc6D

6. Adding a New Chain

Adding a chain is a 3-step config-only process:

Step 1: Add to chains.config.ts

```
// Inside CHAINS_CONFIG array:
{
  id:   42161,
  name: "Arbitrum One",
  shortName: "ARB",
  label: "Arbitrum Mainnet",
  nativeCurrency: { name: "Ether", symbol: "ETH", decimals: 18 },
```

```
  rpcUrls: ["https://arb1.arbitrum.io/rpc"],
  explorerUrl: "https://arbiscan.io",
  explorerTxPath: "/tx/",
  color: "#28A0F0",
}
```

Step 2: Add contracts to contracts.config.ts

```
42161: {
  warpRoute: "0xYOUR_WARP_ROUTE_ON_ARBITRUM",
  mailbox:   "0x979Ca5202784112f4738403dBec5D0F3B9daabB9",
},
```

Step 3: Add token address to tokens.config.ts

```
// Inside the USDC token entry:
addresses: {
  84532: "0x036CbD...",
  97:    "0x779508...",
  42161: "0xYOUR_USDC_ON_ARBITRUM",  // add this
},
```

> ☑ *That's it! The chain selector, balance reader, bridge flow, history, and admin panel all pick up the new chain automatically.*

7. Production Checklist

Item	Status
Replace testnet chains with mainnet chains in chains.config.ts	☐
Update all contract addresses to mainnet deployments	☐
Set NEXT_PUBLIC_WALLETCONNECT_PROJECT_ID in prod environment	☐
Verify contracts on block explorers (builds user trust)	☐
Transfer contract ownership to a Gnosis Safe multisig	☐
Test bridge with small amounts before announcing	☐
Deploy frontend to Vercel (npm run build && vercel --prod)	☐
Get a security audit before handling significant TVL	☐

8. Hook Reference

Hook	File	Purpose
useBridge	hooks/useBridge.ts	Executes approve + transferRemote with fee quoting
useBridgeHistory	hooks/useBridgeHistory.ts	localStorage history with cross-tab sync
useChains	hooks/useChains.ts	Returns configured chains + viem client factory
useTokenPrice	hooks/useTokenPrice.ts	CoinGecko USD price with 60s cache

9. Support

If you have questions or find a bug, please use the comments section on CodeCanyon. Include your Node.js version, browser, wallet, and a description of the issue.

For deployment help, the built-in /docs page inside the app covers every step with code examples and the /deploy wizard generates your config files automatically.

> *Built with on Hyperlane Protocol — the permissionless interoperability layer.*