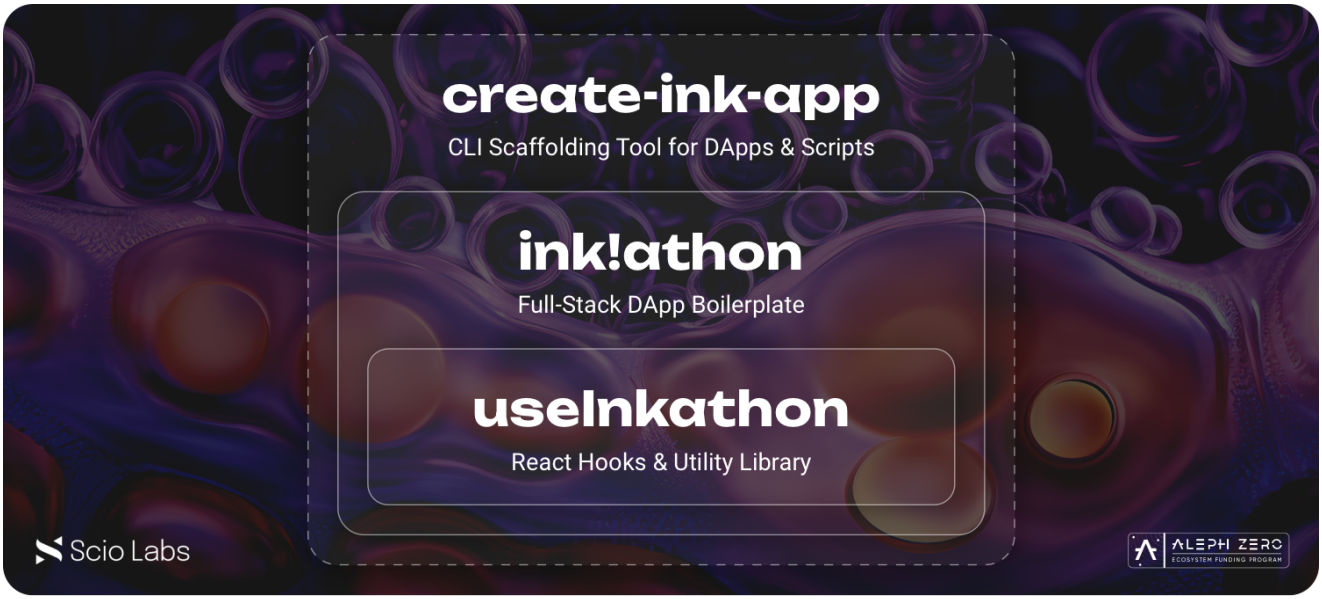
https://ui.use.ink/?rpc=wss://ws.test.azero.dev



ink!athon Boilerplate

License GPLv3
proudly built with ink!
Rust
TypeScript
Next.js

This is a full-stack dApp boilerplate for ink! smart contracts with an integrated frontend. It can be used to quickly start developing your hackathon idea or to scaffold a production-ready Web3 application.

The project is part of a Scio Labs initiative to improve the developer experience in the ink! ecosystem and a proud member of the Aleph Zero EFP. 💜

PROF

Other projects include:

- `create-ink-app` CLI (*Coming soon*)
- `ink!athon` Boilerplate
- `useInkathon` Hooks & Utility Library
- `zink!` Smart Contract Macros

Join the discussion in our Telegram Group 💬

If you want to contribute, please read our Contributor Guidelines 🙏

Table of Contents:

About 📖

The boilerplate comes with a small sample ink! `Greeter` contract which stores a `message` (the "greeting") and allows anyone to update it. The frontend contains simple UI components to connect your wallet and interact with the contract (i.e. read & write the `message`). Try it out live on inkathon.xyz.

Getting started 🚀

1. Run the frontend

The frontend works out of the box, without a local node running, as the sample contract is pre-deployed on certain live testnets (i.e. `alephzero-testnet` and `shibuya`). Necessary deployment metadata and addresses are provided under `contracts/deployments/`.

PROF

> **Pre-requisites:**
>
> - Setup Node.js v18+ (recommended via nvm with `nvm install 18`)
> - Install pnpm (recommended via Node.js Corepack or `npm i -g pnpm`)
> - Clone this repository

▶ **Special Instructions for Windows Users**

> [!IMPORTANT]
> Windows users must either use WSL (recommended) or a custom shell like Git Bash. PowerShell is not supported.

> **Pre-requisites when using WSL for Linux:**
>
> - Install WSL and execute *all* commands in the WSL terminal
> - Setup Node.js v18+ (recommended via nvm with `nvm install 18`)

- Install the following npm packages globally:
- `npm i -g npm`
- `npm i -g pnpm node-gyp make`
- Clone this repository into the WSL file system (e.g. `/home/<user>/inkathon`).

Tip: You can enter `\\wsl$\` in the top bar of the Windows Explorer to access the WSL file system visually.

```
# Install dependencies (once)
# NOTE: This automatically creates an `.env.local` file
pnpm install

# Start Next.js frontend
pnpm run dev
```

Optionally, to enable `simple-git-hooks` (for automatic linting & formatting when committing), you can run the following command once: `pnpm simple-git-hooks`.

2. Build & deploy contracts on a local node

The `contracts/package.json` file contains shorthand scripts for building, testing, and deploying your contracts.

Pre-requisites:

- Install Rust via the Substrate Docs (skip the "Compile a Substrate node" section)
- Install `cargo contract`
- Install `substrate-contracts-node`

```
# Set `contracts/` as the active working directory in your terminal
cd contracts

# Build contracts and move artifacts to
`contracts/deployments/{contract}/` folders
pnpm run build

# Start local node with persistence (contracts stay deployed after
restart)
# NOTE: When using Brave, shields have to be taken down for the UIs
pnpm run node

## IMPORTANT: Open a separate terminal window and keep the node running

# Deploy the contracts on the local node
pnpm run deploy
```

Alternatively, you can also deploy contracts manually using Contracts UI (`pnpm contracts-ui`) in the browser.

3. Connect the frontend to the local node

Open the `frontend/.env.local` file and set the `NEXT_PUBLIC_DEFAULT_CHAIN` variable to `development`. Then restart the frontend and you should be able to interact with the contracts deployed on your local node.

Read more about environment variables and all available chain constants in the Environment Variables section below.

Customization 🎨

1. Project Name

There are multiple places where you need to insert your project's name and identifier. Most of these occurrences are highlighted with a `/* TODO */` comment in the code. You can easily replace them one by one by installing the `todo-tree` plugin.

Additionally, there are the following un-highlighted occurrences:

- the name of the `inkathon.code-workspace` file
- the `package.json`'s name & metadata in the root directory as well as in the `contracts/` and `frontend/` packages
- the workspace dependency (`@inkathon/contracts`) defined in `frontend/package.json` and imported in `frontend/src/deployments/deployments.ts`

2. Custom Contracts

To replace the default `Greeter` contract or add a new one, you need to do the following:

- Add a new contract directory under `contracts/src/`
- Add it as another workspace member to the `contracts/Cargo.toml` file
- Add another deployment script or adjust `contracts/scripts/deploy.ts`
- Adjust the `ContractIds` enum and `getDeployments` function in `frontend/src/deployments/deployments.ts`

3. Custom Scripts

Adding custom scripts is useful to interact with your contracts or test certain functionality. Therefore, just duplicate & reuse the `contracts/scripts/script.template.ts` file and run it via `pnpm run script <script-name>`. This command will run the TypeScript file directly via `tsx`.

For general scripts, the same environment variable initialization & configuration applies as described below in the Deployment section (e.g. to change the target network).

The Stack 🥞

▶ **The Stack in Detail**

- Monorepo Workspace with `contracts/` and `frontend/` directories as packages.
- Package Manager: `pnpm` or `yarn@stable` (Read more in the FAQs section below)
- Smart Contract Development: Rust, ink!, `cargo-contract`, `substrate-contracts-node`
- Frontend: Next.js (app-dir), React, TypeScript
    - Contract Interactions: `polkadot-js`, `useInkathon` React Hooks & Utility Library (alternatively: `useInk`)
    - Styling: `shadcn/ui`, `tailwindcss`
    - Linting & Formatting: `eslint`, `prettier`, `simple-git-hooks`, `lint-staged`
- Type-safe contract generation via `typechain-polkadot`

Styling, linting, and formatting libraries can be fully dropped or replaced with alternatives.

Live Examples 🌐

Below you find live examples that use this boilerplate or have a similar setup inspired by it:

- inkathon.xyz – Live demo deployment of this boilerplate
- AZERO.ID – Domain Name Service for Aleph Zero and beyond
- Multiple hackathon projects from ETHWarsaw, HackOnChain, ETHDam, and the Polkadot ink! Hackathon.

PROF

Deployment 🚢

Spinning up a deployment via Vercel is pretty straightforward as the necessary settings are already configured in `vercel.json`. If you haven't cloned the repository yet, you can also use the **Deploy** button below to create a new repository from this template.

▲ Deploy

Alternatively, you can also use the provided Dockerfiles to deploy to any hosting provider of your choice. Read more here.

Environment Variables

All environment variables are imported from `process.env` in `frontend/src/config/environment.ts` and re-exported from there. For improved type safety, Always only import environment variables from `@/config/environment` and never directly from `process.env`.

Environment Variables	Default Values	Description
`NEXT_PUBLIC_DEFAULT_CHAIN` ✳	`alephzero-testnet`	The network (Substrate-based chain) the frontend should connect to by default and what contract deployment artifacts to import.
`NEXT_PUBLIC_PRODUCTION_MODE`	`false`	Optional boolean flag to differentiate production environment (e.g. for SEO or Analytics).
`NEXT_PUBLIC_URL`	`http://localhost:3000`	Optional string that defines the base URL of the frontend (will be auto-inferred from Vercel environment variables).
`NEXT_PUBLIC_SUPPORTED_CHAINS`	–	Optional array with network identifers (e.g. `["alephzero-testnet", "shibuya"]`) that are supported by the frontend, **if the dApp is supposed to be multi-chain**.

PROF ✳ Required

Supported Chains

One key element making this boilerplate so flexible is the usage of environment variables to configure the active network in the frontend. This is done by setting the `NEXT_PUBLIC_DEFAULT_CHAIN` variable in the `frontend/.env.local` file, or in the deployment settings respectively.

If your network is not provided by the `use-inkathon` library, you can add it manually by creating a new `SubstrateChain` object. If you think a chain is missing, please open an issue or PR.

> [!IMPORTANT]
> All supported chain constants can be found here in the `scio-labs/use-inkathon` repository.

Contract Deployment

In the Getting Started section above, we've already deployed the sample `Greeter` contract on a local node. To target a live network, we can use the `CHAIN` environment variable when running the `deploy` script.

```
CHAIN=alephzero-testnet pnpm run deploy
```

Further, dynamically loaded environment files with the `.env.{chain}` naming convention can be used to add additional configuration about the deployer account.

```
# .env.alephzero-testnet
ACCOUNT_URI=bottom drive obey lake curtain smoke basket hold race lonely
fit walk//Alice
```

When running the same script again, this deployer account defined there will be used to sign the extrinsic.

> [!WARNING]
> These files are gitignored by default, but you should still be extra cautious when adding sensitive information to them.

VSCode Setup 🛠

Workspace

It can be helpful to develop in VSCode by opening the workspace file `inkathon.code-workspace` instead of just the plain directory. This approach offers multiple advantages, like sections in the file explorer, or shortcut actions to open the terminal in the correct directory.

Consider installin the `zoma.vscode-auto-open-workspace` extension to automatically open the workspace file when opening the directory.

PROF

Plugins

Additionally, the VSCode plugins listed below are recommended as they can be very helpful when working with this boilerplate.

▶ **All Recommended Plugins**

Plugin Name	Description
`dbaeumer.vscode-eslint`	Adds ESLint editor support.
`esbenp.prettier-vscode`	Adds Prettier editor support.
`bradlc.vscode-tailwindcss`	Adds tailwindcss editor support.
`rust-lang.rust-analyzer`	Adds Rust language support.

Plugin Name	Description
`ink-analyzer.ink-analyzer`	Adds ink! language support.
`tamasfe.even-better-toml`	Adds `.toml` file support.
`gruntfuggly.todo-tree`	Lists all `TODO` comments in your workspace.
`wayou.vscode-todo-highlight`	Highlights `TODO` comments in your workspace.
`mikestead.dotenv`	Adds syntax highlighting for `.env` files.

DRink! CLI Usage 💧

The DRink! CLI is a convenient command line tool that helps you to play with your ink! contracts without setting up a local node.

1. Install `drink-cli` via `cargo install drink-cli --force --locked`.
2. Build your contracts via `pnpm run build`.
3. Run the following command to prepare & open the CLI for your contract: `pnpm run drink-cli <contract-name>`.

Then, just use the `help` command to see all available commands and start interacting with your contract. For example, you can deploy the greeter example contract via `d --constructor default` or `d "Hello World"`.

FAQs & Troubleshooting 💬

▶ **Which package managers are supported? Do I have to use pnpm?**

For monorepo workspaces, pnpm is likely the fastest and most reliable choice. When using it though, it's strongly recommended everyone on the team uses it. No installs should be performed nor any other lock files should be committed.

As an alternative, yarn is also supported and can be used for installation. Caveats when using yarn:

PROF

- Only the stable version of yarn (currently v3) is supported, not yarn classic (v1).
- `yarn.lock` files should be committed instead of `.pnpm-lock.yaml` files.
- The `pnpm` CLI is still used in many `package.json` scripts, so these would have to be adjusted manually.

> [!IMPORTANT]
> As npm lacks support for the `workspace` import protocol, it's not compatible with ink!athon.

▶ **How to solve `Cannot find module './greeter/development.ts'`?**

Sometimes, Next.js doesn't pick up changes (i.e. file creations) in the `contracts/deployments/{contract}/` folders correctly. E.g., when you just deployed on a local node for the first time and set the frontend's `.env.local` to connect to the `development` network.

To fix this, you can delete the build cache at `frontend/.next`. This is currently the only solution and will force Next.js to rebuild the project and pick up the new files.

> [!NOTE]
> To prevent this behavior, the `contracts/package.json` file contains a small `postinstall` script that creates an empty `development.ts` file if none exists.

▶ **How to approach styling?**

Currently it offers styling via the following options out of the box:

- shadcn/ui - Re-usable components built using Radix UI and Tailwind CSS.
- Vanilla Tailwind CSS styled styles via `className` and `*.module.(s)css` files.
- Default (S)CSS styles.

> [!INFO]
> This boilerplate tries to stay as un-opinonated in regards to styling, which means you can use any styling or component library.

▶ **How do type-safe contract interactions work?**

With `typechain-polkadot`, types for each contract (TypeScript files) are created upon build (via the `build` script or `build-all.sh` command). You can suppress this behavior by passing `--skip-types`.

They are stored under `contracts/typed-contracts/` and imported directly from the frontend. Then, via the new `useRegisteredTypedContract` hook from `useInkathon` instances with pre-defined api, network-dependant contract address, and injected signer are being created. See `greeter-contract-interactions.tsx` for an example.

▶ **Resources to learn more about Substrate, ink!, and polkadot.js**

- ink! Documentation
- polkadot.js Documentation
- Polkadot Wiki ink! Tools
- Aleph Zero Documentation
- ink!athon Workshop Recording
- ink!athon Telegram Group