Algorand dApp Quick Start Guide

This guide will help non-technical founders quickly build and test Web3 ideas on Algorand. By following these steps, you'll be able to set up your project, create a landing page, mint NFTs, and deploy fungible tokens with ease.

1. Getting Started

> **Follow the setup steps outlined in the original README** OR in your forked repo's README to get started!
>
> Check out this demonstration video of the entire process if you prefer a visual walkthrough → Demo of entire process.mp4

2. Designing your Home page

Your home page is the first impression of your dApp - it's where users understand what your project does and how to get started. This section will help you design a simple, modern landing page.

1. Open the file `src/Home.tsx`.
2. Copy its contents, then open an AI tool such as ChatGPT, Claude, or Gemini.
3. Paste the existing `Home.tsx` code into the AI chat.
4. Write your design prompt to improve the layout and style of the page.
5. Try using a prompt like the one below to create a visually appealing layout:

```
None

I'm building an Algorand dApp and want to improve the design of my landing page
in src/Home.tsx. Please redesign the layout using modern web design principles
with TailwindCSS. Include a visually striking hero section, a short headline
and subheading, a call-to-action button, and a clean layout that feels
professional and engaging. Use balanced spacing, responsive design for both
desktop and mobile, and ensure colors and typography match a Web3 or tech-style
theme. Maintain all existing logic for wallet connection, navigation, and
button states exactly as they are — do not change any of the logic.
```

3. Sending ALGO and USDC

Inside the app, you can test simple wallet transactions. The `Transact` component allows users to send either ALGO or USDC between accounts — perfect for understanding how payments work on Algorand.

1. Open the file `src/components/Transact.tsx`.
2. Copy its contents and open an AI tool such as ChatGPT, Claude, or Gemini.
3. Paste the code into the AI and use a prompt to design how this feature could look for your own business idea - for example, a simple payments or tipping app.

None

```
I'm building a payments dApp on Algorand that allows users to send ALGO or USDC to others. I've pasted the existing Transact.tsx code, which already contains the transaction logic. Please redesign this component using TailwindCSS to look like a clean, modern payment interface. Include clear input fields for amount, recipient address, and a send button. Add helpful labels, a simple success message after transactions, and use a minimal Web3 design aesthetic. Keep all wallet and transaction logic exactly as it is — do not change any of the logic.
```

4. Minting Tokens

The `Tokenmint` component allows you to mint your own Algorand Standard Assets (ASAs). These can represent fungible assets such as stablecoins, loyalty points, or community tokens. You can set the token name, total supply, and decimals to define how it behaves.

1. Open the file `src/components/Tokenmint.tsx`.
2. Copy its contents and open an AI tool such as ChatGPT, Claude, or Gemini.
3. Paste the code into the AI and use a prompt to help you redesign this feature for your own business idea — for example, creating a loyalty points system or branded token.

```
None
I'm building a loyalty rewards dApp on Algorand that lets businesses create
their own token to reward customers. I've pasted the existing Tokenmint.tsx
code, which already includes the logic for minting ASAs. Please redesign this
component using TailwindCSS to make it look like a professional token creation
form. Include input fields for token name, symbol, total supply, and decimals,
and a clear Mint Token button. Use a clean, modern style that would fit a
startup dashboard. Keep all the minting and wallet logic exactly as it is — do
not change any of the logic.
```

5. Minting NFTs

To mint NFTs with IPFS metadata, you'll use the `NFTmint` component. This allows users to upload images or files, automatically generate metadata, and mint their NFTs directly on Algorand.

Environment Setup for NFTs

1. Navigate to the folder: QuickStartTemplate-contracts/nft_mint_server
2. In that folder you will find a file called .env.template.
3. Copy the contents of .env.template.
4. Create a new file in the same folder and name it .env.
5. Go to Pinata (https://app.pinata.cloud/developers/api-keys) and create a new API Key (set to Admin).
6. Copy the API Key and Secret Key and paste them into your new .env file, replacing the placeholders.
7. Once your environment file is ready, restart the project with `npm run dev`.

The `NFTmint` interface will now allow you to upload files, add metadata, and mint NFTs linked to IPFS.

1. Open the file `src/components/NFTmint.tsx`.
2. Copy its contents and open an AI tool such as ChatGPT, Claude, or Gemini.
3. Paste the code into the AI and use a prompt to help you redesign this component for your own NFT use case - such as a digital art collection, membership card, or event ticket.

```
None
I'm building an Algorand-based NFT dApp that allows users to mint digital
collectibles. I've pasted the existing NFTmint.tsx code, which already includes
the NFT minting logic. Please redesign the layout using TailwindCSS for a sleek
and modern look. Include clear upload and input fields for image, name, and
description, as well as a visible Mint NFT button. Add small visual feedback
for upload and minting progress. Keep all existing wallet, IPFS, and minting
logic exactly as it is — do not change any of the logic.
```

6. Smart Contract Interaction

The `AppCalls.tsx` component demonstrates how to connect your frontend to a deployed Algorand smart contract. By default, the template includes a simple "Hello World" contract so you can see how contract calls work.

You can find the default smart contract inside the backend folder:

`projects/QuickStartTemplate-contracts/smart_contracts/hello_world/contract.algo.ts`
This file contains the example TypeScript smart contract generated by AlgoKit.

This section is meant to help you understand how to integrate your own business logic in the future, for example, saving user data, creating leaderboards, or automating payments.

To explore how to build and deploy your own contracts, visit the official Algorand developer resources:

- **Algorand Developer Portal:** https://dev.algorand.co/
  Learn how to build, deploy, and interact with smart contracts using official documentation, tutorials, and SDK examples.

- **AlgoKit Workshops:** https://algorand.co/algokit-workshops
  Follow guided workshops that teach you how to use AlgoKit to generate, test, and deploy smart contracts.

- **Algodev YouTube Channel:** https://www.youtube.com/@algodevs
  Watch video tutorials, walkthroughs, and community sessions covering AlgoKit, smart contracts, and other Algorand developer topics.

These resources are the best next step for deepening your technical knowledge and expanding what your dApp can do beyond the starter template.

Local Setup (Optional – if Codespaces doesn't work)

If you prefer to run the project locally, or experience issues with GitHub Codespaces, follow the steps below. This option is slightly more technical but gives you full control of your environment.

1. Install Visual Studio Code
Download and install VSCode from https://code.visualstudio.com.

2. Clone the Repository
Open your terminal and run:

```
git clone
https://github.com/Ganainmtech/Algorand-dApp-Quick-Start-Template-TypeScript.git
```

3. Open the Project in VSCode
Once cloned, open the project folder in VSCode.

4. Continue Setup from the README
From this point, follow along in the project's README for the full setup and environment configuration steps:

Follow setup steps in README →