Startup Challenges

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

Dispenser Links
Testnet ALGO dispenser: Algorand dispenser
Testnet USDC dispenser: Testnet Faucet | Circle

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.

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.

7. Deploying Your Frontend (aka, making your website live!)

- This part gets your dApp's face (the frontend) up and running in minutes.

- The backend stuff (for NFT minting) is totally optional.
- REC: Watch this 2 minute video of the **Frontend deploy** process here.
- REC: Watch this 2 minute video of the **Backend (for NFTs) deploy** process here.

Step 1: Get Your Vercel Project Ready

- Head over to Vercel, sign up, and log in with your GitHub. Easy peasy!
- Start a "New Project" and "Import Git Repository."
- Choose your project's fork (it should pop up automatically).
- For the "Root Directory," put: `QuickStartTemplate/projects/QuickStartTemplate-frontend`
- Leave the "Framework Preset" as "Other."

Step 2: Build & Output Settings (just copy and paste these!)

- **Build Command:** `bash ../../../vercel_build.sh`
- **Output Directory:** `dist`

Step 3: Environment Variables for Your Frontend (important stuff!)

#Required:
`VITE_ENVIRONMENT=local`
#TestNet Configuration (for testing things out):
`VITE_ALGOD_SERVER=https://testnet-api.algonode.cloud`
`VITE_ALGOD_NETWORK=testnet`
`VITE_INDEXER_SERVER=https://testnet-idx.algonode.cloud`
#Backend (Optional - only if you want NFT minting):
`VITE_API_URL=`

(Leave this blank for now if you're not doing NFTs yet!)

- Hit "Deploy" and watch the magic happen!

7.5 Deploying Your Backend (Only if you're doing NFTs, otherwise skip!)

- This is *only* if you want users to be able to upload images and mint NFTs. Your frontend will still work without it.
- Watch this 2 minute video of the **Backend (for NFTs) deploy** process here.

Step 1: Get Your Vercel Project Ready for the Backend

- Same as before: Vercel → Sign up → Login via GitHub.
- "New Project" → "Import Git Repository."
- Select your fork.
- For the "Root Directory" this time, it's:

`QuickStartTemplate/projects/QuickStartTemplate-contracts/nft_mint_server`

- Keep "Framework Preset" as "Other."

Step 2: Environment Variables for Your Backend (for Pinata!)

- You'll need your Pinata API Keys for this. Get them at https://app.pinata.cloud/developers/api-keys.
- **Pinata Credentials (Required):** (replace with your actual key & secret!)

```
PINATA_API_KEY=YOUR_PINATA_API_KEY
PINATA_API_SECRET=YOUR_PINATA_API_SECRET
```

Step 3: Deploy & Test Your Backend

- Click that "Deploy" button! 🚀
    a. [Optional] Once it's live, open `https://YOUR-BACKEND-URL.vercel.app/health` in your browser.
    b. You should see something like: `{ "ok": true, "ts": ... }` if it's all good.
    c. Congrats, your backend is live!

Step 4: Link Your Backend to Your Frontend

- Go back to your *frontend* project in Vercel.
- Find or add this environment variable:

```
VITE_API_URL=https://YOUR-BACKEND.vercel.app
```

(make sure to use your *actual* backend URL here!).

- Redeploy your frontend (and clear the cache just to be safe).

Now, your NFT minting should be fully functional with your hosted backend! 🎉

Bonus Steps:
Making it Public!
Go to: Settings → Deployment Protection → Disable Vercel Authentication (anyone can view).

Enabling analytics:
Then go to Settings → Analytics → Click “Enable Analytics → possibly need to redeploy.

- We already baked into the template what is needed to view analytics from the dashboard

Local Setup (Optional – if you want more control)

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 →

(Deeper instructions on Localsetup to be added)

DevConnect Prompts

Algoday: Build & Vibe - Prompt Engineering Guide

This section shows how to quickly improve your **Algorand dApp Quick Start Template** using **AI tools** like ChatGPT, Claude, or Gemini.
Each prompt builds on the same `src/Home.tsx` file to enhance design, layout, and clarity for your project's landing page.

Prompt 1 – Professional & Polished Landing Page

Goal: Give your project a clean, credible, and startup-ready look.

Instructions:

1. Open `src/Home.tsx`
2. Copy its contents and paste them into your chosen AI chat tool.
3. Paste and run this prompt:

```
TypeScript
I'm building an Algorand dApp using React + TailwindCSS. I will paste my current src/Home.tsx file below. Please return the complete updated file only — no extra setup steps or external files.
Goals:
1. Keep ALL existing logic, imports, and functionality (wallet connection, navigation, handlers). Do NOT change or remove any logic.
2. Improve the visual design for a clean, professional, and modern landing page look.
3. Add a light/dark mode toggle in the top-right beside the wallet connect button. Manage it with simple local state inside this component.
4. Include a subtle animated background (for example, a smooth gradient or wave motion) that looks polished but remains fully contained in this file.
5. Ensure **all text, icons, and buttons are clearly visible and have sufficient contrast in both light and dark mode**.
6. Use consistent Tailwind classes (rounded corners, spacing, typography) and make sure it's fully responsive.
Style direction:
- Modern Web3/startup aesthetic
```

```
- Smooth motion, professional balance
- Both light and dark themes must look equally clear and elegant
```

Prompt 2 – Add a Complete Startup-Style Footer

Goal: Add a professional footer that makes the page feel like a real startup landing site.

Instructions:

1. Open `src/Home.tsx`
2. Copy its contents and paste them into your chosen AI chat tool.
3. Paste and run this prompt:

```
TypeScript
I'm building an Algorand dApp using React + TailwindCSS. I will paste my current src/Home.tsx file below. Please return the complete updated file only — no extra setup steps or external files.
Goals:
1. Keep ALL existing logic, imports, and functionality exactly as they are (wallet connection, navigation, modals, handlers). Do NOT change or remove any logic.
2. Add a professional footer section at the bottom of the page that matches the existing design.
3. The footer should include: - A short "About" text placeholder (1–2 sentences) describing the project or mission. - Social media icon links (Twitter/X, LinkedIn, GitHub, Discord, Telegram) using placeholder URLs. - A small "Built on Algorand" line or tag. - A set of quick navigation links (Home, Learn, Contact, Docs) — placeholder anchors are fine. - A copyright line such as "© 2025 Algorand Builders. All rights reserved."
4. Ensure the footer design is balanced, readable, and looks polished in both light and dark mode.
5. Keep all changes fully self-contained in Home.tsx using TailwindCSS classes only. It must remain fully responsive.
Style direction:
- Clean, modern startup feel
- Subtle divider or border separating the footer
- Layout adapts nicely between desktop (multi-column) and mobile (stacked)
```

Prompt 3 – Add a Waitlist / Feedback Form

Goal: Add an interactive, frontend-only waitlist or feedback form that feels real and user-friendly.

In a real dApp, this form would connect to a backend or service like Formspree or Vercel API routes to collect submissions.

Instructions:

1. Open `src/Home.tsx`
2. Copy its contents and paste them into your chosen AI chat tool.
3. Paste and run this prompt:

```
TypeScript
I'm building an Algorand dApp using React + TailwindCSS.
I will paste my current `src/Home.tsx` file below. Please return the full updated
file only – no external files or backend setup.
Goals:
1. Keep ALL existing logic, imports, and functionality (wallet connection,
navigation, modals, handlers). Do NOT change or remove any logic.
2. Add a "Join Waitlist" or "Send Feedback" section near the bottom of the page,
above the footer.
3. The form should include:
   - Name and Email input fields
   - A Submit button
   - On submit, show a short success message (e.g. "Thanks for joining our
waitlist!") and clear the fields.
4. This should be a **frontend-only demo** – no backend or external services
required.
5. Keep the design simple, responsive, and aligned with the rest of the site. It
should look polished in both light and dark mode.
6. Use TailwindCSS classes only. Keep all logic inside `Home.tsx`.
Style direction:
- Clean and modern layout
- Feels functional and realistic
- Works well in both light and dark themes
```

→ **Next Step: Ready to deploy?** See the guide for 7. Deploying Your Frontend (aka, making your website live!)