StrategyFlow

Visual Low-Code Strategy Builder for Kraken

Kraken Forge Hackathon 2025

Track #3: Strategy Builder

Author: Maurice Boendermaker

Live demo: https://strategyflow.dev

1. Summary

StrategyFlow is a production-quality, low-code trading strategy builder designed for Kraken. It enables traders to visually compose, simulate, and execute strategies using drag-and-drop blocks. No coding required. Built with React, Next.js, and TypeScript, it integrates directly with Kraken's authenticated API endpoints.

Table 1.1 - Overview of important details about the project

Item	Details
Track	#3 - Strategy Builder
Tech stack	React 19.2, Next.js 16, TypeScript 5, React Flow, Tailwind CSS 4
Live demo	https://strategyflow.dev
License	MIT License
Author	Maurice Boendermaker

In the table below is a list of all available page URLs (routes) within the application. They also work on localhost when the app is running. If running locally is preferred, replace the domain name with http://localhost:3000/

Table 1.2 - Overview of all available URL routes in the application

Page	Route
Landing page	https://strategyflow.dev/
About	https://strategyflow.dev/about
Docs	https://strategyflow.dev/docs
Editor	https://strategyflow.dev/editor
Export	https://strategyflow.dev/export
Settings	https://strategyflow.dev/settings

Simulator	https://strategyflow.dev/simulator
Strategies	https://strategyflow.dev/strategies
Templates	https://strategyflow.dev/templates
Validation	https://strategyflow.dev/validation
Version history	https://strategyflow.dev/version-history

2. Problem statement

2.1 Current bottlenecks

Building automated trading strategies on crypto exchanges presents significant challenges that prevent many traders from effectively implementing their ideas:

- **Technical barriers**: Traditional strategy development requires extensive programming knowledge in languages like Python or JavaScript, excluding non-technical traders.
- **API complexity**: Kraken's API authentication requires proper *HMAC-SHA512* signature generation, nonce handling, and secure credential management.
- **No visual feedback**: Code-based strategies lack visual representation, making it difficult to understand, debug, and communicate strategy logic.
- **Testing difficulties**: Safely testing strategies before live deployment typically requires custom simulation infrastructure.

2.2 Solution approach

StrategyFlow addresses these challenges by providing a visual interface where traders can drag and drop pre-built blocks that abstract away API complexity, handle authentication automatically, and provide simulation capabilities before live execution.

3. Key features

Table 3.1 - Core features and their descriptions

Feature	Description
Visual editor	Drag-and-drop node editor powered by React Flow with intuitive block connections and real-time visual feedback
Kraken API integration	Full integration with AddOrder, CancelOrder, AmendOrder, and batch operations using secure HMAC-SHA512 authentication
Execution simulator	Step-by-step dry-run mode with block highlighting, input/output inspection, and error visualization
Template gallery	Pre-built strategies: DCA, Stop-Loss, Grid Trading, RSI Momentum, and more for quick start
Inspector panel	Configure block parameters with validation, real-time preview, and contextual help
Flow debugger	Inspect inputs, outputs, errors, and execution timing for each node during runs
Export system	Export strategies as JSON definitions or TypeScript SDK code skeletons for integration

4. Technical architecture

4.1 Technology stack

- **Front-end**: React 19.2 with Next.js 16 for server-side rendering and API routes
- **Type safety**: TypeScript 5 with strict mode enabled for compile-time guarantees
- **Visual editor**: React Flow for node-based canvas with virtualization support
- **Styling**: Tailwind CSS 4 for utility-first styling, Radix UI for accessible components
- **Authentication**: crypto-js for HMAC-SHA512 signature generation per Kraken API specs

4.2 Design decisions

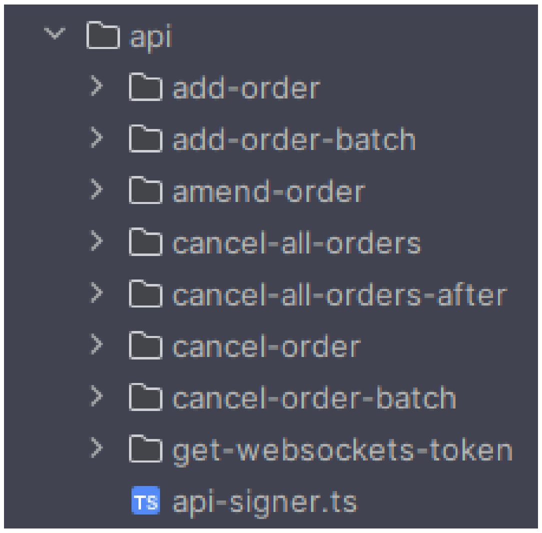
- **Modular components**: Each node type (Order, Condition, Logic, Utility) is a separate file for maintainability and reusability
- **Server-side API routes**: Kraken credentials handled in Next.js API routes, never exposed to client-side code
- **Type-safe blocks**: TypeScript discriminated unions for exhaustive pattern matching across block types
- **Removable landing page**: Demo landing page can be easily removed for production integration

5. API route implementation

5.1 Proxy layer

StrategyFlow uses Next.js API routes as a server-side proxy layer between the client application and Kraken's REST API. This architecture serves two critical purposes: it keeps API credentials secure by ensuring they never reach the browser, and it eliminates *Cross-Origin Resource Sharing (CORS)* restrictions that would otherwise block direct client-to-Kraken requests.

The `/front-end/app/api` directory contains all proxy routes, each corresponding to a specific Kraken endpoint. When a strategy block executes an order action, it calls the local Next.js route (e.g., `/api/add-order`), which then constructs the authenticated request server-side and forwards it to Kraken.



```
api
  add-order
  add-order-batch
  amend-order
  cancel-all-orders
  cancel-all-orders-after
  cancel-order
  cancel-order-batch
  get-websockets-token
  api-signer.ts
```

Figure 5.1.1 - Project structure showing the API routes directory with endpoints.

5.2 Authentication flow

Kraken's private API endpoints require *HMAC-SHA512* signatures for authentication. The signature generation is centralized in `api-signer.ts`, which implements Kraken's specific signing algorithm:

```
import CryptoJS from "crypto-js";

export function getKrakenSignature(
 path: string,
 data: Record<string, string | number>,
 privateKey: string
): string {

 const postData = new URLSearchParams(
   Object.entries(data).map(([k, v]) => [k, String(v)])
 ).toString();

 // 2. SHA256(nonce + postData)
 const nonce = data.nonce;
 if (!nonce) {
   throw new Error("Nonce is required for Kraken signature");
 }

 const sha256Hash = CryptoJS.SHA256(
   nonce.toString() + postData
 );

 // 3. Message = path + sha256Hash (binary)
 const message = CryptoJS.enc.Utf8.parse(path).concat(sha256Hash);

 // 4. Decode private key from base64
 const secret = CryptoJS.enc.Base64.parse(privateKey);

 // 5. HMAC-SHA512
 const hmac = CryptoJS.HmacSHA512(message, secret);

 // 6. Base64 encode signature
 return CryptoJS.enc.Base64.stringify(hmac);
}
```

Figure 5.2.1 - The getKrakenSignature function implements Kraken's HMAC-SHA512 authentication protocol, combining the API path, nonce, and request body into a cryptographic signature.

The signing process follows Kraken's specification: first, the nonce and URL-encoded POST data are concatenated and hashed with SHA256. This hash is then combined with the API path and signed using *HMAC-SHA512* with the base64-decoded private key. The resulting signature is base64-encoded for transmission in the `API-Sign` header.

5.3 Route handler pattern

Each API route follows a consistent pattern. Below is the implementation for the AddOrder endpoint:

```
import { NextRequest, NextResponse } from "next/server";
import { getKrakenSignature } from "../api-signer";

const API_URL = process.env.API_URL;
const ADD_ORDER_PATH = "/0/private/AddOrder";

export async function POST(req: NextRequest) {
 try {
   const params = await req.json();
   const API_KEY = process.env.KRAKEN_API_KEY;
   const PRIVATE_KEY = process.env.KRAKEN_PRIVATE_KEY;

   if (!API_KEY || !PRIVATE_KEY) {
     return NextResponse.json(
       { error: "Missing Kraken API credentials" },
       { status: 500 }
     );
   }

   const nonce = Date.now().toString();
   const body: Record<string, string> = { nonce, ...params };
   const signature = getKrakenSignature(ADD_ORDER_PATH, body, PRIVATE_KEY);

   const response = await fetch(`${API_URL}${ADD_ORDER_PATH}`, {
     method: "POST",
     headers: {
       "Content-Type": "application/x-www-form-urlencoded",
       "API-Key": API_KEY,
       "API-Sign": signature,
     },
     body: new URLSearchParams(body).toString(),
   });

   const data = await response.json();
   return NextResponse.json(data);
 } catch (err: any) {
   return NextResponse.json({ error: err.message }, { status: 500 });
 }
}
```

Figure 5.3.1 - The AddOrder route handler demonstrates the standard pattern: validate credentials, generate nonce, sign request, forward to Kraken, and return response.

The remaining routes in the `/api` directory (including cancel-order, amend-order, add-order-batch, cancel-order-batch, cancel-all-orders, and cancel-all-orders-after) follow this identical pattern, differing only in their Kraken endpoint paths and any endpoint-specific parameter handling.

5.4 Security considerations

This proxy architecture ensures that `KRAKEN_API_KEY` and `KRAKEN_PRIVATE_KEY` environment variables remain server-side only. The client application sends order parameters to the local `/api/*` routes without any knowledge of the actual credentials. Additionally, since these routes run on the same origin as the frontend, no CORS configuration is required.

6. Block reference

6.1 Order actions

Table 6.1.1 - Available order blocks and their corresponding Kraken API endpoints

Block	Description	Kraken endpoint
Place order	Submit buy/sell order	`/0/private/AddOrder`
Cancel order	Cancel existing order by ID	`/0/private/CancelOrder`
Amend order	Modify existing order params	`/0/private/AmendOrder`
Batch add	Submit multiple orders	`/0/private/AddOrderBatch`
Batch cancel	Cancel multiple orders	`/0/private/CancelOrderBatch`

6.2 Other block categories

- **Conditions**: Price thresholds, balance checks, time-based triggers, portfolio percentage conditions
- **Utilities**: Delay, log output, calculate values, format data, variable storage
- **System**: Start/End markers, loop constructs, error handlers, webhooks

7. Installation & Setup

7.1 Prerequisites

- Node.js 18+ and npm
- Kraken Pro account with API credentials
- Git for cloning the repository

7.2 Installation steps

```
# Clone the repository
git clone

https://github.com/MauriceBoendermaker/kraken_forge_hackathon_2025.git

cd kraken_forge_hackathon_2025/front-end

# Install dependencies
npm install

# Configure environment
cp .env.example .env

# Start development server
npm run dev
```

7.3 Environment variables

Table 7.3.1 - Required environment variables for .env

Variable	Description
`KRAKEN_API_KEY`	Your Kraken API public key
`KRAKEN_PRIVATE_KEY`	Your Kraken API private key (base64 encoded)
`API_URL`	https://api.kraken.com

8. User guide

8.1 How to get started

1. **Open the editor**: Navigate to `/editor` to access the visual strategy builder
2. **Add blocks**: Drag blocks from the sidebar categories (Order Actions, Conditions, Utilities, System)
3. **Connect nodes**: Click and drag between node handles to define execution flow
4. **Configure parameters**: Select a node and use the Inspector panel on the right to set values
5. **Test strategy**: Use the Simulator at `/simulator` to dry-run before live execution
6. **Execute**: Click "*Run Strategy*" to execute against Kraken's live API

9. API integration

9.1 Authentication flow

StrategyFlow implements Kraken's authentication protocol using HMAC-SHA512 signatures. The flow is handled server-side via Next.js API routes, ensuring credentials are never exposed to the browser. Each request includes a unique nonce (timestamp), API key, and computed signature.

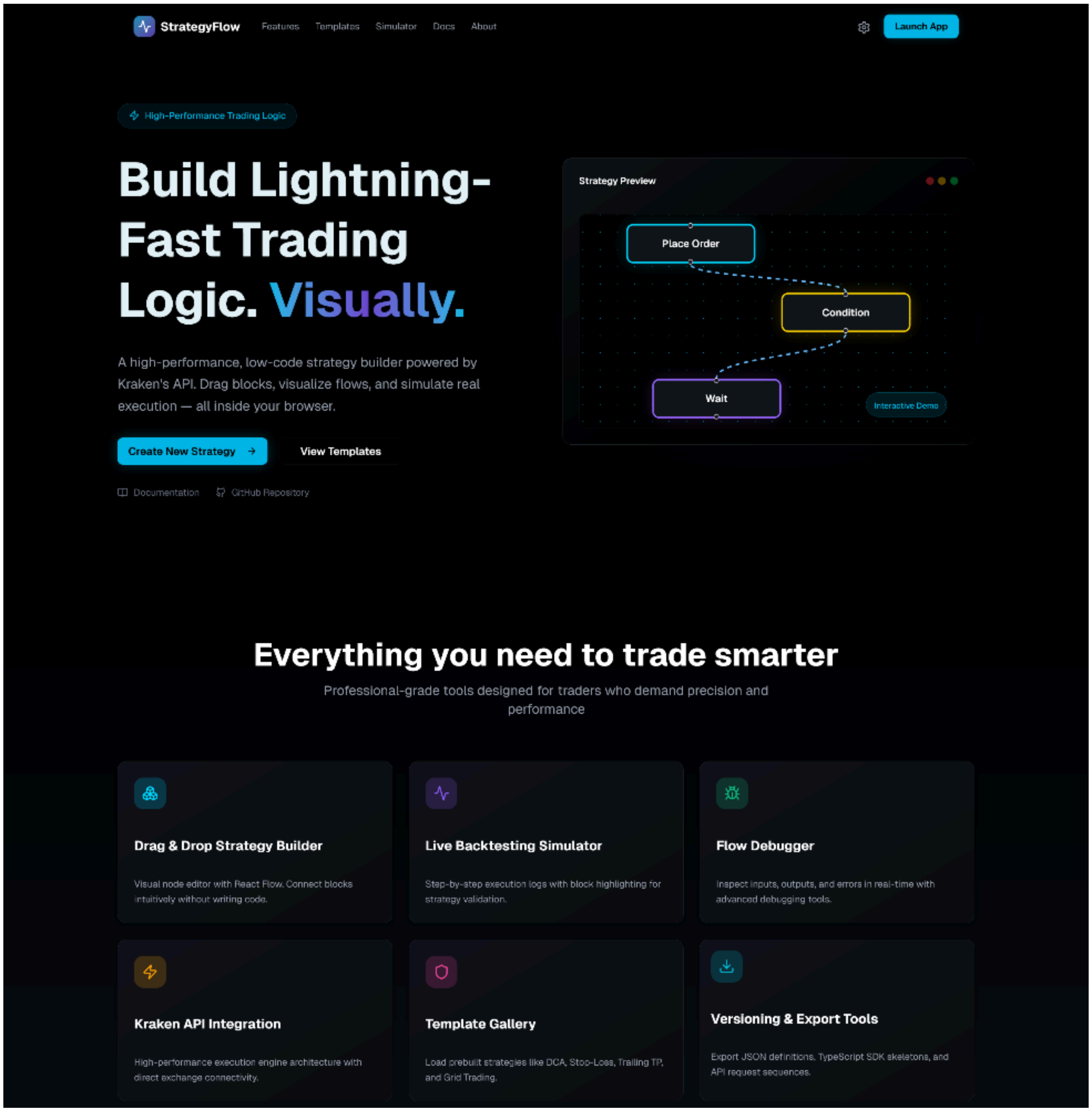
9.2 Demo note

Important: The live demo at strategyflow.dev connects to a Kraken Pro account *without* a funded balance for safety. When executing orders, it will receive "Insufficient funds" errors. This is expected and confirms the API integration is working correctly. For production use, add funds to your Kraken account.

10. Screenshots

The following screenshots demonstrate StrategyFlow's key interfaces and features.

1. The landing page introduces StrategyFlow with a clean, dark-themed design featuring animated elements.



Build Lightning-Fast Trading Logic. Visually.

A high-performance, low-code strategy builder powered by Kraken's API. Drag blocks, visualize flows, and simulate real execution — all inside your browser.

Everything you need to trade smarter

Professional-grade tools designed for traders who demand precision and performance

Drag & Drop Strategy Builder

Visual node editor with React Flow. Connect blocks intuitively without writing code.

Live Backtesting Simulator

Step-by-step execution logs with block highlighting for strategy validation.

Flow Debugger

Inspect inputs, outputs, and errors in real-time with advanced debugging tools.

Kraken API Integration

High-performance execution engine architecture with direct exchange connectivity.

Template Gallery

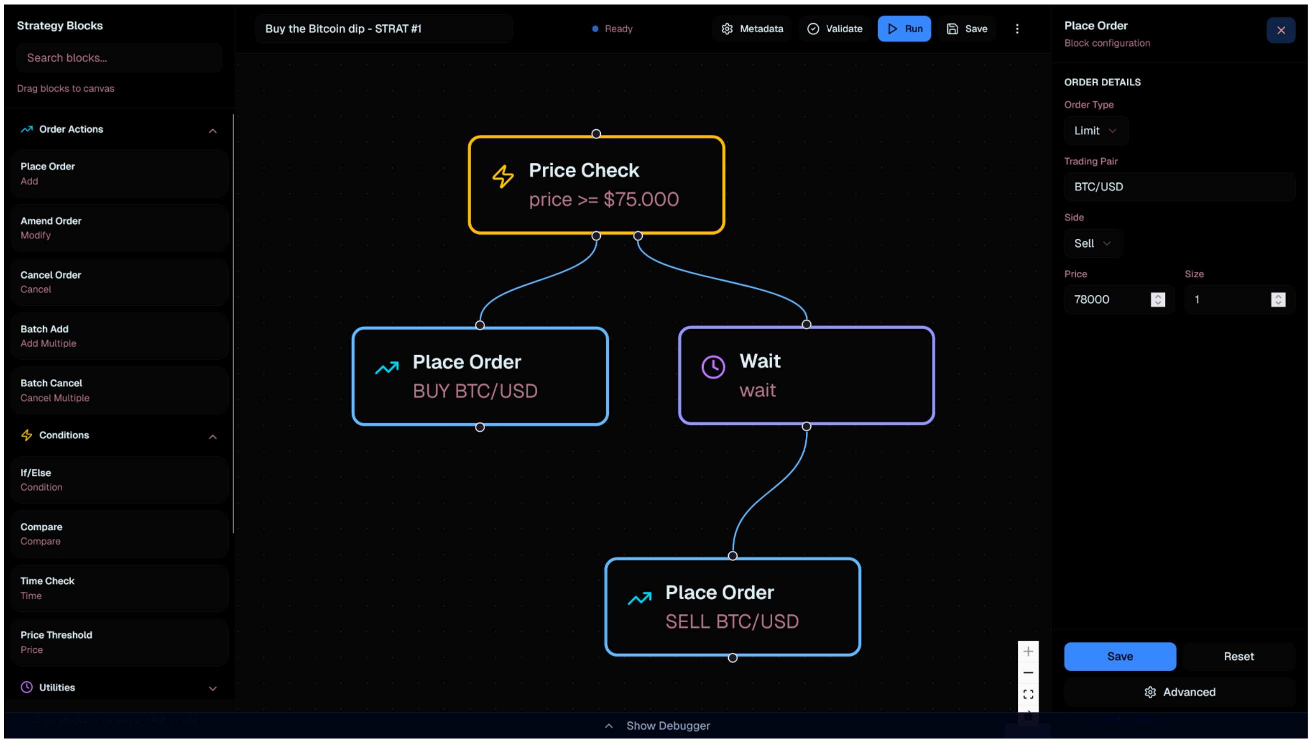
Load prebuilt strategies like DCA, Stop-Loss, Trailing TP, and Grid Trading.

Versioning & Export Tools

Export JSON definitions, TypeScript SDK skeletons, and API request sequences.

Figure 10.1 - Landing page

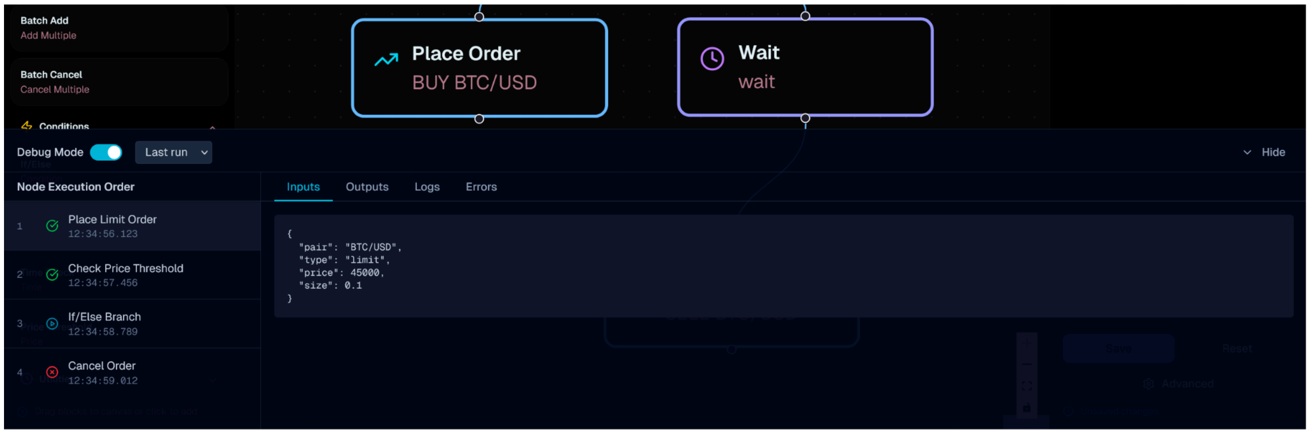
2. The main editor interface shows the node canvas with sidebar, inspector panel, and toolbar.



Order Type: Limit
Trading Pair: BTC/USD
Side: Sell
Price: 78000
Size: 1

Figure 10.2 - Strategy editor

2.1 The main editor interface with the inspector panel opened.

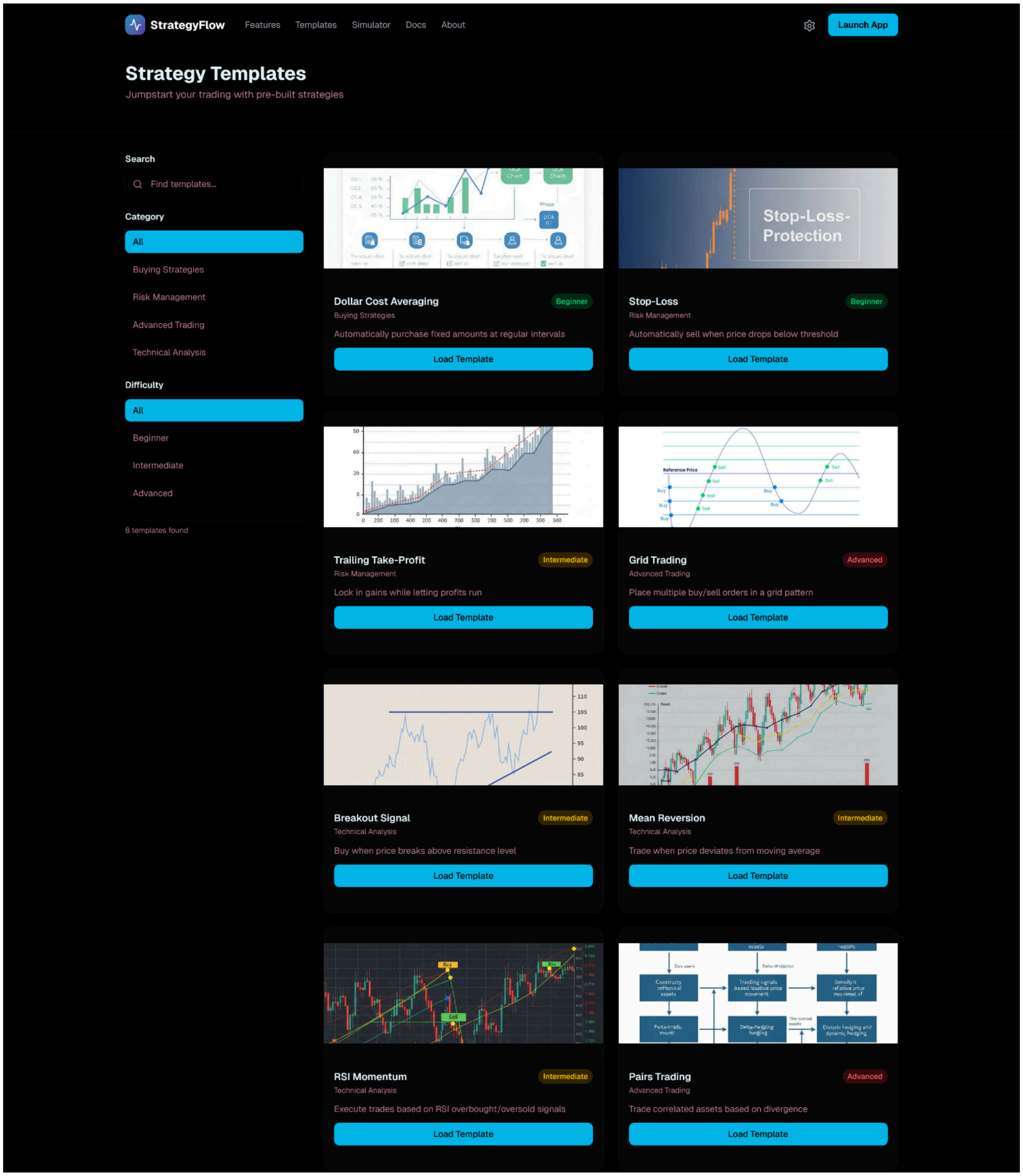


```
1  Place Limit Order       12:34:56.123
2  Check Price Threshold   12:34:57.456
3  If/Else Branch          12:34:58.789
4  Cancel Order            12:34:59.012
```

```
{
  "pair": "BTC/USD",
  "type": "limit",
  "price": 45000,
  "size": 0.1
}
```

Figure 10.3 - Strategy editor with inspector panel

3. Pre-built strategy templates available for quick start, including DCA, Stop-Loss, and Grid Trading.



Strategy Templates

Jumpstart your trading with pre-built strategies

Search

Find templates...

Category

- All
- Buying Strategies
- Risk Management
- Advanced Trading
- Technical Analysis

Difficulty

- All
- Beginner
- Intermediate
- Advanced

8 templates found

Dollar Cost Averaging — Beginner
Buying Strategies
Automatically purchase fixed amounts at regular intervals
Load Template

Stop-Loss — Beginner
Risk Management
Automatically sell when price drops below threshold
Load Template

Trailing Take-Profit — Intermediate
Risk Management
Lock in gains while letting profits run
Load Template

Grid Trading — Advanced
Advanced Trading
Place multiple buy/sell orders in a grid pattern
Load Template

Breakout Signal — Intermediate
Technical Analysis
Buy when price breaks above resistance level
Load Template

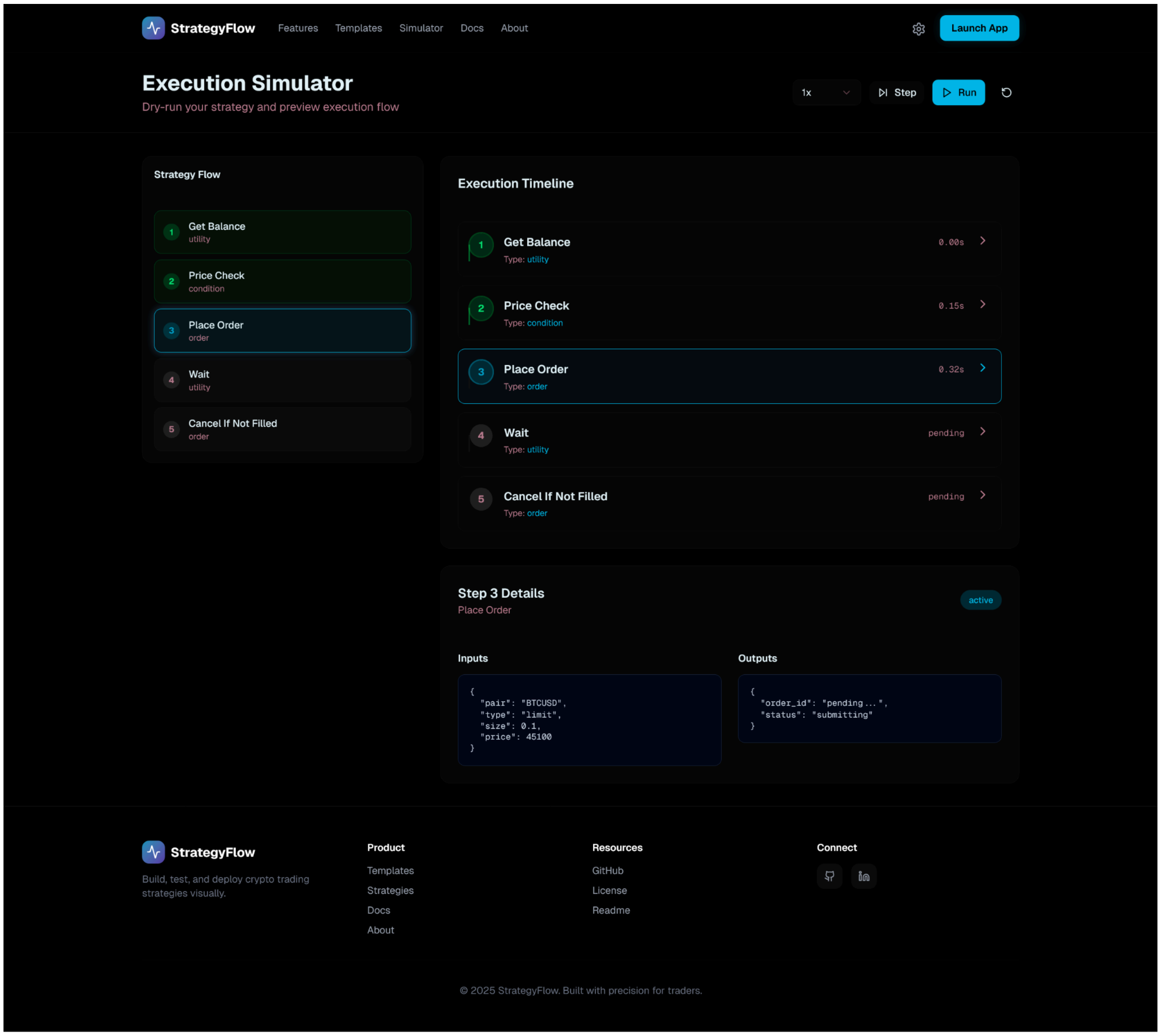
Mean Reversion — Intermediate
Technical Analysis
Trade when price deviates from moving average
Load Template

RSI Momentum — Intermediate
Technical Analysis
Execute trades based on RSI overbought/oversold signals
Load Template

Pairs Trading — Advanced
Advanced Trading
Trade correlated assets based on divergence
Load Template

Figure 10.4 - Template gallery

4. Step-by-step simulation mode with block highlighting and I/O inspection for debugging.



Execution Simulator

Dry-run your strategy and preview execution flow

Strategy Flow

1 Get Balance utility
2 Price Check condition
3 Place Order order
4 Wait utility
5 Cancel If Not Filled order

Execution Timeline

1	Get Balance — Type: utility	0.00s
2	Price Check — Type: condition	0.15s
3	Place Order — Type: order	0.32s
4	Wait — Type: utility	pending
5	Cancel If Not Filled — Type: order	pending

Step 3 Details active

Place Order

Inputs

```
{
  "pair": "BTCUSD",
  "type": "limit",
  "size": 0.1,
  "price": 45100
}
```

Outputs

```
{
  "order_id": "pending...",
  "status": "submitting"
}
```

StrategyFlow
Build, test, and deploy crypto trading strategies visually.
Product
Templates
Strategies
Docs
About
Resources
GitHub
License
Readme
Connect
© 2025 StrategyFlow. Built with precision for traders.

Figure 10.5 - Execution simulator

11. Future enhancements

11.1 Possible features

The following features are planned for future development:

- **OpenAI integration**: Natural language strategy creation where users can type requests like "Create a DCA strategy for BTC every Monday" and have blocks automatically generated and placed on the canvas.
- **WebSocket Real-time data**: Live orderbook and ticker feeds displayed directly within the editor for market-aware strategy building.
- **Backtesting engine**: Historical data simulation with performance metrics, win rate analysis, and risk assessment.
- **Strategy marketplace**: Community platform to share, discover, and remix trading strategies built by other users.

11.2 Scalability considerations

The modular architecture supports scaling through microservice extraction of the strategy engine, database-backed strategy persistence, and queue-based execution for long-running strategies.

12. License

This project is released under the MIT License as required by the Kraken Forge Hackathon rules.

Thank you for reviewing StrategyFlow

Built with care for Kraken Forge Hackathon 2025

Track #3: Strategy Builder

https://strategyflow.dev