Below is an end-to-end example of a full-stack application that lets users chat with a swarm of agents, generate images, and execute blockchain trades (on Base and Solana via Metaplex, Helius, Aura, etc.) while also integrating the Virtuals GAME framework for on-chain actions. This example uses FastAPI (Python) for the back end and Next.js (TypeScript) for the front end. (Note: In a production system you'd further secure keys, use environment variables, and refine error handling.)

Below is the complete file structure and code:

Project Structure:

```
fullstack-app/
├── backend/
│   ├── main.py
│   ├── agents.py
│   ├── blockchain.py
│   ├── image_gen.py
│   └── virtuals.py
└── frontend/
    ├── package.json
    ├── next.config.js
    └── pages/
        ├── index.tsx
        ├── chat.tsx
        ├── image.tsx
        └── trade.tsx
```

Backend Code (Python)

backend/main.py

```
from fastapi import FastAPI, HTTPException, Request
from fastapi.middleware.cors import CORSMiddleware
from agents import get_swarm_response
from blockchain import execute_trade, get_blockchain_data
```

```
from image_gen import generate_image
from virtuals import perform_virtual_action

app = FastAPI()

# Allow cross-origin requests (adjust as needed)
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_methods=["*"],
    allow_headers=["*"],
)

@app.post("/api/chat")
async def chat_endpoint(request: Request):
    data = await request.json()
    query = data.get("query")
    if not query:
        raise HTTPException(status_code=400, detail="Query is required")
    response = get_swarm_response(query)
    return {"response": response}

@app.post("/api/generate-image")
async def image_endpoint(request: Request):
    data = await request.json()
    prompt = data.get("prompt")
    if not prompt:
        raise HTTPException(status_code=400, detail="Prompt is required")
    image_url = generate_image(prompt)
    return {"image_url": image_url}

@app.post("/api/trade")
async def trade_endpoint(request: Request):
    data = await request.json()
    trade_details = data.get("trade_details")
    if not trade_details:
```

```
        raise HTTPException(status_code=400, detail="Trade details required")
    result = execute_trade(trade_details)
    return {"result": result}

@app.post("/api/virtual-action")
async def virtual_action_endpoint(request: Request):
    data = await request.json()
    action = data.get("action")
    if not action:
        raise HTTPException(status_code=400, detail="Action required")
    result = perform_virtual_action(action)
    return {"result": result}

@app.get("/api/blockchain-data")
async def blockchain_data_endpoint():
    data = get_blockchain_data()
    return {"data": data}

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

backend/agents.py

```
# Minimal swarm agents integration using the Cheshire persona.
def get_swarm_response(query: str) → str:
    # Here you would call your multi-agent swarm orchestration (including base
d Chesh personality, etc.)
    return f"Swarm response for query: '{query}'. (Based Chesh: All agents con
cur!)"
```

backend/blockchain.py

```
# Stub implementations for blockchain interactions.
def execute_trade(trade_details: dict) → dict:
    # Simulate executing a trade across Base and Solana (using Metaplex, Heliu
```

```
s, Aura RPC, etc.)
    return {
        "status": "success",
        "details": trade_details,
        "message": "Trade executed on Base & Solana."
    }

def get_blockchain_data() → dict:
    # Return dummy blockchain data from Base, Solana, Metaplex, and Helius.
    return {
        "base": {"price": 100, "volume": 5000},
        "solana": {"price": 30, "volume": 10000},
        "metaplex": {"nfts": 250},
        "helius": {"transactions": 12345}
    }
```

backend/image_gen.py

```
# Stub for image generation (e.g. using a stable diffusion model on an NVIDIA
Orin Nano via BrewLaunchable).
def generate_image(prompt: str) → str:
    # In production, this function would call a GPU-accelerated model.
    # Here we return a dummy image URL.
    return "https://dummyimage.com/1024×1024/000/fff&text=" + prompt.repla
ce(" ", "+")
```

backend/virtuals.py

```
# Stub for performing a virtual action via the Virtuals GAME framework.
def perform_virtual_action(action: dict) → dict:
    # In a full implementation, this would interact with the Virtuals SDK to trigge
r on-chain actions.
    return {"status": "success", "action": action, "message": "Virtual action exe
cuted."}
```

Frontend Code (Next.js / TypeScript)

frontend/package.json

```
{
  "name": "cheshire-terminal-frontend",
  "version": "1.0.0",
  "scripts": {
    "dev": "next dev",
    "build": "next build",
    "start": "next start"
  },
  "dependencies": {
    "next": "^13.0.0",
    "react": "^18.0.0",
    "react-dom": "^18.0.0",
    "axios": "^1.0.0"
  },
  "devDependencies": {
    "typescript": "^4.0.0",
    "@types/react": "^18.0.0",
    "@types/node": "^18.0.0"
  }
}
```

frontend/next.config.js

```
/** @type {import('next').NextConfig} */
const nextConfig = {
  reactStrictMode: true
}
module.exports = nextConfig;
```

frontend/pages/index.tsx

```
import Link from 'next/link';
import React from 'react';

const HomePage: React.FC = () => {
  return (
    <div style={{ padding: '2rem', fontFamily: 'sans-serif' }}>
      <h1>Cheshire Terminal</h1>
      <p>Welcome to the full-stack Cheshire Terminal application.</p>
      <ul>
        <li><Link href="/chat">Chat with Agents</Link></li>
        <li><Link href="/image">Generate Image</Link></li>
        <li><Link href="/trade">Make a Trade</Link></li>
      </ul>
    </div>
  );
};

export default HomePage;
```

frontend/pages/chat.tsx

```
import React, { useState } from 'react';
import axios from 'axios';
import Link from 'next/link';

const ChatPage: React.FC = () => {
  const [query, setQuery] = useState('');
  const [response, setResponse] = useState('');

  const sendQuery = async () => {
    try {
      const res = await axios.post('http://localhost:8000/api/chat', { query });
      setResponse(res.data.response);
    } catch (error) {
      setResponse('Error occurred while communicating with the agents.');
```

```
    }
  };

  return (
    <div style={{ padding: '2rem' }}>
      <h1>Chat with Agents</h1>
      <textarea
        rows={4}
        cols={50}
        value={query}
        onChange={(e) => setQuery(e.target.value)}
        placeholder="Enter your query..."
      />
      <br />
      <button onClick={sendQuery}>Send</button>
      <h2>Response:</h2>
      <pre>{response}</pre>
      <Link href="/">Back to Home</Link>
    </div>
  );
};

export default ChatPage;
```

frontend/pages/image.tsx

```
import React, { useState } from 'react';
import axios from 'axios';
import Link from 'next/link';

const ImagePage: React.FC = () => {
  const [prompt, setPrompt] = useState('');
  const [imageUrl, setImageUrl] = useState('');

  const generateImage = async () => {
    try {
```

```
      const res = await axios.post('http://localhost:8000/api/generate-image', { prompt });
      setImageUrl(res.data.image_url);
    } catch (error) {
      setImageUrl('Error occurred while generating image.');
    }
  };

  return (
    <div style={{ padding: '2rem' }}>
      <h1>Generate Image</h1>
      <input
        type="text"
        value={prompt}
        onChange={(e) => setPrompt(e.target.value)}
        placeholder="Enter prompt for image..."
      />
      <button onClick={generateImage}>Generate</button>
      {imageUrl && (
        <div>
          <h2>Generated Image:</h2>
          <img src={imageUrl} alt="Generated" style={{ maxWidth: '100%' }} />
        </div>
      )}
      <Link href="/">Back to Home</Link>
    </div>
  );
};

export default ImagePage;
```

frontend/pages/trade.tsx

```
import React, { useState } from 'react';
import axios from 'axios';
import Link from 'next/link';
```

```
const TradePage: React.FC = () => {
  const [tradeDetails, setTradeDetails] = useState('');
  const [result, setResult] = useState('');

  const executeTrade = async () => {
    try {
      const res = await axios.post('http://localhost:8000/api/trade', { trade_detai
ls: { detail: tradeDetails } });
      setResult(JSON.stringify(res.data.result, null, 2));
    } catch (error) {
      setResult('Error occurred while executing trade.');
    }
  };

  return (
    <div style={{ padding: '2rem' }}>
      <h1>Make a Trade</h1>
      <input
        type="text"
        value={tradeDetails}
        onChange={(e) => setTradeDetails(e.target.value)}
        placeholder="Enter trade details..."
      />
      <button onClick={executeTrade}>Execute Trade</button>
      <h2>Trade Result:</h2>
      <pre>{result}</pre>
      <Link href="/">Back to Home</Link>
    </div>
  );
};

export default TradePage;
```

Overview

- **Back End:**
  - FastAPI serves endpoints for chatting with agents, image generation, trade execution, and virtual on-chain actions.
  - The agents, blockchain, image generation, and Virtuals integrations are stubbed out here; in a full implementation, they would include calls to actual LLMs, blockchain RPCs (Base RPC, Helius for Solana, Metaplex libraries, etc.), and GPU inference on your NVIDIA Orin Nano (via BrewLaunchable).
- **Front End:**
  - A Next.js application in TypeScript provides a simple user interface for chatting, generating images, and executing trades.
  - Axios is used for calling the back-end API endpoints.
- **Integration:**
  - The Virtuals GAME framework and Web3 integrations would be expanded within the respective modules (virtuals.py and blockchain.py) to interface with Base, Solana, Metaplex, Aura RPC, and Helius RPC.

This full-stack application gives you a starting point for a comprehensive system where users can interact with a multi-agent AI swarm, create images, and perform blockchain transactions—all under the “Cheshire Terminal” brand and powered by modern AI and blockchain tech.

Feel free to extend or modify each module according to your production needs. Enjoy building your next-generation cross-chain, multi-modal AI platform!