# SPEC-1-AutoTradingPlatform

## Background

This project aims to build an automated trading platform powered by DhanHQ and TradingView. It allows users to paste Pine Script strategies, deploy them via a unified UI, and execute trades automatically via broker APIs. The system also supports paper trading, realtime status monitoring, alerting, and detailed reporting.

## Requirements

**Must Have**

* UI page to connect/link DhanHQ broker (OAuth or token-based)
* UI input for Pine Script (TradingView)
* Deploy button that connects Pine Script logic to DhanHQ webhook execution
* Backend integration of TradingView alerts via webhook and script execution
* Realtime execution status display
* Reports dashboard with filters + CSV export of trades
* Paper trading support (simulate orders instead of real trades)
* User authentication & session handling
* Strategy management (view, delete, update)
* Error logging and retry on webhook failures
* Telegram/email alerts for strategy execution
* Admin panel to manage users (enable/disable, reset broker tokens)

## Method

### High-Level Architecture

@startuml  
actor User  
  
package "Frontend (Next.js)" {  
 [Login Page] --> [Dashboard]  
 [Dashboard] --> [Strategy Editor]  
 [Dashboard] --> [Broker Integration UI]  
 [Dashboard] --> [Execution Status]  
 [Dashboard] --> [Reports/Export]  
 [Dashboard] --> [Admin Panel]  
}  
  
package "Backend (FastAPI/NestJS)" {  
 [Auth Service]  
 [Strategy Manager]  
 [Webhook Listener]  
 [Trade Executor]  
 [Paper Trade Engine]  
 [Alert Notifier]  
 [Broker Adapter]  
 [Logging & Retry Handler]  
 [Report Generator]  
 [Admin Controller]  
}  
  
User --> [Login Page]  
[Strategy Editor] --> [Strategy Manager]  
[Broker Integration UI] --> [Broker Adapter]  
[Execution Status] --> [Trade Executor]  
[Webhook Listener] --> [Trade Executor]  
[Webhook Listener] --> [Logging & Retry Handler]  
[Trade Executor] --> [DhanHQ API]  
[Trade Executor] --> [Paper Trade Engine]  
[Trade Executor] --> [Alert Notifier]  
[Report Generator] --> [Reports/Export]  
[Admin Panel] --> [Admin Controller]  
  
[TradingView Alerts] --> [Webhook Listener]  
@enduml

### Database Schema

#### users

id UUID PRIMARY KEY  
email TEXT UNIQUE NOT NULL  
password\_hash TEXT NOT NULL  
role TEXT CHECK (role IN ('user', 'admin')) DEFAULT 'user'  
status TEXT CHECK (status IN ('active', 'disabled')) DEFAULT 'active'  
created\_at TIMESTAMP DEFAULT NOW()

#### brokers

id UUID PRIMARY KEY  
user\_id UUID REFERENCES users(id)  
type TEXT CHECK (type IN ('dhanhq', 'paper'))  
auth\_token TEXT  
token\_expiry TIMESTAMP  
connected\_at TIMESTAMP

#### strategies

id UUID PRIMARY KEY  
user\_id UUID REFERENCES users(id)  
name TEXT  
script TEXT  
broker\_id UUID REFERENCES brokers(id)  
paper\_trading BOOLEAN DEFAULT false  
webhook\_path TEXT UNIQUE  
status TEXT CHECK (status IN ('active', 'paused', 'error'))  
created\_at TIMESTAMP DEFAULT NOW()

#### alerts

id UUID PRIMARY KEY  
strategy\_id UUID REFERENCES strategies(id)  
symbol TEXT  
signal TEXT  
price NUMERIC  
triggered\_at TIMESTAMP  
raw\_payload JSONB

#### executions

id UUID PRIMARY KEY  
alert\_id UUID REFERENCES alerts(id)  
type TEXT CHECK (type IN ('live', 'paper'))  
status TEXT CHECK (status IN ('success', 'fail'))  
response JSONB  
executed\_at TIMESTAMP  
retry\_count INTEGER DEFAULT 0

#### paper\_trades

id UUID PRIMARY KEY  
strategy\_id UUID REFERENCES strategies(id)  
symbol TEXT  
side TEXT CHECK (side IN ('BUY', 'SELL'))  
entry\_price NUMERIC  
exit\_price NUMERIC  
qty INTEGER  
pnl NUMERIC  
entry\_time TIMESTAMP  
exit\_time TIMESTAMP

#### notifications

id UUID PRIMARY KEY  
user\_id UUID REFERENCES users(id)  
method TEXT CHECK (method IN ('email', 'telegram'))  
endpoint TEXT  
enabled BOOLEAN DEFAULT true  
created\_at TIMESTAMP

## Implementation

## 📝 Local Development README

### 1. Prerequisites

* Docker + Docker Compose
* Node.js 18+
* Python 3.11+

### 2. Environment Setup

#### .env.example

#### .env.production

# Backend  
SECRET\_KEY=your-prod-secret-key  
ACCESS\_TOKEN\_EXPIRE\_MINUTES=60  
DATABASE\_URL=postgresql://at\_user:at\_pass@localhost:5432/autotrading  
  
# Frontend  
NEXT\_PUBLIC\_API\_BASE=https://yourdomain.com/api  
  
# Broker Integrations  
DHAN\_API\_KEY=prod-dhan-key  
DHAN\_API\_SECRET=prod-dhan-secret  
  
# Alerts  
TELEGRAM\_BOT\_TOKEN=prod-telegram-token  
SENDGRID\_API\_KEY=prod-sendgrid-key

# Backend  
SECRET\_KEY=changeme  
ACCESS\_TOKEN\_EXPIRE\_MINUTES=60  
DATABASE\_URL=postgresql://at\_user:at\_pass@db:5432/autotrading  
  
# Frontend  
NEXT\_PUBLIC\_API\_BASE=http://localhost:8000  
  
# Broker Integrations  
DHAN\_API\_KEY=your-dhan-key  
DHAN\_API\_SECRET=your-dhan-secret  
  
# Alerts  
TELEGRAM\_BOT\_TOKEN=your-bot-token  
SENDGRID\_API\_KEY=your-sendgrid-api-key

1. Clone the repository:

git clone https://github.com/algodatta/AlgoDatta.git  
cd AlgoDatta

1. Set environment variables:

cp .env.example .env  
# edit .env with DB and JWT secrets

### 3. Start Services

docker-compose up --build

Access UI: <http://localhost:3000>  
Access API: <http://localhost:8000/docs>

### 4. Run Tests (from backend directory)

pytest

### 5. Deploy

Handled via Jenkins pipeline into Amazon Lightsail instance.

### 6. Import Postman Collection

Use AutoTradingPlatform.postman\_collection.json to test endpoints manually.

## System Bootstrapping

### ☁️ CI/CD Deployment (Jenkins + Amazon Lightsail)

#### Jenkinsfile

pipeline {  
 agent any  
  
 environment {  
 REMOTE\_HOST = "ubuntu@43.205.125.42"  
 SSH\_CRED\_ID = "sshKeyPair"  
 }  
  
 stages {  
 stage('Checkout Repo') {  
 steps {  
 git credentialsId: 'github-access', url: 'https://github.com/algodatta/AlgoDatta.git', branch: 'main'  
 }  
 }  
  
 stage('Install Docker') {  
 steps {  
 sshagent (credentials: [env.SSH\_CRED\_ID]) {  
 sh '''  
 ssh -o StrictHostKeyChecking=no $REMOTE\_HOST '  
 if ! command -v docker >/dev/null 2>&1; then  
 echo "Installing Docker..."  
 curl -fsSL https://get.docker.com | sh  
 else  
 echo "Docker is already installed"  
 fi  
 '  
 '''  
 }  
 }  
 }  
  
 stage('Build and Deploy') {  
 steps {  
 sshagent (credentials: [env.SSH\_CRED\_ID]) {  
 sh '''  
 ssh -o StrictHostKeyChecking=no $REMOTE\_HOST '  
 if [ ! -d "AlgoDatta" ]; then  
 git clone https://github.com/algodatta/AlgoDatta.git  
 fi  
 cd AlgoDatta  
 git reset --hard  
 git clean -fd  
 git pull origin main  
 docker compose -f docker-compose.yml up -d --build --remove-orphans  
 '  
 '''  
 }  
 }  
 }  
 }  
}

### ✅ Testing Setup

#### tests/test\_reports.py

def test\_csv\_export(client):  
 email = "csv@test.com"  
 password = "exportme"  
 client.post("/register", json={"email": email, "password": password})  
 res = client.post("/login", json={"email": email, "password": password})  
 token = res.json()["access\_token"]  
 headers = {"Authorization": f"Bearer {token}"}  
  
 res = client.get("/reports/csv", headers=headers)  
 assert res.status\_code == 200  
 assert "text/csv" in res.headers["content-type"]  
 assert "executed\_at" in res.text # basic sanity check

#### tests/test\_broker.py

def test\_link\_broker(client):  
 email = "broker@link.com"  
 password = "linkme"  
 client.post("/register", json={"email": email, "password": password})  
 res = client.post("/login", json={"email": email, "password": password})  
 token = res.json()["access\_token"]  
 headers = {"Authorization": f"Bearer {token}"}  
  
 res = client.post("/broker", json={"auth\_token": "mock\_token\_value"}, headers=headers)  
 assert res.status\_code == 200  
 assert res.json()["status"] == "Broker linked"

#### tests/test\_webhook.py

import uuid  
import pytest  
  
def test\_receive\_webhook\_and\_execute(client):  
 email = "alert@bot.com"  
 password = "trader123"  
 client.post("/register", json={"email": email, "password": password})  
 res = client.post("/login", json={"email": email, "password": password})  
 token = res.json()["access\_token"]  
 headers = {"Authorization": f"Bearer {token}"}  
  
 # Create a strategy first  
 payload = {  
 "name": "Alert Test",  
 "script": "strategy.entry('Buy', strategy.long)",  
 "broker\_id": str(uuid.uuid4()),  
 "paper\_trading": True  
 }  
 res = client.post("/strategies", json=payload, headers=headers)  
 strategy = res.json()  
 webhook\_path = strategy["webhook\_path"]  
  
 # Simulate webhook  
 webhook\_payload = {  
 "signal": "buy",  
 "symbol": "NSE:TCS",  
 "price": 3620.50  
 }  
 res = client.post(f"/webhooks/{webhook\_path}", json=webhook\_payload, headers=headers)  
 assert res.status\_code == 200  
 assert res.json()["status"] == "alert received"  
  
 # Confirm execution was created  
 res = client.get("/executions", headers=headers)  
 assert res.status\_code == 200  
 assert len(res.json()) >= 1

#### tests/conftest.py

import pytest  
from fastapi.testclient import TestClient  
from app.main import app  
  
@pytest.fixture(scope="module")  
def client():  
 with TestClient(app) as c:  
 yield c

#### tests/test\_auth.py

def test\_register\_and\_login(client):  
 email = "test@example.com"  
 password = "strongpass"  
  
 response = client.post("/register", json={"email": email, "password": password})  
 assert response.status\_code == 200  
 data = response.json()  
 assert data["email"] == email  
  
 response = client.post("/login", json={"email": email, "password": password})  
 assert response.status\_code == 200  
 token = response.json().get("access\_token")  
 assert token is not None

#### tests/test\_strategy.py

import uuid  
  
def test\_create\_strategy(client):  
 email = "s1@example.com"  
 password = "password123"  
 client.post("/register", json={"email": email, "password": password})  
 res = client.post("/login", json={"email": email, "password": password})  
 token = res.json()["access\_token"]  
 headers = {"Authorization": f"Bearer {token}"}  
  
 payload = {  
 "name": "Breakout Strategy",  
 "script": "strategy.entry('Buy', strategy.long)",  
 "broker\_id": str(uuid.uuid4()),  
 "paper\_trading": True  
 }  
 res = client.post("/strategies", json=payload, headers=headers)  
 assert res.status\_code == 200  
 assert "webhook\_path" in res.json()

### 📁 Backend Project Structure (FastAPI)

### 🧱 Models & Schemas: User

### 📦 Models & Schemas: Strategy

### 🧾 Models & Schemas: Alert

#### models/alert.py

from sqlalchemy import Column, String, DateTime, ForeignKey, Numeric  
from sqlalchemy.dialects.postgresql import UUID  
from sqlalchemy.types import JSON  
from sqlalchemy.sql import func  
import uuid  
  
from ..db import Base  
  
class Alert(Base):  
 \_\_tablename\_\_ = "alerts"  
  
 id = Column(UUID(as\_uuid=True), primary\_key=True, default=uuid.uuid4)  
 strategy\_id = Column(UUID(as\_uuid=True), ForeignKey("strategies.id"), nullable=False)  
 symbol = Column(String)  
 signal = Column(String)  
 price = Column(Numeric)  
 triggered\_at = Column(DateTime(timezone=True), default=func.now())  
 raw\_payload = Column(JSON)

#### schemas/alert.py

from pydantic import BaseModel  
from uuid import UUID  
from datetime import datetime  
from typing import Dict  
  
class AlertBase(BaseModel):  
 symbol: str  
 signal: str  
 price: float  
 raw\_payload: Dict  
  
class AlertCreate(AlertBase):  
 strategy\_id: UUID  
  
class AlertRead(AlertBase):  
 id: UUID  
 triggered\_at: datetime  
  
 class Config:  
 orm\_mode = True

### 🧾 Models & Schemas: Execution

#### models/execution.py

from sqlalchemy import Column, String, DateTime, ForeignKey, Enum, Integer  
from sqlalchemy.dialects.postgresql import UUID, JSONB  
from sqlalchemy.sql import func  
import uuid  
import enum  
  
from ..db import Base  
  
class ExecutionType(str, enum.Enum):  
 live = "live"  
 paper = "paper"  
  
class ExecutionStatus(str, enum.Enum):  
 success = "success"  
 fail = "fail"  
  
class Execution(Base):  
 \_\_tablename\_\_ = "executions"  
  
 id = Column(UUID(as\_uuid=True), primary\_key=True, default=uuid.uuid4)  
 alert\_id = Column(UUID(as\_uuid=True), ForeignKey("alerts.id"), nullable=False)  
 type = Column(Enum(ExecutionType), nullable=False)  
 status = Column(Enum(ExecutionStatus), nullable=False)  
 response = Column(JSONB)  
 executed\_at = Column(DateTime(timezone=True), server\_default=func.now())  
 retry\_count = Column(Integer, default=0)

#### schemas/execution.py

from pydantic import BaseModel  
from uuid import UUID  
from datetime import datetime  
from enum import Enum  
from typing import Dict  
  
class ExecutionType(str, Enum):  
 live = "live"  
 paper = "paper"  
  
class ExecutionStatus(str, Enum):  
 success = "success"  
 fail = "fail"  
  
class ExecutionBase(BaseModel):  
 type: ExecutionType  
 status: ExecutionStatus  
 response: Dict  
  
class ExecutionRead(ExecutionBase):  
 id: UUID  
 alert\_id: UUID  
 executed\_at: datetime  
 retry\_count: int  
  
 class Config:  
 orm\_mode = True

### 🧪 Paper Trade Engine

#### services/paper\_engine.py

from uuid import uuid4  
from datetime import datetime  
from sqlalchemy.orm import Session  
  
from app.models.strategy import Strategy  
from app.models.paper\_trade import PaperTrade  
  
class PaperTradeEngine:  
 def simulate\_order(self, strategy: Strategy, symbol: str, side: str, qty: int, price: float, db: Session):  
 trade = PaperTrade(  
 id=uuid4(),  
 strategy\_id=strategy.id,  
 symbol=symbol,  
 side=side,  
 entry\_price=price,  
 qty=qty,  
 entry\_time=datetime.utcnow(),  
 exit\_price=None,  
 exit\_time=None,  
 pnl=None  
 )  
 db.add(trade)  
 db.commit()  
 return trade

### 📣 Alert Notifier

#### services/notifier.py

import requests  
  
def send\_telegram\_alert(chat\_id: str, bot\_token: str, message: str):  
 url = f"https://api.telegram.org/bot{bot\_token}/sendMessage"  
 payload = {"chat\_id": chat\_id, "text": message}  
 requests.post(url, data=payload)  
  
# Add similar send\_email\_alert via SendGrid or SMTP

### 🤖 Trade Executor

#### services/trade\_executor.py

from datetime import datetime  
from uuid import uuid4  
from sqlalchemy.orm import Session  
  
from app.models.alert import Alert  
from app.models.execution import Execution  
from app.models.strategy import Strategy  
  
# Simulated order response from Dhan or Paper Engine  
class OrderStatus:  
 SUCCESS = "success"  
 FAIL = "fail"  
  
def execute\_trade(alert: Alert, strategy: Strategy, db: Session):  
 # Simulate sending order to broker or paper engine  
 # In a real version, use DhanAdapter or PaperEngine based on strategy.paper\_trading  
 try:  
 response\_data = {"order\_id": str(uuid4()), "status": "mocked"}  
  
 execution = Execution(  
 id=uuid4(),  
 alert\_id=alert.id,  
 type="paper" if strategy.paper\_trading else "live",  
 status=OrderStatus.SUCCESS,  
 response=response\_data,  
 executed\_at=datetime.utcnow(),  
 retry\_count=0  
 )  
 db.add(execution)  
 db.commit()  
 except Exception as e:  
 db.rollback()  
 execution = Execution(  
 id=uuid4(),  
 alert\_id=alert.id,  
 type="paper" if strategy.paper\_trading else "live",  
 status=OrderStatus.FAIL,  
 response={"error": str(e)},  
 executed\_at=datetime.utcnow(),  
 retry\_count=1  
 )  
 db.add(execution)  
 db.commit()

### 🛡️ Admin API

#### Role-based Dependency Guard

##### core/deps.py

from fastapi import Depends, HTTPException, status  
from jose import JWTError, jwt  
from fastapi.security import OAuth2PasswordBearer  
from sqlalchemy.orm import Session  
from app.db import get\_db  
from app.models.user import User, UserRole  
  
SECRET\_KEY = "your-secret-key"  
ALGORITHM = "HS256"  
  
oauth2\_scheme = OAuth2PasswordBearer(tokenUrl="/login")  
  
def get\_current\_user(token: str = Depends(oauth2\_scheme), db: Session = Depends(get\_db)):  
 try:  
 payload = jwt.decode(token, SECRET\_KEY, algorithms=[ALGORITHM])  
 user\_id = payload.get("sub")  
 if user\_id is None:  
 raise HTTPException(status\_code=401, detail="Invalid token")  
 except JWTError:  
 raise HTTPException(status\_code=401, detail="Invalid token")  
  
 user = db.query(User).filter(User.id == user\_id).first()  
 if not user:  
 raise HTTPException(status\_code=404, detail="User not found")  
 return user  
  
def require\_admin(user: User = Depends(get\_current\_user)):  
 if user.role != UserRole.admin:  
 raise HTTPException(status\_code=403, detail="Admin access required")  
 return user

#### api/v1/admin.py

from fastapi import APIRouter, Depends, HTTPException  
from sqlalchemy.orm import Session  
from app.db import get\_db  
from app.models.user import User, UserStatus  
from app.models.strategy import Strategy  
from uuid import UUID  
  
from app.core.deps import require\_admin  
  
router = APIRouter(dependencies=[Depends(require\_admin)])  
  
@router.get("/admin/users")  
def list\_users(db: Session = Depends(get\_db)):  
 return db.query(User).all()  
  
@router.patch("/admin/users/{user\_id}/status")  
def toggle\_user\_status(user\_id: UUID, status: UserStatus, db: Session = Depends(get\_db)):  
 user = db.query(User).filter(User.id == user\_id).first()  
 if not user:  
 raise HTTPException(status\_code=404, detail="User not found")  
 user.status = status  
 db.commit()  
 return {"status": f"User status updated to {status}"}  
  
@router.get("/admin/users/{user\_id}/strategies")  
def user\_strategies(user\_id: UUID, db: Session = Depends(get\_db)):  
 return db.query(Strategy).filter(Strategy.user\_id == user\_id).all()  
  
@router.post("/admin/users/{user\_id}/reset-broker")  
def reset\_broker\_token(user\_id: UUID, db: Session = Depends(get\_db)):  
 user\_brokers = db.query(Broker).filter(Broker.user\_id == user\_id).all()  
 for broker in user\_brokers:  
 broker.auth\_token = None  
 broker.token\_expiry = None  
 db.commit()  
 return {"status": "Broker tokens reset"}

### 🔗 Broker Integration UI

### 🛠️ Broker Backend API

#### api/v1/broker.py

from fastapi import APIRouter, Depends  
from sqlalchemy.orm import Session  
from uuid import uuid4  
from datetime import datetime  
  
from app.db import get\_db  
from app.models.broker import Broker  
from app.core.deps import get\_current\_user  
  
router = APIRouter(dependencies=[Depends(get\_current\_user)])  
  
@router.post("/broker")  
def link\_broker(payload: dict, db: Session = Depends(get\_db), user = Depends(get\_current\_user)):  
 auth\_token = payload.get("auth\_token")  
 if not auth\_token:  
 return {"error": "auth\_token required"}  
  
 broker = db.query(Broker).filter(Broker.user\_id == user.id).first()  
 if broker:  
 broker.auth\_token = auth\_token  
 broker.connected\_at = datetime.utcnow()  
 else:  
 broker = Broker(  
 id=uuid4(),  
 user\_id=user.id,  
 type="dhanhq",  
 auth\_token=auth\_token,  
 connected\_at=datetime.utcnow()  
 )  
 db.add(broker)  
  
 db.commit()  
 return {"status": "Broker linked"}

#### 📄 pages/broker.tsx

import { useState } from 'react';  
import axios from 'axios';  
  
export default function BrokerPage() {  
 const [authToken, setAuthToken] = useState('');  
 const [linked, setLinked] = useState(false);  
 const [error, setError] = useState('');  
  
 const linkBroker = async () => {  
 try {  
 const token = localStorage.getItem('token');  
 await axios.post('/api/broker', { auth\_token: authToken }, {  
 headers: { Authorization: `Bearer ${token}` }  
 });  
 setLinked(true);  
 } catch (e) {  
 setError('Failed to link broker');  
 }  
 };  
  
 return (  
 <div className="min-h-screen bg-gray-100 p-8">  
 <div className="max-w-lg mx-auto bg-white p-6 rounded shadow space-y-4">  
 <h1 className="text-2xl font-bold">Connect to Dhan Broker</h1>  
 <input  
 className="w-full p-2 border rounded"  
 placeholder="Enter Dhan Auth Token"  
 value={authToken}  
 onChange={(e) => setAuthToken(e.target.value)}  
 />  
 <button  
 className="bg-blue-600 text-white px-4 py-2 rounded hover:bg-blue-700"  
 onClick={linkBroker}  
 >  
 Link Broker  
 </button>  
 {linked && <p className="text-green-600">Broker linked successfully!</p>}  
 {error && <p className="text-red-600">{error}</p>}  
 </div>  
 </div>  
 );  
}

### 📊 Execution Status API

### 📤 Reports & CSV Export

#### Frontend Integration Example (React/Next.js)

<button  
 onClick={() => {  
 const token = localStorage.getItem("token");  
 const a = document.createElement("a");  
 a.href = `/api/reports/csv?token=${token}`;  
 a.download = "executions.csv";  
 a.click();  
 }}  
 className="bg-green-600 text-white px-4 py-2 rounded hover:bg-green-700"  
>  
 Download CSV  
</button>

#### api/v1/reports.py

from fastapi import APIRouter, Depends, Query  
from fastapi.responses import StreamingResponse  
from sqlalchemy.orm import Session  
from io import StringIO  
import csv  
  
from app.db import get\_db  
from app.models.execution import Execution  
from app.core.deps import get\_current\_user  
  
router = APIRouter(dependencies=[Depends(get\_current\_user)])  
  
@router.get("/reports/csv")  
def export\_csv(db: Session = Depends(get\_db)):  
 output = StringIO()  
 writer = csv.writer(output)  
 writer.writerow(["executed\_at", "type", "status", "response"])  
  
 results = db.query(Execution).order\_by(Execution.executed\_at.desc()).limit(500).all()  
 for row in results:  
 writer.writerow([  
 row.executed\_at.isoformat(),  
 row.type,  
 row.status,  
 str(row.response)  
 ])  
  
 output.seek(0)  
 return StreamingResponse(output, media\_type="text/csv", headers={  
 "Content-Disposition": "attachment; filename=executions.csv"  
 })

#### api/v1/executions.py

from fastapi import APIRouter, Depends  
from sqlalchemy.orm import Session  
from app.db import get\_db  
from app.models.execution import Execution  
from app.schemas.execution import ExecutionRead  
from app.core.deps import get\_current\_user  
  
router = APIRouter(dependencies=[Depends(get\_current\_user)])  
  
@router.get("/executions", response\_model=list[ExecutionRead])  
def list\_executions(db: Session = Depends(get\_db)):  
 return db.query(Execution).order\_by(Execution.executed\_at.desc()).limit(50).all()

### 🌐 Webhook Listener

#### api/v1/webhooks.py

from fastapi import APIRouter, Request, HTTPException, Depends  
from sqlalchemy.orm import Session  
from app.db import get\_db  
from app.models.strategy import Strategy  
from app.models.alert import Alert  
from uuid import uuid4  
from datetime import datetime  
  
from app.core.deps import get\_current\_user  
  
router = APIRouter(dependencies=[Depends(get\_current\_user)])  
  
@router.post("/webhooks/{webhook\_id}")  
def receive\_webhook(webhook\_id: str, request: Request, db: Session = Depends(get\_db)):  
 strategy = db.query(Strategy).filter(Strategy.webhook\_path == webhook\_id).first()  
 if not strategy:  
 raise HTTPException(status\_code=404, detail="Strategy not found")  
  
 payload = await request.json()  
 signal = payload.get("signal")  
 symbol = payload.get("symbol")  
 price = payload.get("price")  
  
 if not all([signal, symbol, price]):  
 raise HTTPException(status\_code=400, detail="Missing signal data")  
  
 alert = Alert(  
 id=uuid4(),  
 strategy\_id=strategy.id,  
 symbol=symbol,  
 signal=signal,  
 price=price,  
 triggered\_at=datetime.utcnow(),  
 raw\_payload=payload  
 )  
 db.add(alert)  
 db.commit()  
 return {"status": "alert received"}

#### models/strategy.py

from sqlalchemy import Column, String, Boolean, Enum, DateTime, ForeignKey  
from sqlalchemy.dialects.postgresql import UUID  
from sqlalchemy.sql import func  
import uuid  
  
from ..db import Base  
  
class StrategyStatus(str, enum.Enum):  
 active = "active"  
 paused = "paused"  
 error = "error"  
  
class Strategy(Base):  
 \_\_tablename\_\_ = "strategies"  
  
 id = Column(UUID(as\_uuid=True), primary\_key=True, default=uuid.uuid4)  
 user\_id = Column(UUID(as\_uuid=True), ForeignKey("users.id"), nullable=False)  
 name = Column(String)  
 script = Column(String)  
 broker\_id = Column(UUID(as\_uuid=True), ForeignKey("brokers.id"))  
 paper\_trading = Column(Boolean, default=False)  
 webhook\_path = Column(String, unique=True)  
 status = Column(Enum(StrategyStatus), default=StrategyStatus.active)  
 created\_at = Column(DateTime(timezone=True), server\_default=func.now())

#### schemas/strategy.py

from pydantic import BaseModel  
from uuid import UUID  
from datetime import datetime  
from enum import Enum  
  
class StrategyStatus(str, Enum):  
 active = "active"  
 paused = "paused"  
 error = "error"  
  
class StrategyBase(BaseModel):  
 name: str  
 script: str  
 broker\_id: UUID  
 paper\_trading: bool = False  
  
class StrategyCreate(StrategyBase):  
 pass  
  
class StrategyRead(StrategyBase):  
 id: UUID  
 status: StrategyStatus  
 webhook\_path: str  
 created\_at: datetime  
  
 class Config:  
 orm\_mode = True

#### api/v1/strategies.py

from fastapi import APIRouter, Depends, HTTPException  
from sqlalchemy.orm import Session  
from uuid import uuid4  
  
from app.schemas.strategy import StrategyCreate, StrategyRead  
from app.models.strategy import Strategy  
from app.db import get\_db  
  
from app.core.deps import get\_current\_user  
  
router = APIRouter(dependencies=[Depends(get\_current\_user)])  
  
@router.post("/strategies", response\_model=StrategyRead)  
def create\_strategy(payload: StrategyCreate, db: Session = Depends(get\_db)):  
 webhook\_path = f"webhook-{uuid4()}"  
 strategy = Strategy(  
 id=uuid4(),  
 name=payload.name,  
 script=payload.script,  
 broker\_id=payload.broker\_id,  
 paper\_trading=payload.paper\_trading,  
 webhook\_path=webhook\_path,  
 )  
 db.add(strategy)  
 db.commit()  
 db.refresh(strategy)  
 return strategy

### 🔐 Auth Service & Route

#### services/auth\_service.py

from passlib.context import CryptContext  
from jose import jwt, JWTError  
from datetime import datetime, timedelta  
  
SECRET\_KEY = "your-secret-key"  
ALGORITHM = "HS256"  
ACCESS\_TOKEN\_EXPIRE\_MINUTES = 60  
pwd\_context = CryptContext(schemes=["bcrypt"], deprecated="auto")  
  
def hash\_password(password: str) -> str:  
 return pwd\_context.hash(password)  
  
def verify\_password(plain: str, hashed: str) -> bool:  
 return pwd\_context.verify(plain, hashed)  
  
def create\_access\_token(data: dict, expires\_delta: timedelta = None):  
 to\_encode = data.copy()  
 expire = datetime.utcnow() + (expires\_delta or timedelta(minutes=ACCESS\_TOKEN\_EXPIRE\_MINUTES))  
 to\_encode.update({"exp": expire})  
 return jwt.encode(to\_encode, SECRET\_KEY, algorithm=ALGORITHM)

#### api/v1/auth.py

from fastapi import APIRouter, Depends, HTTPException  
from sqlalchemy.orm import Session  
from app.schemas.user import UserCreate, UserRead  
from app.models.user import User  
from app.db import get\_db  
from app.services.auth\_service import hash\_password, verify\_password, create\_access\_token  
from uuid import uuid4  
  
from app.core.deps import get\_current\_user  
  
router = APIRouter(dependencies=[Depends(get\_current\_user)])  
  
@router.post("/register", response\_model=UserRead)  
def register\_user(user: UserCreate, db: Session = Depends(get\_db)):  
 existing = db.query(User).filter(User.email == user.email).first()  
 if existing:  
 raise HTTPException(status\_code=400, detail="Email already registered")  
 new\_user = User(  
 id=uuid4(),  
 email=user.email,  
 password\_hash=hash\_password(user.password)  
 )  
 db.add(new\_user)  
 db.commit()  
 db.refresh(new\_user)  
 return new\_user  
  
@router.post("/login")  
def login\_user(user: UserCreate, db: Session = Depends(get\_db)):  
 db\_user = db.query(User).filter(User.email == user.email).first()  
 if not db\_user or not verify\_password(user.password, db\_user.password\_hash):  
 raise HTTPException(status\_code=401, detail="Invalid credentials")  
 token = create\_access\_token({"sub": str(db\_user.id)})  
 return {"access\_token": token, "token\_type": "bearer"}

#### models/user.py

from sqlalchemy import Column, String, Enum, DateTime  
from sqlalchemy.dialects.postgresql import UUID  
from sqlalchemy.sql import func  
import enum  
import uuid  
  
from ..db import Base  
  
class UserRole(str, enum.Enum):  
 user = "user"  
 admin = "admin"  
  
class UserStatus(str, enum.Enum):  
 active = "active"  
 disabled = "disabled"  
  
class User(Base):  
 \_\_tablename\_\_ = "users"  
  
 id = Column(UUID(as\_uuid=True), primary\_key=True, default=uuid.uuid4)  
 email = Column(String, unique=True, nullable=False)  
 password\_hash = Column(String, nullable=False)  
 role = Column(Enum(UserRole), default=UserRole.user)  
 status = Column(Enum(UserStatus), default=UserStatus.active)  
 created\_at = Column(DateTime(timezone=True), server\_default=func.now())

#### schemas/user.py

from pydantic import BaseModel, EmailStr  
from uuid import UUID  
from datetime import datetime  
from enum import Enum  
  
class UserRole(str, Enum):  
 user = "user"  
 admin = "admin"  
  
class UserStatus(str, Enum):  
 active = "active"  
 disabled = "disabled"  
  
class UserBase(BaseModel):  
 email: EmailStr  
 role: UserRole = UserRole.user  
 status: UserStatus = UserStatus.active  
  
class UserCreate(UserBase):  
 password: str  
  
class UserRead(UserBase):  
 id: UUID  
 created\_at: datetime  
  
 class Config:  
 orm\_mode = True

backend/  
├── app/  
│ ├── main.py # FastAPI app instance  
│ ├── api/ # Route definitions  
│ │ └── v1/  
│ │ ├── auth.py  
│ │ ├── strategies.py  
│ │ ├── webhooks.py  
│ │ └── admin.py  
│ ├── models/ # SQLAlchemy models  
│ │ ├── user.py  
│ │ ├── strategy.py  
│ │ ├── alert.py  
│ │ └── execution.py  
│ ├── schemas/ # Pydantic schemas  
│ │ ├── user.py  
│ │ ├── strategy.py  
│ │ └── alert.py  
│ ├── services/ # Business logic  
│ │ ├── auth\_service.py  
│ │ ├── trade\_executor.py  
│ │ ├── notifier.py  
│ │ └── paper\_engine.py  
│ ├── core/ # Settings and utils  
│ │ ├── config.py  
│ │ └── security.py  
│ └── db.py # DB session / engine  
├── tests/ # Pytest-based test suite  
└── Dockerfile

This structure supports modular service layering, multi-version APIs, and clear domain separation.

### 🐳 Docker Setup (FastAPI + Next.js + PostgreSQL)

#### docker-compose.yml

version: '3.9'  
  
services:  
 db:  
 image: postgres:15  
 restart: always  
 environment:  
 POSTGRES\_USER: at\_user  
 POSTGRES\_PASSWORD: at\_pass  
 POSTGRES\_DB: autotrading  
 volumes:  
 - postgres\_data:/var/lib/postgresql/data  
 ports:  
 - "5432:5432"  
  
 backend:  
 build:  
 context: ./backend  
 command: uvicorn app.main:app --host 0.0.0.0 --port 8000 --reload  
 volumes:  
 - ./backend:/app  
 ports:  
 - "8000:8000"  
 depends\_on:  
 - db  
 environment:  
 DATABASE\_URL: postgresql://at\_user:at\_pass@db:5432/autotrading  
  
 frontend:  
 build:  
 context: ./frontend  
 volumes:  
 - ./frontend:/app  
 ports:  
 - "3000:3000"  
 command: npm run dev  
 depends\_on:  
 - backend  
  
volumes:  
 postgres\_data:

### 📦 Backend Dockerfile (backend/Dockerfile)

FROM python:3.11-slim  
  
WORKDIR /app  
  
COPY ./requirements.txt /app/requirements.txt  
RUN pip install --no-cache-dir -r requirements.txt  
  
COPY . /app  
  
CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000"]

#### backend/requirements.txt

fastapi  
uvicorn[standard]  
sqlalchemy  
psycopg2-binary  
python-jose  
passlib[bcrypt]  
python-dotenv

### 🌐 Frontend Dockerfile (frontend/Dockerfile)

FROM node:18-alpine  
  
WORKDIR /app  
  
COPY package.json package-lock.json ./  
RUN npm install  
  
COPY . .  
  
CMD ["npm", "run", "dev"]

### AI-Enhanced Tooling

#### 🔧 Developer Productivity

* **GitHub Copilot**: AI-assisted code generation for backend and frontend
* **Cursor IDE**: Smart IDE for debugging and architectural code refactoring
* **Mintlify / Documatic**: Auto-generated API documentation from source code

#### 📈 Strategy Simulation & Debugging

* **ChatGPT / Claude.ai**: Assist with Pine Script debugging and webhook parsing
* **Deepnote**: Collaboratively analyze paper trades and visualize P&L

#### 📊 Testing & Monitoring

* **Testim / Playwright AI**: AI-generated UI test automation for trading workflows
* **New Relic AI / Datadog AI Assist**: Automated log/error pattern analysis for performance issues

#### 💬 Customer Support + Internal Ops

* **Tidio / Intercom AI Bots**: Strategy-aware support bots for users
* **Langchain + Pinecone (optional)**: Admin assistant for logs, order audit, or error retry flows

### Tech Stack

* DhanHQ prioritized via BrokerAdapter interface (default implementation)
* Frontend: Next.js + Tailwind CSS
* Backend: FastAPI or NestJS (Node.js)
* DB: PostgreSQL
* Auth: JWT + bcrypt
* Infra: Docker, Jenkins, Amazon Lightsail
* Alerts: SendGrid (Email), Telegram Bot API

### Steps

1. Auth system + broker token linking (UI + backend)
2. StrategyManager API + webhook URL generation
3. Webhook listener for TradingView
4. TradeExecutor service for DhanHQ + PaperTradeEngine
5. AlertNotifier integration (email + Telegram)
6. Reports dashboard + CSV export
7. Admin Panel (user list, strategy view, disable/reset broker)
8. CI/CD with Docker + Jenkins
9. Auth system + broker token linking (UI + backend)
10. StrategyManager API + webhook URL generation
11. Webhook listener for TradingView
12. TradeExecutor service for DhanHQ + PaperTradeEngine
13. AlertNotifier integration (email + Telegram)
14. Reports dashboard + CSV export
15. Admin Panel (user list, strategy view, disable/reset broker)
16. CI/CD with Docker + Jenkins

## Milestones

**Week 1:** Setup, auth, Docker, Jenkins, Lightsail deployment  
**Week 2:** Broker integration + StrategyManager  
**Week 3:** Webhook handling + PaperTradeEngine  
**Week 4:** Live trading via DhanHQ + retries  
**Week 5:** Telegram/email alerting  
**Week 6:** Reports UI + export CSV  
**Week 7:** Admin panel + testing + edge cases

## Gathering Results

### Functional

* Strategies can be deployed, triggered, and logged (paper/live)
* Broker integration and trade execution reliable
* Alert delivery works with <2s latency
* Reports export correct and filterable
* Admin can manage users and view activity

### Performance

* Webhook handling ≥10/sec
* Execution success rate ≥98%
* Paper trade logic matches expected P&L logic
* System uptime ≥99%

## Need Professional Help in Developing Your Architecture?

Please contact me at [sammuti.com](https://sammuti.com) :)