

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 $\geq 10/\text{sec}$
- Execution success rate $\geq 98\%$
- Paper trade logic matches expected P&L logic
- System uptime $\geq 99\%$

Need Professional Help in Developing Your Architecture?

Please contact me at sammuti.com :)