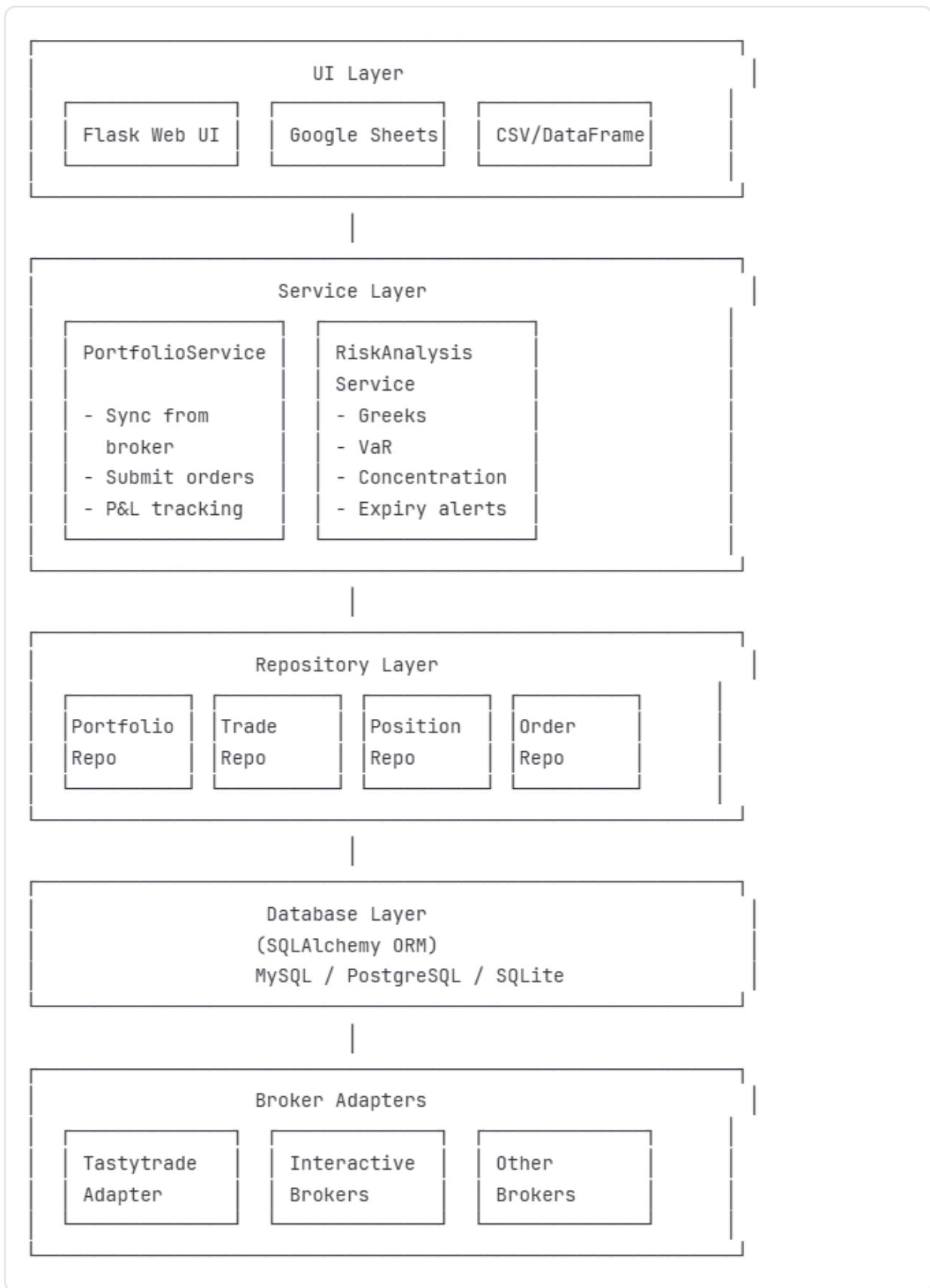


## Architecture Overview



# Core Domain Models

## Symbol

Represents any tradeable instrument (stocks, options, futures)

- Handles OCC option symbol format
- Asset type abstraction
- Multiplier support

## Leg

Individual component of a trade

- Tracks entry/exit prices and times
- Side (buy/sell, open/close)
- P&L calculation
- Greeks (for options)

## Trade

Logical grouping of legs

- One or more legs
- Associated with a Strategy
- Tracks planned entry/exit/stop loss
- Open/closed status
- P&L aggregation

## Strategy

Defines option strategy templates

- Iron Condor, Vertical Spreads, etc.
- Max profit/loss calculations
- Break-even points
- Aggregated Greeks

## Position

Current holdings (broker snapshot)

- Net position across all trades
- Real-time pricing
- Greeks aggregation
- Cost basis tracking

## Order

## Submitted orders

- Multiple legs for complex orders
- Status tracking (pending, filled, cancelled)
- Price types (market, limit, stop)

## Portfolio

### Top-level container

- Multiple trades and positions
- Cash and buying power
- Portfolio-level Greeks
- Total equity and P&L

### Pip Install

```
pip install sqlalchemy
pip install requests
pip install gspread oauth2client
pip install pandas
pip install flask
```

### Database set up

```
from core_models import init_database, get_session

# SQLite (for development)
engine = init_database("sqlite:///portfolio.db")

# PostgreSQL (for production)
# engine = init_database("postgresql://user:password@localhost/portfolio")

# MySQL
# engine = init_database("mysql+pymysql://user:password@localhost/portfolio")

session = get_session(engine)
```

### Usage example

```
from broker_adapters import TastytradeAdapter
from service_layer import PortfolioService

# Initialize broker adapter
broker = TastytradeAdapter(
    account_id="your_account",
    username="your_username",
    password="your_password"
)
broker.authenticate()

# Create service
service = PortfolioService(session, broker)
```

```

# Create and sync portfolio
portfolio = service.create_portfolio_from_broker(
    broker_name="tastytrade",
    account_id="your_account",
    name="My Trading Portfolio"
)

# Get portfolio summary
summary = service.get_portfolio_summary(portfolio.id)
print(f"Total Equity: ${summary['total_equity']:.2f}")
print(f"Total P&L: ${summary['total_pnl']:.2f}")

```

#### Build and track strategies

```

from service_layer import StrategyBuilder
from datetime import datetime, timedelta
from decimal import Decimal

# Build an Iron Condor
expiration = datetime.now() + timedelta(days=45)
trade = StrategyBuilder.iron_condor(
    underlying="SPY",
    expiration=expiration,
    put_short_strike=Decimal("450"),
    put_long_strike=Decimal("445"),
    call_short_strike=Decimal("470"),
    call_long_strike=Decimal("475"),
    quantity=1
)

# Save trade
trade_repo = TradeRepository(session)
trade_repo.create(trade, portfolio.id)

```

#### Risk Analysis

```

from service_layer import RiskAnalysisService

risk_service = RiskAnalysisService(session)

# Get portfolio risk metrics
risk = risk_service.calculate_portfolio_risk(portfolio.id)
print(f"Portfolio Delta: {risk['greeks']['delta']:.2f}")
print(f"Portfolio Theta: {risk['greeks']['theta']:.2f}")

# Get expiring positions
expiring = risk_service.get_expiring_positions(portfolio.id, days=7)
for pos in expiring:
    print(f"{pos['symbol']} expires in {pos['days_to_expiry']} days")

# Calculate Value at Risk
var = risk_service.calculate_var(portfolio.id, confidence=0.95)
print(f"95% VaR: ${var:.2f}")

```

[Export to GoogleSheets](#)

```

from ui_export_layer import GoogleSheetsExporter

# Setup Google Sheets exporter
exporter = GoogleSheetsExporter('credentials.json')

# Export portfolio
summary = service.get_portfolio_summary(portfolio.id)
positions = service.get_positions_summary(portfolio.id)
trades = service.get_trades_summary(portfolio.id)

url = exporter.export_portfolio(
    summary,
    positions,
    trades,
    spreadsheet_name="My Portfolio Dashboard"
)
print(f"View at: {url}")

```

#### Run web ui

```

from ui_export_layer import PortfolioWebUI

# Create web UI
ui = PortfolioWebUI(service)

# Run server
ui.run(host='0.0.0.0', port=5000)
# Visit http://localhost:5000

```

#### Submit Orders

```

from core_models import Order, Leg, Symbol, OrderSide, OrderType

# Create a simple buy order
order = Order(
    order_type=OrderType.LIMIT,
    limit_price=Decimal("1.50"),
    legs=[
        Leg(
            symbol=Symbol("SPY", AssetType.OPTION, OptionType.CALL,
                         Decimal("470"), expiration, multiplier=100),
            quantity=1,
            side=OrderSide.BUY_TO_OPEN
        )
    ]
)

# Submit to broker
broker_order_id = service.submit_order(portfolio.id, order)
print(f"Order submitted: {broker_order_id}")

```

#### Google Sheets Setup

1. Create a Google Cloud Project
2. Enable Google Sheets API and Google Drive API
3. Create a Service Account
4. Download credentials JSON file
5. Share your Google Sheet with the service account email

python

```
exporter = GoogleSheetsExporter('path/to/credentials.json')
```

## Database Schema

### Key Tables

#### portfolios

- Portfolio metadata
- Account balances
- Portfolio-level Greeks

#### trades

- Trade details
- Strategy association
- Open/closed status
- P&L tracking

#### legs

- Individual leg details
- Entry/exit prices
- Associated with trades or orders

#### positions

- Current positions snapshot
- Real-time pricing
- Greeks

#### orders

- Order history
- Status tracking
- Execution details

#### symbols

- Master symbol table
- Option contract details

#### strategies

- Strategy templates
- Risk parameters

## Adding New Brokers

# ..To add a new broker, implement the `BrokerAdapter` interface:

```
python
from broker_adapters import BrokerAdapter

class MyBrokerAdapter(BrokerAdapter):
    def authenticate(self) -> bool:
        # Implement authentication
        pass

    def get_positions(self) -> List[Position]:
        # Implement position fetching
        pass

    # ... implement other required methods
```

Then register in the factory:

```
python
# In broker_adapters.py
class BrokerFactory:
    @staticmethod
    def create_adapter(broker_name: str, **kwargs) -> BrokerAdapter:
        adapters = {
            "tastytrade": TastytradeAdapter,
            "mybroker": MyBrokerAdapter, # Add here
        }
```

## Key Design Principles

1. **Broker Agnostic:** All broker-specific logic isolated in adapters
2. **Rich Domain Models:** Business logic in domain objects
3. **Repository Pattern:** Data access abstraction
4. **Service Layer:** Orchestrates business operations
5. **Multi-leg Support:** Native support for complex strategies

6. **Flexible Storage:** Works with any SQL database

## Performance Considerations

- Use database indexes on frequently queried fields (ticker, dates, status)
- Batch position updates during sync
- Consider caching for real-time quotes
- Use connection pooling for database
- Implement rate limiting for broker API calls

## Security Best Practices

- Never commit credentials to version control
- Use environment variables for sensitive data
- Encrypt database in production
- Use HTTPS for web UI
- Implement proper authentication/authorization
- Rotate API keys regularly

## Future Enhancements

- Backtesting framework
- Alert system (price, Greeks thresholds)
- Advanced P&L analytics
- Tax lot tracking
- Mobile app
- Real-time WebSocket updates
- Machine learning for strategy optimization
- Integration with charting libraries

