MT5 Trading System - Implementation Action Plan

📋 Project Overview

Goal: Implement a complete MetaTrader 5 Python trading system with 10 classes and ~120 methods **Estimated Timeline**: 4-6 weeks **Team Size**: 1-3 developers

---

Phase 1: Project Setup & Foundation (Week 1)

1.1 Environment Setup

- ☑ Install Python 3.8+ (recommended 3.10+)
- ☑ Install MetaTrader 5 terminal
- ☑ Create virtual environment

```
python -m venv venv
source venv/bin/activate  # Linux/Mac
venv\Scripts\activate     # Windows
```

- ☑ Install required packages

```
pip install MetaTrader5 pandas numpy python-dateutil
```

- ☑ Install development tools

```
pip install pytest pytest-cov black flake8 mypy
```

1.2 Project Structure

- ☑ Create project directory structure:

```
mt5_trading_system/
├── mymt5/
│   ├── __init__.py
│   ├── client.py
│   ├── account.py
│   ├── symbol.py
│   ├── data.py
│   ├── trade.py
│   ├── history.py
│   ├── risk.py
│   ├── terminal.py
│   ├── validator.py
│   ├── utils.py
│   └── enums.py
├── tests/
│   ├── __init__.py
│   ├── test_client.py
│   ├── test_account.py
│   ├── test_symbol.py
│   ├── test_data.py
│   ├── test_trade.py
│   ├── test_history.py
│   ├── test_risk.py
│   ├── test_terminal.py
│   ├── test_validator.py
│   └── test_utils.py
├── config/
│   ├── config.json
│   └── config.example.json
├── logs/
├── data/
├── docs/
│   ├── architecture.md
│   └── api_reference.md
├── examples/
│   ├── basic_usage.py
│   ├── trading_example.py
│   └── risk_management_example.py
├── requirements.txt
├── setup.py
├── README.md
└── .gitignore
```

1.3 Configuration Files

- ☑ Create `requirements.txt`
- ☐ Create `setup.py`
- ☑ Create `config.example.json` with template

- ☑ Create `.gitignore` for Python projects
- ☑ Create `README.md` with project overview
- ☑ Set up logging configuration

1.4 Version Control

- ☑ Initialize Git repository
- ☑ Create `.gitignore`
- ☑ Make initial commit
- ☑ Create development branch
- ☑ Set up GitHub/GitLab repository (if team project)

Phase 2: Core Implementation - Enums & Utils (Week 1)

2.1 Enumerations (enums.py)

- ☑ Create `ConnectionState` enum
  - ☑ DISCONNECTED
  - ☑ CONNECTED
  - ☑ FAILED
  - ☑ INITIALIZING
  - ☑ RECONNECTING
- ☑ Create `OrderType` enum
  - ☑ BUY
  - ☑ SELL
  - ☑ BUY_LIMIT
  - ☑ SELL_LIMIT
  - ☑ BUY_STOP
  - ☑ SELL_STOP
  - ☑ BUY_STOP_LIMIT
  - ☑ SELL_STOP_LIMIT

- ☑ Create `TimeFrame` enum
  - ☑ M1, M5, M15, M30
  - ☑ H1, H4
  - ☑ D1, W1, MN1
- ☑ Write unit tests for enums

2.2 Utilities Class (utils.py)

Time Operations

- ☑ Implement `convert_time()`
- ☑ Implement `get_time()`
- ☑ Test time conversions

Price Operations

- ☑ Implement `convert_price()`
- ☑ Implement `format_price()`
- ☑ Implement `round_price()`
- ☑ Test price operations

Volume Operations

- ☑ Implement `convert_volume()`
- ☑ Implement `round_volume()`
- ☑ Test volume operations

Type Conversions

- ☑ Implement `convert_type()`
- ☑ Test type conversions

Data Formatting

- ☑ Implement `to_dict()`
- ☑ Implement `to_dataframe()`
- ☑ Test data formatting

File Operations

- ☑ Implement `save()` (JSON, CSV, pickle)
- ☑ Implement `load()`
- ☑ Test file operations

Calculations

- ☑ Implement `calculate()` helper
- ☑ Test calculations
- ☑ Write comprehensive unit tests for MT5Utils
- ☑ Document all utility functions

Phase 3: Core Layer - Client (Week 1-2)

3.1 MT5Client Class (client.py)

Basic Structure

- ☑ Create class skeleton
- ☑ Initialize attributes
- ☑ Set up logging system

Connection Management

- ☑ Implement `__init__()`
- ☑ Implement `initialize()`
- ☑ Implement `connect()`
- ☑ Implement `disconnect()`
- ☑ Implement `shutdown()`
- ☑ Implement `is_connected()`
- ☑ Implement `ping()`
- ☑ Test connection lifecycle

Authentication

- ☑ Implement `login()`

- ☑ Implement `logout()`
- ☑ Test authentication

Auto-Reconnection

- ☑ Implement `reconnect()`
- ☑ Implement `enable_auto_reconnect()`
- ☑ Implement `disable_auto_reconnect()`
- ☑ Implement `set_retry_attempts()`
- ☑ Implement `set_retry_delay()`
- ☑ Implement `_handle_reconnection()` (private)
- ☑ Test auto-reconnection logic

Configuration

- ☑ Implement `configure()`
- ☑ Implement `get_config()`
- ☑ Implement `load_config()`
- ☑ Implement `save_config()`
- ☑ Test configuration management

Multi-Account Support

- ☑ Implement `switch_account()`
- ☑ Implement `save_account()`
- ☑ Implement `load_account()`
- ☑ Implement `list_accounts()`
- ☑ Implement `remove_account()`
- ☑ Test multi-account features

Event System

- ☑ Implement `on()` (register callback)
- ☑ Implement `off()` (unregister callback)
- ☑ Implement `trigger_event()`
- ☑ Test event callbacks

Status & Diagnostics

- ☑ Implement `get_status()`
- ☑ Implement `get_connection_statistics()`
- ☑ Test status methods

Error Handling

- ☑ Implement `get_error()`
- ☑ Implement `handle_error()`
- ☑ Test error handling

Utility Methods

- ☑ Implement `reset()`
- ☑ Implement `export_logs()`
- ☑ Test utility methods

Testing & Documentation

- ☑ Write unit tests for all methods
- ☑ Write integration tests
- ☑ Document all public methods
- ☑ Create usage examples

Phase 4: Information Layer (Week 2)

4.1 MT5Account Class (account.py)

- ☑ Create class skeleton with dependencies
- ☑ Implement `__init__()`

Account Information

- ☑ Implement `get()` unified getter
  - ☑ Support for 'balance', 'equity', 'margin', etc.
  - ☑ Return all info when attribute=None
- ☑ Implement `_fetch_account_info()` (private)
- ☑ Test account info retrieval

Account Status

- ☑ Implement `check()` for status checks
  - ☑ 'demo', 'authorized', 'trade_allowed', 'expert_allowed'
- ☑ Test status checks

Account Metrics

- ☑ Implement `calculate()` for metrics
  - ☑ 'margin_level'
  - ☑ 'drawdown'
  - ☑ 'health'
  - ☑ 'margin_required'
- ☑ Implement `_calculate_margin_level()` (private)
- ☑ Implement `_calculate_drawdown()` (private)
- ☑ Implement `_calculate_health_metrics()` (private)
- ☑ Test calculations

Credentials & Export

- ☑ Implement `validate_credentials()`
- ☑ Implement `get_summary()`
- ☑ Implement `export()`
- ☑ Test validation and export

Testing & Documentation

- ☑ Write unit tests with mocked client
- ☑ Write integration tests
- ☑ Document all methods
- ☑ Create usage examples

4.2 MT5Symbol Class (symbol.py)

- ☑ Create class skeleton
- ☑ Implement `__init__()`

Symbol Discovery

- ☑ Implement `get_symbols()`
  - ☑ 'all', 'market_watch', 'group', 'search'
- ☑ Test symbol discovery

Market Watch Management

- ☑ Implement `initialize()`
- ☑ Implement `manage()`
  - ☑ 'add', 'remove', 'select', 'deselect'
- ☑ Test market watch operations

Symbol Information

- ☑ Implement `get_info()` unified getter
  - ☑ Support all symbol attributes
  - ☑ Return all info when attribute=None
- ☑ Implement `_fetch_symbol_info()` (private)
- ☑ Implement `_update_cache()` (private)
- ☑ Test symbol info

Symbol Status

- ☑ Implement `check()`
  - ☑ 'available', 'visible', 'tradable', 'market_open'
- ☑ Test status checks

Real-Time Prices

- ☑ Implement `get_price()`
  - ☑ 'current', 'bid', 'ask', 'last'
- ☑ Test price retrieval

Market Depth

- ☑ Implement `get_depth()`
- ☑ Test market depth

Subscriptions

- ☑ Implement `subscribe()`

- ☑ Implement `unsubscribe()`
- ☑ Test subscriptions

Validation

- ☑ Implement `validate()`
  - ☑ 'exists', 'tradable', 'volume'
- ☑ Implement `validate_volume()`
- ☑ Test validation

Utility

- ☑ Implement `get_summary()`
- ☑ Implement `export_list()`
- ☑ Test utility methods

Testing & Documentation

- ☑ Write unit tests
- ☑ Write integration tests
- ☑ Document all methods
- ☑ Create usage examples

4.3 MT5Terminal Class (terminal.py)

- ☑ Create class skeleton
- ☑ Implement `__init__()`

Terminal Information

- ☑ Implement `get()` unified getter
  - ☑ All terminal attributes
- ☑ Implement `_fetch_terminal_info()` (private)
- ☑ Test info retrieval

Terminal Status

- ☑ Implement `check()`
  - ☑ All status checks

- ☑ Test status checks

Terminal Properties

- ☑ Implement `get_properties()`
  - ☑ 'resources', 'display', 'limits', 'all'
- ☑ Implement `_get_resources()` (private)
- ☑ Implement `_get_display_info()` (private)
- ☑ Implement `_get_limits()` (private)
- ☑ Test property retrieval

Utility

- ☑ Implement `get_summary()`
- ☑ Implement `print_info()`
- ☑ Implement `export()`
- ☑ Implement `check_compatibility()`
- ☑ Test utility methods

Testing & Documentation

- ☑ Write unit tests
- ☑ Document all methods
- ☑ Create usage examples

Phase 5: Data Layer (Week 2-3)

5.1 MT5Data Class (data.py)

- ☑ Create class skeleton
- ☑ Implement `__init__()`

OHLCV Data

- ☑ Implement `get_bars()`
  - ☑ Support count parameter
  - ☑ Support start/end date range

    - ☑ Return DataFrame or dict
- ☑ Test bar retrieval with different parameters

Tick Data

- ☑ Implement `get_ticks()`
    - ☑ Support count parameter
    - ☑ Support start/end date range
    - ☑ Support different tick flags
- ☑ Test tick retrieval

Streaming

- ☑ Implement `stream()`
    - ☑ 'ticks', 'bars'
    - ☑ Callback mechanism
- ☑ Implement `stop_stream()`
- ☑ Test streaming functionality

Data Processing

- ☑ Implement `process()`
    - ☑ 'normalize' operation
    - ☑ 'clean' operation
    - ☑ 'resample' operation
    - ☑ 'fill_missing' operation
    - ☑ 'detect_gaps' operation
- ☑ Implement `_normalize_data()` (private)
- ☑ Implement `_clean_data()` (private)
- ☑ Implement `_fill_missing()` (private)
- ☑ Implement `_detect_gaps()` (private)
- ☑ Test all processing operations

Caching

- ☑ Implement `cache()`
- ☑ Implement `get_cached()`
- ☑ Implement `clear_cache()`
- ☑ Test caching mechanism

Export

- ☑ Implement `export()`
  - ☑ CSV format
  - ☑ JSON format
  - ☑ Parquet format
  - ☑ Database export
- ☑ Test all export formats

Timeframe Utilities

- ☑ Implement `get_timeframes()`
- ☑ Implement `convert_timeframe()`
- ☑ Test timeframe operations

Statistics

- ☑ Implement `get_summary()`
- ☑ Implement `calculate_stats()`
- ☑ Test statistics

Testing & Documentation

- ☑ Write unit tests
- ☑ Write integration tests
- ☑ Document all methods
- ☑ Create usage examples

5.2 MT5History Class (history.py)

- ☑ Create class skeleton
- ☑ Implement `__init__()`

History Retrieval

- ☑ Implement `get()`
  - ☑ 'deals', 'orders', 'both'
  - ☑ Support filters
- ☑ Implement `_fetch_deals()` (private)

- ☑ Implement `_fetch_orders()` (private)
- ☑ Test history retrieval

Quick Access

- ☑ Implement `get_today()`
- ☑ Implement `get_period()`
  - ☑ 'day', 'week', 'month', 'year'
- ☑ Test quick access methods

Performance Metrics

- ☑ Implement `calculate()`
  - ☑ 'win_rate'
  - ☑ 'profit_factor'
  - ☑ 'avg_win', 'avg_loss'
  - ☑ 'largest_win', 'largest_loss'
  - ☑ 'sharpe_ratio'
  - ☑ 'max_drawdown'
  - ☑ 'total_trades', 'total_profit'
  - ☑ 'total_commission', 'total_swap'
- ☑ Implement `_calculate_win_rate()` (private)
- ☑ Implement `_calculate_profit_factor()` (private)
- ☑ Implement `_calculate_sharpe_ratio()` (private)
- ☑ Implement `_calculate_max_drawdown()` (private)
- ☑ Test all calculations

Trade Analysis

- ☑ Implement `analyze()`
  - ☑ 'by_symbol'
  - ☑ 'by_hour', 'by_day', 'by_weekday', 'by_month'
  - ☑ 'winning_trades', 'losing_trades'
  - ☑ 'statistics'
- ☑ Implement `_analyze_by_symbol()` (private)
- ☑ Implement `_analyze_by_time()` (private)
- ☑ Test analysis methods

Reports

- ☑ Implement `generate_report()`
  - ☑ 'performance', 'trade_log', 'summary', 'detailed'
  - ☑ Support dict, dataframe, html, text formats
- ☑ Test report generation

Export & Summary

- ☑ Implement `export()`
- ☑ Implement `get_summary()`
- ☑ Implement `print_report()`
- ☑ Test export and summary

Testing & Documentation

- ☑ Write unit tests
- ☑ Write integration tests
- ☑ Document all methods
- ☑ Create usage examples

Phase 6: Trading Layer (Week 3-4) ✅ COMPLETED ✅ VERIFIED

6.1 MT5Trade Class (trade.py)

- ☑ Create class skeleton with dependencies
- ☑ Implement `__init__()`

Order Execution

- ☑ Implement `execute()` unified method
  - ☑ Support all order types
  - ☑ Build request dict
  - ☑ Send to MT5
- ☑ Implement `buy()` simplified
- ☑ Implement `sell()` simplified
- ☑ Implement `build_request()` helper

- ☑ Implement `_send_request()` (private)
- ☑ Test order execution

Order Management

- ☑ Implement `get_orders()`
  - ☑ Support all filter types (symbol, ticket, group)
- ☑ Implement `modify_order()`
- ☑ Implement `cancel_order()`
  - ☑ Single order
  - ☑ By filter (symbol)
  - ☑ All orders
- ☑ Filtering implemented inline (no separate `_filter_orders()` needed)
- ☑ Test order management

Position Management

- ☑ Implement `get_positions()`
  - ☑ Support all filter types (symbol, ticket, group)
- ☑ Implement `modify_position()`
- ☑ Implement `close_position()`
  - ☑ Full close
  - ☑ Partial close
  - ☑ By filter (symbol)
  - ☑ All positions
- ☑ Implement `reverse_position()`
- ☑ Filtering implemented inline (no separate `_filter_positions()` needed)
- ☑ Test position management

Position Analytics

- ☑ Implement `analyze_position()`
  - ☑ 'profit', 'profit_points', 'duration'
  - ☑ 'current_price', 'entry_price', 'volume'
  - ☑ 'all' (return dict)
- ☑ Implement `get_position_stats()`
- ☑ Calculation implemented inline (no separate `_calculate_position_profit()` needed)
- ☑ Test analytics

Validation & Utility

- ☑ Implement `validate_request()`
- ☑ Implement `check_order()`
- ☑ Implement `get_summary()`
- ☑ Implement `export()`
- ☑ Test validation and utility

Testing & Documentation

- ☑ Write unit tests with mocked dependencies
- ☑ Write integration tests
- ☑ Test error scenarios
- ☑ Document all methods
- ☑ Create comprehensive trading examples

6.2 MT5Risk Class (risk.py)

- ☑ Create class skeleton with dependencies
- ☑ Implement `__init__()`

Position Sizing

- ☑ Implement `calculate_size()`
  - ☑ 'percent' method
  - ☑ 'amount' method
  - ☑ 'ratio' method
- ☑ Implement `_calculate_position_size_percent()` (private)
- ☑ Implement `_calculate_position_size_amount()` (private)
- ☑ Test position sizing

Risk Calculation

- ☑ Implement `calculate_risk()`
  - ☑ 'amount' metric
  - ☑ 'percent' metric
  - ☑ 'reward_ratio' metric
  - ☑ 'all' (return dict)

- ☑ Implement `_calculate_risk_amount()` (private)
- ☑ Implement `_calculate_risk_percent()` (private)
- ☑ Test risk calculations

Risk Limits

- ☑ Implement `set_limit()`
  - ☑ 'max_risk_per_trade'
  - ☑ 'max_daily_loss'
  - ☑ 'max_positions'
  - ☑ 'max_symbol_positions'
  - ☑ 'max_total_exposure'
- ☑ Implement `get_limit()`
- ☑ Test limit management

Risk Validation

- ☑ Implement `validate()`
  - ☑ Check all limits
  - ☑ Return violations
- ☑ Implement `check()`
  - ☑ 'trade_allowed'
  - ☑ 'margin_available'
  - ☑ 'risk_within_limits'
  - ☑ 'stop_loss_valid'
  - ☑ 'take_profit_valid'
- ☑ Implement `_check_risk_limits()` (private)
- ☑ Test validation

Portfolio Risk

- ☑ Implement `get_portfolio_risk()`
  - ☑ 'total_exposure'
  - ☑ 'total_risk'
  - ☑ 'correlation_risk'
  - ☑ 'margin_usage'
  - ☑ 'all' (return dict)
- ☑ Implement `_calculate_total_exposure()` (private)
- ☑ Implement `_calculate_correlation_risk()` (private)

- ☑ Test portfolio risk

Utility

- ☑ Implement `get_summary()`
- ☑ Implement `export_limits()`
- ☑ Test utility methods

Testing & Documentation

- ☑ Write unit tests
- ☑ Write integration tests
- ☑ Test edge cases
- ☑ Document all methods
- ☑ Create risk management examples

Phase 7: Utility Layer (Week 4)

7.1 MT5Validator Class (validator.py)

- ☑ Create class skeleton
- ☑ Implement `__init__()`

Master Validation

- ☑ Implement `validate()` unified method
  - ☑ Route to specific validators
  - ☑ Return (bool, error_message)

Specific Validators (Private Methods)

- ☑ Implement `_validate_symbol()`
- ☑ Implement `_validate_volume()`
- ☑ Implement `_validate_price()`
- ☑ Implement `_validate_stop_loss()`
- ☑ Implement `_validate_take_profit()`
- ☑ Implement `_validate_order_type()`

- ☑ Implement `_validate_magic()`
- ☑ Implement `_validate_deviation()`
- ☑ Implement `_validate_expiration()`
- ☑ Implement `_validate_timeframe()`
- ☑ Implement `_validate_date_range()`
- ☑ Implement `_validate_trade_request()`
- ☑ Implement `_validate_credentials()`
- ☑ Implement `_validate_margin()`
- ☑ Implement `_validate_ticket()`

Batch Validation

- ☑ Implement `validate_multiple()`
- ☑ Test batch validation

Utility

- ☑ Implement `get_validation_rules()`
- ☑ Test validation rules

Testing & Documentation

- ☑ Write unit tests for each validator
- ☑ Test edge cases
- ☑ Test invalid inputs
- ☑ Document validation rules
- ☑ Create validation examples

Phase 8: Integration & Testing (Week 4-5)

8.1 Integration Testing

- ☑ Create end-to-end test scenarios
- ☑ Test complete trading workflow
  - ☑ Connect → Get account info → Execute trade → Monitor → Close

- ☑ Test error recovery scenarios
- ☑ Test auto-reconnection
- ☑ Test multi-account switching
- ☑ Test concurrent operations
- ☑ Test data streaming
- ☑ Test caching mechanisms

8.2 Performance Testing

- ☑ Benchmark connection speed
- ☑ Benchmark data retrieval
- ☑ Benchmark order execution
- ☑ Test with high-frequency operations
- ☑ Profile memory usage
- ☑ Optimize slow operations

8.3 Error Scenario Testing

- ☑ Test network disconnection
- ☑ Test invalid credentials
- ☑ Test insufficient margin
- ☑ Test invalid symbols
- ☑ Test market closed scenarios
- ☑ Test API errors
- ☑ Test edge cases

8.4 Code Quality

- ☑ Run code coverage analysis (aim for 80%+)
- ☑ Run linting (flake8, pylint)
- ☑ Run type checking (mypy)
- ☑ Format code (black)
- ☑ Review and refactor
- ☑ Code review (if team)

Phase 9: Documentation (Week 5) ✅ COMPLETED

9.1 Code Documentation

- ☑ Add docstrings to all classes
- ☑ Add docstrings to all public methods
- ☑ Add inline comments for complex logic
- ☑ Generate API documentation (Sphinx)

9.2 User Documentation

- ☑ Write comprehensive README.md
- ☑ Create installation guide
- ☑ Create quick start guide
- ☑ Create configuration guide
- ☑ Create troubleshooting guide

9.3 Examples & Tutorials

- ☑ Basic connection example
- ☑ Account information example
- ☑ Market data retrieval example
- ☑ Simple trading strategy example
- ☑ Risk management example
- ☑ Multi-symbol trading example
- ☑ Backtesting example
- ☑ Error handling example

9.4 API Reference

- ☑ Generate API documentation
- ☑ Document all classes
- ☑ Document all methods

- ☑ Document parameters and return types
- ☑ Add usage examples

Phase 10: Packaging & Deployment (Week 5-6) ✅ COMPLETED

10.1 Package Setup

- ☑ Finalize setup.py
- ☑ Create MANIFEST.in
- ☑ Create pyproject.toml
- ☑ Set version number (1.0.0)
- ☑ Create LICENSE file

10.2 Build & Test Package

- ☑ Build package

  ```
  python -m build
  ```

- ☑ Install in clean environment
- ☑ Test installation
- ☑ Test imports

10.3 Configuration Templates

- ☑ Create config templates
- ☑ Create example scripts
- ☑ Create starter project template

10.4 Distribution (Optional)

- ☐ Create PyPI account

- ☐ Upload to Test PyPI
- ☐ Test installation from Test PyPI
- ☐ Upload to PyPI
- ☐ Verify installation from PyPI

Phase 11: Production Readiness (Week 6)

11.1 Security Review

- ☑ Review credential storage
- ☑ Implement secure config loading guidance (.env / secrets)
- ☑ Review logging (no sensitive data; redaction guidance)
- ☑ Add input sanitization guidance

11.2 Monitoring & Logging

- ☑ Set up structured logging template (logging.conf.example)
- ☑ Add performance metrics guidance (ping/latency, heartbeat)
- ☑ Add error tracking guidance
- ☑ Create log rotation template
- ☐ Create monitoring dashboard (optional)

11.3 Deployment Checklist

- ☑ Create deployment guide (docs/DEPLOYMENT.md)
- ☑ Create production configuration template (config.ini.example)
- ☑ Document system requirements
- ☑ Document firewall requirements (guidance in SECURITY.md)
- ☑ Create backup strategy guidance
- ☑ Create disaster recovery plan outline

11.4 Maintenance Plan

- ☑ Set up CI/CD guidance (optional)
- ☑ Create issue templates (covered in docs guidance)
- ☑ Create contribution guidelines (already present)
- ☑ Plan update schedule (MAINTENANCE_PLAN.md)
- ☑ Create changelog (CHANGELOG.md)

Continuous Tasks (Throughout Project)

Daily/Weekly Tasks

- ☐ Commit code regularly
- ☐ Write tests as you code
- ☐ Update documentation
- ☐ Run test suite
- ☐ Review code quality
- ☐ Check for bugs

Code Quality Checks

```
# Run tests
pytest tests/ -v --cov=mymt5

# Run linting
flake8 mymt5/ tests/

# Run type checking
mypy mymt5/

# Format code
black mymt5/ tests/
```

Milestones & Review Points

Milestone 1: Foundation Complete (End of Week 1)

- ☑ Project structure created
- ☑ Enums implemented
- ☑ Utils implemented
- ☑ Basic tests passing

Milestone 2: Core Complete (End of Week 2)

- ☑ MT5Client fully implemented
- ☑ Information layer complete
- ☑ Connection works reliably
- ☐ Multi-account tested

Milestone 3: Data Layer Complete (End of Week 3)

- ☑ MT5Data implemented
- ☑ MT5History implemented
- ☑ Data retrieval tested
- ☑ Caching working

Milestone 4: Trading Complete (End of Week 4)

- ☑ MT5Trade implemented
- ☑ MT5Risk implemented
- ☑ MT5Validator implemented
- ☑ Can execute trades successfully

Milestone 5: Testing Complete (End of Week 5)

- ☑ All unit tests passing
- ☑ Integration tests passing
- ☑ Code coverage > 80%
- ☑ Documentation complete

Milestone 6: Production Ready (End of Week 6)

- ☑ Package built
- ☑ Examples working
- ☑ Security reviewed
- ☑ Ready for deployment

Risk Management & Contingency

Potential Risks

1. **MT5 API Changes**
   - Mitigation: Version pinning, regular updates
2. **Connection Issues**
   - Mitigation: Robust reconnection logic, timeouts
3. **Performance Issues**
   - Mitigation: Profiling, caching, optimization
4. **Testing Challenges**
   - Mitigation: Mocking, test accounts, CI/CD
5. **Time Overruns**
   - Mitigation: Prioritize core features, iterative development

Fallback Plans

- ☐ Identify critical vs nice-to-have features
- ☐ Plan phased rollout if needed
- ☐ Keep architecture flexible for future additions

Success Criteria

Functional Requirements

✅ All 10 classes implemented ✅ ~120 methods working ✅ Can connect to MT5 ✅ Can execute trades ✅ Can retrieve data ✅ Error handling works ✅ Auto-reconnection works

Quality Requirements

✅ Test coverage > 80% ✅ No critical bugs ✅ Code follows style guide ✅ Documentation complete ✅ Examples working

Performance Requirements

✅ Connection < 2 seconds ✅ Order execution < 1 second ✅ Data retrieval efficient ✅ Memory usage reasonable

---

Post-Launch Tasks

Week 7+

- ☐ Monitor system in production
- ☐ Collect user feedback
- ☐ Fix reported bugs
- ☐ Plan improvements
- ☐ Consider additional features
- ☐ Update documentation

- ☐ Release updates

Feature Backlog (Future Enhancements)

- ☐ Advanced strategy backtesting
- ☐ Machine learning integration
- ☐ Web dashboard
- ☐ Mobile notifications
- ☐ Multi-broker support
- ☐ Cloud deployment options
- ☐ Advanced analytics
- ☐ Trade journal features

Resources & References

Documentation

- MT5 Python Documentation: https://www.mql5.com/en/docs/python_metatrader5
- Python Best Practices: PEP 8, PEP 257
- Testing: pytest documentation

Tools

- Version Control: Git
- Testing: pytest, pytest-cov
- Linting: flake8, pylint, black
- Type Checking: mypy
- Documentation: Sphinx

Learning Resources

- ☐ MT5 Python API examples
- ☐ Trading system design patterns
- ☐ Risk management principles

- ☐ Software testing best practices

Team Roles (If Applicable)

Solo Developer

✅ All tasks Focus: Core functionality first, optimization later

2-Person Team

- Developer 1: Core + Information + Data layers
- Developer 2: Trading + Risk + Utility layers

3-Person Team

- Developer 1: Core + Information layers + Integration
- Developer 2: Data + History + Testing
- Developer 3: Trading + Risk + Validator + Documentation

Final Checklist Before Launch

Code

- ☐ All features implemented
- ☐ All tests passing
- ☐ No known critical bugs
- ☐ Code reviewed
- ☐ Dependencies documented
- ☐ Version tagged

Documentation

- ☐ README complete
- ☐ API docs generated
- ☐ Examples working
- ☐ Installation tested
- ☐ Troubleshooting guide ready

Deployment

- ☐ Package built
- ☐ Configuration examples ready
- ☐ Security reviewed
- ☐ Backup plan ready
- ☐ Monitoring in place

Support

- ☐ Issue tracking ready
- ☐ Communication channel set
- ☐ Update plan ready
- ☐ Maintenance schedule set

Notes

- Adjust timeline based on team size and experience
- Prioritize core features over nice-to-haves
- Test frequently, commit often
- Document as you go
- Keep architecture flexible
- Listen to early user feedback

Project Start Date: **_** **Target Completion Date**: **_** **Project Manager**: **_** **Lead Developer**: **_**

Last Updated: [Date] Version: 1.0