Share Profit Tracker Software - Project Report

1. Executive Summary

Project Name: Share Profit Tracker
Version: 1.0
Target Platform: Windows Desktop Application (.exe)
Development Language: Python with Tkinter GUI
Project Duration: 2-3 weeks
Budget Estimate: Low-cost (primarily time investment)

Project Objective

Develop a lightweight desktop application that allows users to track their stock purchases and automatically calculate profits/losses based on real-time market prices.

2. Project Requirements

2.1 Functional Requirements

- **Portfolio Input**: Allow users to input share details (symbol, quantity, purchase price, purchase date)
- **Real-time Price Fetching**: Automatically retrieve current market prices
- **Profit/Loss Calculation**: Display current profit/loss for each stock and total portfolio
- **Data Persistence**: Save portfolio data locally
- **User-friendly Interface**: Simple, intuitive GUI
- **Export Functionality**: Export portfolio summary to CSV/Excel

2.2 Technical Requirements

- **Minimum PC Requirements**:
  - Windows 7/8/10/11
  - 2GB RAM
  - 100MB free disk space
  - Internet connection for price updates
- **Single executable file** (no installation required)
- **Offline data storage** (SQLite database)
- **Lightweight** (< 50MB file size)

2.3 Non-Functional Requirements

- Fast startup time (< 5 seconds)

- Responsive UI (< 2 seconds for calculations)
- Data accuracy (99.9% uptime for price feeds)
- Error handling for network issues

3. System Architecture

3.1 Technology Stack

- **Programming Language**: Python 3.9+
- **GUI Framework**: Tkinter (built-in, no external dependencies)
- **Database**: SQLite (lightweight, serverless)
- **API**: Yahoo Finance API (free tier)
- **Packaging**: PyInstaller (creates single .exe file)

3.2 Application Structure

```
ShareProfitTracker/
├── main.py              # Entry point
├── gui/
│   ├── main_window.py     # Main interface
│   ├── add_stock_dialog.py # Add/Edit stock dialog
│   └── settings_window.py  # Configuration settings
├── data/
│   ├── database.py       # SQLite operations
│   └── models.py         # Data models
├── services/
│   ├── price_fetcher.py   # Market data API
│   └── calculator.py      # Profit/loss calculations
└── utils/
    ├── config.py         # Application settings
    └── helpers.py        # Utility functions
```

4. Feature Specifications

4.1 Core Features

Portfolio Management

- Add new stock entries with:
  - Stock symbol (e.g., AAPL, GOOGL)
  - Company name
  - Quantity purchased
  - Purchase price per share

    - Purchase date
    - Broker/platform (optional)
- Edit existing entries
- Delete stock entries
- Bulk import from CSV file

Real-time Price Updates

- Automatic price fetching every 15 minutes during market hours
- Manual refresh option
- Display last updated timestamp
- Handle market closures and holidays

Profit/Loss Analytics

- **Individual Stock Metrics**:
    - Current value
    - Total profit/loss amount
    - Profit/loss percentage
    - Days held
    - Annualized return
- **Portfolio Summary**:
    - Total investment
    - Current portfolio value
    - Overall profit/loss
    - Best/worst performing stocks

Data Management

- Local SQLite database
- Automatic backup creation
- Data export to CSV/Excel
- Import portfolio from other formats

4.2 Advanced Features (Phase 2)

- Price alerts and notifications
- Historical performance charts
- Dividend tracking

- Tax calculation assistance
- Multiple portfolio support

5. User Interface Design

5.1 Main Window Layout

- **Header**: Application title, last update time, refresh button
- **Portfolio Table**: Scrollable table showing all stocks with columns:
  - Symbol | Company | Quantity | Purchase Price | Current Price | P&L Amount | P&L %
- **Summary Panel**: Total investment, current value, overall P&L
- **Action Buttons**: Add Stock, Edit Stock, Delete Stock, Settings, Export

5.2 Add/Edit Stock Dialog

- Form with input fields for stock details
- Symbol validation and company name auto-fill
- Date picker for purchase date
- Save/Cancel buttons

6. Implementation Plan

Phase 1 (Week 1): Core Development

- Set up project structure
- Implement basic GUI with Tkinter
- Create SQLite database schema
- Develop stock data input functionality
- Basic profit/loss calculations

Phase 2 (Week 2): API Integration

- Integrate Yahoo Finance API
- Implement real-time price fetching
- Add error handling for network issues
- Implement data refresh mechanisms

Phase 3 (Week 3): Polish & Packaging

- UI improvements and validation
- Export functionality
- Testing and bug fixes

- Package into executable with PyInstaller
- Create user documentation

7. Technical Implementation Details

7.1 Database Schema

```sql
CREATE TABLE stocks (
    id INTEGER PRIMARY KEY,
    symbol TEXT NOT NULL,
    company_name TEXT,
    quantity REAL NOT NULL,
    purchase_price REAL NOT NULL,
    purchase_date TEXT NOT NULL,
    broker TEXT,
    created_at TEXT DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE price_cache (
    symbol TEXT PRIMARY KEY,
    current_price REAL,
    last_updated TEXT
);
```

7.2 API Integration

- Use `yfinance` library for market data
- Implement caching to reduce API calls
- Handle rate limiting and errors gracefully
- Fallback to alternative data sources if needed

7.3 Executable Creation

```bash
# PyInstaller command for creating single executable
pyinstaller --onefile --windowed --icon=app.ico main.py
```

8. Risk Analysis

8.1 Technical Risks

- **API Limitations**: Free tier restrictions on Yahoo Finance

- **Network Dependency**: Requires internet for price updates
- **Data Accuracy**: Potential delays or errors in market data

8.2 Mitigation Strategies

- Implement multiple data source fallbacks
- Cache recent prices for offline viewing
- Add user notifications for data issues
- Regular testing with different market conditions

9. Testing Strategy

9.1 Unit Testing

- Database operations
- Profit/loss calculations
- API response handling

9.2 Integration Testing

- GUI interactions
- Database-API integration
- File import/export functionality

9.3 User Acceptance Testing

- Test with different portfolio sizes
- Validate calculations manually
- Performance testing with large datasets

10. Deployment & Distribution

10.1 Packaging Requirements

- Single .exe file (no installation required)
- Include necessary DLLs and dependencies
- File size optimization (target < 50MB)
- Digital signature for trust

10.2 Distribution Channels

- Direct download from website
- GitHub releases

- Microsoft Store (future consideration)

11. Maintenance & Support

11.1 Regular Updates

- API compatibility maintenance
- Bug fixes and performance improvements
- New feature additions based on user feedback

11.2 User Support

- User manual and FAQ
- Email support for technical issues
- Feature request tracking

12. Budget Estimation

12.1 Development Costs

- Development time: 80-120 hours @ $0 (self-development)
- Tools and services: $0 (using free/open-source tools)
- Testing devices: $0 (using existing hardware)

12.2 Operational Costs

- Domain/hosting (optional): $50-100/year
- Code signing certificate: $200-400/year
- Cloud backup service: $60/year

Total First Year Cost: $310-560

13. Success Metrics

- Application startup time < 5 seconds
- Price update accuracy > 99%
- User satisfaction score > 4.0/5.0
- Zero critical bugs in production
- File size under 50MB

14. Future Enhancements

Version 2.0 Features

- Web-based version
- Mobile app companion
- Advanced charting and analytics
- Integration with popular brokers
- Social features (share portfolios)
- Cryptocurrency support

15. Conclusion

The Share Profit Tracker software addresses a clear need for individual investors to monitor their portfolio performance easily. With its lightweight design and minimal system requirements, it will be accessible to a wide range of users. The phased development approach ensures a robust, feature-complete application that can grow with user needs.

The use of Python with Tkinter ensures broad compatibility while keeping the application lightweight. The integration with free market data APIs makes it cost-effective to operate while providing accurate, real-time information.

Next Steps:

1. Finalize technical specifications
2. Set up development environment
3. Begin Phase 1 implementation
4. Establish testing procedures
5. Plan user feedback collection