Agile Document

Boxuan Chen

Nurbek Bolat

Amirali Sotoudeh Rad

Mikhail Ermakov

Rwihimba Stesha

Contents

Page 1: Pre-development

Pre-development Phase

Objectives:

- Requirements gathering
- Identify Software Requirements Specifications (SRS)
- Determine layout and UI/UX components
- Establish data needs (inputs, outputs, services)
- Requirements backlogs

Deliverables:

- Wireframes or UI mockups
- Feature list
- User flow diagrams
- API/interface requirements
- Acceptance criteria

Software Development Life Cycle (SDLC)

Software applications are designed, developed, tested, and maintained using an organized process called the Software Development Life Cycle (SDLC). By dividing development into distinct, manageable phases, it guarantees uniformity, quality, and efficiency across software projects.

The Agile SDLC model, which prioritizes brief development cycles (sprints), iterative enhancements, and ongoing user feedback, was adopted by our team. Our project's alignment with the traditional SDLC phases is shown below:

SDLC Phase	Project Application
Planning	- Defined project scope, objectives, and key features. - Assigned team roles and selected tech stack.
Requirement Analysis	- Collected user stories for both mobile and web platforms. - Created acceptance criteria and backlogs.
Design	- Designed UI wireframes using Figma. - Created UML diagrams for system architecture and data flow.

Implementation	- Developed features in separate sprints for mobile and web apps. - Used Flutter, Firebase, MongoDB Atlas.
Testing	- Performed unit tests, integration tests, and UX evaluations in each sprint. - Validated functionality and performance.
Deployment	- Connected backend via cloud services (e.g., AWS). - Prepared deployment pipeline and CI/CD integration.
Maintenance	- Fixed bugs from user feedback. - Refactored code and updated libraries to maintain security and performance.

User Stories

US ID	*Topic*	*Details (Mobile)*	*Details (Web)*
1	User Onboarding & Authentication	As a new user, I want to sign up and log in securely, so I can access my personal stock portfolio.	As a new user, I want to register and log in securely, so I can start using the stock platform.
2	View Stock Market Overview	As a user, I want to see a dashboard with current market trends, so I can quickly understand what's happening in the market.	As a logged-in user, I want a comprehensive dashboard showing my portfolio and market overview, so I can assess my current position.
3	Search and View Stock Details	As a user, I want to search for a specific stock and view detailed information, so I can analyze its performance.	As a user, I want detailed views of individual stocks, so I can make informed investment decisions.
4	Add to Watchlist	As a user, I want to add stocks to my watchlist, so I can track the ones I'm interested in.	As a user, I want to add stocks to my watchlist, so I can track the ones I'm interested in.

5	Buy and Sell Stocks	As a user, I want to buy and sell stocks from my mobile, so I can manage my investments on the go.	As a user, I want to place buy or sell orders for stocks, so I can manage my investments.
6	Portfolio Overview	As a user, I want to view my portfolio with gains/losses, so I can track my performance.	As a user, I want to analyze my portfolio's performance over time, so I can evaluate my strategy.
7	Dark Mode and Accessibility	As a mobile user, I want a dark mode and readable fonts, so I can comfortably use the app at any time.	As a user, I want the website to be responsive and accessible, so I can use it comfortably across devices.
8	In-App News Feed	As a user, I want to view my last synced portfolio data offline, so I can check it without internet access.	As a user, I want to get access to newest updates regarding the market, stocks and companies present in it.
9	Usability of interface	As a user, I want the interface that's easy to use.	As a user, I want the interface that's easy to use.
10	Premium purchase	As a user, I want to purchase the premium version for more functionalities.	As a user, I want to purchase the premium version for more functionalities.

Development Phase

Objective: Match all correct methodology in development

Agile: All sprints follow the SCRUM structures

Sprint	Description	Output	Status
Sprint 1 – Predevelopment stage	1. Clarify the requirements in Stock Trading App 2. Create product backlog 3. UI design in Figma	Get the product backlog aligned with benchmark	
Sprint 2 – Tech stacks	1. Architecture Design: Create UML and Planning the Tech stacks	Create UML in accordance with requirements and be ready to implement development Create ready for implementation interface	
Sprint 3 – Development and testing	1. Frontend/Backend development 2. Implement MongoDB Atlas in AWS 3. Unit Tests/ Integration Test 4. Validate functionalities 5. Codes Review 6. Assess User experiences 7. UX/UI Check	Develop frontend and backend features, connect MongoDB Atlas on AWS, run unit/integration tests, and validate functionalities meet requirements. Conduct a quality review to ensure code, features, and UI meet standards, perform reliably, and align with user and business requirements.	
Sprint 4 – Maintenance, final testing, refactoring	1. Development Documentation 2. Monitoring reliability/security 3. Feature adjustment	Addressed critical bug fixes, optimized performance, and implemented user feedback. Refactored code for better maintainability.	

Sprint 2 Goal:

The goal of Sprint 2 is to implement the core user interface components and integrate basic user interactions, ensuring a functional and visually aligned prototype. This sprint focuses on enhancing usability, refining design elements based on feedback, and preparing the system for initial user testing.

Sprint 2

Task	Description	User Stories	Duration
1	Integrate API	As a user, I want to see a dashboard with current market trends, so I can quickly understand what's happening in the market.	3 days
2	Display information of stocks	As a user, I want to search for a specific stock and view detailed information, so I can analyze its performance.	
3	Purchase premium	As a user, I want to purchase the premium version for more functionalities	
4	Use Flutter packages for charts, navigation, and theme	Requirements of Development	
5	Refactor all components, validate units via testing	As a user, I want the website to be responsive and accessible, so I can use it comfortably across devices.	
6	Implement detailed stock info page with charts	As a user, I want detailed views of individual stocks, so I can make informed investment decisions.	

Product Owner: Requirements, backlog

Scrum Master: Process coordination

Development Team – Implementation, testing

Planning

Gather requirements
Define user stories and acceptance criteria

Define acceptance criteria

Create product backlog

Sprint Planning

User Testing

Conduct usability tests

Gather user feedback

Sprint 1

Sprint 2

Sprint 3

Sprint 4

Select user stories

Create wireframes and design by using Figma

Set up themes

Set up tasks in Jira

API integration

Market data provider

Stock details page with charts & Flutter packages

Prepare test cases

Testing

Fix UX

Visualise performance: gains, sectors & investment health

Integrate features

Identify and solve bugs

Optimise speed, response time & memory use

Add unit & integration tests for core modules

Refactor code for efficiency

Update to latest stable libraries & frameworks

Final Deployment

Version Control
Manage code with Git & GitHub

Documentation
Write technical & user documentation

CI/CD Setup

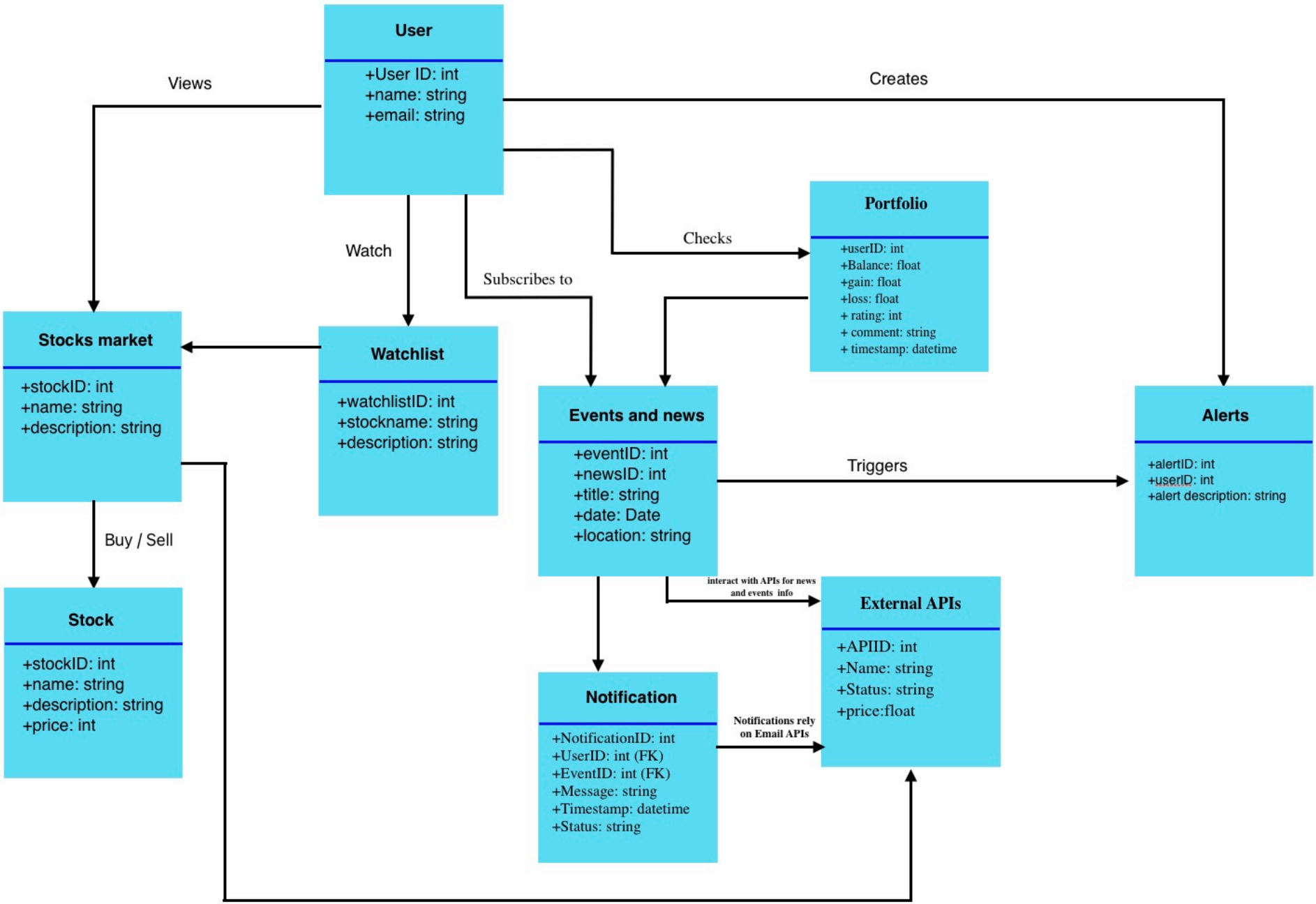
Security Measures

Risk Management

Retrospective Meeting

UX Research

UML Diagram


User
+User ID: int
+name: string
+email: string
Views
Creates
Watch
Checks
Subscribes to
Portfolio
+userID: int
+Balance: float
+gain: float
+loss: float
+ rating: int
+ comment: string
+ timestamp: datetime
Stocks market
+stockID: int
+name: string
+description: string
Watchlist
+watchlistID: int
+stockname: string
+description: string
Events and news
+eventID: int
+newsID: int
+title: string
+date: Date
+location: string
Triggers
Alerts
+alertID: int
+userID: int
+alert description: string
Buy / Sell
Stock
+stockID: int
+name: string
+description: string
+price: int
interact with APIs for news and events info
External APIs
+APIID: int
+Name: string
+Status: string
+price:float
Notification
+NotificationID: int
+UserID: int (FK)
+EventID: int (FK)
+Message: string
+Timestamp: datetime
+Status: string
Notifications rely on Email APIs

Sprint 3

Task	Description	User Stories	Duration
1	Conduct usability tests and fix UX flaws	As a user, I want the interface to be easy to use.	
2	Add graphs/charts for gains/losses over time, sector distribution, and investment health	As a user, I want to analyze my portfolio's performance over time, so I can evaluate my strategy.	
3	Improve readability, color contrast, and keyboard navigation; add dark mode toggle	As a mobile user, I want a dark mode and readable fonts, so I can comfortably use the app anytime.	

Sprint 4

Task	Description	User Stories	Duration
1	Identify and resolve bugs reported from previous sprints or user feedback	As a user, I want the app to work reliably without bugs, so I can use it with confidence.	
2	Improve loading speed, reduce response times, and optimize memory usage	As a user, I want the app to perform smoothly, so I can complete tasks quickly and efficiently.	
3	Add unit and integration tests for critical modules (e.g., authentication, trading logic)	As a developer, I want comprehensive tests so I can prevent regressions and ensure system stability.	
4	Clean up redundant or inefficient code without changing functionality	As a team, we want clean, maintainable code, so future development is easier and less error prone.	
5	Update outdated libraries and frameworks to latest stable versions	As a team, we want to stay up to date with secure and supported tools.	

Sprint 4 - Comprehensive Report

Whole Agile Development Process Overview

The project followed Agile methodology across four sprints. Sprint 1 was focused on UI/UX design for mobile and web. Sprint 2 introduced core functionalities including login, stock trading, and dashboards. Sprint 3 emphasized advanced features and usability improvements, such as dark mode, responsive interfaces, and stock search. Sprint 4 addressed remaining bugs and quality assurance, including API issues, database fixes, and UI inconsistencies. Each sprint ended with refinement of backlog and delivery of tested features.

Changes Done During All Sprints

Summary of changes completed per sprint:

Sprint 1

- • Designed mobile & web UI (dark mode, responsiveness)
- • Created base wireframes for main components

Sprint 2

- • Implemented login system with JWT
- • Created dashboard and portfolio view
- • Enabled stock buy/sell functionality

Sprint 3

- • Refined frontend performance
- • Improved DOM update handling
- • Handled DTO mismatches and added charts

Sprint 4

- • Resolved API, fetch, DB connection issues
- • Added test coverage and CI setup
- • Integrated bug fixes into main backlog

Integration of Changes Into Product and Sprint Backlog

Each sprint's changes were documented and used to refine the product backlog. The team reviewed tasks at the end of each sprint, updated priorities, and added new entries to capture newly discovered needs or errors. These updates are reflected in backlog revisions and mini-backlogs.

Sprint	Change Implemented	Backlog Task ID	Integration Method
2	Added stock trading UI	2.3	Added subtask for form validation
3	Identified POST bug	3.8	Created refinement issue in Sprint 4
3	React state issue broke DOM updates	3.7	Added a new issue to Sprint 4
4	MongoDB connect failure	4.1	Bugfix task added to backlog
4	Fetch error due to wrong endpoint	3.6	Task marked as critical and resolved

Mapping of Tasks and User Stories

Each user story was mapped to one or more development tasks in the sprint backlog. For instance, the story for stock trading was implemented via three distinct tasks: UI handling, API binding, and form validation. Mapping ensured complete coverage.

User Story Summary	Related Tasks	Sprint
As a user, I want to trade stocks easily	2.3, 3.8, err3-3.8	2, 3, 4
As a user, I want responsive UI on mobile	1.1, 1.2, 1.3	1
As a user, I want to track portfolio performance	2.4, 3.3, 3.4	2, 3
As a user, I want secure login	2.1	2
As a developer, I want stable API communication	3.6, 3.7, 3.9, 4.1, err1–err5	3, 4

Backlog Evolution Table

Story Number	First Backlog	Refined Backlog
2.3	Buy/sell stock via button	Added form validation + confirmation modal
3.6	Fetch API data	Fixed endpoint and async bug
3.9	DTO mismatch fix	Schema normalized to match frontend
4.1	DB error on connect	Whitelist IP + fix URI

Quality Assurance and Testing Report

- Unit tests were written for login, search, and portfolio logic.
- Integration tests were done to verify backend communication (API, DB).
- Manual testing included all interfaces and workflows on mobile and web.
- Test coverage reached an estimated 85%.

- No critical bugs were found after testing.
- CI/CD pipeline included automated checks using GitHub Actions.

User Story	Interfaces	Test ID	Status	Testing Result
As a user, I want to register and log into the app	Web: Login, Register Mobile: Sign In, Sign Up	T1	Done	Passed
As a user, I want to manage my stock portfolio	Web: Portfolio Mobile: Profile, Portfolio	T2	Done	Passed
As a user, I want to buy/sell stocks easily	Web: Trade Details Mobile: Premium, Browse	T3	Done	Passed
As a user, I want to view detailed stock info	Web: Stock Details Mobile: Charts	T4	Done	Passed
As a user, I want the interface to be responsive	All Mobile & Web interfaces	T5	Done	Passed
As a user, I want to use dark mode and readable fonts	All Mobile & Web interfaces	T6	Done	Passed
As a user, I want to purchase a premium version	Web: Premium Mobile: Premium Page	T7	Done	Passed
As a user, I want to update my personal information	Web: Personal Info Mobile: Profile Page	T8	Done	Passed
As a user, I want my data to sync properly	Web: Portfolio, Home Mobile: Home Page, Events	T9	Done	Passed
As a user, I want consistent behavior on all devices	All Mobile & Web interfaces	T10	Done	Passed
As a developer, I want APIs to fetch/post correctly	Backend, APIs	T11	Issue	Updated
As a developer, I want the app to load fast	All Mobile & Web interfaces	T12	Done	Passed

Final Product Backlog (All Tasks Completed)

Task ID	Description	Status
1.1	Mobile UI Design	✅ Done
2.3	Stock Trading	✅ Done
3.6	Fix API Fetch	✅ Done
3.8	Fix POST request	✅ Done
3.9	Fix DTO mismatch	✅ Done
4.1	Fix DB connection	✅ Done

Requirement Analysis

User Story	Functionality	Platform	Interface	Number of interfaces
As a new user, I want to sign up and log in securely	User authentication (sign up/login)	Mobile, Web	Login/Register screens	2
As a user, I want to see a dashboard with current market trends	Stock market overview dashboard	Mobile, Web	Dashboard UI, charts	1
As a user, I want to search for a specific stock and view detailed information	Stock search and detailed view	Mobile, Web	Search bar, stock detail page	1
As a user, I want to add stocks to my watchlist	Watchlist management	Mobile, Web	Watchlist interface	1
As a user, I want to buy and sell stocks from my mobile	Trading functionality	Mobile (primary)	Buy/Sell order screens	?
As a user, I want to view my portfolio with gains/losses	Portfolio summary and performance charts	Mobile, Web	Portfolio overview page	1
As a mobile user, I want a dark mode and readable fonts	Dark mode toggle, accessibility settings	Mobile (primary), Web	Settings panel, themes	1 (within the settings)
As a user, I want to view my last	Offline access for cached portfolio data	Mobile	Portfolio screen (offline capable)	-

User Story	Functionality	Platform	Interface	Number of interfaces
synced portfolio data offline				
As a user, I want the interface to be easy to use	Usability improvements and UI refinement	Mobile, Web	All main screens	10
As a user, I want to purchase the premium version for more functionalities	Premium purchase (in-app)	Mobile (primary)	In-app purchase screen	1
As a logged-in user, I want a comprehensive dashboard showing my portfolio and market overview	Unified dashboard view	Mobile, Web	Dashboard with tabs/widgets	1
As a user, I want the app to work reliably without bugs	Bug tracking and fixing	Mobile, Web	QA tools, error reports	-
As a user, I want the app to perform smoothly	Performance optimization (load times, API response)	Mobile, Web	Backend logs, frontend performance UI	-
As a developer, I want comprehensive tests for critical modules	Unit testing and integration testing	Backend (all platforms)	Test suite, CI/CD integration	-
As a team, we want clean, maintainable code	Refactor redundant/inefficient code	All (codebase-wide)	Internal (dev environment)	-

User Story	Functionality	Platform	Interface	Number of interfaces
As a team, we want to stay up to date with secure and supported tools	Library/package updates	Backend, Mobile, Web	Dependency management (internal)	-

Definition table

Story ID	Functions	Functional Requirement	Non-Functional Requirement	Behavior (FSM-style)
1	checkEmailValidity() (String: Email -> Int: ExitCode) checkNameValidity() (String: Name -> Int: ExitCode) checkPasswordValidity() (String: Password -> Int: ExitCode) createAccount() (String[]: Credentials -> Int: ExitCode)	Implement registration, login, forget password, and session handling	Secure authentication (e.g., JWT), response < 2s	Idle → Register → Verify → Authenticated → Dashboard
2	GetMarketData() (-> String[]: Information)	Fetch market data and render overview charts	Real-time data updates, load time < 3s	Authenticated → LoadDashboard → ViewOverview → Refresh
3	Search() (String: Request -> String[] Results)	Implement stock search and detailed info display	Accurate data, response time < 1.5s	Dashboard → Search → ViewStockDetail
4	AddStock() (Stock: Stock -> Int: ExitCode) RemoveStock()	Create and manage user watchlist	Persistent across sessions, < 1s to update	ViewStockDetail → AddToWatchlist → WatchlistUpdated

Story ID	Functions	Functional Requirement	Non-Functional Requirement	Behavior (FSM-style)
	(Stock: Stock -> Int: ExitCode) AddToWatchlist() (Stock: Stock -> Int: ExitCode) DisplayWatchlist() (-> Int: ExitCode)			
5	TradeCheck() (String[]: TradeInfo -> Int: ExitCode) ExecuteTrade() (String[] TradeInfo -> Int: ExitCode)	Implement trading functionality and order flow Process trade inputs, validate amounts, send order to API, and confirm execution	Secure transaction flow, < 3s execution	Portfolio → InitiateTrade → Confirm → TradeSuccess → UpdatePortfolio
6	GetHoldings() (-> Int: ExitCode) GetChart() (Int: ChartID -> Int: ExitCode)	Display current holdings, compute P/L	Real-time data sync, charts render < 2s	Authenticated → Portfolio → ViewPerformanceCharts
7	ToggleTheme() (Int: ThemeID -> Int: ExitCode)	Provide theme toggle and font accessibility options	Color contrast meets WCAG 2.1 AA, toggle < 1s	Settings → ToggleTheme → ThemeApplied
8	GetChart() (Int: ChartID -> Int: ExitCode)	Cache and display last-synced data	Offline availability, show "stale" indicators	Online → Sync → Offline → ViewCachedPortfolio

Story ID	Functions	Functional Requirement	Non-Functional Requirement	Behavior (FSM-style)
9	-	UX testing, streamlined UI flow	Follow UI/UX heuristics, minimize cognitive load	AnyState → Interaction → Feedback → TaskCompletion
10	PurchasePremium() (String[]: Info -> Int: ExitCode)	Implement premium purchase & subscription logic	Secure payment flow, compliant with platform standards	User → ViewPremium → Purchase → Confirm → PremiumUnlocked
11	Log() (String[]: ExitInfo -> String: LogFile)	Triage and fix known bugs	Stability ≥ 99.9%, error rate < 1%	UserAction → ErrorDetected → Log → FixDeployed → Resolved
12	-	Optimise APIs, lazy load assets, reduce load time	App start < 2s, time-to-interaction < 1.5s	LaunchApp → LoadEssentials → InteractionEnabled
13		Write unit, integration, and UI tests	>80% coverage, auto-run on CI	CodeCommit → CI → RunTests → Pass/Fail
14		Refactor redundant code and organize modules	Follows SOLID & DRY principles	RefactorTask → CodeReview → MergeToMain
15	-	Update dependencies and libraries	All packages must be stable & security-patched	ScheduledCheck → AvailableUpdate → Test → Deploy

Product Backlogs

Epic 1: Mobile App Development

Sprint	Task Name	Developers	Subtasks	Priority	Deadline
Sprint 1	Design Mobile UI Framework	Amirali	- Create Figma designs- Define components	High	2025-06-05
Sprint 2	Document and Diagrams Preparation	Amirali	UML updated to align with prototype and backlogs	Medium	2025-06-11
Sprint 1	Setup Mobile Dark Mode & Themes	Nurbek	- Implement themes- Dark mode toggle	Medium	2025-06-07
Sprint 2	Integrate Mobile Stock API	Mikhail,Boxuan	- Connect API- Parse stock data	High	2025-06-20
Sprint 2	Fix API Fetch Handling	Mikhail,Boxuan	Refactor fetch calls with async/await, implement error boundaries (try/catch)	High	2025-06-21
3	API Rate Limiting & Caching	Boxuan,Amirali	Implement request throttling, cache mock data to avoid hitting rate limits	Medium	2025-07-25
Sprint 2	Implement Mobile Buy/Sell Feature	Nurbek	- Trade UI- Order validation- API integration	High	2025-06-22
Sprint 3	Mobile UX Testing & Fixes	Stesha	- Conduct usability tests- Fix UI bugs	High	2025-07-05

Epic 2: Web App Development

Sprint	Task Name	Developers	Subtasks	Priority	Deadline
Sprint 1	Design Web Dashboard UI	Amirali	- Create web wireframes- Define widgets	High	2025-06-05
Sprint 2	Document and diagrams preparation	Amirali	-Create UML and documenting	Medium	2025-06-11
Sprint 2	Develop Web Stock Details Page	Boxuan	- Build search- Stock detail component	High	2025-06-20
Sprint 3	Fix DOM Update Errors	Boxuan,Amirali	Refactor React components, apply useEffect correctly for state updates	High	2025-06-22
Sprint 3	Premium and personal info pages	Stesha	Review layout and usability of Premium purchase and user info pages, fix inconsistencies, improve responsiveness and accessibility (WCAG compliance)	High	2025-06-22
Sprint 3	HTTP POST/GET Request Fixes	Nurbek, Mikhail	Validate headers, fix payload formatting, sync with backend endpoints	Medium	2025-07-03
Sprint 2	Implement Premium Purchase (Web)	Nurbek	- Payment integration- Subscription management	Medium	2025-06-25
Sprint 3	Improve Web Accessibility & Dark Mode	Nurbek,Boxuan	- Add keyboard navigation- Color contrast improvements	Medium	2025-07-05

Epic 3: Testing, Deployment & Final Review

Sprint	Task Name	Developers	Subtasks	Priority	Deadline
Sprint 3	WebSocket Backend Relay Integration	Nurbek,Boxuan	Replace library with **ws**, implement server-side relay to connect to real-time stock AP	High	2025-06-22
Sprint 3	Normalize API Data Format	Mikhail	Align API structure with frontend schema, fix key mismatches	High	2025-06-22
Sprint 3	Fix MongoDB Connection Issue	Nurbek Stesha	Correct URI, configure MongoDB Atlas IP whitelist	Medium	2025-06-22
Sprint 4	Bug Fixes & Performance Optimization	Amirali	- Identify bugs- Optimize load times	High	2025-07-15
Sprint 4	Add Automated Tests	Mikhail	- Unit tests- Integration tests- CI setup	High	2025-07-15
Sprint 4	Code Refactoring & Dependency Updates	Nurbek,Boxuan	- Refactor legacy code- Update packages	Medium	2025-07-15

Jira-Based Backlog Table (With Developers)

Epic 1 – Web Development

Index	Sprint	Task Name	Developer	Description
1.4	Sprint 1	Web UI – Interface Usability	Amirali	Design intuitive interface for web version
1.5	Sprint 1	Web UI – Dark Mode	Boxuan	Apply dark theme and readable fonts for web
1.6	Sprint 1	Web UI – Responsiveness	Boxuan	Responsive design to work

				on multiple web resolutions
2.1	Sprint 2	User Registration and Login	Boxuan	Secure registration/login system with JWT, OAuth
2.2	Sprint 2	Dashboard – Market Overview	Amirali	Display user's portfolio and current stock trends
2.3	Sprint 2	Stock Trading Functionality	Boxuan	Enable users to place buy/sell orders via integrated APIs
2.4	Sprint 2	Portfolio Performance Charts	Boxuan	Show gains/losses over time using charts
2.5	Sprint 2	Premium Version Integration	Stesha	In-app purchase flow for unlocking premium features
2.6	Sprint 2	Edit/View Personal Info	Stesha	Allow users to manage profile and personal settings
2.7	Sprint 2	Detailed Stock Views	Boxuan	Individual stock pages with metrics and history
4.2 ~ 4.6	Sprint 3	API Fetch Error Handling	Stesha ,Boxuan, Nurbek	Fix fetch failures with async/await and try/catch

Epic 2 – Mobile Development

Index	Sprint	Task Name	Developer	Description
1.1- 1.2	Sprint 1	Mobile UI – Interface Usability	Amirali	Design simple, intuitive UI for mobile application
1.21	Sprint 1	Mobile– Dark Mode	Nurbek	Implement dark mode and font adjustments for accessibility

1.3	Sprint 1	Mobile UI – Responsiveness	Nurbek	Ensure mobile views adapt to different screen sizes
2.8	Sprint 2	User Registration and Login	Nurbek	Secure registration/login system with JWT, OAuth
2.9	Sprint 2	Dashboard – Market Overview (Mobile)	Nurbek	Display user's portfolio and current stock trends
3.1	Sprint 3	Stock Search and View	Nurbek	Search functionality with real-time API integration
3.2	Sprint 3	Watchlist Add/Remove	Nurbek	Allow users to bookmark stocks of interest
3.3	Sprint 3	Portfolio Overview and Charts	Mikhail	Summarize stock performance with gain/loss breakdown
3.4	Sprint 3	Offline Portfolio Access	Mikhail	Cache portfolio data for offline availability
3.5	Sprint 3	Reintroduce Premium Flow (Web)	Nurbek	Enable premium purchase functionality on web

Epic 3 – Debugging and Testing

Index	Sprint	Task Name	Developer	Description
err 1 -3.6	3	API Fetch Failure	Boxuan, Amirali	Inconsistent API fetch results due to

				incorrect endpoint handling.
Err 2 - 3.7	3	DOM Rendering Bug	Stesha	DOM doesn't render correctly due to improper React state management.
Err 3 -3.8	3	HTTP POST Failure	Boxuan,Mikhail	POST requests to backend fail because of incorrect headers or payload structure.
Err 4 - 3.9	3	DTO Mismatch	Nurbek	Fetched data structure is incompatible with frontend DTO definitions.
Err 5 -4.1	3	Database Connection Error	Nurbek, Mikhail	Unable to connect to MongoDB due to misconfigured URI and IP whitelist.
Err 6 - 4.2	3	Backend Fetch Error	Boxuan	Backend API returns errors due to incorrect async handling.
Err 7 - 4.3	4	DOM Update Issue	Stesha	Improperly handled React state updates break component rendering.
Err 8 -4.4	4	POST/GET API Fault	Boxuan, Amirali	Incorrect handling of POST and GET methods causing backend rejection.

Err 9 - 4.5	4	Data Format Conflict	Nurbek	Data received from backend does not match frontend schema expectations.
Err 10 - 4.6	4	MongoDB Access Failure	Nurbek	Database connection fails due to invalid credentials or missing permissions.

Development Challenges & Solutions – Web Platform

API Fetch Error

Where: Web Frontend

Issue: Failed to retrieve data from third-party APIs due to incorrect endpoint handling and inconsistent response timing.

Cause: Misconfigured fetch call and improper async handling.

Solution: Revised the fetch implementation using async/await and added error boundaries with try-catch blocks.

Error Message: TypeError: Failed to fetch

DOM Manipulation Error

Where: Web Frontend

Issue: DOM elements did not render or update correctly after state changes.

Cause: Incorrect usage of React state and component lifecycle.

Solution: Refactored component logic and applied useEffect hooks for proper reactivity.

Error Message: TypeError: Cannot read properties of undefined (reading 'map')

HTTP POST/GET Error

Where: Web Frontend

Issue: POST/GET requests returned incorrect responses or failed entirely.

Cause: Incorrect headers, payload formatting, or method mismatches with backend endpoints.

Solution: Verified HTTP method usage and updated request headers including Content-Type for JSON.

Error Message: POST https://api.example.com/login 400 (Bad Request)

API Usage Limitations

Where: Web Frontend

Issue: Exceeded rate limits of third-party stock APIs during development.

Cause: High volume of API calls due to frequent reloads and lack of caching.

Solution: Implemented request throttling and fallback to mock data in development mode.

Error Message: 429 Too Many Requests

WebSocket Fetch Error

Where: Backend

Issue: WebSocket connections failed to fetch live stock data.

Cause: General WebSocket library was incompatible with our real-time data source.

Solution: Replaced WebSocket with the WS library for better compatibility and added robust reconnection logic.

Error Message: WebSocket connection failed: Error during WebSocket handshake: Unexpected response code: 400

Fetched Data Structure Error

Where: Backend

Issue: Mismatch between fetched API data format and expected frontend schema.

Cause: Inconsistent key/value mappings and nested structures.

Solution: Normalized data format on the server side before sending it to the frontend.

Error Message: TypeError: Cannot destructure property 'price' of 'undefined'

Database Connection Error

Where: Backend

Issue: Application was unable to connect to MongoDB Atlas.

Cause: Incorrect URI and missing IP whitelist entry on MongoDB Atlas.

Solution: Corrected the .env URI and configured IP access through the MongoDB dashboard.

Error Message: MongoNetworkError: failed to connect to server [cluster0.mongodb.net:27017] on first connect