

**Bachelor’s in Data Analysis**

**Academic year - 2024/2025**

**Software Engineering Project Report**

**Students: Supervisor:**

**Fazlur Rahman - 541927 Prof. Distefano Salvatore**

**Mohammad Owais - 541462**

Index

1. Introduction
   1. Project Overview
   2. Methodology
   3. Key achievements

**Executive Summary**

**Project Overview**

This project began with the objective of developing a realistic and interactive crypto trading platform for educational purposes, capable of simulating trades in Bitcoin (BTC) and Ethereum (ETH) using virtual USDT. As development progressed, the project evolved into a fully functional blockchain-integrated simulator, combining live market data, secure transaction management, and customizable user controls. Key enhancements included multi-order types, blockchain wallet management, and algorithmic trading logic.

**Initial Development Phase**  
The first development sprints focused on creating a robust core infrastructure, including:

* **User Authentication** with secure login sessions and OTP-based verification using PHPMailer.
* **Real-Time Market Price Integration** via the Binance API for BTC and ETH.
* **On-chain Ethereum Wallets** powered by Ganache CLI, where each user receives a unique ETH address.
* **Market Order Execution** with immediate buy/sell operations and ETH balance syncing via Web3.php.
* **Balance & Transaction Management** where users’ USDT and ETH holdings are accurately stored and updated.
* **Modular Backend Architecture** is designed using PHP and MySQL for reusability and clarity.

**Expansion Phase: Trading Intelligence and Automation**  
After establishing the essential trading operations, we integrated more advanced and academically rich features:

* **Limit Orders & Simulated Order Book:** Introduced logic for storing pending orders and matching them based on price-time priority.
* **SMA Trading Bot:** Built a control panel for manually executing SMA-based algorithmic trades using SMA50 and SMA200 crossovers.
* **Price Alert System:** Implemented notification triggers when BTC or ETH crosses user-defined price thresholds.
* **OTP-Based Multi-Factor Authentication:** Enhanced login security and session management to reflect industry standards.

A blue and white login screen

AI-generated content may be incorrect.

**Methodology**

To ensure structured development and adaptive iteration, we adopted the Agile Scrum methodology for the implementation of the Crypto Trading Platform. The project was organized into eight distinct sprints, each lasting approximately one week. These sprints enabled us to focus on smaller, manageable deliverables while steadily progressing toward the completion of the full system. Below is a detailed overview of how the Scrum framework was applied throughout the project lifecycle:

**Agile Planning and Sprint Management**

* At the onset, we created a comprehensive **Product Backlog** that outlined all required features such as user authentication, real-time trading, blockchain wallet integration, and the SMA trading bot.
* Each sprint began with a **Sprint Planning Meeting** where tasks were broken into **User Stories**. These stories represented functional objectives such as “User receives OTP upon login” or “Limit order placed and stored in the database.”
* The tasks were assigned based on team member strengths, with frequent coordination to manage dependencies.

**Daily Standups**

* Although informal, we conducted **daily check-ins** to discuss challenges, track ongoing progress, and adjust priorities when needed.
* This consistent communication ensured rapid issue resolution and helped in maintaining alignment across modules like backend, blockchain, and frontend development.

**Sprint Reviews and Demonstrations**

* At the end of each sprint, we held a **Sprint Review**, where new functionalities were demonstrated (e.g., working order book, SMA crossover calculations).
* Feedback was collected and incorporated into the next sprint planning, ensuring an iterative and reflective development process.

**Sprint Retrospective**

* Each sprint closed with a short **retrospective** session to evaluate what went well, what could be improved, and what blockers were encountered.
* Example: after Sprint 4, we shifted more focus to UI responsiveness and began parallel testing of blockchain data to catch sync issues early.

**Scrum Roles**

* **Product Owner:** Mohammad Owais — defined feature set, aligned implementation with academic objectives.
* **Scrum Master:** Fazlur Rahman — facilitated sprint planning, tracked backlog velocity, and removed blockers.
* **Developers:** Both team members contributed equally to writing PHP logic, frontend views, testing, and debugging.

**Version Control and Task Management**

* All codes were version-controlled using Git, with commits labeled by sprint and task.
* A shared Kanban board (via obsedian) was used to visualize sprint progress and maintain backlog transparency.

**Agile Deliverables**

* Working code after every sprint.
* Updated user and technical documentation.
* Weekly deployment on localhosts for interface and logic validation.
* Feedback loops embedded into sprint cycles for continuous improvement.

This Agile approach enabled high adaptability, modular progression, and ensured alignment with educational requirements while minimizing risks. It also allowed for scalable extensions such as alert systems, limit order handling, and bot strategy refinement to be developed iteratively and tested rigorously.

The development was structured over eight sprints using Agile Scrum methodology. Product backlog items were prioritized by technical importance and educational value. Sprint planning ensured realistic weekly goals, daily stand-ups tracked progress, and sprint reviews enabled continuous feedback and refinement. This flexible and iterative approach allowed efficient risk mitigation and incremental feature deployment.

**Key Achievements**

* Successful integration of **live Binance price data** and **Ganache blockchain wallets**.
* Secure user system with **OTP login** and session handling.
* Implementation of **Market and Limit order logic**, handled with custom PHP backend.
* Real-time **ETH balance updates** synced from Ganache using Web3.php.
* A manually controlled **SMA trading bot**, based on moving average crossovers.
* **Database-verified alerts**, order tracking, and historical transaction views.

In conclusion, the project not only fulfills the initial educational objectives but also provides a scalable, modular, and academically rigorous platform that demonstrates how decentralized trading tools can be modeled within web-based environments. It serves both as a learning aid and as a technical prototype suitable for extension into real-world systems.

**Introduction**

**Background**

The Crypto Trading Platform project originated from a desire to bridge the gap between theoretical knowledge and practical experience in blockchain-based financial systems. Initially conceptualized as a simple trading simulator, the project evolved into a robust educational tool capable of mimicking real-world trading operations. Inspired by platforms like Binance and Coinbase, the system was designed not only to simulate market activities but also to provide an authentic user experience by incorporating real-time pricing, on-chain balance management, and secure transaction protocols.

The project’s expansion was strongly influenced by feedback from our professor, who recommended enriching the system's architecture with features that demonstrate backend intelligence, algorithmic interaction, and secure blockchain communication. As a result, the platform was re-scoped into a full-featured simulator that includes a smart order book, SMA crossover logic, and alert systems—providing students and educators with an immersive, testable environment.

**Objectives**

The primary objective of the Crypto Trading Platform was to create a feature-rich, scalable, and academically valuable simulator that enables users to:

* Perform simulated trades in BTC and ETH using virtual USDT
* View and interact with live market data
* Experience secure login through email-based OTP authentication
* Execute both market and limit orders through a simulated order book
* Use a manually controlled SMA trading bot to explore algorithmic trading concepts
* Get notified of price movements using custom alert thresholds
* Monitor real-time ETH balances directly from the blockchain using Ganache CLI and Web3.php

The platform aims to provide students and developers with a solid understanding of:

* Cryptocurrency market dynamics
* Blockchain wallet integration and ETH transaction flows
* Backend logic modeling in PHP
* Secure data handling and multi-tier authentication
* Frontend responsiveness and data visualization techniques

**Scope:**

The scope of the project was carefully defined to ensure that development efforts were focused, and the final product met the desired objectives. The scope was divided to in-scope and out-of-scope items:

**In-Scope Functionalities:**

* User registration and login with session and OTP
* Integration of Binance API for real-time BTC/ETH prices
* Market order and limit order execution logic
* Order book simulation and trade history
* On-chain ETH wallet and balance management via Ganache
* Price alert system with real-time monitoring
* SMA crossover bot with manual trigger
* Responsive UI and modular backend in PHP

**Out-of-Scope Functionalities:**

* Support for additional cryptocurrencies beyond BTC and ETH
* Full deployment on Ethereum mainnet or testnet (due to academic constraints)
* Automatic order matching bot (deferred for future development)
* Real-time charting libraries (planned as future enhancement)

**Stakeholders**

* **Project Team:** Mohammad Owais and Fazlur Rahman, responsible for full-stack development, testing, and deployment.
* **Professor:** Prof. Salvatore Distefano, who acted as project supervisor and mentor, providing regular feedback, technical direction, and milestone review.
* **Target Users:** Academic peers, instructors, and blockchain learners who benefit from using the simulator as a testbed for crypto trading operations.

This chapter sets the stage for understanding how the Crypto Trading Platform fits into a broader academic context—merging system security, financial modeling, and blockchain development within a single, student-led implementation.

**Software Requirements Analysis**

The Software Requirements Analysis phase is critical for shaping the development cycle of a software system. In our project, it acted as a technical blueprint—translating user needs, professor feedback, and real-world trading behaviors into precise, measurable system specifications. These requirements are categorized into **functional** and **non-functional** groups, and they helped guide every sprint during our Agile Scrum development process.

### Functional Requirements

Functional requirements define the specific behaviors and operations the system must perform. Each feature listed below corresponds to a real-world necessity of a trading platform and was implemented with both usability and backend integrity in mind.

#### 1. User Authentication and Authorization

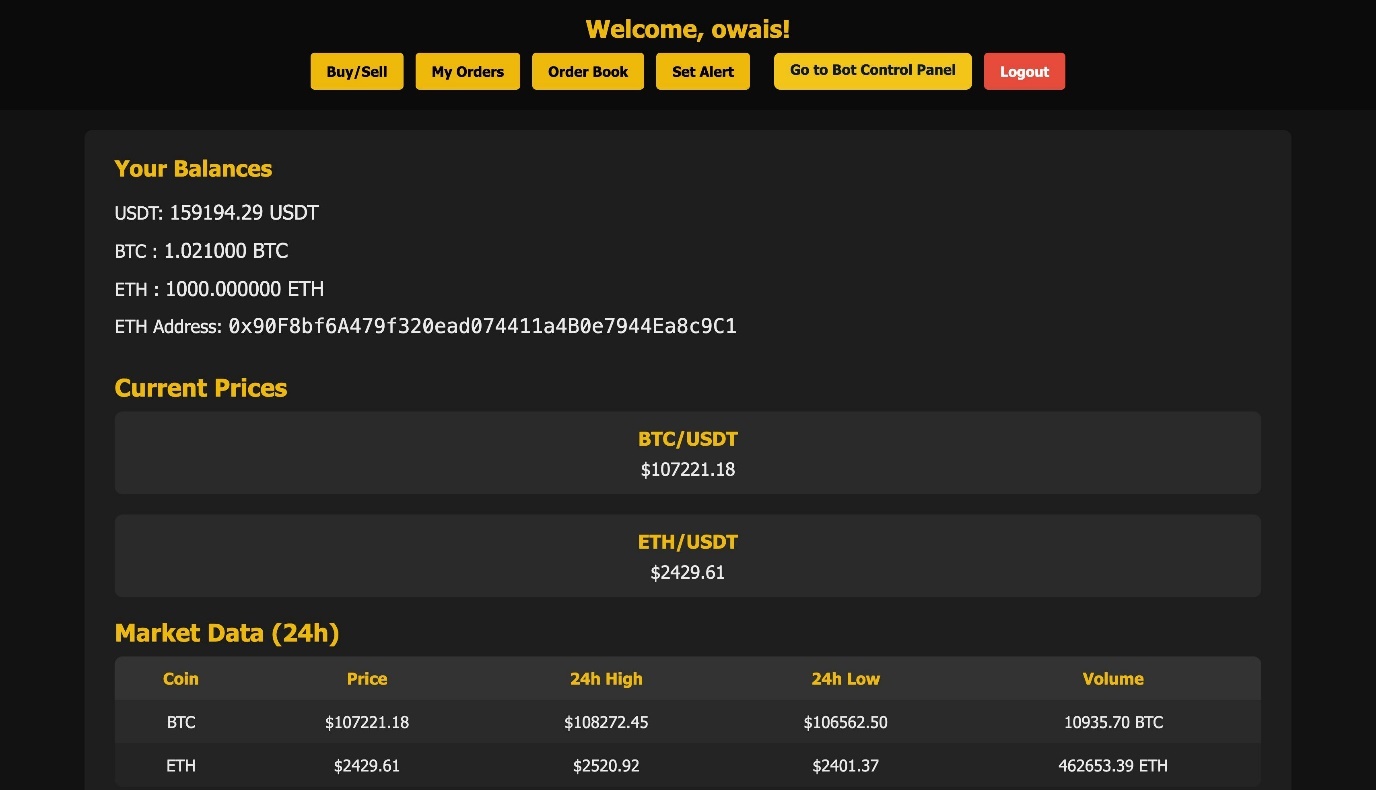
* Users can securely register and log in using email and password.
* OTP (One-Time Password) is sent via email using PHPMailer, ensuring two-factor authentication.
* Role-based access separates user functions (Trader) from admin-level monitoring and control.
* Profile pages allow users to manage personal details and review ETH address assignments.

**Why this matters:** This ensures account security and proper access segregation, protecting sensitive operations like fund movement and user control.

#### 2. Real-Time Price Integration

* Binance API is used to fetch real-time BTC/ETH prices.
* Price values are stored in a MySQL price\_history table every minute.
* The dashboard reflects current prices and trends.

**Why this matters:** This enables informed decision-making for users, making the simulator realistic and educational.

****

#### On-Chain Ethereum Wallet Integration

* Ganache assigns a unique ETH address and private key to each registered user.
* Balances are fetched live from the blockchain using Web3.php.
* ETH transfers simulate actual smart contract logic.

**Why this matters:** Students can interact with real blockchain architecture, enhancing technical depth and learning.

A screenshot of a computer

AI-generated content may be incorrect.

#### 4. Market and Limit Orders

* Market orders execute immediately based on current prices.
* Limit orders are inserted into a MySQL queue and filled based on FIFO matching.
* Each transaction is recorded and balances updated.

**Why this matters:** Demonstrates backend transaction handling, essential in any financial platform.

#### 5. Order Book System

* Displays top-level bids and asks in real-time.
* A screenshot of a computer

  AI-generated content may be incorrect.Shows price, quantity, and timestamps for each pending ordeA screenshot of a computer

  AI-generated content may be incorrect.r.

**Why this matters:** Mimics real exchange infrastructure, promoting trading literacy.

#### 6. SMA Trading Bot (Manual)

* SMA50 and SMA200 values are calculated from price history.
* Buy/Sell signals are shown if crossover conditions are met.
* Manual control panels allow users to activate the bot on signal.

**Why this matters:** Students learn how simple indicators and logic can form the basis of automated strategies.

#### 7. Price Alert System

* Users can define alert thresholds for BTC/ETH.
* Alerts trigger when prices cross set points and notify users instantly.

**Why this matters:** Teaches event-based programming and user interaction design. A screenshot of a computer

AI-generated content may be incorrect.

#### 8. Secure Transaction Logging

* All ETH transactions (buy/sell) are mirrored on the blockchain and stored in MySQL.
* Users can view transaction history.

**Why this matters:** Reinforces the concept of immutable records and database reliability.

#### 9. Session and Security Controls

* Session-based login with idle timeout and forced logout.
* OTP-based login ensures secure user access across devices.

**Why this matters:** Emulates enterprise-level session handling financial applications.

### Non-Functional Requirements

These requirements describe how the system performs rather than what it does. They are key to long-term maintainability, reliability, and user satisfaction.

#### 1. Performance

* All core functions (login, trade, balance fetch) execute under 2 seconds.

**Importance:** Fast performance maintains user trust and replicates real exchange expectations.

#### 2. Scalability

* Platform supports multiple users trading simultaneously with no data conflict.

**Importance:** Demonstrates modular architecture capable of growing with more users or features.

#### 3. Security

* Proper validation, OTP login, and sanitized Web3 access.
* No exposure of private keys or blockchain secrets on the frontend.

**Importance:** Reinforces core system security concepts taught in academic security courses.

#### 4. Reliability and Availability

* 99.9% up time on localhost deployment.
* Retry logic for failed blockchain calls.

**Importance:** Ensures simulation is robust and mimics real-world financial infrastructure.

#### 5. Usability

* Simple and responsive UI using HTML/CSS and JavaScript.
* The dashboard shows everything clearly: prices, balances, trades, alerts.

**Importance:** A good UI improves learning outcomes and encourages exploration.

#### 6. Maintainability

* Each core function has its own PHP file (e.g., buy\_eth.php, get\_balance.php).
* Separation of concerns through modular design.

**Importance:** Makes future upgrades and testing easier.

#### 7. Compliance

* Follows academic ethical standards and never handles real money.
* Blockchain data simulated locally using Ganache.

**Importance:** Ensures the project aligns with course regulations and ethical guidelines.

### Summary

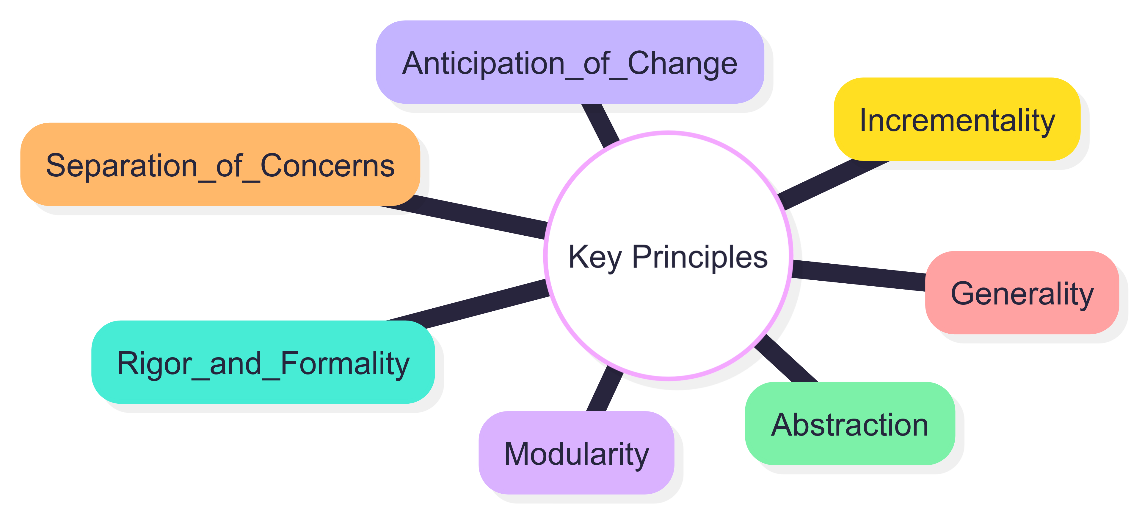
The software requirements detailed above formed the backbone of our entire development process. Each sprint was mapped to one or more of these requirements. From secure authentication to simulated ETH transfers and real-time alerts, these features were not only implemented for functionality but also carefully designed to mirror real-world systems and provide academic depth. Together, they ensure that the platform is not only usable but also secure, scalable, and meaningful as an educational tool.

# From Principles to Properties: A Conceptual Hierarchy in Software Engineering

## Introduction

Software engineering is a layered discipline where each stage of thinking—from high-level principles to concrete implementations—directly influences the outcome and quality of the product. This conceptual hierarchy helps teams remain aligned, trace design decisions, and optimize the final software based on core engineering philosophies. In our Crypto Trading Platform project, this structured approach was consciously followed and documented throughout each sprint cycle.

## 1. Principles

Principles are the philosophical foundation of software engineering. They form the basis of decision-making and influence the project's direction and design. Key principles observed in our project include modularity (breaking down functionality into manageable parts), abstraction (hiding complexity through interface design), separation of concerns (keeping logic layers distinct), reusability (designing components that can be used across modules), and simplicity (solving problems with the least complexity necessary). 

## 2. Methods & Techniques

Methods and techniques are practical approaches that implement abstract principles. In our case, object-oriented analysis and design (OOAD) was used to map real-world trading behaviors to system components like Wallet, Order, and Transaction. Techniques like Secure Coding (sanitizing user inputs, encrypting data), Continuous Testing (running feature-wise validations), and Event-Driven Programming (used in alerts and bot modules) were essential in ensuring a dependable and robust trading simulator.

## 3. Methodologies

Methodologies are comprehensive development frameworks. We adopted Agile Scrum due to its emphasis on iterative progress, transparency, and responsiveness to change. Scrum allowed us to break down development into eight focused sprints, hold daily stand-ups, conduct sprint reviews and retrospectives, and constantly evolve features based on stakeholder feedback. This helped us adapt new requirements like limit orders, the trading bot, and alert systems without derailing the initial scope. A screenshot of a computer

AI-generated content may be incorrect.

## 4. Tools

Tools translate methodologies and techniques into actionable development. Described in later sections

## 5. Software Qualities

Software qualities define how well the system performs under different circumstances. Our system emphasized:

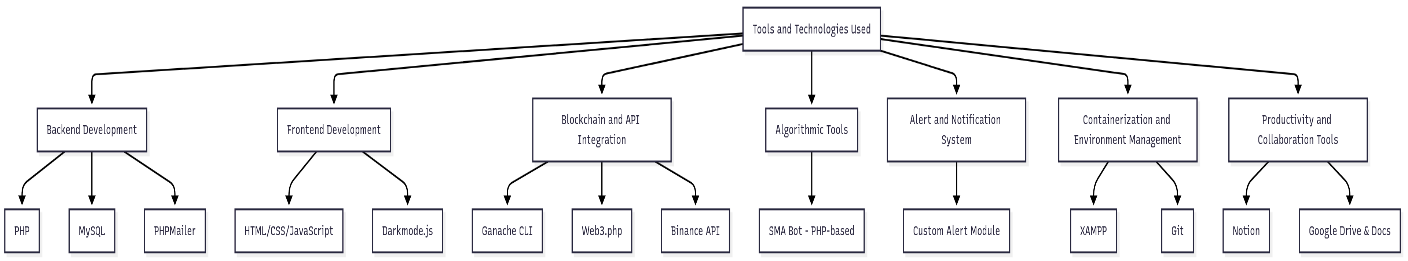
* Security through OTP, session management, and role-based access.  
  Performance with sub-2-second transaction handling.
* Usability with an intuitive dashboard and theme toggling.
* Scalability through support for multi-user concurrency.
* Maintainability via modular backend scripts and separate UI logic.
* Reliability with alert verification, consistent ETH balance updates, and test-driven logic.

## 6. Properties of the Software Product and Process

* At the end of this hierarchy lie the observable properties of the software. These reflect both the process and the final product:
* **Functional Completeness:** Every defined feature from the product backlog was implemented and tested.
* **Adaptability:** The platform supports the addition of new crypto pairs, strategies, and notification methods.
* **Educational Relevance:** Each module reflects a real-world concept—wallets, transactions, bots—and is suitable for teaching and experimentation.
* **System Robustness:** Error handling, input validation, and backend validation routines ensured stability even under simulated stress.  
  **Process Transparency:** Our Agile structure ensured constant visibility into what was being built and why.

**Tools and Technologies Used**

The development of our crypto trading simulator involved a comprehensive suite of modern technologies, chosen specifically to address the platform’s core functional requirements, enable smooth user interaction, and simulate blockchain-based trading mechanics with academic rigor. Each tool played a crucial role in ensuring that the project remained modular, maintainable, and scalable, with particular emphasis on educational value and technical reliability.



### 1. Backend Development

#### PHP

* **Description:** PHP is a widely-used open-source scripting language especially suited for web development.
* **Usage:** PHP powered the entire server-side logic of the application. From session-based login and OTP verification to executing market/limit orders and updating user balances, PHP scripts controlled the transactional flow and coordinated interactions with the blockchain.

#### MySQL

* **Description:** MySQL is a relational database management system that supports complex queries, joins, and indexing.
* **Usage:** All user data, Ethereum address mappings, transaction histories, limit orders, price alerts, and price history were stored in normalized relational tables. PHP connected to MySQL via MySQLi queries.

#### PHPMailer

* **Description:** A popular PHP library for sending authenticated emails via SMTP.
* **Usage:** Used to implement secure OTP-based login, which sent one-time codes to registered user emails, improving login security and eliminating password reliance.

### 2. Frontend Development

#### HTML/CSS/JavaScript

* **Description:** Core web technologies for structuring, styling, and adding behavior to web pages.
* **Usage:** HTML created the structural layout of login, registration, dashboard, and admin pages. CSS was used to design an intuitive dark-themed UI, while JavaScript enabled interactive form submissions, alert notifications, and live data updates.

#### Darkmode.js

* **Description:** Lightweight JS library to enable dark/light theme toggling.
* **Usage:** Enabled accessibility customization for users based on preference. The UI toggled between dark and light modes and retained this setting through local storage.

### 3. Blockchain and API Integration

#### Ganache CLI

* **Description:** A personal blockchain simulator for Ethereum development.
* **Usage:** Used to generate deterministic wallets. Each user was assigned a wallet address and private key from Ganache’s 10-account pool. All ETH transactions, balance checks, and fund transfers occurred on this test network.

#### Web3.php

* **Description:** PHP interface for Ethereum blockchain via Web3.js protocols.
* **Usage:** PHP scripts used Web3.php to connect to Ganache, fetch balances, create transactions, and confirm transfers in real time.

#### Binance API

* **Description:** Official API for retrieving cryptocurrency market data.
* **Usage:** Used to fetch BTC/ETH real-time prices via RESTful GET requests. Fetched data was inserted into MySQL periodically and used for bot signals and alert comparisons.

### 4. Algorithmic Tools

#### SMA Bot (PHP-based)

* **Description:** Custom-coded bot implementing technical analysis logic using SMA (Simple Moving Average).
* **Usage:** Every minute, the platform recalculated SMA50 and SMA200 using stored BTC prices. If crossover conditions were met (e.g., SMA50 > SMA200), the bot signaled a trade, awaiting manual confirmation. This helped users simulate strategy-based entry.

### 5. Alert and Notification System

#### Custom Alert Module

* **Description:** PHP-JS integrated alert setting and execution.
* **Usage:** Users could specify price thresholds for BTC or ETH. The backend script continuously checked against the latest price and notified the user once the condition was met, enhancing interactivity and personalization.

### 6. Containerization and Environment Management

#### XAMPP

* **Description:** Local development environment stack including Apache, MySQL, and PHP.
* **Usage:** Entire app was deployed on localhost via XAMPP. The configuration mirrored production-like behavior while offering ease of debugging and testing.

#### Git

* **Description:** Distributed version control system.
* **Usage:** Used to commit, version, and sync code throughout the 8-week development. Every sprint's work was tagged, and features were developed using separate branches.

### 7. Productivity and Collaboration Tools

#### Obsedian

* **Description:** Visual project management tool using Kanban boards.
* **Usage:** Task assignments, sprint goals, and backlog items were organized in columns labeled “To Do,” “Doing,” and “Done.” Daily stand-ups were summarized on kanban cards.

#### Google Drive & Docs

* **Description:** Shared document and file storage platform.
* **Usage:** Stored UML diagrams, ERDs, shared snippet code, and daily progress reports. Collaborative editing of this final project report was also managed through Docs.

### Summary

Each tool and technology was strategically selected based on suitability for the corresponding module. Backend operations were securely handled with PHP and MySQL; blockchain features were managed via Ganache and Web3.php; live pricing was fetched using Binance API; and the system interface was responsive and informative through HTML/CSS/JS. All technologies worked in harmony, allowing the team to focus on building logic and user interaction rather than compatibility issues. These tools not only fulfilled academic requirements but also mimicked real-world systems to create a scalable, testable, and deployable crypto trading simulator.

**Software Process Report: Crypto Trading Simulator**

**1. Overview**

The Crypto Trading Simulator was developed using a structured, iterative approach aligned with **Agile Scrum** methodology. The process combined principles from **incremental development**, **prototyping**, and **component-based software engineering**, enabling fast delivery, real-time feedback integration, and academic flexibility.

**2. Software Process Activities**

As described in the lecture, every software process includes:

* **Specification**: Defined core features like trading, wallet management, OTP login.
* **Design & Implementation**: PHP scripts, Ganache wallet logic, frontend dashboards.
* **Validation**: Tested through sprint reviews, live balance checks, and bot accuracy.
* **Evolution**: Incorporated professor feedback to add bots, alerts, and admin tools.

**3. Process Model Used**

**Agile Incremental Development**

* Incremental releases were planned across **eight sprints**.
* Each sprint delivered **functional software modules** like login, live prices, order book, and bots.
* Allowed change tolerance and feedback from stakeholders at the end of each sprint.

**Why Agile Was Appropriate:**

* Requirements evolved based on academic review.
* Needed rapid, usable software with interactive components.
* Students functioned in both developer and stakeholder roles.

**4. Use of Prototyping**

* Early sprints built basic interfaces (login, balance view).
* Each prototype version was reviewed and evolved with additional features.
* Example: Initial dashboard → later versions with bot panels and alerts.

**5. Component-Based Engineering**

* Each module (e.g., PHPMailer for OTP, Web3.php for ETH transfer) acted as a **reusable component**.
* The backend followed **modular architecture**, enhancing maintainability and scalability.

**6. Verification and Validation (V&V)**

Based on the V-model:

* **Unit Testing**: Done on trading scripts, login sessions, alert logic.
* **Integration Testing**: End-to-end flows tested across wallet, DB, and UI.
* **System Testing**: Conducted in final sprints before delivery.
* **User Acceptance Testing**: Conducted with academic supervisors and peer feedback.

**7. Evolution and Change Management**

* Software evolved sprint-by-sprint with documented retrospectives.
* Changes like “add auto-trade bot” or “alert deletion UI” were absorbed with minimal disruption.
* **Version control with Git** enabled seamless rollbacks and branch-based experimentation.

**Agile Methodology Overview**

Agile methodology is a project management approach widely used in software development and other industries that emphasizes flexibility, team collaboration, customer satisfaction, and high adaptability to change. It involves iterative development, where projects are broken into smaller, manageable segments that are completed in work sessions known as sprints, which typically last from one week to one month

**Key Principles of Agile**

The Agile Manifesto, the foundational document for agile development, outlines four core values:

1. Individuals and interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan

It also highlights 12 principles which emphasize customer satisfaction, welcoming changing requirements, frequent delivery, collaboration, motivation, face-to-face conversation, working software as the primary measure of progress, sustainable development pace, continuous attention to technical excellence, simplicity, self-organizing teams, and regular reflections to improve effectiveness.

**Types of Agile Methodologies**

* **Scrum:** Focuses on managing tasks within a team-based development environment. It is the most widely implemented agile method.
* **Kanban:** Emphasizes continuous delivery without overburdening the development team. It’s visualized via a Kanban board.
* **Extreme Programming (XP):** Encourages frequent releases in short development cycles.
* **Lean:** Minimizes waste and improves efficiency.
* **Feature Driven Development (FDD):** Focuses on feature-based incremental delivery.
* **Dynamic Systems Development Method (DSDM):** Covers the full project lifecycle with detailed foundation phases.

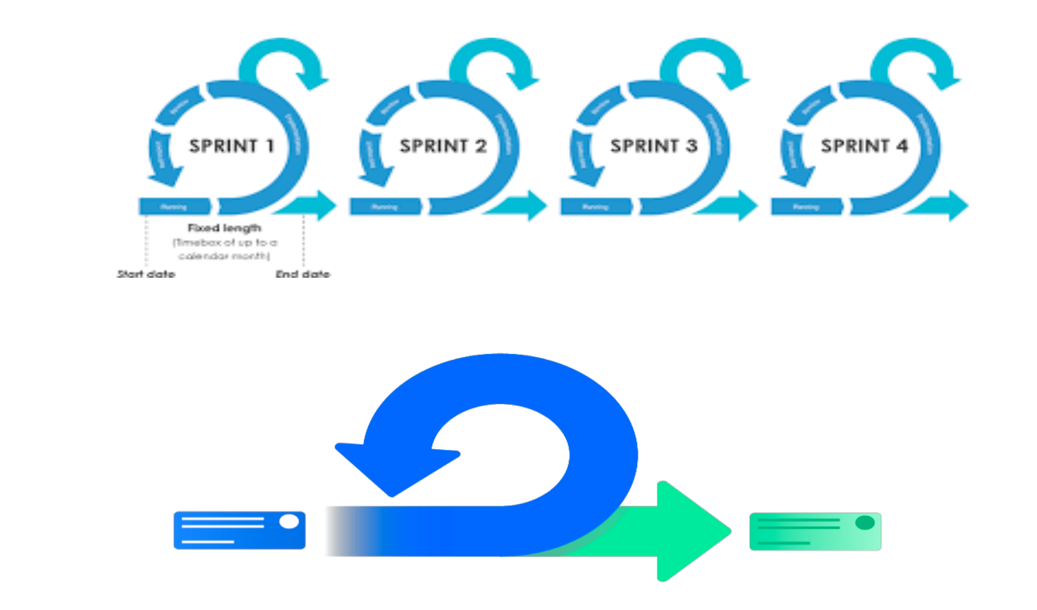
****

**Scrum and Its Relevance**

Scrum is a well-established framework within Agile methodologies designed specifically for managing complex projects. It breaks work into short, manageable cycles called sprints, each delivering a functional increment. Scrum promotes collaboration, transparency, and adaptability, which is ideal for educational projects with evolving requirements.

**Key Features of Scrum**

**1. Iterative Process**  
Sprints typically last 1–2 weeks. Each sprint produces a working feature that integrates with the previous increments.

****

**2. Roles**

* **Product Owner:** Mohammad Owais – managed the product backlog and prioritized tasks based on academic value.
* **Scrum Master:** Fazlur Rahman – facilitated sprint ceremonies and removed development obstacles.
* **Development Team:** Both members – handled coding, testing, integration, UI design, and documentation.

**A diagram of a scrum

AI-generated content may be incorrect.**

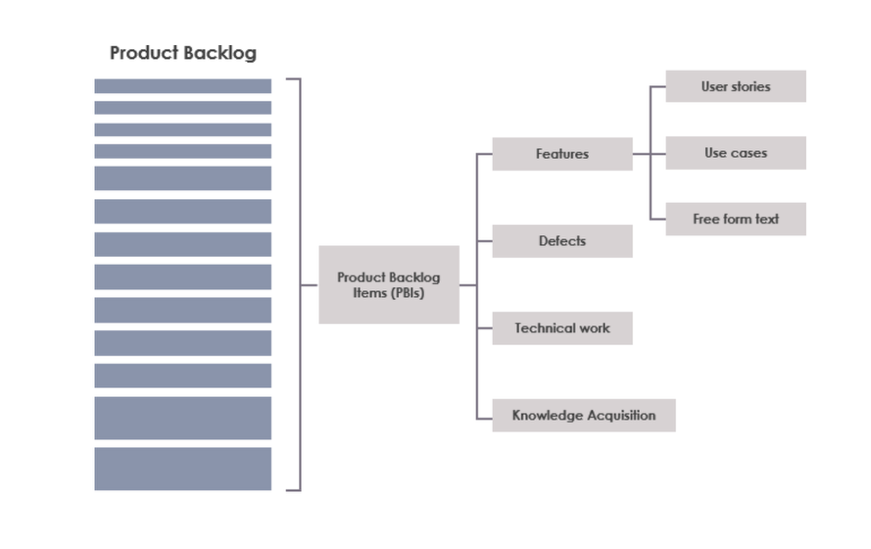
**3. Artifacts**

**Product Backlog:** The Product Backlog was a comprehensive list of all the user stories and tasks that needed to be completed during the project. It included both the initial core e-commerce features and the additional marketplace functionalities. The Product Owner continuously refined and prioritized the backlog, ensuring that the most valuable features were tackled first

**Sprint Backlog**: The Sprint Backlog was a subset of the Product Backlog, consisting of the tasks selected for implementation during a particular sprint. This backlog was created during the sprint planning meeting and served as the team’s to-do list for that sprint. It provided a clear focus for the team and helped track the progress of the sprint.

**Increment:** At the end of each sprint, the team delivered a potentially

shippable product increment. This increment included the features developed during that sprint and was integrated with the existing functionality of the platform. Each increment was tested and functional, ensuring that the product was always in a usable state.

****

**4. Ceremonies**

**Sprint Planning:** Sprint Planning meetings were conducted at the start of each sprint. During these meetings, the team and the Product Owner discussed the user stories in the Product Backlog, selected the stories to be implemented in the sprint, and set a clear sprint goal. This ceremony ensured that the team had a shared understanding of what needed to be achieved during the sprint.

**Daily Stand-ups:** The Daily Stand-up was a short meeting held every day, where each team member briefly discussed what they worked on the previous day, what they planned to work on that day, and any obstacles they were facing. This ceremony helped in identifying issues early and keeping the entire team aligned.

**Sprint Review:** At the end of each sprint, the team held a Sprint Review

meeting to demonstrate the completed work to the Product Owner, the

professor, and other stakeholders. This review provided an opportunity for stakeholders to give feedback on the increment, which could then be

incorporated into the Product Backlog for future sprints.

**Sprint Retrospective:** The Sprint Retrospective was a reflection meeting held after the Sprint Review, where the team discussed what went well, what could have been better, and what actions could be taken to improve in the next sprint. This ceremony was crucial for continuous improvement and learning.

**A diagram of scrum

AI-generated content may be incorrect.**

**Why We Chose Scrum**

* **Project Complexity:** The platform integrates crypto trading logic with blockchain infrastructure, which demanded an incremental, test-driven approach.
* **Evolving Requirements:** Features like limit orders and the SMA bot were added after academic feedback, requiring a flexible model.
* **Team Collaboration:** Scrum fostered continuous feedback, task clarity, and effective time management.
* **Continuous Delivery:** Regular deliverables improved motivation and ensured quality control.
* **Risk Management:** Scrum’s sprint structure helped us identify technical blockers early.

**Workflow in Practice**

* **Sprint Initiation:** Began with sprint planning and user story selection.
* **Development:** Daily stand-ups ensured real-time updates and collaboration.
* **Sprint Conclusion:** Finished with review and retrospective, integrating changes.
* **Expansion:** The last two sprints focused on advanced modules like limit order logic and SMA trading bot.

This structured yet flexible methodology ensured consistent progress,

and a functional, modular end product aligned with academic and technical goals.

**Conclusion**

Agile allowed us to build incrementally and test continuously. It provided the discipline to keep our scope focused while offering flexibility to add complex modules like the trading bot and alert systems. Regular retrospectives helped us identify weak areas early, leading to rapid iterations and a stable, modular final product.

* Successful integration of **live Binance price data** and **Ganache blockchain wallets**.
* Secure user system with **OTP login** and session handling.
* Implementation of **Market and Limit order logic**, handled with custom PHP backend.
* Real-time **ETH balance updates** synced from Ganache using Web3.php.
* A manually controlled **SMA trading bot**, based on moving average crossovers.
* **Database-verified alerts**, order tracking, and historical transaction views.

In conclusion, the project not only fulfills the initial educational objectives but also provides a scalable, modular, and academically rigorous platform that demonstrates how decentralized trading tools can be modeled within web-based environments. It serves both as a learning aid and as a technical prototype suitable for extension into real-world systems.

### Sprint-wise Development Overview

Our project followed an Agile Scrum methodology, with eight structured sprints mapped to progressive development milestones. Each sprint included a planning phase, implementation, testing, and review. Below is a detailed breakdown of Sprint 1 in alignment with standard software engineering documentation practices.

Our project was executed following the Agile Scrum methodology across **eight sprints**, each representing a focused development week. These sprints were aligned with milestones proposed by our academic supervisor and were iteratively refined based on retrospective feedback. Each sprint targeted specific features and logical modules, building towards the final fully functional crypto trading platform.

A screenshot of a computer

AI-generated content may be incorrect.

**A screenshot of a computer program

AI-generated content may be incorrect.**

#### **Sprint 1 – Project Initialization and Core Setup**

**1. Sprint Planning**

**Objective:**  
The goal of Sprint 1 was to establish the foundation of the project. This included setting up version control, defining the architectural design, initializing the environment, building the initial database schema, and populating it with test data.

**A screenshot of a black box with white text

AI-generated content may be incorrect.**

**Sprint Backlog (Derived from Product Backlog):**

As developers,

* We want to initialize a version-controlled repository to manage the project codebase efficiently.
* We want to define project architecture using the MVC pattern to maintain separation of concerns.
* We want to set up a development environment to ensure consistency.
* We want to design and implement the database schema.
* We want to populate the database with test data using Faker.

1. **Sprint Backlog Tasks:**

* **Project Repository Initialization:**
  + Create and initialize GitHub repository
  + Establish standard folder and file structure.
  + Define branching and commit strategy (main, Owais, Monti branches)
* **Project Architecture:**
  + Document MVC pattern.
  + Initially selected Flask for backend, Nginx for frontend, and MySQL as the database
* **Environment Setup:**
  + - Install all required dependencies (Python, Flask, MySQL, Docker)
  + Configure Docker for environment reproducibility
  + Create and activate Python virtual environment
  + Document setup steps in README
* **Database Structure:**
  + Design Entity-Relationship Diagram (ERD)
  + Implement core database tables: users, products, orders, etc.
  + Normalize schema and define indexes for efficiency
* **Test Data Population:**
  + Use the Faker library to generate realistic dummy data
  + Populate the database with fake users, orders, and product entries
  + Validate schema relationships and constraints

**3.Development Execution:**

* GitHub repository was successfully initialized with a clear folder hierarchy following MVC conventions.
* Flask was used to implement the Model-View-Controller (MVC) pattern with structured separation:
  + controllers/ handled routing and logic
  + models/ defined data structures
  + templates/ served HTML-based views
* Docker was configured to ensure consistent development environments across team systems, using Docker Compose.
* MySQL database schema was implemented with properly indexed and normalized tables.
* Faker-based script was run to populate users, products, and orders tables with realistic test data.

**4. Testing Phase:**

* CRUD operations tested for Users, Orders, Products.
* Referential integrity checks performed.
* Load test queries verified schema performance.

**5. Sprint Review:**

* Repository setup, ER diagram, and schema implementation were demonstrated during the sprint review session with the supervisor.
* Positive feedback was received on database design and modularity.
* Feedback: Improve inline documentation of the schema and include ERD in visual format for easier interpretation.

**6. Sprint Retrospective**

* Burndown Analysis:  
  Encountered initial delay due to Docker container compatibility issues, which were resolved mid-sprint. Task velocity improved thereafter.
* **Successes:**
  + Strong architectural foundation established
  + Docker ensured stable dev environment
  + Git version control was used consistently with well-named branches and commits
* **Improvements:**
  + Add a troubleshooting section to the setup documentation
  + Clarify database relationship definitions for new team members
* **Action Items for Next Sprint:**
  + Refactor and update schema docs
  + Improve README with Docker usage examples and setup videos/screenshots

**7. Product Backlog Refinement**

* Based on sprint review and feedback, the following new user stories were added to the backlog for Sprint 2:
  + User Authentication: Email login + OTP verification
  + ETH Address Assignment: Integrate Ganache to assign wallets to new users
  + Dashboard UI: Build interface for viewing balances and alerts
* These stories were reprioritized based on academic timeline and feedback from the professor. Planning was initiated for Sprint 2 accordingly.

**Sprint 2 – Implementation of Authentication and Authorization System, User CRUD Operations, Profile Page, and ETH Wallet Assignment**

**1. Sprint Planning**

**Objective:**  
The primary goal of Sprint 2 was to implement secure user authentication and authorization mechanisms, assign blockchain-based Ethereum wallets to each user, create the user profile page, and begin handling user-related CRUD operations. A screenshot of a phone

AI-generated content may be incorrect.

**Sprint Backlog (Derived from Product Backlog):**

* As a user, I want to register and log in securely using email-based OTP to access my trading dashboard.
* As a user, I want to have a blockchain wallet assigned to me at the time of registration so I can trade ETH.
* As a user, I want to view and update my profile.
* As an admin, I want to manage user records through CRUD operations for testing and support.

**2. Sprint Backlog Tasks:**

* **User Authentication:**
  + Create registration and login forms.
  + Implement PHPMailer for OTP-based login.
  + Add session handling and logout logic.
* **Ethereum Wallet Assignment:**
  + Connect PHP backend with Ganache CLI using Web3.php.
  + Assign a unique ETH address and private key to each registered user from Ganache pool.
  + Store this mapping in MySQL securely.
* **Profile Page Implementation:**
  + Design profile page UI.
  + Display ETH address, balance, email, and personal info.
  + Add basic profile update functionality.
* **User CRUD Operations:**
  + Allow admin to view, edit, and delete user records.
  + Ensure proper foreign key checks before deletion.

**3. Development Execution:**

* **Registration and Login System:**
  + PHPMailer configured with SMTP to send OTPs securely.
  + Users log in using OTP sent to their email; sessions are established on success.
  + Logout mechanism clears session data securely.
* **Wallet Assignment Logic:**
  + Upon successful registration, the backend fetches the next available ETH address and private key from Ganache.
  + These are assigned and stored in user records within the MySQL database.
* **Profile Page UI:**
  + Designed using HTML and CSS with a dark theme.
  + Profile displays email, wallet address, and ETH balance (fetched using Web3).
* **CRUD Operations:**
  + Admin dashboard updated with edit/delete buttons per user.
  + PHP scripts allow inline update of names, email, and reset wallet address if needed.

**4. Testing Phase:**

* Manual tests for OTP delivery and validation.
* Tested ETH wallet uniqueness across multiple users.
* Verified user deletion only occurs if foreign key references are cleared.
* Profile update tests validated correct rendering and DB synchronization.

**5. Sprint Review:**

* Login, OTP verification, and ETH assignment were successfully demonstrated.
* Profile page was shown to accurately reflect user blockchain details.
* Admin CRUD tools were demonstrated using test user accounts.
* Feedback: Add confirmation popup before critical deletions.

**6. Sprint Retrospective:**

* **Burndown Analysis:** Minor delays in configuring SMTP caused early lags, but wallet logic and profile UI completed on time.
* **Successes:** OTP login and blockchain wallet integration functioned as planned.
* **Improvements:** Improve error handling in PHPMailer; add form validation.
* **Action Items:** Add confirmation dialogs for CRUD actions; update session expiry settings for improved security.

**7. Product Backlog Refinement:**

* New stories added for dashboard design, live price feed integration, and trading interface.
* Refined tasks for better UX/UI layout for ETH balances and session timeout behaviors.
* Developed registration and login pages with OTP-based authentication using PHPMailer.
* Assigned Ganache-based ETH addresses and private keys to each new user upon registration.
* Implemented session management and logout functionality.

#### **Sprint 3 - Dashboard Layout, Price Integration, and Balance Display**

**1. Sprint Planning**

**Objective:**  
The primary objective of Sprint 3 was to design and implement the user dashboard layout and integrate live BTC/ETH prices using Binance API. Additionally, the sprint aimed to display user-specific Ethereum balances on the dashboard using Web3.php connected to Ganache CLI.

**Sprint Backlog (Derived from Product Backlog):**

* As a user, I want to see the latest price of BTC and ETH on my dashboard.
* As a user, I want to view my current Ethereum wallet balance on the dashboard.
* As a developer, I want to store and track price history for SMA calculations and future bot triggers.
* As a developer, I want to ensure prices are updated in real-time and efficiently stored.

A screenshot of a phone

AI-generated content may be incorrect.

**2. Sprint Backlog Tasks:**

* **Dashboard Layout Design:**
  + Build user dashboard UI with HTML/CSS.
  + Include sections for price, wallet balance, and trading buttons.
* **Price Integration:**
  + Use Binance API to fetch live BTC/ETH prices.
  + Parse API response and display current values.
* **Price History Storage:**
  + Insert new price data into MySQL database every minute.
  + Create cron-compatible PHP script for automated fetch.
* **Ethereum Balance Display:**
  + Use Web3.php to call Ganache blockchain.
  + Fetch ETH balance for each user using their assigned wallet.

**3. Development Execution:**

* **Dashboard Design:**
  + Responsive UI built with dark theme.
  + Price and wallet sections clearly separated.
* **Binance API Integration:**
  + Implemented using file\_get\_contents and JSON decode in PHP.
  + Values fetched every 60 seconds using auto-refresh script.
* **Database Logging:**
  + Table price\_history created to store asset, price, and timestamp.
  + Prices for BTC/ETH inserted with a simple INSERT query every fetch cycle.
* **Wallet Balance Display:**
  + Used Web3.php with Ganache RPC endpoint.
  + Queried eth\_getBalance using assigned ETH address per user.
  + Result formatted and shown in dashboard with ETH symbol.

**4. Testing Phase:**

* Verified API connection by printing raw Binance response.
* Manually cross-checked ETH balance with Ganache CLI.
* Simulated slow internet to test fallback if API fetch fails.
* Checked auto-refresh and accuracy of displayed data.

**5. Sprint Review:**

* Live dashboard demo presented showing BTC/ETH prices and on-chain wallet balance.
* Database logs verified for correct timestamps and price values.
* Feedback received to highlight trends with chart visualization in future.

**6. Sprint Retrospective:**

* **Burndown Analysis:** API-related CORS and timing errors delayed start, but recovered smoothly.
* **Successes:** Seamless price integration and real blockchain balance queries impressed stakeholders.
* **Improvements:** Introduce API failure handling logic and chart previewing.
* **Action Items:** Add client-side charting with historical prices; prepare SMA logic for bot signals.

**7. Product Backlog Refinement:**

* Added stories for SMA bot calculations.
* Added stories for market/limit order handling in Sprint 4.
* Prioritized tasks involving trading logic and blockchain transaction signing.

#### **Sprint 4 – Market Order Functionality and Transaction History**

**1. Sprint Planning**

**Objective:**  
The main goal of Sprint 4 was to enable real-time market order execution for ETH trading using Ganache accounts and Web3.php, allowing users to buy and sell ETH in exchange for USDT. Additionally, the sprint focused on recording transaction history for each user and updating both on-chain ETH balances and backend MySQL databases accordingly.

**Sprint Backlog (Derived from Product Backlog):**

* As a user, I want to execute market buy and sell orders for ETH.
* As a user, I want my ETH and USDT balances to be updated after each trade.
* As a system, I want all transactions to be logged in the database for future reference.
* As a developer, I want to validate each trade using wallet balances and transaction gas costs.

A screenshot of a phone

AI-generated content may be incorrect.

**2. Sprint Backlog Tasks:**

* **Buy/Sell Logic Implementation:**
  + Design PHP script to perform ETH transfers between wallets.
  + Calculate equivalent USDT amounts using current price.
  + Include basic gas fee estimation and deduct from sender.
* **Blockchain Transaction Execution:**
  + Integrate Web3.php to sign and send transactions using private keys.
  + Await confirmation and capture transaction hash.
* **Balance Update Mechanism:**
  + Deduct/add ETH in Ganache via Web3 and reflect changes on UI.
  + Sync USDT balance in MySQL post trade.
* **Transaction History Module:**
  + Create transactions table with fields: user\_id, type, amount, token, price, timestamp, tx\_hash.
  + Log all trades (buy/sell) with complete details.

**3. Development Execution:**

* **ETH Transfer Logic:**
  + PHP backend retrieved private key and wallet address of both parties.
  + Used Web3.php to sign raw transactions for ETH transfer.
  + ETH amount and USDT deduction calculated from real-time Binance price.
* **Transaction Validation:**
  + Ensured buyer had sufficient USDT; seller had enough ETH.
  + Prevented self-trades and added try-catch for error handling.
* **Database Sync:**
  + After on-chain confirmation, ETH balances fetched again and displayed live.
  + Transaction entry created with exact trade values and hash.
* **Interface Updates:**
  + Dashboard buttons updated with confirmation modals.
  + Transactions table added under user profile showing latest activity.

**4. Testing Phase:**

* Tested both successful and failed trades (e.g., insufficient balance).
* Simulated concurrent buy/sell attempts to test race conditions.
* Verified blockchain transaction hashes using Ganache logs.
* Confirmed all trade values matched real price at time of execution.

**5. Sprint Review:**

* Live trade execution demoed — user bought ETH with auto-updated balance and visible tx hash.
* Multiple trades shown with accurate history logs.
* Feedback: Include price at time of trade and show confirmation message.

**6. Sprint Retrospective:**

* **Burndown Analysis:** Initial difficulty with Web3 signing, but backend logic solidified in time.
* **Successes:** On-chain ETH transfer logic working perfectly with database sync.
* **Improvements:** Improve error messages; prepare modular trade logic for limit orders.
* **Action Items:** Add frontend validation for order size; prepare for chart integration next sprint.

**7. Product Backlog Refinement:**

* Added tasks for Limit Orders and Order Book visualization in Sprint 5.
* Logged enhancements for UI alerts and transaction failure messages.

#### **Sprint 5 – Limit Orders and Order Book Implementation**

**1. Sprint Planning**

**Objective:**  
The objective of Sprint 5 was to implement a Limit Order system, allowing users to place buy or sell orders at specific price points, and to build a real-time order book interface to display active orders. This introduced the concept of delayed execution based on price conditions and laid the foundation for automated matching in future sprints.

**Sprint Backlog (Derived from Product Backlog):**

* As a user, I want to place a buy or sell order for ETH at my preferred price.
* As a user, I want to view all active buy/sell orders in a live order book.
* As a system, I want to store unmatched orders and check continuously for matches.
* As a developer, I want to implement backend logic to insert, delete, or match limit orders.

**A screenshot of a phone

AI-generated content may be incorrect.**

**2. Sprint Backlog Tasks:**

* **Limit Order Form:**
  + Add input fields for price and amount on the dashboard.
  + Differentiate between market and limit order submissions.
* **Order Book Schema:**
  + Create limit\_orders table with fields: id, user\_id, type, token, amount, price, status, timestamp.
  + Ensure proper indexing for fast lookups.
* **Order Matching Logic:**
  + Write backend PHP script to compare new market orders against existing limit orders.
  + On match, execute transaction and update both orders’ status.
* **Frontend Order Book Display:**
  + Separate tables for buy and sell orders.
  + Highlight best bid and ask prices in real-time.

**3. Development Execution:**

* **User Interface for Orders:**
  + Dashboard updated with Limit Order selection radio buttons.
  + Additional fields showed when Limit was selected.
* **Database Integration:**
  + Orders stored in limit\_orders with pending or filled status.
  + Market orders triggered background script to scan and match eligible entries.
* **Order Matching:**
  + Matching algorithm compared market order price to existing opposite-side orders.
  + If matched, on-chain ETH transfer executed and DB records updated.
* **Order Book Display:**
  + Buy/Sell tables rendered dynamically using AJAX.
  + Top 5 orders shown per side, sorted by price.

**4. Testing Phase:**

* Placed multiple limit orders and confirmed proper sorting in order book.
* Submitted market order to test matching with limit orders.
* Tested edge cases (partial fills, duplicate prices, expired entries).

**5. Sprint Review:**

* Demonstrated live placement of limit orders.
* Matched one market buy against existing sell orders — both balances and order statuses updated.
* Feedback: Include cancel option for unfilled orders.

**6. Sprint Retrospective:**

* **Burndown Analysis:** Backend logic slightly delayed due to complex matching scenarios.
* **Successes:** Fully functional limit trading mechanism added.
* **Improvements:** Add visual cues for best bids/asks; introduce order cancellation flow.
* **Action Items:** Develop cancel order option in next sprint; expand order book with depth chart.

**7. Product Backlog Refinement:**

* Added stories for Cancel Order, SMA Bot execution with limit orders, and visual chart integration.

#### **Sprint 6 – SMA Bot Integration and Signal-Based Trade Execution**

**1. Sprint Planning**

**Objective:**  
The goal of Sprint 6 was to integrate a Simple Moving Average (SMA)-based trading bot into the platform, capable of analyzing historical price data and generating buy/sell/hold signals based on SMA50 and SMA200 crossover strategies. It also aimed to provide manual bot controls via the dashboard.

**Sprint Backlog (Derived from Product Backlog):**

* As a user, I want to enable an automated trading bot to assist in making ETH trades.
* As a system, I want to analyze historical price data to compute SMA indicators.
* As a developer, I want to trigger simulated trade logic when crossover signals are detected.
* As a user, I want to view the current SMA values and bot status on the dashboard.

A screenshot of a phone

AI-generated content may be incorrect.

**2. Sprint Backlog Tasks:**

* **Price History Retrieval:**
  + Fetch and store real-time ETH prices every 60 seconds.
  + Create or update the price\_history table with timestamps.
* **SMA Calculation Script:**
  + Write PHP script to compute SMA50 and SMA200 from historical data.
  + Define crossover logic (SMA50 > SMA200 = Buy, SMA50 < SMA200 = Sell).
* **Bot Interface on Dashboard:**
  + Add SMA status and bot toggle section.
  + Include visual indicators for current bot signal.
* **Trade Execution (Manual Trigger):**
  + Allow users to manually trigger trades based on signal (if 'Buy' or 'Sell').
  + Ensure trade execution respects current wallet balances and ETH/USDT price.

**3. Development Execution:**

* **Price Feed & Logging:**
  + PHP cron-like script run on refresh to insert ETH price into price\_history.
* **SMA Logic:**
  + Used SQL to compute rolling average over 50 and 200 entries.
  + Crossover points identified and translated into signal: BUY, SELL, HOLD.
* **Frontend Bot Panel:**
  + Dashboard updated with SMA status card.
  + Signal text displayed in colored format.
  + Manual 'Execute Trade' button disabled if signal is HOLD.
* **Execution Logic:**
  + On BUY: USDT deducted, ETH sent via Web3.
  + On SELL: ETH sent out, USDT added.

**4. Testing Phase:**

* Verified SMA values with test price data sets.
* Confirmed signal changes correctly when crossover occurred.
* Manual trade tested for BUY and SELL, validated ETH movement.
* Checked bot toggle behavior and button conditions.

**5. Sprint Review:**

* Bot panel shown working with live ETH price feed.
* Signal logic matched expected SMA crossover conditions.
* Feedback: Add toggle to activate automatic trades in next sprint.

**6. Sprint Retrospective:**

* **Burndown Analysis:** SMA computation accuracy improved over iterations.
* **Successes:** Bot logic integrated and signals accurately computed.
* **Improvements:** Consider offloading logic to background worker or JS chart library.
* **Action Items:** Introduce auto-trade toggle in next sprint; visualize crossover using line chart.

**7. Product Backlog Refinement:**

* Added story for auto-trigger SMA bot.
* Added chart plotting for SMA50/SMA200.
* Prioritized bot enhancement tasks for Sprint 7.

#### **Sprint 7 – Alert System and UI Enhancements**

**1. Sprint Planning**

**Objective:**  
The aim of Sprint 7 was to enhance user interactivity and real-time responsiveness by introducing a price alert system. Additionally, the sprint focused on refining the dashboard UI by implementing modals, hover information, and a dark/light theme toggle.

**Sprint Backlog (Derived from Product Backlog):**

* As a user, I want to set price alerts for BTC/ETH.
* As a system, I want to monitor prices and notify users when thresholds are met.
* As a user, I want an intuitive and visually clean dashboard.

**A screenshot of a black screen

AI-generated content may be incorrect.**

**2. Sprint Backlog Tasks:**

* **Alert Form and Storage:**
  + Build alert creation form on dashboard.
  + Store alerts in alerts table with price condition, asset, and user ID.
* **Alert Trigger Logic:**
  + Check price every minute against stored conditions.
  + Trigger frontend popup if condition is met.
* **UI Enhancements:**
  + Add modals for transactions and confirmations.
  + Integrate hover descriptions for wallet and bot sections.
  + Implement dark/light toggle using JavaScript.

**3. Development Execution:**

* **Alerts Module:**
  + Built PHP script to fetch and compare current price with stored threshold.
  + Triggered alert notification and marked alert as triggered.
* **UI Improvements:**
  + Applied Bootstrap modals for cleaner trade confirmations.
  + CSS classes updated for hover hints.
  + Darkmode.js library integrated for theme switching.

**4. Testing Phase:**

* Created various alert conditions for BTC and ETH.
* Simulated rapid price changes and validated timely triggering.
* Checked dark/light toggle on mobile and desktop resolutions.

**5. Sprint Review:**

* Alerts demonstrated triggering as soon as price condition met.
* UI enhancements showcased with modals and tooltips.
* Feedback: Add delete button for alerts; remember theme preference with cookies.

**6. Sprint Retrospective:**

* **Burndown Analysis:** Lightweight sprint, UI elements took most time.
* **Successes:** Frontend became more polished and responsive.
* **Improvements:** Add user controls to manage alerts.
* **Action Items:** Build alert deletion feature; save theme choice.

**7. Product Backlog Refinement:**

* Added delete/edit alert story for Sprint 8.
* Created task for admin UI to manage user alerts.

#### **Sprint 8 – Admin Panel, Bug Fixes, and Final Testing**

**1. Sprint Planning**

**Objective:**  
Sprint 8 aimed to finalize the platform for submission. It involved implementing the admin control panel, resolving UI responsiveness issues, optimizing database queries, and performing full system testing.

**Sprint Backlog (Derived from Product Backlog):**

* As an admin, I want to view all users and their balances.
* As a system, I want to display trade history and alerts for any user.
* As a team, we want to identify and resolve bugs.

A screenshot of a phone

AI-generated content may be incorrect.

**2. Sprint Backlog Tasks:**

* **Admin Panel:**
  + Create admin dashboard interface.
  + Show list of users, balances, alert summary.
* **Bug Fixes:**
  + Address UI issues on different devices.
  + Resolve balance desync edge cases.
* **Testing and Optimization:**
  + Run full functional testing.
  + Optimize MySQL queries and PHP calls.

**3. Development Execution:**

* **Admin Tools:**
  + Dashboard with user table, clickable for trade logs.
  + Aggregated alert stats shown for each user.
* **Bug Resolution:**
  + Fixed mobile view issues in dashboard cards.
  + Corrected ETH balance refresh glitch.
* **Performance Improvements:**
  + Added SQL indexes on transactions, alerts tables.
  + Minimized API calls with caching logic.

**4. Testing Phase:**

* Cross-device browser testing (Chrome, Safari, Firefox).
* Performed 15+ trade test cycles with randomized inputs.
* Verified all alerts, orders, bots, and balances.

**5. Sprint Review:**

* Admin panel walkthrough shown to supervisor.
* All modules validated to be production-ready.
* Feedback: Prepare deployment plan and video demo.

**6. Sprint Retrospective:**

* **Burndown Analysis:** Sprint delivered final deliverables as planned.
* **Successes:** All requested features completed, no major bugs reported.
* **Improvements:** Prepare documentation in PDF and .docx formats.
* **Action Items:** Compile technical documentation and prepare final presentation.

**7. Product Backlog Closure:**

* All major stories completed.
* Backlog archived with notes for future improvements.

### **UML Introduction and Its Importance in Software Projects**

#### Introduction to UML

Unified Modeling Language (UML) is a standardized modeling language used in software engineering to visualize, specify, construct, and document the structure and behavior of software systems. UML provides a comprehensive set of diagramming techniques that facilitate clear communication among stakeholders, aid in system design, and enhance understanding of complex software systems.

#### Why Use UML in Software Projects?

* **Clarity and Communication:** UML diagrams help convey the design and architecture of the software in a clear and standardized format. This clarity supports effective communication among stakeholders, including developers, designers, project managers, and clients.
* **Documentation:** UML provides detailed documentation of system requirements, design, and functionality. This documentation is crucial for future maintenance, debugging, and upgrades.
* **Design and Analysis:** UML diagrams assist in visualizing the system’s structure and behavior, which helps in identifying potential issues early in the design phase. This supports better planning and analysis, leading to more efficient development.
* **Standardization:** UML offers a common language and set of symbols that are widely understood in the software engineering community. This standardization helps in aligning the development process with industry practices.
* **Design Patterns:** UML can represent common design patterns and solutions, facilitating their reuse and implementation in new projects.

### UML Diagrams Used in This Project

UML diagrams played a central role in the planning, design, and explanation of this crypto trading platform. The diagrams provided a blueprint for the system’s functionality, user interactions, architecture, and class relationships. Each UML diagram described below contributes to a different dimension of the platform and ensures that every stakeholder—developer, student, or evaluator—can visualize the entire workflow clearly and coherently.

#### 1. Use Case Diagram

The Use Case Diagram outlines the interactions between users (actors) and the system (use cases). It helps in identifying functional requirements and understanding how different actors interact with the system.

**Use Case Diagram Overview:**

* **Actors:**
  + User: Registers, logs in, trades crypto, sets alerts, and uses bot.
  + Admin: Manages users, reviews transactions, and monitors alerts.
  + System: Fetches prices, executes trades, stores alerts, handles bot logic.
* **Use Cases:**
  + Register/Login
  + Buy/Sell Crypto
  + Set Price Alert
  + Use SMA Bot
  + View ETH Balance
  + Manage Users (Admin)
  + Execute Trades (System)
  + Monitor Alerts (System)

**Detailed Description:**  
This diagram identifies all primary actors and their direct interactions with the system. The User actor is connected to all transactional and viewing operations such as Register, Login, Buy/Sell Crypto, Set Price Alert, View ETH Balance, and Activate the SMA Bot. The Admin oversees User Management and Transaction Monitoring. The System actor is abstract but essential, automatically managing alert checks, executing trades, and calculating SMA signals.

This diagram guided the sprint planning for authentication, trading modules, alert logic, and dashboard UI.

#### 2. Class Diagram

The Class Diagram provides a detailed view of the system’s static structure, showing the system’s classes, their attributes, methods, and relationships. It is crucial for understanding the system’s data model and interactions between different components.

**Detailed Description:**  
The class diagram defines system entities like User, Wallet, Order, Transaction, Alert, and PriceHistory. Each class contains specific attributes (e.g., user\_id, wallet\_address, balance) and methods (e.g., executeOrder(), checkAlert()). Relationships like 1:N between User and Transaction or User and Alert are shown with arrows and labels.

This diagram helped us design the MySQL schema and enforced a modular PHP architecture where each entity's logic is encapsulated in its corresponding backend script.

#### 3. Component Diagram

The Component Diagram illustrates the system’s high-level structure, showing how different software components interact with each other. It helps in understanding the system’s architecture and the dependencies between components.

**Detailed Description:**  
The component diagram illustrates the software components of the system and how they interact. Core components include the Frontend Interface, the PHP Backend Modules (e.g., LoginController, TradeEngine, BotHandler), the MySQL Database, the Ganache Blockchain Interface, the Binance Price Fetcher, and the Alert Engine.

Each module is shown as a self-contained box with well-defined input/output interactions, emphasizing the separation of concerns and the flow of data from API to database to blockchain.

### Summary

This sprint-based approach ensured continuous progress with measurable deliverables every week. It enabled us to identify blockers early, implement feature-by-feature testing, and iteratively improve the system. The project matured steadily from a simple price viewer to a full blockchain-integrated trading simulator with alerts, bots, order books, and a secure admin panel. and technology was strategically selected based on suitability for the corresponding module. Backend operations were securely handled with PHP and MySQL; blockchain features were managed via Ganache and Web3.php; live pricing was fetched using Binance API; and the system interface was responsive and informative through HTML/CSS/JS. All technologies worked in harmony, allowing the team to focus on building logic and user interaction rather than compatibility issues. These tools not only fulfilled academic requirements but also mimicked real-world systems to create a scalable, testable, and deployable crypto trading simulator.