Bilkent University
Department of Computer Engineering

Senior Design Project
T2438
Para-Meter

Final Report

22003598, Abdullah Samed Uslu, samed.uslu@ug.bilkent.edu.tr

22102566, Tuna Saygın, tuna.saygin@ug.bilkent.edu.tr

22102825, Sıla Özel, sila.ozel@ug.bilkent.edu.tr

22002756, Muti Kara, muti.kara@ug.bilkent.edu.tr

Selim Aksoy

Mert Bıçakçı, Atakan Erdem

02.05.2025

1. Introduction

This section explains our project's functionality, purpose, and goal, along with the explanations and definitions of technical terms that will be used in the report.

1.1. Purpose of the System

The financial market has seen a growing demand for accessible, data-driven investment tools that enable users to optimize their portfolios with precision and flexibility. While effective for institutional investors with specialized knowledge, retail investors find traditional portfolio optimization solutions hard to use due to their technical complexity, high costs, and reliance on conventional analysis techniques.

Para-Meter is a machine learning-powered portfolio optimization tool designed to bridge this gap. By integrating advanced data analysis, machine learning models, and a simple yet intuitive interface, Para-Meter empowers users to easily optimize their investment strategies, manage risk, and achieve their financial goals.

1.2. Design Goals

Our design goals are as follows:

Usability

- Para-Meter should have an intuitive UI that is accessible to non-technical users and has easy-to-use options for model selection, customization, and analysis.

Reliability

- High availability to ensure consistent access and minimal downtime, particularly for real-time portfolio updates and rebalancing.
- Robust security protocols to protect user data and ensure compliance with data protection regulations.

Performance

- The system must be designed to deliver low-latency portfolio optimization and simulations through efficient parallelization techniques. Performance optimization extends to model training, backtesting, and real-time analytics to ensure responsive user interactions even when processing large datasets or complex calculations.

Supportability

- Modular code structure allows easier updates, model additions, and future feature integration.

Marketability

- Para-Meter must differentiate itself from competitors by combining advanced machine learning capabilities with odd customization options and user accessibility. The platform's ability to allow users to import alternative datasets and train custom algorithms should create a unique value proposition in the market.

Flexibility

- The system should allow for extensive customization, enabling users to select from various machine learning models, integrate alternative data sources, create custom features, and define precise risk and return parameters according to their investment strategy and goals.

1.3. Definitions, Acronyms, and Abbreviations

- **AI**: Artificial Intelligence
- **API**: Application Programming Interface
- **CCPA**: California Consumer Privacy Act
- **GDPR**: General Data Protection Regulation
- **GPU**: Graphics Processing Unit
- **KVKK**: Personal Data Protection Law (Turkish data protection regulation)
- **ML**: Machine Learning
- **OAuth 2.0**: Industry-standard protocol for authorization
- **UI**: User Interface
- **UML**: Unified Modeling Language

1.4. Overview

The Para-Meter system introduces a comprehensive portfolio optimization solution that integrates machine learning capabilities with financial expertise to deliver personalized investment strategies. The platform is built on a modular architecture that separates concerns into distinct services for data retrieval, preprocessing, algorithm training, portfolio optimization, and user interface components.

This report details the transformation of our analytical model into a functional system design. It begins by outlining the high-level system architecture and describes the subsystem decomposition, hardware/software mappings, and data management strategies. The report then addresses integration aspects, including access control, control flow, and the handling of boundary conditions.

Subsequent sections elaborate on comprehensive testing strategies—both functional and non-functional—designed to validate system performance, security, stability, and reliability.

Additionally, we detail the incorporation of various engineering considerations such as public health, safety, security, global, cultural, social, environmental, and economic factors. A dedicated chapter on teamwork discusses individual contributions, collaborative practices, and leadership sharing within the project team.

Overall, this introduction sets the stage for the detailed design and testing procedures that follow, providing a clear roadmap for stakeholders to understand the purpose, structure, and guiding principles behind Para-Meter.

2. Requirements Details

2.1. Functional Requirements

User Features

- Users can create accounts, log in, manage profiles, store their preferences, save portfolios, and select features.
- A dashboard provides real-time visualization of portfolio metrics, asset allocations, and performance over time through graphs and charts.
- Users can compare different machine learning models side-by-side based on performance metrics, accuracy, and historical returns before selecting a model for portfolio optimization.

Customization

- Users can import alternative datasets (e.g., social sentiment, satellite data, economic indicators) to enhance portfolio predictions.
- Users can choose from various machine learning models, including decision trees, SVMs (Support Vector Machine), neural networks, and ensemble methods.
- Users can set return goals, risk tolerance levels (using metrics like volatility and VaR), and investment horizons.
- Users can configure portfolios as long-only or long-short, set net and gross exposure limits, and manage leverage constraints.

Performance Analysis

- The platform uses cross-validation, out-of-sample testing, and rolling window testing for robust model validation.
- Integrated stress testing and Monte Carlo simulations assess portfolio performance under various market conditions.
- The platform provides detailed performance tracking, including metrics like Sharpe ratio, drawdown, turnover, and sector or factor exposures.

Optimization and Maintenance

- Models update periodically to incorporate the latest market data, maintaining relevance over time.
- The tool preprocesses data, including normalization and feature engineering, to ensure high-quality input for models.
- Automated tuning methods (e.g., grid search and Bayesian optimization) are available to optimize model performance.

2.2. Non-Functional Requirements

Usability

- The tool should have an intuitive UI, accessible to non-technical users, with easy-to-use options for model selection, customization, and analysis.

Reliability

- High availability to ensure consistent access and minimal downtime, particularly for real-time portfolio updates and rebalancing.
- Robust security protocols to protect user data and ensure compliance with data protection regulations.

Performance

- The platform should ensure low latency for portfolio rebalancing and simulations.
- Models and calculations should be optimized to minimize processing time without sacrificing accuracy, even under heavy computational loads.

Supportability

- Modular code structure allows easier updates, model additions, and future feature integration.

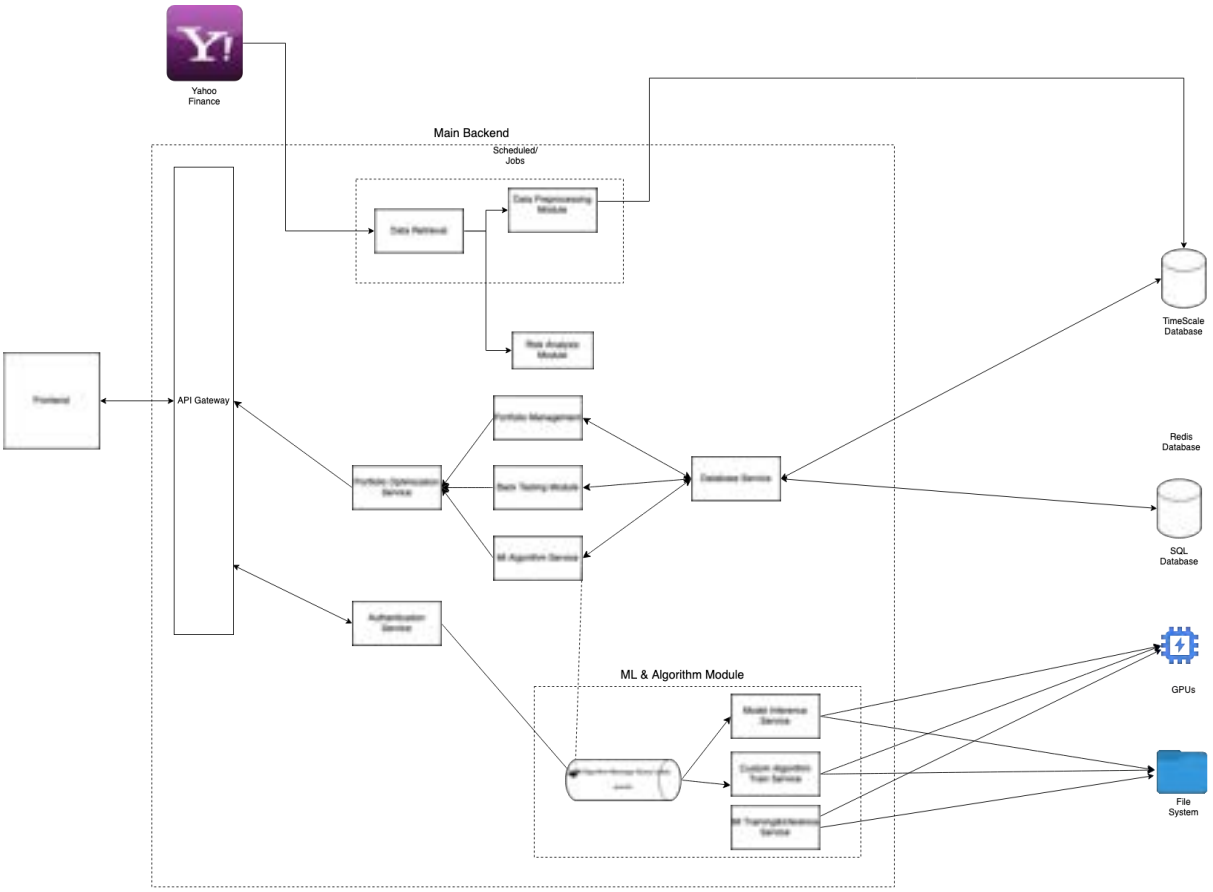
Scalability

- Support for portfolios of varying sizes, with efficient handling of large datasets and concurrent requests.
- Support for large-scale data scraping and processing.

3. Final Architecture and Design Details

3.1. System Architecture

3.1.1. High-Level Architecture



The main components of our architecture are:

Frontend Layer: This is what users see and interact with. We built it using React.js, which helps us create a responsive and interactive user interface. Users can view their portfolios, create new ones, and use different optimization strategies through this interface.

Backend Layer: This is the brain of our system. We used FastAPI to build a RESTful API that handles all the business logic. The backend processes requests from the frontend, runs portfolio optimization algorithms, manages user authentication, and communicates with the database.

Database Layer: We store our data in TimescaleDB, which is a special type of PostgreSQL database that's good at handling time-series data (like stock prices over time). It stores user information, portfolio details, stock data, and model configurations. For processing large files, we used dedicated folders and the filesystem.

External Data Sources: Our system gets stock price data from the Yahoo Finance API [1]. This gives us real-time and historical stock price information that we need for our portfolio optimization algorithms.

3.1.2 Monolithic Architecture

Even though we didn't fully separate our backend into independent microservices, we organized our code into modular components that act like microservices. This approach gives us some benefits of microservices while keeping development simpler. Our backend is divided into these main service areas:

Authentication Service: Handles user registration, login, token management, and admin access control. It uses JWT (JSON Web Tokens) for secure authentication.

User Management Service: Manages user profiles, preferences.

Portfolio Service: Handles creation, retrieval, update, and deletion of user portfolios. It also manages the stocks within each portfolio.

Stock Data Service: Fetches and caches stock data from external APIs like Yahoo Finance. It provides historical price data needed for portfolio analysis.

Predictor Service: Contains all the portfolio optimization algorithms and strategies that users can apply to their portfolios.

Backtesting Service: Allows users to test investment strategies against historical data to see how they would have performed.

Model Training Service: Provides infrastructure for training machine learning models that can be used in portfolio optimization.

These service components communicate with each other through function calls since they're all part of the same FastAPI application. Specially Model Training Service, also uses async pool listeners to keep track of changes in the database. In a future version, we could separate them into true microservices if needed.

3.1.3 Component Interactions

The components of our system interact in specific ways to deliver the functionality users need. Here are the key component interactions:

Authentication Flow:

- User enters login credentials in the React frontend
- Frontend sends credentials to the Authentication API endpoint
- Backend validates credentials and returns a JWT token
- Frontend stores the token and includes it in all future requests

Portfolio Management Flow:

- Frontend sends portfolio creation/update requests to Backend API
- Backend validates the request data and updates the database
- Database confirms changes
- Backend sends a success/failure response to Frontend

Stock Data Retrieval Flow:

- Backend requests stock data from the Yahoo Finance API
- External API returns stock price data
- Backend processes and stores data in TimescaleDB
- Frontend requests stock data from the Backend
- Backend queries the database and returns processed data to the Frontend

Portfolio Optimization Flow:

- User selects the optimization strategy in Frontend
- Frontend sends optimization request to Backend
- Backend runs the selected algorithm on the portfolio data
- The algorithm produces optimized portfolio weights
- Backend stores results and returns them to the Frontend
- Frontend displays optimization results to the user

Model Training Flow:

- The user configures model parameters in Frontend
- Frontend sends a training request to Backend
- Backend initiates the model training process
- Training progress is stored in the database
- Frontend polls for training status updates
- When complete, the model is stored for future use

For data persistence, all components interact with TimescaleDB through our Tortoise-ORM models. This gives us a clean way to handle database operations.

3.1.4 Technology Stack Overview

We chose specific technologies for each part of our system based on our requirements and team skills:

Frontend Technologies:

- **React.js:** We used React for our frontend because it lets us build a responsive single-page application with reusable components. It's also what our team had the most experience with.
- **CSS:** For styling our components, we used custom CSS. We organized our styles in separate files for each component.
- **React Router:** This helped us handle navigation between different pages in our application.

Backend Technologies:

- **FastAPI:** We chose FastAPI for our backend because it's fast, easy to use, and has built-in support for async operations. It also generates API documentation automatically [2].
- **Python 3.10:** Our backend is written in Python because it has great libraries for data analysis and machine learning.
- **Tortoise-ORM:** This is an async ORM that works well with FastAPI and makes database operations easier [3].
- **JWT:** We used JSON Web Tokens for user authentication because they're secure and don't require server-side storage.
- **NumPy and Pandas:** These libraries helped us process financial data and implement optimization algorithms.
- **PyTorch:** We used PyTorch to build and train our models. With ts utilization of CUDA, it enabled us to utilize GPUs for heavy load processes like model training and inference.

Database Technology:

- **TimescaleDB:** We chose TimescaleDB (a PostgreSQL extension) because it's specially designed for time-series data like stock prices. It can efficiently handle large volumes of timestamped data.

Deployment and Infrastructure:

- **Docker:** We containerized our application using Docker to make sure it works consistently across different environments.
- **Docker Compose:** This helped us define and run multi-container Docker applications easily.

3.1.5 Authentication System

Our authentication system is designed to be secure yet simple. Here's how it works:

Registration: Users register by providing their email, username, and password. Passwords are hashed using bcrypt before being stored in the database.

Login: Users log in with their credentials. If valid, the server generates a JWT token containing the user's ID and role.

Token Validation: For protected endpoints, the backend validates the token in the request header. If the token is valid, the request is processed; otherwise, it returns a 401 Unauthorized error.

Role-Based Access: Different user roles (regular user, admin) have different permissions. For example, only admins can access certain management endpoints.

The JWT tokens have an expiration time, after which users need to refresh. This helps maintain security while providing a good user experience.

3.2. Backend Design

3.2.1 API Design and RESTful Endpoints

Our backend API follows RESTful principles with a clear and consistent structure. We organized our endpoints by resource type to make the API easy to understand and maintain. The main endpoint groups in our application are:

1. Authentication Endpoints

- These handle user registration, login, and token management
- We implemented secure JWT-based authentication to protect user data
- Endpoints follow standard authentication patterns like login/register/logout

2. User Management Endpoints

- For managing user profiles and account settings
- Include permission checks to ensure users can only access their own data
- Admin-specific endpoints for user management

3. Stock and Portfolio Endpoints

- Endpoints for creating, viewing, updating, and deleting portfolios
- Stock data retrieval with filtering options
- Portfolio performance calculation endpoints

4. Predictor Endpoints

- For creating and managing portfolio optimization algorithms
- Running predictions on existing portfolios
- Storing and retrieving prediction results

5. Backtest Endpoints

- Testing investment strategies against historical data
- Parameter-based configuration for different backtest scenarios
- Results storage and comparison capabilities

6. Model Training Endpoints

- Creating and managing machine learning training requests
- Monitoring training status and progress
- Accessing training history and results

All our endpoints follow consistent patterns for request parameters, response formats, and error handling. We use HTTP status codes appropriately (200 for success, 400 for bad requests, 401 for unauthorized access, etc.) and structure our responses consistently using Pydantic schemas.

3.2.2 Asynchronous Processing Model

The Para-Meter backend uses an asynchronous processing model to handle multiple requests efficiently. This is especially important for operations that might take time, like retrieving financial data or running complex algorithms. Key aspects of our async implementation:

Async Request Handling

- All API endpoints use FastAPI's async capabilities
- This allows the server to handle many simultaneous connections without blocking
- Improves overall system responsiveness and throughput

Async Database Operations

- We use Tortoise-ORM, which supports asynchronous database queries
- Database operations don't block the event loop
- Allows the server to process other requests while waiting for database results

Background Tasks

- Long-running operations like ML model training run in the background
- Users receive an immediate response while processing continues
- Status polling allows the frontend to track progress

Async Data Fetching

- Financial data retrieval happens asynchronously
- Multiple stock data requests can run in parallel
- Improves performance when loading portfolio data with many stocks

This asynchronous approach helps our system remain responsive even under heavy load or when performing complex calculations. It also makes better use of server resources by not blocking threads during I/O operations.

3.2.3 Module Organization and Structure

We organized our backend code into logical modules based on functionality. This organization helps keep the code maintainable and makes it easier for team members to find and work on specific parts. The main modules in our backend are:

Core Module

- Contains fundamental components used throughout the application
- Includes settings management, security utilities, and shared resources
- Houses the core functionality that other modules depend on

Models Module

- Defines the database models and relationships
- Includes models for users, stocks, portfolios, predictors, and training data
- Uses Tortoise-ORM to define model structure and relationships

Schema Module

- Contains Pydantic models for request and response validation
- Ensures data consistency between frontend and backend
- Provides automatic documentation for API endpoints

Routes Module

- Defines all API endpoints grouped by resource type
- Implements the business logic for each endpoint
- Handles authentication, validation, and response formatting

Finance Module

- Implements financial algorithms and data processing
- Handles stock data retrieval and transformation
- Contains portfolio optimization and backtesting functionality

Model Training Module

- Manages machine learning model training workflows
- Handles background processing for training jobs
- Implements model evaluation and storage
- Can in as separate workers to maximize hardware utilization and queueing.

This modular structure follows the principle of separation of concerns, making the codebase easier to understand, test, and maintain. Each module has a clear responsibility, and dependencies between modules are well-defined.

3.3 Frontend Design

3.3.1. Component Hierarchy

We designed our React frontend using a component-based architecture to promote reusability and maintainability. Our component hierarchy is organized into several layers:

Application Container

- The top-level component that handles routing and provides the Context API providers
- Manages authentication state and routing protection

Layout Components

- Components that define the overall layout of pages
- Include navigation bar, sidebar menu, and page containers
- Ensure a consistent look and feel across the application

Page Components

- Represent full pages in the application, such as Dashboard, Portfolio View, and Model Creation
- Compose smaller components to build complete interfaces
- Handle page-specific state and data fetching

Feature Components

- Specialized components for specific functionality
- Include charts for portfolio visualization, data tables for stock information
- Implement both technical and non-technical user interfaces for different user needs

Form Components

- Components for data input with built-in validation
- Range from simple inputs to complex multi-step forms

- Dynamic forms that adapt based on user selection

Graph Components & Infoboxes

- For comprehension of the forms, which are multi-step forms, we designed graphs to make users understand what they are doing.
- Each field has a specific infoboxes that explain what the field is for non-technical users and technical users who have less experience in machine learning.

This hierarchy allows us to maintain a clear organization of our user interface and supports our goal of creating interfaces suitable for both technical and non-technical users. The component design also ensures we can reuse common elements across different parts of the application.

3.3.2. State Management Approach

For state management in our application, we designed a solution that aligns with React's principles and our application's needs:

Authentication State

- Designed a Context API-based solution for application-wide authentication
- Provides login state, user information, and authentication tokens to all components
- Handles token expiration and logout functionality

Page-Level State

- Each page component manages its own state
- State includes UI state (like open/closed panels) and data state (like loaded portfolios)
- Uses React hooks for state management

Component State

- Individual components manage their internal state
- Form components track input values and validation state
- Chart components manage visual state, like selected data points

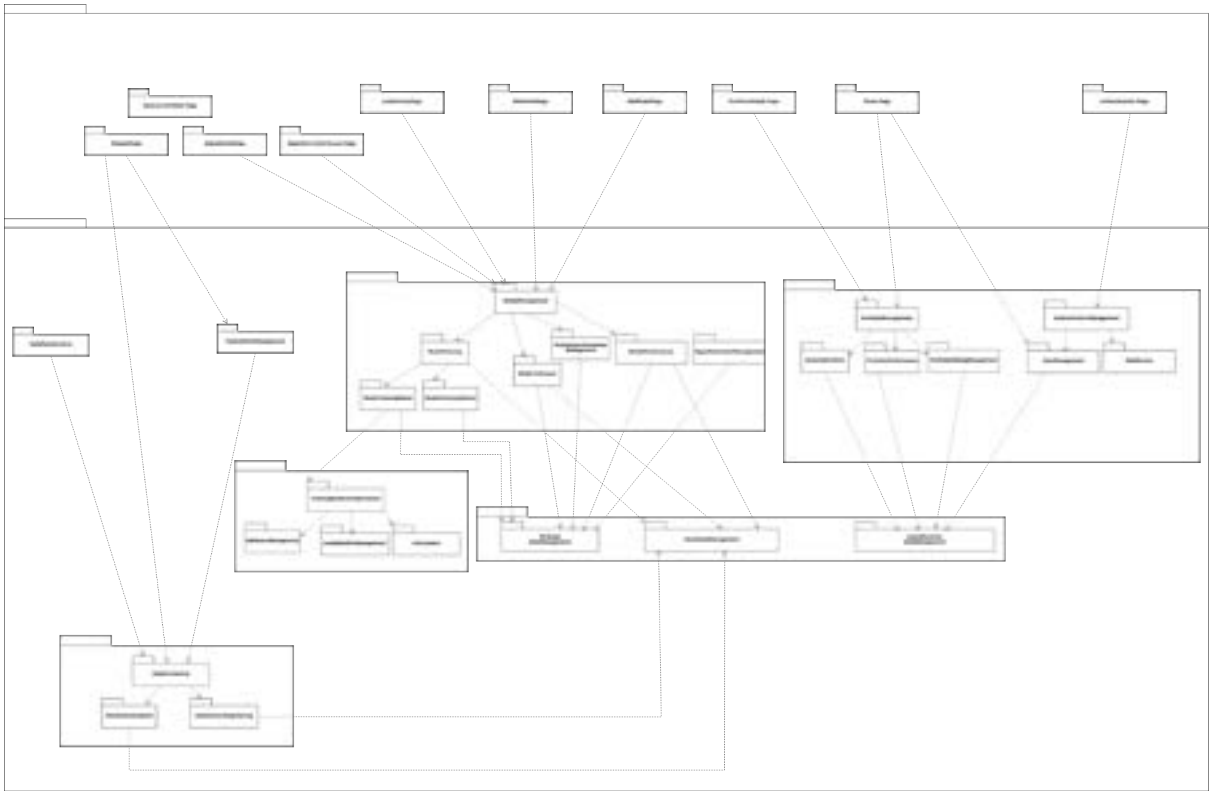
Custom Hooks

- Designed reusable hooks for common state patterns
- Data fetching hooks that handle loading, error states, and caching
- Form state hooks for complex form handling

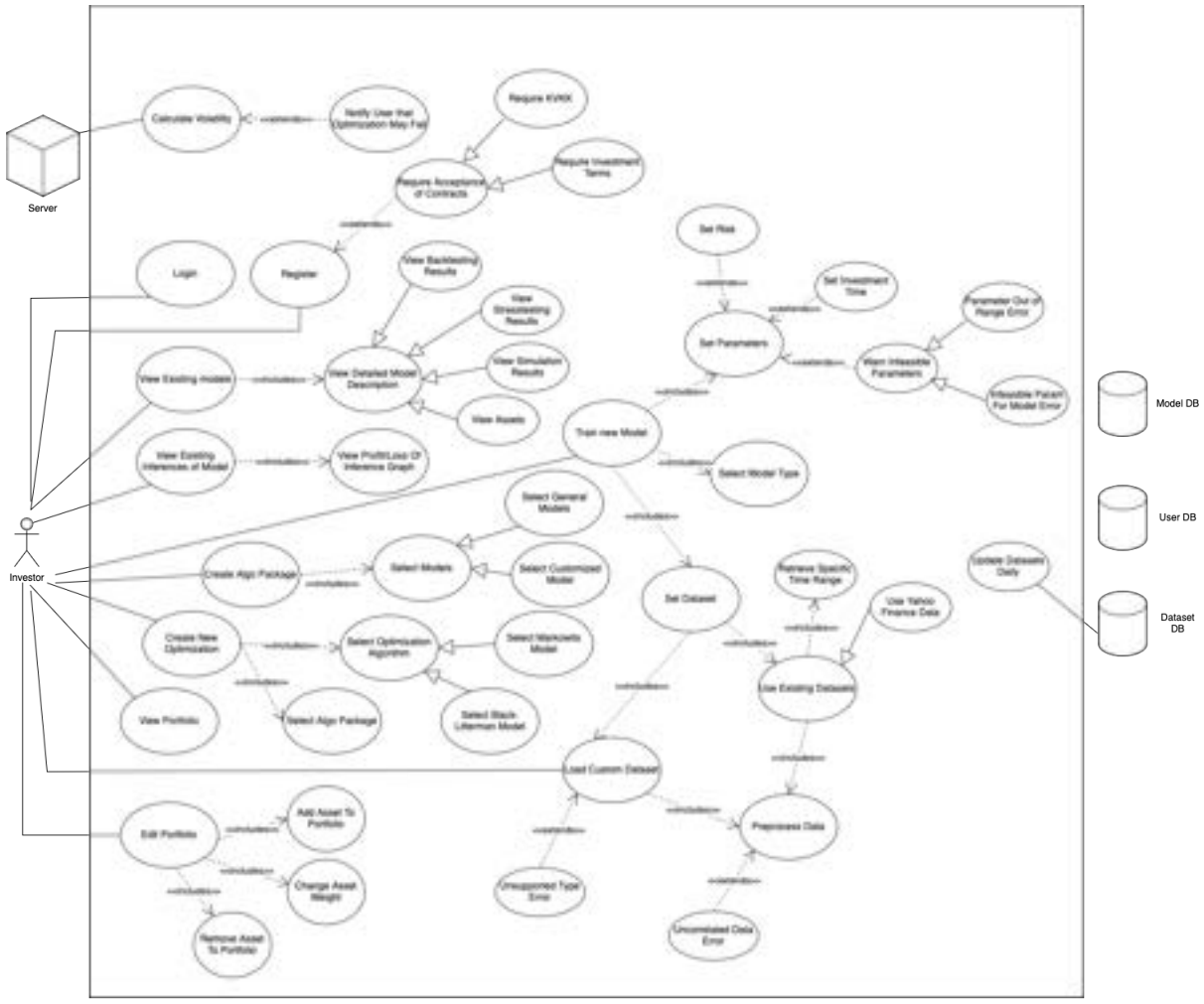
This design avoids unnecessary complexity while providing enough state management capability for our application. By using React's built-in state management features rather than third-party libraries, we kept our architecture simpler and more maintainable.

3.4. Architecture Diagrams

3.4.1. Subsystem Decomposition Diagram



3.4.2. Use Case Diagram



4. Development/Implementation Details

This section discusses the development and implementation details of our application. We will start by explaining our development methodology and will move on to the implementation details of the frontend and backend of the application.

4.1. Development Methodology

For our Para-Meter project, we set up a development environment that allowed our team to work efficiently both individually and together. We made sure everyone had the same setup to avoid the "it works on my machine" problem.

4.1.1. Development Tools

We used these main tools for development:

Visual Studio Code: Our primary code editor because it has good support for both Python and JavaScript. We shared a common set of extensions like Python, ESLint, and Prettier to maintain consistent code formatting.

Cursor IDE: Our secondary code editor because it has really good support for LLM models and AI code writing.

Claude Code: We used Claude Code Research Preview on several occasions for solving bugs and implementing some modules.

Git and GitHub: For version control and code sharing. We created a shared repository with proper branch protection rules to prevent accidental changes to the main branch.

Docker and Docker Compose: To create consistent development environments. This helped us avoid dependency issues between team members with different operating systems.

Postman: For testing API endpoints before integrating them with the frontend.

Jira: For tracking the development and task management, we used Atlassian Jira as our sprint manager.

4.1.2. Development Workflow

Our workflow followed these steps:

Local Development: Each developer worked on their assigned tasks in their local environment. We used Docker to prevent environmental issues.

Code Review: At least one other team member reviewed the other's code.

Integration Testing: After merging to the main branch, we tested to make sure everything still worked together.

End-to-End Testing: For the backend, we created a test_api_flow.py script to test our complex user flows, such as simulating a user registering and logging followed by creating a portfolio and requesting optimizations with backtest results.

Our development methodology was essential for maintaining productivity across the team and ensuring consistent behavior across different deployment stages.

4.2. Backend Implementation

4.2.1. FastAPI Application Implementation

We built our backend using FastAPI, which helped us create a fast and reliable API with automatic documentation. For the implementation, we followed a modular approach as described in the architecture section.

Our main application file connects all the different parts together. We organized the application using routers for different resource types (authentication, users, stocks, portfolios, etc.), which kept our code organized and easy to maintain. We also added CORS middleware to allow our frontend to communicate securely with the backend.

We used FastAPI's dependency injection system extensively, which allowed us to reuse common functionality like authentication across different endpoints. This made our code cleaner and more maintainable, while also making testing easier.

4.2.2. Database Implementation

For database operations, we implemented Tortoise-ORM to handle our interactions with TimescaleDB. The database connection gets initialized when the application starts up, using configuration settings from environment variables.

For the **Model Training Module,** we used Python's asyncpg library to create our pool listeners and used hand-crafted SQL queries for handling model training workers.

We developed several scripts to manage database migrations, which allowed us to update our database schema as our application evolved during development. These migration scripts were crucial for maintaining database consistency across different environments and between team members.

4.2.3. Authentication Implementation

Our authentication system consists of several key components:

- A secure password handling system using bcrypt for hashing

- JWT token generation and verification for stateless authentication
- Authentication middleware to protect restricted endpoints

When users register, their passwords are securely hashed before storage. During login, we verify the password and generate a JWT token that contains the user's ID and role. This token is then used to authenticate subsequent requests.

We implemented proper security practices like:

- Storing only hashed passwords, never plaintext
- Setting appropriate token expiration times
- Validating user permissions for protected actions

4.2.4. Financial Data Processing

Efficient processing of financial data was one of our main challenges. We developed a caching system to balance data freshness with performance:

- We first check our database for requested stock data
- If the data is missing or outdated, we fetch it from Yahoo Finance
- New data is stored in the database for future use
- External scripts update the data periodically

This approach significantly improved our application's performance and reduced the load on external data sources.

4.2.5 Machine Learning Models

We implemented two main portfolio optimization strategies in our backend:

- Mean-Variance Optimization (Markowitz) [4]
- Black Litterman [5]

For the detailed implementations of these models, we developed a model management system that handles the process from parameter input to model deployment. This system:

- Processes user-defined training parameters
- Allows users to customize model structures, such as using a custom MLP network for predicting the expected return in the Markowitz model
- Creates a training job that runs in the background
- Updates the database with progress information
- Stores the trained model for later use

A significant challenge was managing long-running model training tasks without affecting API responsiveness. We solved this by implementing a background worker system that processes training requests asynchronously.

4.2.6. Error Handling and Logging

We implemented comprehensive error handling throughout our application:

- Custom exception handlers for different error types
- Structured error responses with clear messages
- Appropriate HTTP status codes for different situations
- Detailed logging for debugging and monitoring
- Testing scripts to ensure the robustness and correctness of user flows

This approach helped us identify and fix issues during development and will make future maintenance easier.

Through our implementation process, we focused on creating clean, maintainable code with good documentation. We used Python's type hints throughout the codebase to make it more self-documenting and to catch potential errors early in the development process.

4.3. Frontend Implementation

4.3.1. React Application Setup

We implemented our frontend using React to create a single-page application. We chose React because it allowed us to build reusable components and manage state efficiently across our application. Our implementation started with a basic project structure using Vite as our build tool, which provided faster development and better performance compared to other tools.

When implementing the application, we focused on creating a clear separation between different parts of our application:

- Pages for full-screen views like Dashboard, Portfolio View, and Model Creation
- Reusable components for common UI elements
- API integration for communication with our backend by using Axios
- Context providers for global state management

We implemented proper routing using React Router, which allowed users to navigate between different sections of the application while maintaining state. We also added route protection to ensure that users must be authenticated to access protected pages.

4.3.2. Component Implementation

We implemented our components using a functional approach with React hooks. For each component, we focused on making it independent and reusable where possible. Some of our key implementation decisions included:

- Creating specialized chart components that wrap around the Recharts library to display portfolio data in a consistent way
- Building form components with built-in validation to ensure data quality
- Implementing table components for displaying financial data with features like pagination and sorting
- Creating modal components for confirmation dialogs and multi-step forms

For example, in the portfolio view implementation, we created components that fetch portfolio data and pass it to specialized visualization components. This separation made our code more maintainable and testable.

We also implemented different UIs for technical and non-technical users. For technical users, we built complex interfaces with detailed configuration options, while for non-technical users, we implemented simplified interfaces with guided workflows and clear explanations.

4.3.3. State Management Implementation

We implemented state management using a combination of approaches based on the scope of the state:

- For application-wide authentication state, we implemented a custom AuthContext using React's Context API. This provider component handles:
    - Token storage and retrieval
    - Login and logout functions
    - Automatic token expiration
- For component-specific state, we used React's useState hook. For example, in form components, we implemented state for form inputs, validation errors, and submission status.
- For more complex states in specific features, we implemented custom hooks. For example, we created hooks for portfolio data loading that handle loading states, error handling, and data caching.

We chose not to implement a full state management library like Redux because our application's state requirements weren't overly complex, and React's built-in state management tools were sufficient for our needs.

4.3.4. Chart and Data Visualization Implementation

Financial data visualization was a critical part of our frontend implementation. We implemented several types of visualizations:

- Portfolio allocation pie charts to show asset distribution
- Performance line graphs to display portfolio returns over time
- Comparison charts for backtesting results
- Stock price history charts with volume indicators

We implemented these visualizations using the Recharts library, which provided a good balance between customization options and ease of use. For each chart type, we created wrapper components that handle data formatting and styling to ensure consistent appearance throughout the application. We paid special attention to making our charts responsive so they work well on different screen sizes.

4.3.5. Form Handling Implementation

Form handling was implemented with several key features:

- Input validation with clear error messages
- Multi-step forms for complex workflows like model creation
- Dynamic form generation based on selected options
- File upload handling for CSV portfolio imports

For the technical user form, we implemented a complex form that dynamically changes based on the selected algorithm. The form displays different parameters based on the chosen algorithm type and provides helpful explanations for each parameter.

In the portfolio creation form, we implemented both manual asset entry and CSV file upload options. For manual entry, we included validation to ensure weights add up to 100% and provide immediate feedback to users.

4.3.6. Authentication Implementation

We implemented a secure authentication system in the frontend with these features:

- JWT token storage in browser session storage
- Automatic token validation and expiration handling
- Protected routes that redirect unauthenticated users to the login page
- Login/registration forms with validation

When a user logs in, our implementation stores the JWT token and includes it in all subsequent API requests. We also implemented automatic logout when the token expires.

Through our frontend implementation, we focused on creating a user-friendly interface that makes complex financial operations accessible while still providing powerful tools for advanced users. We paid special attention to error handling, loading states, and validation to ensure users always understand what's happening in the application.

5. Test Cases and Results

In this section, we will give a detailed explanation, steps, expected results, and test results of the test cases we executed in our application.

5.1. Functional Test Cases

Test ID	1	**Category**	Functional	**Severity**	Minor
Objective	Verify the login functionality works				
Expected	The user successfully logs in if the username and password fields are correct else; the login operation should fail and show the “Username and password do not match.” message.				
Steps	Successful Login: 1. Navigate to the login page 2. Enter the username correctly 3. Enter the password correctly 4. Click on the log-in button and log in to the account. Failed Login: 1. Enter the username or password incorrectly. 2. Click on the log-in button and get the correct error message.				
Date-Result	30.04.2025 - Passed.				

Test ID	2	**Category**	Functional	**Severity**	Minor
Objective	Verify that the user registration process works correctly.				
Expected	The user should be registered, and a verification email is sent to the user if the username and password are valid. Otherwise, the operation fails if the username is already taken, the password does not comply with the password restrictions, and the “Username or password invalid” message is shown.				
Steps	Successful Registration:				

	1. Navigate to the registration page. 2. Enter a valid username, email address, and password. 3. Click on the "Register" button. 4. Check the email inbox for an activation email. 5. Click on the activation link to activate the account. Failed Registration: 1. Navigate to the registration page 2. Enter an already-used username or an invalid password. 3. Check if the correct error message appears on the screen.
Date-Result	30.04.2025 - Passed.

Test ID	3	**Category**	Functional	**Severity**	Minor
Objective	Verify the forgot password functionality.				
Expected	When a valid email is provided, the system sends a password reset link to the user's registered email address, and the user successfully resets their password.				
Steps	1. Navigate to the login page and click on "Forgot Password." 2. Enter the registered email address. 3. Click on the "Submit" button. 4. Check the email inbox for a password reset link. 5. Click the reset link and set a new valid password.				
Date-Result	30.04.2025 - Passed.				

Test ID	4	**Category**	Functional	**Severity**	Critical
Objective	Verify that the unauthenticated users cannot access restricted pages via URL.				

Expected	The user will be redirected to the login page upon unauthenticated access trial with an error message.
Steps	1. Log out if you are logged in. 2. Clear the browser cache or open an incognito/private window to ensure no cached tokens or session data are present. 3. Type the URL of a restricted page and hit enter. 4. Verify that the system automatically redirects to the login page with the expected error message.
Date-Result	30.04.2025 - Passed.

Test ID	5	**Category**	Functional	**Severity**	Medium
Objective	Verify that the user can add a portfolio with valid input values.				
Expected	A new portfolio appears in the user's portfolio list when valid data is provided.				
Steps	1. Log in and navigate to the "Create Portfolio" section. 2. Enter a portfolio name and required parameters. 3. Click on the "Create Portfolio" button. 4. Verify that the new portfolio appears in the portfolio list.				
Date-Result	30.04.2025 - Passed.				

Test ID	6	**Category**	Functional	**Severity**	Critical
Objective	Verify that the user can update their portfolios.				
Expected	Changes made to the portfolio details are saved and correctly reflected in the portfolio summary.				

Steps	1. Log in and select an existing portfolio. 2. Click the "Edit Portfolio" button. 3. Modify portfolio parameters (e.g., name, asset allocation) with proper values. 4. Click "Save Portfolio" and verify that the updated information is displayed.
Date-Result	30.04.2025 - Passed.

Test ID	7	**Category**	Functional	**Severity**	Major
Objective	Verify that the user can delete their portfolios.				
Expected	The selected portfolio is permanently removed from the portfolio list after confirmation.				
Steps	1. Log in and navigate to the portfolio list. 2. Select a portfolio to delete. 3. Click the "Delete Portfolio" button and confirm the deletion when prompted. 4. Verify that the portfolio no longer appears in the list.				
Date-Result	30.04.2025 - Passed.				

Test ID	8	**Category**	Functional	**Severity**	Major
Objective	Verify that the user can see their portfolios on the dashboard. Portfolio Cards must be responsive to the different devices				
Expected	The user should see their portfolios in the “My Portfolios” section if they have portfolio(s).				

	The portfolios must be adjusted to small, medium, and big-sized screens.
Steps	1. Navigate or be redirected to the dashboard page. 2. The system displays the dashboard with the “My Portfolios” section visible. 3. Verify that the portfolio cards are correctly displayed within the “My Portfolios” section. 4. Resize the browser window or use device emulation tools to simulate small, medium, and large screen sizes. 5. Confirm that the portfolio cards adjust responsively and maintain proper layout and usability across different devices.
Date-Result	30.04.2025 - Passed.

Test ID	9	**Category**	Functional	**Severity**	Critical
Objective	Verify that the users can upload their portfolio as a CSV file, given that it is in the correct format.				
Expected	The upload should be successful if the file is in the correct format. Otherwise, the user will see an error message, “The CSV file is in incorrect format. Please upload a CSV file with expected columns.”				
Steps	1. Navigate to the “Add Portfolio” page. 2. Fill out the necessary input fields. 3. Choose to export their portfolio via CSV file upload. 4. Click the submit button to validate that the portfolio appears in the list with the correct values.				
Date-Result	30.04.2025 - Passed.				

Test ID	10	**Category**	Functional	**Severity**	Critical
Objective	Verify that the user can select from predefined machine-learning models for				

	portfolio optimization.
Expected	The options should be chosen from the form, and no errors should occur upon submission.
Steps	1. Navigate to the “Balance Portfolio” page and select the “Technical User” option. 2. Select a model from the predefined options and fill out the form with the correct inputs. 3. Click on the “Next” and verify the selected models.
Date-Result	30.04.2025 - Passed.

Test ID	11	**Category**	Functional	**Severity**	Critical
Objective	Verify that the users can upload a custom dataset to train their models with their own data.				
Expected	The upload should be successful if the dataset is in the correct file format. Otherwise, the system should show an error message.				
Steps	1. Click on the “Balance Portfolio” button under the portfolio to balance the portfolio. 2. Click on the “Technical User” option. 3. Choose among model options and click the next button. 4. In the “Select Dataset” part of the form, choose to use the custom dataset option. A file input field will appear as the user chooses this option. 5. Click the “Browse Files” button to choose a file to upload a custom dataset. 6. Verify the uploading of the dataset from the preview.				
Date-Result	30.04.2025 - Implementation not finished.				

Test ID	12	**Category**	Functional	**Severity**	Critical
Objective	Verify that the system gives information about the progress of machine-learning models that the user trains.				
Expected	The system should show an informative message on the status of the training.				
Steps	1. Click on the “Balance Portfolio” button under the portfolio to balance the portfolio. 2. Select any type of user. 3. Continue to model training. 4. Verify the training status information is informative and valid.				
Date-Result	04.05.2025 - Passed.				

Test ID	13	**Category**	Functional	**Severity**	Major
Objective	Verify that the user can abort the training process before it is completed.				
Expected	If the abortion is successful, the user will be notified via notification. If a failover happens during abortion (connection loss), the request must be persisted until abortion successfully occurs. Compute units must be restored accordingly.				
Steps	1. Click on the “Balance Portfolio” button under the portfolio to balance the portfolio. 2. Choose any type of user. 3. Proceed with the model training on the portfolio. 4. Before completing training, click on the “Abort Training” button on the “Past Activities” page. 5. Validate the behavior of the abortion process and the amount of compute units.				
Date-Result	30.04.2025 - Implementation not finished.				

Test ID	14	**Category**	Functional	**Severity**	Minor
Objective	Verify that the user cannot train a model if they do not have enough compute units threshold for that workload.				
Expected	The user is notified that they don’t have enough training compute unit threshold.				
Steps	1. Log in with an account with fewer compute units than required for the selected model workload. 2. Navigate to the “Balance Portfolio” page. 3. Select the technical knowledge level as a technical user and choose custom models. 4. Check if the necessary compute units are higher than the account holds. 5. Ensure the training process does not start and that the notification appears on the notifications page.				
Date-Result	30.04.2025 - Will not be implemented.				

Test ID	15	**Category**	Functional	**Severity**	Critical
Objective	Verify that the system uses the selected model to optimize the portfolio of the user.				
Expected	The assets' final weights and the selected models' backtesting results appear.				
Steps	1. Click on the “Balance Portfolio” button under the portfolio to balance the portfolio. 2. Select any type of user. 3. Continue with the training and optimization process. 4. Validate that the selected models’ results appear on the screen after finishing the training process.				

Date-Result	04.05.2025 - Passed.

Test ID	16	**Category**	Functional	**Severity**	Major
Objective	Verify that the users can download the optimization backtesting result as a report.				
Expected	The user should download the optimization results report. Results will consist of backtesting and will contain all metrics without anything absent that is shown on the page.				
Steps	1. Navigate to the Past Activities page. 2. Click on the backtesting report option. 3. The backtesting results are displayed on the screen. 4. Click on the “Download PDF” button. 5. The system downloads a PDF report containing all the displayed backtesting metrics.				
Date-Result	03.05.2025 - Implementation not finished.				

Test ID	17	**Category**	Functional	**Severity**	Critical
Objective	Verify that the users can purchase compute units to train their models.				
Expected	Upon successful transaction, the compute unit quota should be increased by the user's purchase amount.				
Steps	1. Navigate to the purchase compute units page from the sidebar. 2. Enter the amount of compute units to purchase. 3. The total amount will be calculated, and the user will be charged automatically according to the unit price. 4. Enter the debit/credit card details correctly. 5. Click the submit button to purchase, and the payment system will redirect you to transaction approval.				

	6. Verify that the user is redirected to the application after approval and that the compute units have increased accordingly.
Date-Result	30.04.2025 - Will not be implemented.

Test ID	18	**Category**	Functional	**Severity**	Critical
Objective	Verify that the users are charged correctly for the compute units they want.				
Expected	The amount of money taken from the user is the same as the calculated amount for the compute units they purchase.				
Steps	1. Navigate to the purchase compute units page from the sidebar. 2. Enter a valid amount of compute units. 3. The total amount will be calculated, and the user will be charged automatically according to the unit price. 4. Enter the debit/credit card details. 5. Click on the submit button to purchase, and the payment system will redirect you to transaction approval. 6. Check the bank account to verify the amount of money withdrawn.				
Date-Result	30.04.2025 - Will not be implemented.				

Test ID	19	**Category**	Functional	**Severity**	Critical
Objective	Verify that the users can see their purchase and usage history in detail.				
Expected	All activities related to consuming or purchasing compute units are listed on the Past Activities page.				
Steps	1. Apply training and purchase operations to spend or buy compute units. 2. Record these operations. 3. Navigate to the “Past Activities” page from the sidebar menu.				

	4. View all past activities performed and verify that every consumption and purchase of compute units is listed there.
Date-Result	30.04.2025 - Will not be implemented.

Test ID	20	**Category**	Functional	**Severity**	Critical
Objective	Verify that the user can update their portfolio upon the optimization recommendations.				
Expected	The user's selected portfolio is updated according to the recommended optimized portfolio.				
Steps	1. Train a model/models to optimize the portfolio. 2. After completing the optimization process, click on the update portfolio button to update the portfolio according to the recommended portfolio. 3. Validate that the portfolio is rebalanced with the suggested asset weights on the Dashboard page.				
Date-Result	30.04.2025 - Implementation not finished.				

Test ID	21	**Category**	Functional	**Severity**	Critical
Objective	Verify that the users receive a notification when the training job is finished.				
Expected	Receive a notification about the completed job.				
Steps	1. Start training to optimize the portfolio. 2. Verify the related notification appears on the notifications page when training is done.				
Date-Result	30.04.2025 - Implementation not finished.				

Test ID	22	**Category**	Functional	**Severity**	Critical
Objective	Verify that the user is not charged if the transaction for the compute unit purchase fails. Also, the user is informed in case the transaction fails.				
Expected	Receive an error message about the failure, and the user will not be charged.				
Steps	1. Navigate to the “Purchase Compute Unit” page. 2. Fill out the amount of compute units to be purchased with a valid input. 3. Enter the wrong credentials, or the card with not enough account balance. 4. Verify that no purchase is made and the error notification is sent to the user.				
Date-Result	30.04.2025 - Will not be implemented.				

Test ID	23	**Category**	Functional	**Severity**	Major
Objective	Verify that the logout functionality works correctly.				
Expected	Logged-out accounts cannot access the restricted pages without logging in again				
Steps	1. Log in with valid credentials. 2. Click on the "Logout" button in the navigation bar. 3. Verify that the user is redirected to the login page with the message “You have been logged out.” and that the user cannot access the restricted pages.				

Date-Result	30.04.2025 - Passed.

Test ID	24	**Category**	Functional	**Severity**	Minor
Objective	Verify that the homepage profit/loss plot is working correctly.				
Expected	The profit/loss plot only contains the current portfolios.				
Steps	1. Create a new portfolio. 2. Validate it appears on the profit/loss plot. 3. Delete a portfolio. 4. Validate it disappears from the profit/loss graph.				
Date-Result	30.04.2025 - Passed.				

Test ID	25	**Category**	Functional	**Severity**	Major
Objective	Verify that adding a custom model addition makes the model usable.				
Expected	● Step 3: Recently added models should be seen in the list of custom models. ● Step 5: See the recently added model in the options of selecting a model. ● Step 7: Model trains without error. ● Step 8: See the backtesting result associated with the custom model.				
Steps	1. Log in with a valid account. 2. Create a model with random valid parameters. 3. Navigate to the “My Models” page. 4. Go back to the homepage and click on the “balance portfolio” button of a portfolio. 5. Select the recently added custom models in model addition. 6. Fill in additional parameters such as risk threshold. 7. The recently added custom model is trained and inferred.				

	8. Backtesting results are given accordingly.
Date-Result	03.05.2025 - Passed.

Test ID	26	**Category**	Functional	**Severity**	Major
Objective	Verify that the user email verification works.				
Expected	Unverified users cannot log in directly and require verification. After the verification, their accounts are activated.				
Steps	1. Register a new user. 2. Try to log in to see the error message indicating the user is not verified. 3. Using the link sent with the verification email, activate the registered account. 4. Log in to see if the account is activated, and the user can see the restricted pages.				
Date-Result	30.04.2025 - Passed.				

Test ID	27	**Category**	Functional	**Severity**	Critical
Objective	Verify that the system correctly handles datasets containing missing values or invalid data entries during upload.				
Expected	If the custom dataset contains missing information, the system should warn users that their data is impartial.				
Steps	1. Upload a dataset with missing information. 2. Verify that the warning about the custom data appears.				
Date-Result	30.04.2025 - Implementation not finished.				

Test ID	28	**Category**	Functional	**Severity**	Medium
Objective	Verify that the users are notified when their portfolios need a new rebalance operation.				
Expected	The system should send a notification when the investment horizon of the portfolio has reached and rebalance is needed.				
Steps	1. Train a model/models to optimize the portfolio. 2. Rebalance the portfolio accordingly, and note the investment horizon. 3. Validate that the notification is sent when the investment horizon is reached.				
Date-Result	30.04.2025 - Implementation not finished.				

Test ID	29	**Category**	Functional	**Severity**	Medium
Objective	Verify that users can compare different portfolios in the backtesting.				
Expected	The system should allow users to see different balanced portfolios side by side and compare them.				
Steps	1. Create 2 portfolios. 2. Balance them with the models available in the system. 3. Validate that after the training, the backtest results for the old versions of those portfolios and the suggested versions of them can be comparable simultaneously.				
Date-Result	04.05.2025 - Passed.				

Test ID	30	**Category**	Functional	**Severity**	Critical
Objective	Verify that the system doesn't allow training with empty custom data or empty portfolios.				
Expected	The system must prevent the initiation of model training when either the portfolio data or custom data are empty. It should display the error message "Cannot train the model: portfolio data is empty." if the portfolio lacks stocks or "Cannot train the model: custom data is empty." if the custom data is missing.				
Steps	1. Log in as a user with sufficient compute units. 2. Create a new portfolio without adding any stocks. 3. Attempt to start model training with the empty portfolio. 4. Verify that the error "Cannot train the model: portfolio data is empty." is displayed. 5. Add valid stock data to the portfolio. 6. Attempt to start model training with an empty custom dataset. 7. Verify that the error "Cannot train the model: custom data is empty." is displayed. 8. Provide valid custom data and verify that training proceeds successfully.				
Date-Result	30.04.2025 - Passed.				

Test ID	31	**Category**	Functional	**Severity**	Critical
Objective	Verify add model functionality.				
Expected	The created model must appear on the My Models page.				
Steps	1. Navigate to the Add Model page. 2. Select a model and arrange the parameters. 3. After adding the model, validate that it appears on the My Models page with the correct parameters.				

Date-Result	30.04.2025 - Passed.

Test ID	32	**Category**	Functional	**Severity**	Critical
Objective	Verify update model functionality.				
Expected	The updated parameters must be stored and displayed on the My Models page.				
Steps	1. Navigate to the My Model page. 2. Select a model and click Edit. 3. Update the parameters of the model. 4. Click Save. 5. Verify that the parameters of the model are updated on the My Models page.				
Date-Result	30.04.2025 - Passed.				

Test ID	33	**Category**	Functional	**Severity**	Critical
Objective	Verify delete model functionality.				
Expected	The deleted model must be unlisted from the My Models page.				
Steps	1. Create a model if there is no one. 2. Navigate to the My Models page. 3. Select a model and click the Delete button. 4. Approve the delete operation. 5. Verify that the model is unlisted from the My Models page.				
Date-Result	30.04.2025 - Passed.				

Test ID	34	**Category**	Functional	**Severity**	Critical
Objective	Verify that the user can update their profile information.				
Expected	The updated information is saved and displayed correctly on the profile page.				
Steps	1. Log in and navigate to the "Profile" section. 2. Modify fields such as name, email, or contact details. 3. Click the "Save Changes" button. 4. Refresh the page and verify that the updated information is displayed.				
Date-Result	30.04.2025 - Passed.				

Test ID	35	**Category**	Functional	**Severity**	Critical
Objective	Verify that the session times out after a period of inactivity.				
Expected	The system automatically logs out the user after the defined inactivity period and prompts for re-login.				
Steps	1. Log in and remain inactive for the preset timeout period. 2. Attempt to interact with the system after the timeout. 3. Confirm the system will redirect you to the login page with a timeout message.				
Date-Result	30.04.2025 - Passed.				

Test ID	36	**Category**	Functional	**Severity**	Critical

Objective	Verify that user settings can be updated successfully.
Expected	Changes in user settings are saved and correctly reflected on the settings page.
Steps	1. Log in and navigate to "User Settings." 2. Modify settings such as notification preferences or display options. 3. Click "Save" and then refresh the page to verify changes.
Date-Result	30.04.2025 - Implementation not finished.

5.2. Non-Functional Test Cases

Test ID	37	**Category**	Non-functional	**Severity**	Critical
Objective	Verify that the training subsystem can handle multiple model training without performance loss with respect to time.				
Expected	The system scales according to the number of training jobs, and the user experience and time efficiency do not decrease dramatically.				
Steps	1. From multiple profiles, train multiple models in each profile. 2. Measure the completion time of the jobs and compare them to the completion time of the identical jobs that work in isolation. 3. Verify that no significant performance loss occurred in the system.				
Date-Result	30.04.2025 - Passed.				

Test ID	38	**Category**	Non-functional	**Severity**	Critical
Objective	Verify that the app functions properly on multiple browsers.				
Expected	The user interface does not differ much across browsers, and the				

	functionalities work as expected.
Steps	1. Log in to the application across mostly used browsers, such as Chrome, Safari, and Firefox. 2. Try out the main functionalities of the application on different browsers. 3. Validate that there is no problem with the interface or the functionality.
Date-Result	30.04.2025 - Passed.

Test ID	39	**Category**	Non-functional	**Severity**	Critical
Objective	Verify that the portfolio optimization is completed within the estimated timeframe.				
Expected	The optimized portfolio results are calculated within the estimated time after the process is finished.				
Steps	1. Start a training process to optimize the portfolio. 2. Note the reported expected time. 3. Measure the real-time to finish the training process. 4. Validate that the realized time is no more than the threshold (e.g. $1.1 \times estimated\ time$).				
Date-Result	03.05.2025 - Passed.				

Test ID	40	**Category**	Non-Functional	**Severity**	Major
Objective	Verify that the system supports training requests more than the available GPUs.				
Expected	● Step 2: The system should accept training requests that exceed available GPU clusters.				

	• Step 3,4: For each running job, the status should be either pending or running.
Steps	1. From a client (e.g., using a curl command), query the training scheduling service to retrieve the current resource status, specifically the number of available GPUs. 2. Submit training requests equal to three times the number of existing GPUs in the system. 3. Using Kubernetes (or the orchestration platform), check the status of each training job submitted. 4. Verify that each job's status is either pending (queued for execution) or running (actively using GPU resources). 5. Continuously monitor the status of the training jobs until all jobs have either been completed or transitioned out of the pending state. 6. Validate that the system correctly manages the over-subscription of GPU resources without causing failures or resource conflicts.
Date-Result	30.04.2025 - Passed.

Test ID	41	**Category**	Security	**Severity**	Critical
Objective	Verify that the system is resilient to the SQL injection attacks.				
Expected	If an ill-intended actor tries to inject SQL queries, the system should not execute those queries.				
Steps	1. Try all user input areas, such as editing profiles, creating portfolios, etc. 2. Verify that SQL injection attacks are unsuccessful.				
Date-Result	30.04.2025 - Passed.				

Test ID	42	**Category**	Non-functional	**Severity**	Medium

Objective	Verify the scalability and stability of the backend under heavy concurrent usage.
Expected	The system should be able to handle many users using the website concurrently.
Steps	1. Write a script that simulates various user's behavior. 2. Run the script in large quantities in parallel. 3. Verify that the system works fine by performance monitoring.
Date-Result	30.04.2025 - Failed.

Test ID	43	**Category**	Non-functional	**Severity**	Medium
Objective	Verify that the market data update time doesn't introduce any inconsistency in rebalancing and backtesting.				
Expected	The system should use the updated data available only after the update process.				
Steps	1. Before the market data update time, run backtesting and portfolio optimization tasks. 2. During market data update time, run backtesting and portfolio optimization tasks and validate that results are the same and correct. 3. After the market data update time, run backtesting and portfolio optimization tasks to validate that the results are updated with the new market data.				
Date-Result	30.04.2025 - Passed.				

Test ID	44	**Category**	Non-functional	**Severity**	Critical
Objective	Verify the system is resilient against brute force attacks for stealing				

	passwords and accounts.
Expected	The system should prevent brute force attacks by presenting a cooldown or email verification.
Steps	1. Register a user account and verify it. 2. Log out from that user account. 3. Try the wrong passwords multiple times. 4. Validate that the system blocks this brute force approach after the threshold times attempts.
Date-Result	30.04.2025 - Passed.

Test ID	45	**Category**	Non-functional	**Severity**	Critical
Objective	Verify that the compute unit estimation system works accurately within an error rate.				
Expected	The system should be able to estimate the computing cost of training beforehand within a decided error range.				
Steps	1. Create different training tasks with different combinations. 2. Compare the estimated computing unit usage and realized computing unit usage. 3. Verify that the difference is within the reasonable error range.				
Date-Result	30.04.2025 - Will not be implemented.				

Test ID	46	**Category**	Non-functional	**Severity**	Critical
Objective	Verify that the system updates historical market data accordingly during stock splits.				

Expected	The historical stock prices and calculated values must be adjusted according to the stock split.
Steps	1. Upload test data, including stock split. 2. Validate the handling of the stock split via portfolio optimizations and backtesting results.
Date-Result	30.04.2025 - Failed.

Test ID	47	**Category**	Non-functional	**Severity**	Critical
Objective	Verify that the system maintains market data integrity during market data updates.				
Expected	The system should be resilient to crashes during market data updates. If a crash occurs, the system must detect missing or incomplete data and automatically resume the update process.				
Steps	1. Prepare a historical market dataset with the last week's data intentionally removed. 2. Start the market data update process. 3. Inject simulated accurate data for each missing day of the last week sequentially. 4. Intentionally crash the application during the update process. 5. Restart the application. 6. Verify that the system automatically detects the missing data and resumes the update process to complete the dataset.				
Date-Result	30.04.2025 - Passed.				

Test ID	48	**Category**	Non-functional	**Severity**	Critical
Objective	Verify that multiple concurrent logins from different devices are handled correctly.				

Expected	The system allows concurrent logins while maintaining session integrity and data consistency.
Steps	1. Log in on one device with valid credentials. 2. Log in on a second device using the same account. 3. Verify that both sessions remain active and data is synchronized between devices.
Date-Result	30.04.2025 - Passed.

Test ID	49	**Category**	Non-functional	**Severity**	Critical
Objective	Verify that file uploads respect the maximum size limit.				
Expected	The system rejects files that exceed the specified size limit with an appropriate error message.				
Steps	1. Navigate to the "Balance Portfolio" section as a Technical User. 2. During the training steps, choose to upload a custom dataset. 3. Select a file larger than the allowed limit. 4. Click "Upload" and observe the error message.				
Date-Result	30.04.2025 - Passed.				

Test ID	50	**Category**	Security	**Severity**	Critical
Objective	Verify that sensitive data is encrypted in storage.				
Expected	Data such as passwords and personal information are stored in an encrypted format.				
Steps	1. Create or update a user account with sensitive data. 2. Access the backend storage (using a secure tool or test interface).				

	3. Confirm that sensitive fields are not stored in plain text.
Date-Result	30.04.2025 - Passed.

6. Maintenance Plan and Details

For our Para-Meter project, we currently do not plan to maintain the application for public use. This decision is mainly because of our limited financial resources and time constraints as students. While we created a working application with many features, we acknowledge that a financial application requires ongoing support and updates to be useful for real users.

We have several parts of the application that we could not fully implement due to our project timeline. These include some advanced machine learning models and additional portfolio optimization strategies that we planned in our initial design. We might complete these features in the future.

7. Other Project Elements

7.1. Consideration of Various Factors in Engineering Design

This chapter details the key constraints that have influenced our design decisions and outlines the standards we follow to ensure a robust, secure, and high-quality system.

7.1.1. Constraints

7.1.1.1. Implementation Constraints

- Accessing reliable and high-quality financial data is important for accurate modeling and optimization. However, many financial APIs have rate limits or limited historical data. Also, the data might be inaccurate or incomplete. These might hinder the accuracy of the results.
- Training machine learning models and running optimization algorithms require significant computational resources, including processing power and memory. The lack of access to high-performance hardware, such as GPUs or distributed computing systems, can slow down model training and testing, particularly when working with large datasets or performing hyperparameter tuning.
- The limited development timeline of 8 months can restrict the scope of development and testing. This limited duration requires careful prioritization of features and functionalities to ensure the delivery of a functional, high-quality system within the allotted time.

7.1.1.2. Financial Constraints

- The development of a machine learning-based portfolio optimization tool faces significant financial limitations. The initial budget for the project restricts access to advanced cloud infrastructure and premium financial datasets. Cloud services, which are essential for computation and storage, incur substantial costs, especially when utilizing high-performance configurations like GPUs or distributed systems. Similarly, high-quality financial datasets, often critical for accurate modeling, can be expensive and may exceed budgetary limits. These constraints could affect the accuracy and speed of the system's development.
- Operational costs are another concern, as the platform aims to remain affordable for retail investors who are sensitive to pricing. Maintaining low operational expenses is important to ensure the tool is accessible, which may necessitate adopting a freemium or tiered pricing model. Cost-effective development practices, such as using open-source tools and optimizing resource usage, will be critical to staying within budget.

7.1.1.3. Ethical Constraints

- A primary ethical concern is user data privacy. Compliance with data protection regulations such as GDPR (General Data Protection Regulation), CCPA (California Consumer Privacy Act), and KVKK (Personal Data Protection Law) is crucial since the tool handles sensitive financial and personal information.
- Avoiding algorithmic bias is one of the most important ethical constraints for Para-Meter. Machine learning models trained on financial data may unintentionally favor certain asset classes, industries, or demographic groups, leading to biased recommendations. Ensuring fairness in portfolio suggestions requires diverse, high-quality training datasets, which can be hard to detect and obtain.

7.1.1.4. Public Health, Safety, and Welfare Factors

- **Impact Level:** 3/10
  Although the system does not have a direct impact on public health and safety, it does contribute to broader welfare by empowering users to make informed financial decisions.
- **Financial Well-being:** By enabling users to optimize their investment portfolios and better manage risk, the system indirectly contributes to the financial well-being of individuals. As people become more informed and make smarter investment choices, their overall financial security could improve, supporting their long-term welfare.
- **Accessibility to Financial Tools:** By providing a user-friendly and affordable tool, the system helps democratize access to financial optimization tools that were previously only available to institutional investors. This can improve the financial outcomes of individuals who might otherwise not have had access to advanced investment strategies.

7.1.1.5. Global and Cultural Factors

- **Impact Level:** 4/10
  Global and cultural factors play an important role in shaping how the system is used across different regions and by diverse groups of investors.
- **Market Accessibility:** The system will focus on ensuring seamless access to the most widely traded stocks across different regions. It must consider any regional restrictions, such as access to foreign markets, to allow investors from any region to build diversified portfolios without being limited to local stocks.

7.1.1.6. Economic Factors

- **Impact Level:** 10/10
  The system must account for various economic considerations to ensure it is accessible, efficient, and sustainable in the long term.
- **Affordability for Retail Investors:** The platform must remain affordable, particularly for retail investors, who are typically sensitive to costs. This necessitates a pricing model that balances feature availability with affordability. A freemium or tiered pricing model would allow users to access essential functionalities while offering premium features for more advanced users or institutional clients.
- **Operational Costs and Cloud Infrastructure:** High-performance cloud computing and financial data APIs are crucial for training machine learning models and executing real-time optimization. However, using advanced cloud infrastructure such as GPUs, distributed computing systems, and premium data sources can significantly increase costs. The system must find a cost-effective solution that balances computational needs with budgetary constraints.
- **Economic Sensitivity of Market Conditions:** The platform must adjust its models to reflect changing economic conditions, such as market volatility, interest rates, and inflation. These economic factors can impact the performance of investment portfolios and must be integrated into the system's predictive models to ensure that portfolio optimizations remain relevant under different economic climates.

	Effect level	Effect
Public health	0	N/A - The project has no direct impact on public health.
Public safety	0	N/A - The project has no direct impact on public safety.

Public welfare	3	The project indirectly contributes to financial well-being by helping users optimize their investments.
Global Factors	4	The project ensures accessibility to global markets and compliance with international regulations.
Cultural factors	0	N/A - The project has no direct impact on cultural factors.
Social factors	5	The project democratizes access to advanced investment tools, fostering inclusivity and financial literacy.
Environmental factors	1	Minimal impact due to hosting infrastructure and cloud service usage.
Economic factors	10	The project heavily focuses on economic considerations, especially affordability for retail investors.

Table 1. Factors that can affect analysis and design.

7.1.2. Standards

To ensure that Para-Meter is developed with best practices and meets appropriate requirements, we followed several recognized standards where applicable to our project scope:

IEEE 830: This standard provides guidance for writing clear requirements specifications. We used it as a reference when documenting our functional and non-functional requirements, though we adapted it to fit our project's scale.

UML 2.5.1: We used elements of the Unified Modeling Language (UML) to create diagrams that helped us design our system. This included basic use-case diagrams and class diagrams that guided our implementation.

JWT Authentication: Instead of implementing the full OAuth 2.0 protocol, we used JSON Web Tokens (JWT) for authentication, which was more appropriate for our project scope. This provided us with secure, stateless authentication while being simpler to implement.

Basic Security Practices: While we didn't implement a complete ISO/IEC 27001 information security management system, we followed basic security practices, including:

- Password hashing using BCrypt
- HTTPS for data transmission
- Input validation to prevent common attacks
- Proper authorization checks for API endpoints

Password Security: We followed key recommendations from NIST SP 800-63B regarding password storage:

- Passwords are hashed using the BCrypt algorithm
- We implemented minimum password strength requirements
- We don't store passwords in plain text

Data Privacy Considerations: While our project doesn't currently have plans for public deployment requiring full GDPR compliance, we designed our data handling with privacy in mind:

- We only collect necessary user information
- Personal data is stored securely
- We implemented user data access controls

As a student project, our implementation of these standards was focused on learning and applying key principles rather than achieving formal certification or complete compliance. If Para-Meter were to be developed for production use in the future, a more comprehensive implementation of these standards would be necessary.

7.2. Ethics and Professional Responsibilities

This section discusses the ethical and professional responsibilities we acquired during this project.

7.2.1. Ethical Responsibilities

Terms and Conditions: Due to the nature of our project, we deal with financial data. So, there is always a risk of losing or gaining money if the users take our application’s suggestion. We inform the users about this risk, and they have to accept these terms and take responsibility for their financial decisions.

Ensuring Custom Dataset Privacy: We provide users the flexibility of uploading their own datasets. We do not use their dataset for our own models or other users’ models. The dataset

will be stored during the training, and it will be deleted and not held for future use to ensure that the users ' data is not used for other purposes.

Transparency: One of the main goals of Para-Meter was to create a more transparent application than the ones already successful in the market. The issue with already existing solutions is that they are not transparent in terms of what happens on the backend while optimizing a portfolio. This decreases trust in those who are not tech-savvy. So, we kept that in mind while implementing our application.

7.2.2. Professional Responsibilities

Throughout this project, we upheld several key professional responsibilities:

Technical Accuracy and Transparency: We ensured that our financial prediction models were built with statistical transparency. All assumptions, limitations, and performance metrics are clearly documented to prevent misrepresentation of the system's capabilities.

Continuous Learning and Improvement: We maintained a commitment to staying current with best practices in financial modeling and machine learning. This included regular review of relevant literature and incorporating feedback during the development.

Quality Assurance and Testing: We tested all the features we implemented to validate our models and application functionality.

Documentation and Knowledge Transfer: We created comprehensive documentation to facilitate future maintenance and development of the system, ensuring sustainability beyond our direct involvement.1

7.3. Teamwork Details

This section describes the teamwork approach during the project, outlining how each member contributed to the project, fostered a collaborative and inclusive work environment, and took lead roles in different parts of the project.

7.3.1. Contributing and functioning effectively on the team to establish goals, plan tasks, and meet objectives

Each team member played a vital role in ensuring that the project advanced efficiently and effectively:

- **Tuna Saygın**
  He took charge of the ML module and contributed to the overall architectural design. His contributions were central to integrating machine learning components with the rest of the system. His role ensured that the architecture remained modular and scalable, supporting the project’s advanced functionalities. Additionally, he

implemented some components on the frontend and helped with the frontend-backend integration.

- **Sıla Özel:**
  She was responsible for designing and implementing a user-friendly frontend. She focused on ensuring that the interface was both efficient and accessible, contributing to a seamless user experience. Her dedication to UI/UX design ensured that the system remained intuitive, which is critical for the target users. She also took the lead for frontend-backend integration.
- **Abdullah Samed Uslu**
  He managed the JIRA system to track tasks and deadlines, ensuring that all project activities were clearly documented and scheduled. In addition, he contributed significantly to portfolio optimization techniques and managed the financial data aspect of the system. This contribution helped shape the core analytical functions and maintain a structured approach to data management.
- **Muti Kara:**
  He handled the backend development, including communication protocols and database management. He created the backbone of the backend system in terms of both high-level architecture and code. His work ensured that data flowed smoothly between the system components and that backend services were efficient and robust. His contributions to backend systems were key in maintaining system performance and reliability.

Overall, each member not only focused on their specific areas of responsibility but also collaborated across disciplines, providing support and insights that enriched the overall quality of the project.

7.3.2. Helping create a collaborative and inclusive environment

Our team prioritized creating an open, collaborative, and inclusive environment through several key practices:

- **Weekly Meetings:**
  We held regular weekly meetings to discuss progress, address challenges, and plan upcoming tasks. These meetings provided a dedicated space for every team member to share updates, ask questions, and contribute ideas. They were essential for keeping everyone aligned and ensuring transparency across all aspects of the project.

- **Open Communication:**
  In addition to weekly meetings, we maintained open communication channels through messaging apps. This constant dialogue helped resolve issues quickly and fostered a sense of camaraderie among team members.

- **Mutual Support and Peer Learning:**
  Each member willingly shared their expertise and assisted colleagues when needed. For example, when technical challenges emerged in the frontend or backend, team members collaboratively discussed solutions and exchanged knowledge. This peer learning approach enhanced our collective skills and improved project outcomes.

- **Inclusive Decision-Making:**
  Major decisions, such as tool selection, system architecture, and project milestones, were discussed openly with input from all members. This inclusive process ensured that diverse perspectives were considered, resulting in well-rounded and effective solutions.

7.3.3. Taking a lead role and sharing leadership on the team

Leadership within our team was shared based on individual strengths, allowing everyone to take charge of their respective areas while supporting each other:

- **Tuna Saygın:**
  He took the lead in the ML module and architectural design efforts. His initiative in organizing discussions around system scalability and modularity helped ensure that our ML components were seamlessly integrated and future-proofed.
- **Sıla Özel:**
  She led the front-end development, guiding design choices and implementation strategies to ensure an intuitive user experience. Her proactive approach to UI/UX reviews significantly shaped our user interface.
- **Abdullah Samed Uslu:**
  He frequently led discussions on task management and portfolio optimization, setting clear objectives and ensuring smooth progress through effective JIRA management.
- **Muti Kara:**
  He led backend development, coordinating efforts to establish robust communication protocols and manage database operations. His leadership was pivotal in integrating backend components with the overall system.

In every instance, leadership was not confined to one person but was a shared responsibility. Each member stepped up to lead in their domain while actively participating in group decisions, contributing to a sense of collective ownership, and ensuring the success of the project.

7.3.4. Meeting objectives

This subsection compares the goals defined in the **Analysis Report** (Work Packages 1–8) with the actual outcomes at project completion on **May 2, 2025**.

WP	Objective	Planned Deadline	Finish Time	Status	Notes

1	Creating a scalable and modular system design while adhering to engineering standards.	10.02.2025	12.02.2025	Met.	We designed and created UML diagrams, high-level architecture sketches, and sequence diagrams for a scalable backend.
2	Setting up the data pipeline and ensuring clean, accurate, and reliable input for ML models.	15.02.2025	14.02.2025	Met.	We successfully utilized external APIs such as Yahoo Finance and applied imputing techniques for missing data.
3	Training and validating machine learning models for portfolio optimization.	31.03.2025	14.04.2025	Partially Met.	While we can train machine learning models on custom data, we still get high loss values on real stock market data.
4	Understanding and implementing financial mathematical models such as risk-return optimization, Value at Risk (VaR), Sharpe ratio calculations, and portfolio rebalancing strategies, ensuring alignment with modern financial theories..	15.03.2025	30.04.2025	Partially Met.	We are supporting only a limited number of models, which are Markowitz and Black Litterman.
5	Developing an intuitive and accessible interface for users.	28.02.2025	24.02.2025	Met.	We developed an intuitive UI with all of the main pages included. The development of core components of the front-end had finished by the due date.

6	Implementing backend services and APIs for seamless data and model handling.	31.03.2025	25.03.2025	Met.	The backend systems for auth management and basic CRUD of stocks and portfolios have been completed. The backbone for training and inferencing AI models and backtesting utilities had been implemented by the due date.
7	Integrating system modules and ensuring seamless operation through comprehensive testing.	01.05.2025	30.04.2025	Partially Met.	In the end, we managed to merge the frontend and backend, but there wasn't enough time for us to test comprehensively.
8	Creating comprehensive documentation and presenting the project.	01.05.2025	02.05.2025	Met.	We have completed this report and prepared our application for the demo presentation.

7.4 New Knowledge Acquired and Applied

Throughout the Para-Meter project, our team learned many new skills and gained valuable knowledge in different areas. This learning process was an important part of our project, and we used several strategies to gain and apply new knowledge.

Technical Knowledge Gained

Before starting this project, most of us had only basic experience with web development. During the project, we learned:

- **FastAPI Framework**: Most of us haven't used FastAPI before. We learned how to create RESTful APIs with automatic documentation and how to use its async features for better performance.
- **React Development**: We improved our React skills, especially with hooks and context API. We learned how to create reusable components and manage the state effectively.
- **TimescaleDB**: We had no significant experience with time-series databases before. We learned how to design database schemas for financial data and how to optimize queries for time-series data.

- **Docker and Containerization**: We learned how to containerize our application using Docker, which helped us maintain consistent environments across our team.
- **Authentication with JWT**: We learned how to implement secure token-based authentication and how to protect API endpoints.

Financial Knowledge Gained

Most of us started with limited knowledge about portfolio optimization and financial algorithms:

- **Portfolio Theory**: We learned about Modern Portfolio Theory, efficient frontiers, and how to balance risk and return.
- **Optimization Algorithms**: We studied different methods for portfolio optimization, like Mean-Variance Optimization and Black Litterman.
- **Financial Data Analysis**: We learned techniques for analyzing financial time-series data and how to calculate important metrics.

Learning Strategies

To gain these new skills and knowledge, we used several learning strategies:

- **Pair Programming**: We often worked in pairs, which helped us share knowledge quickly. When one person knew something the other didn't, they could teach it directly.
- **Weekly Knowledge Sharing**: We had weekly meetings where team members would present what they learned about specific topics to the rest of the team.
- **Books and Articles:**
    - **Modern Portfolio Theory and Investment Analysis:** Used to develop portfolio optimization models.
    - **Machine Learning and Data Science:** Used to develop machine learning models.
    - **Mathematics for Finance: An Introduction to Financial Engineering:** Used to develop portfolio optimization models.
- **University Courses:** Our group members took some courses to apply their outcomes to our project:
    - **CS464 - Introduction to Machine Learning:** Helped to understand the basics of machine learning.
    - **IE440 - Introduction to Financial Engineering:** Helped to understand the basics of portfolio optimization.
- **Expert Opinions:** We had a meeting with Prof. Hüseyin Çağrı Sağlam, who is an economics professor at Bilkent University. He gave us advice on financial markets and what our target users might expect from us. Since we target people who know about finance but don't know about coding, his comments on user expectations were quite insightful. He also recommended a few economics books, which contributed to our sources. Humorously, he also suggested that we should forget about the stock market and pivot our app to betting on the Trabzonspor Football Club.

Application of Knowledge

We applied our new knowledge directly to the project:

- **Backend:** We used our FastAPI knowledge to create efficient API endpoints that handle financial data processing and user authentication.
- **Frontend:** We applied our React knowledge to build an interactive frontend with reusable components and proper state management.
- **Portfolio Optimization:** Our financial knowledge helped us implement accurate portfolio optimization algorithms and properly visualize portfolio performance.
- **Data Management:** We used our TimescaleDB knowledge to design a database schema that efficiently stores and queries time-series stock price data.
- **Machine Learning:** One of our biggest learning challenges was creating the predictive engine for our application. We started with simple linear regression as a baseline to predict asset returns. As we learned more, we moved to more complex models:
    - We implemented multi-layer perceptrons (MLPs) with techniques like batch normalization and dropout to prevent overfitting
    - We built recurrent neural networks (RNNs) to better capture patterns in stock price time series
    - We experimented with ensemble methods like random forests and gradient-boosted trees to make our predictions more robust against market noise
    - We designed our code to allow easy swapping of algorithms and automatic hyperparameter tuning

This project pushed us to learn many new skills quickly and apply them to solve real problems. The knowledge we gained will be valuable for our future careers, especially in fintech or web development. Even though we don't plan to maintain Para-Meter for public use right now, the experience of building it taught us how to learn and apply new technologies effectively.

8. Conclusion and Future Work

8.1 Project Summary

Our Para-Meter portfolio optimization platform successfully meets the main goals we set at the beginning of the project. We created a web application that helps both technical and non-technical users optimize their investment portfolios using machine learning algorithms. The system allows users to create portfolios, select stocks, implement optimization strategies, and visualize performance.

We successfully implemented key features, including:

- User authentication and account management
- Portfolio creation and management
- Stock data retrieval and storage

- Multiple portfolio optimization algorithms
- Machine learning prediction models
- Backtesting capabilities
- Different interfaces for technical and non-technical users

The application uses modern technologies like FastAPI for the backend, React for the frontend, and TimescaleDB for storing financial data. We designed the system to be modular and flexible, which made it easier for our team to work on different parts at the same time.

8.2 Lessons Learned

During this project, we learned many valuable lessons about software development and teamwork:

Planning is important: Having a clear architecture design from the beginning helped us avoid major changes later in the project.

Start simple, then add complexity: Beginning with simple steps before adding more complex features helped us make steady progress. Also, instead of directly designing the complete project, an iterative approach enables us to see problems and errors in our designs on the go.

Regular communication is key: Our weekly meetings and daily check-ins helped us identify and solve problems quickly.

Learning new technologies takes time: We sometimes underestimate how long it would take to learn new technologies and concepts.

Financial domain knowledge matters: Understanding the financial concepts behind our algorithms was just as important as coding skills.

8.3 Future Work

While we're not planning to maintain Para-Meter for public use right now, we identified several areas for potential future improvement:

Additional Optimization Strategies: We could add more portfolio optimization methods like Risk-Parity and Factor-Based optimization.

Advanced Machine Learning Models: We started implementing some advanced ML models, but there's room to add more sophisticated approaches like transformers or reinforcement learning for portfolio management.

Real-Time Data: Currently, our system uses end-of-day stock data. Adding real-time data would make it more useful for active traders.

Performance Improvements: The portfolio optimization algorithms could be made faster, especially for portfolios with many stocks.

If we decide to continue this project in the future, these enhancements would be valuable additions to Para-Meter. However, even in its current state, the project demonstrates a functional portfolio optimization platform and represents the significant knowledge and skills we gained throughout its development.

9. Glossary

RNN: A recurrent neural network that processes sequences by maintaining and updating a hidden state to capture temporal dependencies in data.

Multi-Layer Perceptron (MLP): A feed-forward neural network composed of multiple fully connected layers that learns complex, non-linear mappings between inputs and outputs.

Linear Regression: A statistical method that models the relationship between a dependent variable and one or more independent variables by fitting a linear equation to observed data.

Ensemble Methods: Techniques that combine predictions from multiple models, such as bagging, boosting, or stacking, to improve overall accuracy and robustness.

Machine Learning (ML): A subset of artificial intelligence that enables systems to learn from data

Portfolio Optimization: The process of selecting the optimal allocation of assets in a portfolio

Mean-Variance Optimization (MVO): A mathematical framework for assembling a portfolio of assets

FastAPI: A modern, fast web framework for building APIs with Python

React: A JavaScript library for building user interfaces

TimescaleDB: A time-series database built on PostgreSQL

10. References

[1] "Yahoo Finance API Documentation," Yahoo Finance. [Online]. Available: https://finance.yahoo.com/. [Accessed: Apr. 12, 2025].

[2] "FastAPI," FastAPI. [Online]. Available: https://fastapi.tiangolo.com/. [Accessed: Apr. 25, 2025].

[3] "Tortoise-ORM," Tortoise-ORM. [Online]. Available: https://tortoise-orm.readthedocs.io/. [Accessed: Apr. 10, 2025].

[4] H. Markowitz, "Portfolio Selection," The Journal of Finance, vol. 7, no. 1, pp. 77-91, Mar. 1952.

[5] F. Black and R. Litterman, "Global Portfolio Optimization," Financial Analysts Journal, vol. 48, no. 5, pp. 28-43, Sep. 1992.