Implementation Document

for

PHInance

Version 1.00

Prepared by

Group 15: **Group Name:** Team FEINance

Anisha Srivastava	220149	anishas22@iitk.ac.in
Dilbar Singh Lamba	220368	dilbars22@iitk.ac.in
Pranshu Thirani	220801	pranshut22@iitk.ac.in
Sameer Yadav	220950	sameer22@iitk.ac.in
Sangam Gupta	220961	sangamg22@iitk.ac.in
Aruz Awasthi	230209	aruza23@iitk.ac.in
Anirudh	220146	anirudhm22@iitk.ac.in
Aadi Singh	220004	aadis22@iitk.ac.in
Aadya Dhir	220010	aadyad22@iitk.ac.in
Anish Sahu	220148	anishs22@iitk.ac.in

Course: CS253

Mentor TA: Naman Baranwal

Date: 28/03/2025

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
v1.00	Anisha Srivastava Dilbar Singh Lamba Pranshu Thirani Sameer Yadav Sangam Gupta Aruz Awasthi Anirudh Aadi Singh Aadya Dhir Anish Sahu	The First version of Implementation Document.	28/03/2025

1 Implementation Details

PHInance is a comprehensive web-based financial trading software built using PostgreSQL, Golang(Gin/Gorm) and Next.js with React and Tailwind CSS. This technology stack was chosen for its ability to seamlessly handle the complex requirements of our software.

PostgreSQL, a reliable and feature-rich relational database, ensures efficient management and querying of complex financial data.

Golang, paired with the Gin framework and Gorm ORM, delivers a high-performance, scalable backend with clean routing and seamless database integration.

React.js, a powerful JavaScript library, enables us to create dynamic and interactive user interfaces, ensuring a seamless user experience. Tailwind CSS enhances development speed and design consistency, providing a polished and intuitive interface tailored for financial trading.

Programming Language, Framework, and Libraries

Programming Languages: PHInance has been developed primarily using programming language. Billing 360 follows the model-view-controller architectural pattern.

For the Backend :

1. **Golang**: We picked Golang to run the behind-the-scenes part of PHInance. It’s super fast, simple to work with, and great at handling lots of things at once—like trades happening in real time. We use a tool called Gin to manage web requests quickly and Gorm to talk to our database easily.

For the Frontend :

1. **HTML**: This is the basic building block for our web pages. It works on every browser and keeps things simple.
2. **CSS (Tailwind CSS)**: We use Tailwind CSS to make PHInance look good and work well on any screen. It’s easy to update the design across the whole site with just a few changes.
3. **JavaScript (React.js)**: JavaScript makes the site interactive and engaging for users. We built it with React.js, which helps create smooth, dynamic pages that respond quickly to user actions, providing a lively experience for traders.

For the Database (Storing Data):

1. **PostgreSQL**: We chose PostgreSQL to store all the financial info. It’s strong, trustworthy, and perfect for keeping complicated trading data safe and organized.

Frameworks (Extra Helpers): Build Tools: We use Go modules for the backend and npm for the frontend to keep everything organized and grab extra tools when we need them.

1. **Go Modules**: This helps us manage the Golang stuff so the backend stays clean and works smoothly.
2. **NPM**: This lets us add tons of ready-made tools to the frontend, making it quicker to build and improve.

Backend Framework - Golang (Gin/Gorm):

1. Gin keeps the backend fast and simple, handling lots of requests without slowing down.
2. Gorm makes it easy to save and pull data from PostgreSQL, so the trading info flows without hiccups.
3. Golang's special way of juggling tasks keeps PHInance quick, even when tons of people are using it.

Libraries:

PHInance incorporates several key libraries, including Gin, Gorm, and React (via Next.js), chosen for their efficiency and suitability in meeting the project's requirements. Below are the benefits of these libraries:

GIN (Backend Library):

1. Gin is a widely used web framework for Golang, designed to simplify backend development.
2. It provides a lightweight and flexible structure for building web applications and APIs, making it easier to manage server-side logic in PHInance.
3. Gin supports middleware functions, which allow tasks like request processing, authentication, and logging to be handled in a modular and reusable way.
4. It offers a clear and efficient system for defining routes to manage different HTTP requests (e.g., GET, POST, PUT, DELETE), ensuring smooth handling of operations.
5. Gin enhances error management by providing built-in tools to detect and address issues, improving the reliability and maintainability of the application.

GORM (Backend Library):

1. Gorm is a popular object-relational mapping (ORM) library for Golang, used to connect PHInance's backend to the PostgreSQL database.
2. It simplifies database operations by allowing developers to interact with the database using Golang code instead of complex SQL queries.
3. Gorm supports efficient data handling, making it ideal for managing the structured financial data required by PHInance.

React.js (Frontend Library):

1. React.js is a leading JavaScript library for building user interfaces, ideal for dynamic web applications like PHInance. It uses a component-based approach, allowing us to create interactive and reusable UI elements tailored for financial trading.
2. React.js enhances performance with a virtual DOM, a lightweight version of the browser's DOM. It compares the virtual DOM to the real DOM and updates only the parts that change, ensuring faster and smoother rendering of real-time data like stock prices.
3. React.js follows a one-way data flow, where data moves from parent to child components through props. This provides a clear and predictable way to manage data, such as user portfolios and market updates, across the PHInance platform.

Authentication:

PHInance ensures secure user authentication by employing industry-standard cryptographic techniques to protect user credentials and sessions.

- **Password Hashing & Salting:**
  To safeguard user passwords, we utilize the **Bcrypt** library, which hashes passwords with salting to prevent exposure in case of data leaks. This enhances security by making it computationally expensive for attackers to reverse-engineer stored passwords.
- **Token-Based Authentication (JWT):**
  PHInance employs **JSON Web Tokens (JWT)** for user authentication. Upon successful login, a JWT is generated and signed using **HMAC-SHA256**, embedding the user ID and an expiration timestamp (72 hours). The token is then provided to the user for secure session management.
- **Request Authorization:**
  For each API request, PHInance requires a valid JWT in the **Authorization Header** (formatted as `Bearer <token>`). The backend verifies the token's validity, extracts the user ID, and grants access to protected routes. If the token is missing, expired, or malformed, access is denied, ensuring a secure authentication flow.

By integrating these authentication mechanisms, PHInance provides a **robust, secure, and scalable** authentication system, protecting user data while ensuring seamless access to the platform.

2 Codebase

GitHub Repository- https://github.com/notdilbarsl/PHInance

Hosted Website- https://phinance-2.onrender.com/auth/signup

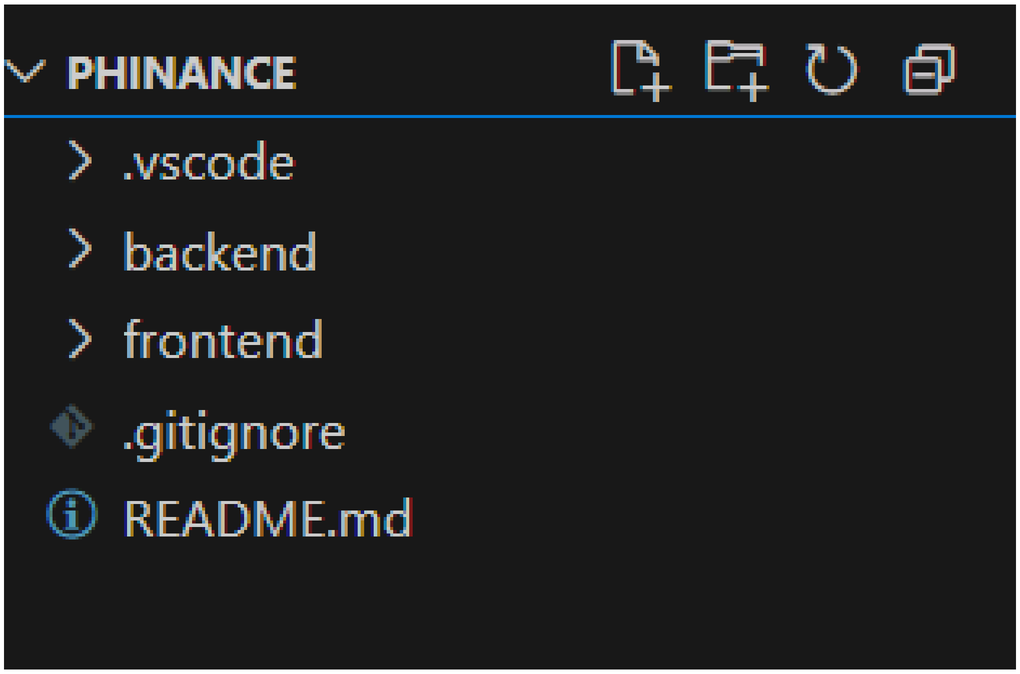
Code Structure:

The above link takes you to the Github repository of PHInance , which contains all the source code for our application.

The project repository is mainly composed of two parts:

- Frontend
- Backend

The overall image of the PHInance repository when cloned in VSCode is as shown:



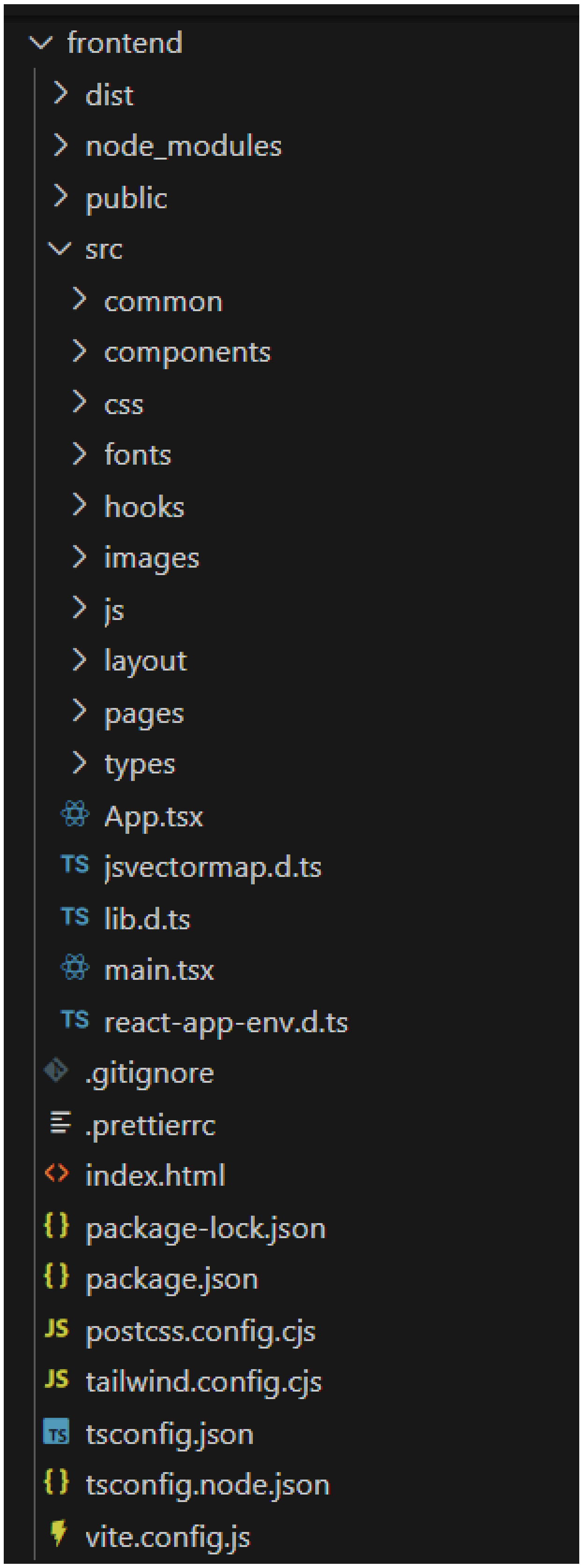
```
PHINANCE
  > .vscode
  > backend
  > frontend
    .gitignore
    README.md
```

When you open the Frontend, the subsequent images are shown. It has mainly the following parts:

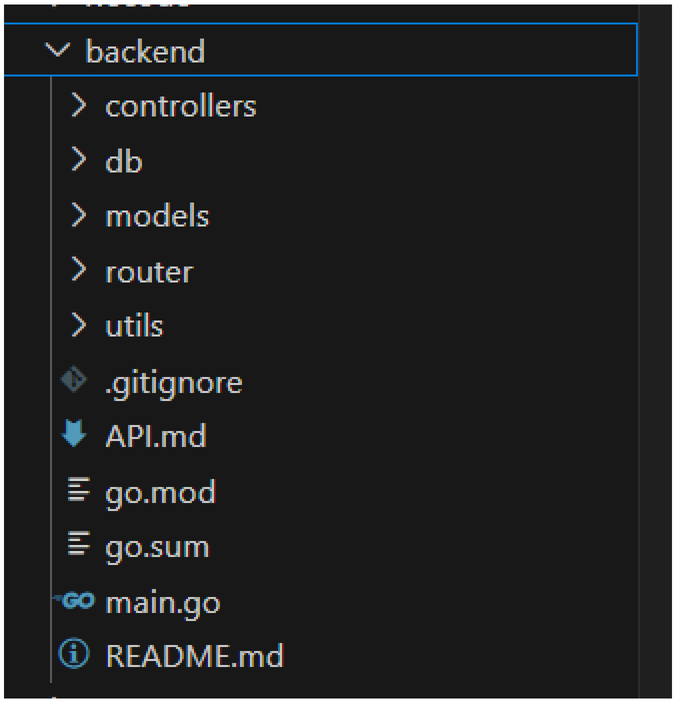
- Dist : This folder contains the compiled and optimized version of the frontend code, ready for deployment and it includes the final JavaScript, HTML, and CSS files that are served to the user’s browser.
- Node_modules : This folder stores all the external libraries and dependencies required for the frontend, such as React.js, Tailwind CSS, and other tools installed via npm.

- Public : The public folder holds static assets that are directly accessible to the frontend, such as images, fonts, or other files needed for the user interface. These assets are not processed by the build system and are served as-is to the browser.

- Src : The src folder is the main working directory for the frontend code. It contains the source files for the React.js application, including directories such as components, pages, and styles. This is where the core logic for rendering the user interface is developed and organized.
  - Common
  - Componenets
  - Css
  - Fonts
  - Hooks
  - Images
  - Js
  - Layout
  - Pages
  - Types



```
frontend
  dist
  node_modules
  public
  src
    common
    components
    css
    fonts
    hooks
    images
    js
    layout
    pages
    types
    App.tsx
    jsvectormap.d.ts
    lib.d.ts
    main.tsx
    react-app-env.d.ts
  .gitignore
  .prettierrc
  index.html
  package-lock.json
  package.json
  postcss.config.cjs
  tailwind.config.cjs
  tsconfig.json
  tsconfig.node.json
  vite.config.js
```



```
backend
  controllers
  db
  models
  router
  utils
  .gitignore
  API.md
  go.mod
  go.sum
  main.go
  README.md
```

The backend has the following parts:

- Controllers : Houses the business-logic layer and request handlers for different parts of the application. It contains files like auth.go for authorization handling.

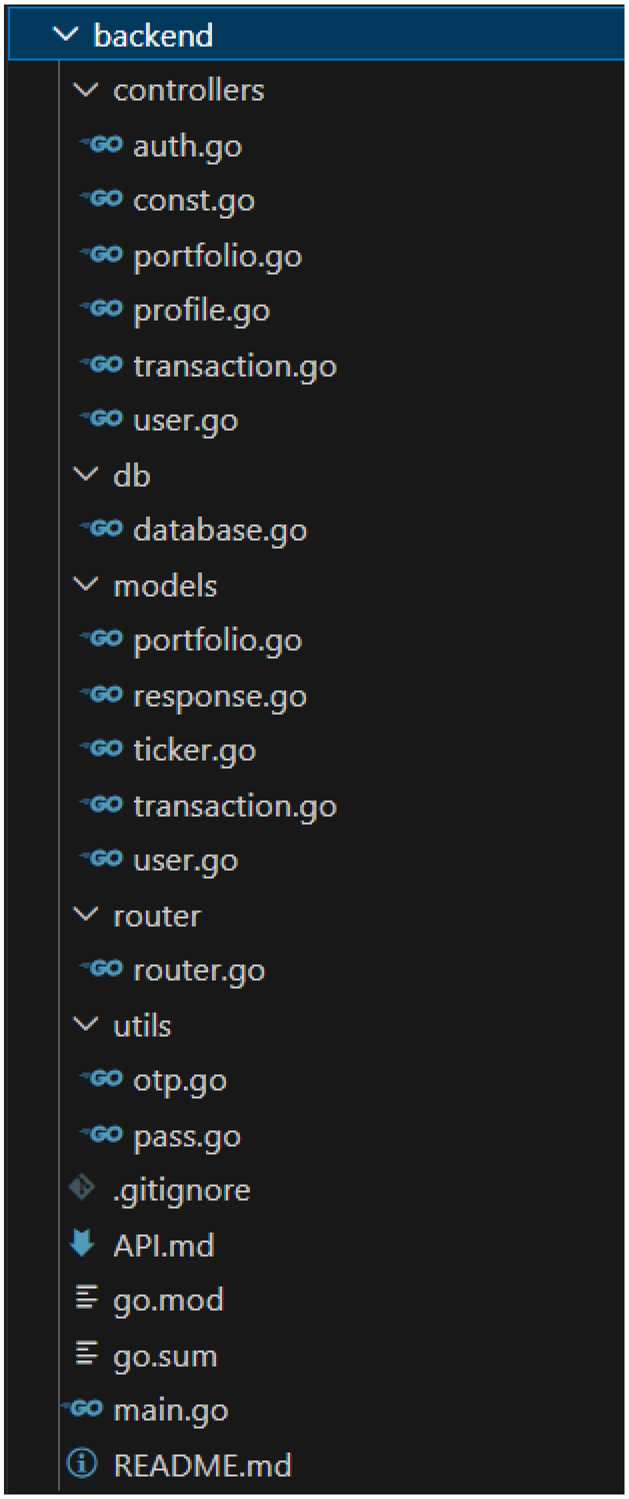
- Db : handles the database initialization. It contains only one file names database.go which loads .env variables (using godotenv.Load()). Constructs the PostgreSQL DSN and opens a GORM connection. Runs AutoMigrate on models: User, Transaction, Ticker, PortfolioStock. Pre-seeds the database with default ticker data from models.GetNiftyFifty().

- Models : Contains the data models and related logic. For eg, user.go contains the User struct (includes email, balance, etc)

- Router : Responsible for defining the API's routes and linking them to controller functions.

- Utils : Holds small, reusable utility functions. Eg, otp.go contains GenerateOtp() for generating a numeric 6-digit OTP.

# The readme file for backend is inside the backend directory.



```
backend
  controllers
    auth.go
    const.go
    portfolio.go
    profile.go
    transaction.go
    user.go
  db
    database.go
  models
    portfolio.go
    response.go
    ticker.go
    transaction.go
    user.go
  router
    router.go
  utils
    otp.go
    pass.go
  .gitignore
  API.md
  go.mod
  go.sum
  main.go
  README.md
```

Other Important Files:

- **main.tsx and app.tsx**: These files are located in the frontend directory. main.tsx is the starting point for the PHInance frontend, setting up the application and preparing it to display webpages to the user. It ensures the app loads properly and applies the overall look, including styles like Tailwind CSS. app.tsx is a key file that manages the user's session, such as keeping track of who is logged in, and helps the app move smoothly between different webpages, like from the dashboard to the profile page. App.tsx is a higher-level component that sets up the initial layout or routing.

- **main.go**: main.go is the Entry point for the Go application. It configures Gin (optionally sets Gin to release mode in production). It creates a default Gin router instance. Calls the router setup function (router.FinRoutes(r)). Starts the server with r.Run().

- **router**: This is a folder within the backend directory, containing the router.go file. It organizes how the backend responds to user actions, such as signing up, logging in, or checking their profile. It ensures that requests from the user are directed to the correct part of the system, which then works with the database to get or update information and send the results back to the user.

Architectural Design

PHInance is developed using the Model-View-Controller (MVC) architecture, an approach well-suited to our financial trading software. This architecture enables multiple ways to interact with the same data across different views while allowing data modifications independent of the user interface. The code structure mirrors this architectural pattern, consisting of three primary components:

1. **Model**:
   The Model handles the data and underlying logic of the website. It comprises PostgreSQL database schemas defined using Gorm, a Golang ORM library. These schemas represent the various data objects essential to the platform, with detailed class descriptions outlined in the Design document. The code for this component is located in the models folder within the backend directory, where each file defines the schema and structure for a specific data class, facilitating seamless integration with PostgreSQL.

2. **View**:
   The View controls what users see and how they interact with PHInance. It is built using

React components within the Next.js framework, with Tailwind CSS providing a consistent and responsive design. The code for this component is housed in the pages and components folders within the frontend directory. Each JavaScript file in pages manages a specific view presented to the user, while the components folder contains reusable React components for handling user interactions. Tailwind CSS is integrated into these files to manage styling and layout. Supporting assets, such as images, are stored in the public folder.

3. **Controller**:
   The Controller contains the business logic linking the Model and View. It is implemented in Golang using the Gin framework to manage HTTP requests and routing. The code for this component resides in the controllers folder within the backend directory. Each .go file in this folder encapsulates logic for specific tasks, such as processing trades or retrieving data. Additional utility functions are organized in the utils folder within the same directory, providing reusable tools to support the controllers.

The code is organised in accordance with the functionalities, which in turn correspond to webpages rendered before the user:

- **Authentication** : This is the sign-in/sign-up page that allows a user to create a new accountand login to access his data and use the software's functionalities.
- **Dashboard**: This serves as the main homepage for users, displayed upon logging into PHInance. It provides an overview of key information and quick access to the platform's features. It displays some stocks and the user's watchlist.
- **Profile**: On this page, users can view and manage their personal information, account settings, and preferences, ensuring a personalized experience. The user may also view his transaction history and stocks owned.
- **Buy And Sell Stocks**: This page enables users to execute stock trading activities, allowing them to buy and sell stocks seamlessly within the platform.
- **AI Powered Predictions**: This feature provides users with data-driven insights and forecasts for stock performance, leveraging artificial intelligence to support informed trading decisions.
- **Portfolio Management**: On this page, users can track and manage their investment portfolios, including viewing performance metrics and adjusting their holdings.
- **Behavioural Analytics**: This page offers insights into user trading patterns and behaviors, helping users understand their decision-making tendencies and improve their strategies.
- **Risk Management**: This section allows users to assess and manage potential risks associated with their trading activities, providing tools to evaluate and mitigate financial exposure.
- **Support**: This page offers assistance to users, providing resources, FAQs, and contact options to help them navigate the platform and resolve any issues.

3 Completeness

We have successfully implemented the features outlined in the SRS for PHInance. Below, we detail the completed functionalities, organized by their respective sections:

Sign In/Sign Up:

- We have implemented the sign-in feature, allowing users to log in using their email ID and password, as specified in the SRS.
- For new users, the Sign-Up feature allows registration by entering their name, email and a secure password.

Light Mode / Dark Mode:

- We have implemented a simple switch to toggle between light mode and dark mode on our website. The switch is accessible on the header once a user is logged in.

Dashboard:

- The dashboard serves as the central hub for users, providing an intuitive and data-driven interface to track their portfolio, buy and sell stocks, monitor stock movements, access AI-powered predictions, and explore learning resources, as outlined in the SRS.
- We have implemented a persistent header/navigation bar at the left side, featuring the platform logo (clickable to redirect to the dashboard), a search bar to find stocks by symbol or company name, and navigation links to key sections: Dashboard, Portfolio Management, Risk Management, Behavioural Analytics, AI Powered Predictions, Buy and Sell Stocks, Learning, Support and Profile/Settings.
- The main dashboard is divided into sections, offering a quick overview of the stocks and watchlist.

Portfolio Management:

- The Portfolio Summary section provides a clear overview of the user's financial standing in the simulated trading environment, displaying key metrics such as Virtual Balance (starting at ₹10,000 and updating after trades), Total Portfolio Value (cash plus current market value of investments), Total ROI (Return on Investment), and Daily PnL (Profit and Loss), as specified in the SRS. The user can also view his Holdings in detail. Eg, he can see for every stock, his buying history, quantity of stock owned, return % and current value.

Buying and Selling Stocks:

- Quick action buttons have been implemented, allowing users to buy stocks (opening a trading window to input stock symbol, quantity, and order type), sell stocks (showing real-time market price and potential PnL), and view transaction history (displaying details like stock name, date, buy/sell price, total trade value, and PnL per trade).

AI-Powered Predictions (End-of-Day Insights) :

- This section provides the user with AI Powered predictions about various stocks, their market sentiment (bullish/bearish) and advice to buy/sell said stocks with a confidence metric in the prediction. The predictions are made by an LSTM model trained on a vast expanse of data to predict long term trends. The predictions are updated at the end of every trading day.

Behavioural Analytics:

- The Behavioural Analytics section provides insights into the user's trading patterns, helping them understand their decision-making habits, as outlined in the SRS.
- We have implemented a line graph showing the total money invested and total profit, allowing users to track their financial performance over time.
- A pie chart displays the distribution of trades, categorizing them into winning trades, losing trades, high-risk trades, and low-risk trades, helping users assess their trading behavior and risk preferences.

Risk Management:

- The Risk Management section helps users evaluate and manage potential risks in their trading activities.
- We have implemented a feature to display the Value at Risk (VaR) for both short-term(1 day VaR) and long-term (1 month VaR) periods.
- The VaR metrics are presented for individual stocks, allowing users to identify which stocks pose the highest risk and adjust their trading strategy accordingly.

Learning & Gamification Section:

- The All Courses section offers a comprehensive learning experience, combining both video lessons and interactive quizzes to facilitate in-depth understanding of stock market concepts.
- Videos Section: This section includes 3 instructional videos, each designed to guide users through fundamental and advanced concepts of the stock market. The videos aim to provide clear and structured learning, covering topics such as market trends, investment strategies, and other essential financial knowledge.
- Quizzes Section: The quizzes section comprises 6 interactive quiz cards, each focused on testing the user's comprehension of the material covered in the videos. These quizzes are strategically designed to reinforce key concepts, allowing users to assess their knowledge and track their progress.

Frequently Asked Questions (FAQs):

- We have implemented the FAQs section, providing answers to common questions and doubts users may have while using PHInance, serving as a quick reference guide, as mentioned in the SRS. We also included the contact details in case the user wants to approach us.

Future Improvements:

- Support for multiple languages to accommodate users from diverse regional backgrounds has not yet been implemented. In future updates, we aim to add language options, enabling users to interact with PHInance in their preferred language, making the platform more accessible.
- Currently, PHInance handles a moderate number of users efficiently, but as the platform grows, we have not yet implemented advanced load balancing and caching mechanisms to support a significantly larger user base.
- Currently, we have not included a regular feedback system for users to provide ratings or suggestions to the developers. We intend to implement this feature in a future version, allowing users to share their feedback directly within the platform to help us improve the user experience.
- We plan to introduce a customizable dashboard feature, where users can rearrange sections (e.g., Portfolio Summary, Market Overview) based on their preferences, providing a more personalized trading experience.
- The option to export transaction history and portfolio reports as downloadable PDFs or Excel files has not been implemented yet. This feature will be added in a future update, enabling users to save and share their trading data for record-keeping or analysis. We aim to enhance the AI-Powered Predictions section by adding a feature to compare AI predictions with actual stock performance after the trading day, helping users evaluate the accuracy of the predictions and improve their trading strategies over time.

Appendix A - Group Log

The frontend part (View) was first written, followed by the backend part (Model and Controller). In each stage, tasks were divided among team members, and pair programming was used to carry out tasks. Thus, all team members are familiar with all stages of the development process as well as all aspects of the software.
Prior to the mid-semester examinations, the frontend part was divided among individual members and was mostly completed before the midsems and before the submission of the design document. After midsem, work started on backend and integration and on the AI model.

Date	Timing	Duration	Agenda
13 Feb 2025	9pm-11pm	2hrs	• Review the frontend work of all the members of the team and discuss the problems that they had faced in completing their work and resolve them. • Installation of PostgreSQL and using basic commands.
3 March 2025	9:30pm-11pm	1.5hrs	• Divided segments of the software among the groups of 2-3 persons where each group was assigned a segment to gather information and explain how it can be implemented.
5 March 2025	6pm-8pm	2hrs	• Discuss the previous work assigned and divide the further work among groups of 2-3-3-2 depending upon the intensity of the workload.
7 March 2025	2pm-5pm	3hrs	• Discuss the work assigned and decide how to do the rest of the work on GitHub so that everyone can work in a collaborative manner. • At this time, we finally divided the team into groups of size 2-2-2-2-2 for the implementation part and these groups remained intact till the implementation was completed. • Each pair was assigned the complete responsibility of a particular webpage.
16 March 2025- 24 March 2025	-	-	• During these days, everyone did their work in pairs as assigned and if anyone faced any error, it was solved by the other members.

			• We also had meets on alternate days to report the work each person and each pair had done .
26 March 2025	4pm-8pm	4hrs	• Major tasks on each webpage were done and reported. • The task of writing this document was assigned to a pair and other pairs were asked to implement the optional features that had been mentioned in SRS. • Also, there were a lot of errors we were facing while running the application by integrating everyone's work. Hence, each group was also asked to debug and to do the best they could to remove errors and make the software run smoothly.
28 March 2025	12 noon – 1pm 2pm-5pm	4hrs	• Did the final edits to the code. • Everything was now working fine. • The software was deployed. • This document was reviewed and finalized.