

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"Jnana Sangama", Belagavi-590 018



A Project-Based Learning (Full Stack Development-BIS601) Report on

“Coinly- A Crypto dashboard and Portfolio Manager”

A Dissertation work submitted in partial fulfillment of the requirement for the award of the degree

Bachelor of Engineering
in
Information Science and Engineering

Submitted by

Rohan Ravi Vernekar **1AY22IS078**
Someshwar R Halewadimath **1AY22IS106**

Under the Guidance of
Dr. Pankaj Kumar
Assistant Professor



DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING
ACHARYA INSTITUTE OF TECHNOLOGY

(AFFILIATED TO VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELAGAVI. ACCREDITED BY NACC, RECOGNISED BY AICTE, NEW DELHI)
Acharya Dr. Sarvepalli Radhakrishnan Road, Soldevanahalli, Bengaluru - 560107

2024-2025

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

ACHARYA INSTITUTE OF TECHNOLOGY

(Affiliated to Visvesvaraya Technological University, Belagavi, Recognized by NAAC, New Delhi)

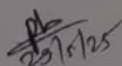
Acharya Dr. Sarvepalli Radhakrishnan Road, Soldevanahalli, Bengaluru-560107

2024-2025

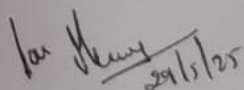


Certificate

This is to certify that the Project based Learning using Full Stack Development (BIS601) entitled "Coinly – A Crypto Dashboard and Portfolio Manager" carried out by Rohan Ravi Vernekar (1AY22IS078) and Someshwar R Halewadimath (1AY22IS106), are bonafide students of Acharya Institute of Technology, Bengaluru in partial fulfillment for the award of the degree of Bachelor of Engineering in Information Science and Engineering of the Visvesvaraya Technological University, Belagavi during the year 2024-25. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The project based learning report has been approved as it satisfies the academic requirements in respect of Project Phase I prescribed for the said degree.


23/5/25

Signature of the Guide
Dr. Pankaj Kumar


29/5/25

Signature of the HOD
Dr. Kala Venugopal
Head of the Department
Department of Information Science & Engg
Acharya Institute of Technology
Soldevanahalli, Bengaluru - 560 107

DECLARATION

Rohan Ravi Vernekar (1AY22IS078) and Someshwar R Halewadimath (1AY22IS106), students of B.E. Information Science and Engineering, Acharya Institute of Technology, Bangalore, hereby declare that the Project Based Learning on Full Stack Development (**BIS601**) entitled “**Coinly- A Crypto dashboard and Portfolio Manager**” is an authentic work carried out under the supervision and guidance of **Dr. Pankaj Kumar**, Assistant Professor, Department of Information Science and Engineering, Acharya Institute of Technology, Bangalore. We have not submitted the matter embodied to any other university or institution for the award of any other degree.

Place: Bengaluru.

Date: 29/05/2025

Name	USN	Signature
Rohan Ravi Vernekar	1AY22IS078
Someshwar R Halewadimath	1AY22IS106

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of a **Project-Based Learning (Full Stack Development – BIS601)** report would be incomplete without mentioning the people who made it possible through their constant guidance and encouragement.

We would take this opportunity to express our gratitude to **Sri. B. Premnath Reddy**, Founder Chairman, Acharya Institutes, **Dr. C K Marigowda**, Principal, Acharya Institute of Technology for providing the necessary infrastructure to complete this **Project Based Learning** report.

We wish to express our deepest gratitude to **Dr. Kala Venugopal**, Head of the Department, Information Science and Engineering.

We wish to express our sincere thanks to our guide **Dr. Pankaj Kumar**, Assistant Professor, Department of Information Science and Engineering for helping us throughout, and guiding us from time to time.

A warm thanks to all the faculty of Department of Information Science and Engineering, who have helped us with their views and encouraging ideas.

Rohan Ravi Vernekar (1AY22IS078)

Someshwar R Halewadimath (1AY22IS106)

ABSTRACT

This system encompasses CoinlyCrypto, a web-based portfolio management application designed to streamline the tracking and analysis of cryptocurrency investments on a single, unified platform. Through CoinlyCrypto, users can securely register and log in to maintain personalized portfolios, adding or removing coins with ease. The application fetches real-time price data and historical market trends from the CoinGecko API, enabling investors to view up-to-date valuations and visualize their holdings through interactive line and pie charts. By automating data retrieval and calculation of profit/loss metrics, CoinlyCrypto removes the need for manual spreadsheet updates and disparate tools. As cryptocurrency investing becomes more prevalent, the need for robust yet user-friendly management tools has become critical. The primary objective of CoinlyCrypto is to provide a maintainable, scalable MERN-based solution that consolidates portfolio tracking, data visualization, and secure user management into one cohesive application. By leveraging MongoDB for flexible data storage, Express/Node.js for performant API services, and React/Chart.js for dynamic front-end presentation, CoinlyCrypto addresses the limitations of manual tracking and fragmented platforms, ultimately enhancing investors' ability to monitor and optimize their cryptocurrency portfolios.

TABLE OF CONTENTS

Acknowledgement	i
Abstract	ii
Table of Contents	iii
1 INTRODUCTION	1
1.1 Introduction to MERN	1
1.2 Why MERN?	2
1.3 MERN applications	3
1.4 Advantages of MERN	3
1.5 Components of MERN	4
1.6 MERN architecture	6
2 SYSTEM REQUIREMENTS	8
2.1 Hardware Requirements	8
2.2 Software Requirements	8
3 SYSTEM DESIGN	9
3.1 System Design	9
3.1.1 ER Diagram	9
3.2 Detailed Design	11
3.2.1 Class Diagram	11
3.2.2 Activity Diagram	12
3.2.3 Use Case Diagram	14

4	IMPLEMENTATION	16
4.1	Modules	16
4.2	Table	18
5	SNAPSHOTS	19
	CONCLUSION	21
	BIBLIOGRAPHY	22

LIST OF FIGURES

1.1	MERN Stack Architecture	6
1.2	MongoDB Architecture	6
3.1	ER Diagram	9
3.1	Class Diagram	11
3.2	Activity Diagram	12
3.4	Use Case Diagram	14
5.1	Dashboard	19
5.2	Live Coins	19
5.3	Add Coin	20
5.4	Portfolio	20

LIST OF TABLES

4.1	Implementation Table	18
------------	-----------------------------	-----------

CHAPTER 1

INTRODUCTION

1.1 Introduction to MERN

The modern web is built on a diverse set of technologies, each playing a crucial role in delivering dynamic and interactive experiences. Among the most popular and efficient combinations for full-stack web development is the **MERN stack**. An acronym derived from its four core technologies, MERN stands for **MongoDB, Express.js, React, and Node.js**. It represents a powerful, end-to-end JavaScript-based framework that allows developers to build robust, scalable, and high-performance web applications entirely within a unified language ecosystem. At its heart, the MERN stack is about consistency and efficiency. By leveraging JavaScript across the entire application lifecycle – from the database to the server to the client-side user interface – it significantly streamlines the development process, reduces context switching for developers, and promotes code reusability. This "JavaScript everywhere" paradigm is a key differentiator and a major reason for MERN's widespread adoption.

- **MERN is a full-stack JavaScript framework** that allows developers to build web applications using a single language—JavaScript—across all layers of the stack, from the browser frontend to the backend database.
- **React** powers dynamic and interactive user interfaces on the client side.
- **Node.js** and **Express.js** manage the server logic, handle requests, and connect to the database.
- **MongoDB**, a NoSQL database, stores application data in flexible, JSON-like documents.

Together, these four technologies create a cohesive ecosystem that dramatically simplifies and accelerates development. MERN empowers developers to prototype quickly, iterate easily, and scale efficiently.

The MERN stack empowers developers to build everything from simple websites to complex enterprise-level applications.

1.2 Why MERN?

The MERN stack has become one of the most popular and compelling choices for modern web development, and for good reason. Its appeal stems from a combination of factors that significantly enhance the development process, application performance, and long-term maintainability.

Here's a breakdown of the key reasons why developers and businesses choose MERN:

1. Unified JavaScript Environment ("JavaScript Everywhere")

This is arguably the most significant advantage. MERN allows you to use **JavaScript for every layer of your application**:

- **Frontend (React):** Built entirely with JavaScript and JSX.
- **Backend (Node.js & Express.js):** JavaScript runs on the server.
- **Database (MongoDB):** Stores data in JSON-like (BSON) documents, which aligns perfectly with JavaScript's native data structures. Even MongoDB's shell commands are JavaScript-based.

2. Why this matters:

- **Reduced Context Switching:** Developers don't need to switch between different programming languages (e.g., Python for the backend, Java for the database, JavaScript for the frontend). This reduces cognitive load, speeds up development, and minimizes errors.
- **Simplified Hiring:** It's easier to find full-stack developers proficient in JavaScript, or to cross- train frontend developers to backend tasks, and vice versa. This can streamline the hiring process and reduce recruitment costs.
- **Code Reusability:** Some logic, such as validation rules or utility functions, can potentially be shared and reused between the frontend and backend, leading to less redundant code and more consistent behaviour..

1.3 MERN Applications

MERN is well suited to a wide variety of web applications, particularly those requiring real-time interactivity or dynamic single-page experiences. Common use cases include:

- **E-commerce platforms**, where live inventory updates and personalized dashboards are critical.
- **Social networking sites**, which depend on instant notifications, chat features, and live feeds.
- **Dashboard and analytics tools**, leveraging React's data-visualization libraries and MongoDB's aggregation framework.
- **Content management systems (CMS)**, taking advantage of React's editing components and Node's API endpoints.
- **SaaS products**, where rapid iteration and full-stack JavaScript speed up prototyping and deployment.

1.4 MERN Applications

The MERN stack offers several key advantages for modern web development:

- **Unified Language:** Developers write both client- and server-side code in JavaScript, minimizing cognitive load and facilitating code reuse.
- **Rapid Development:** A wealth of community-maintained packages and boilerplates accelerates project setup and feature implementation.
- **Scalability:** Node.js's event-driven architecture and MongoDB's sharding capabilities support high-throughput applications.
- **Flexibility:** MongoDB's schema-less data model adapts easily to changing requirements without costly migrations.
- **Maintainability:** Modularization in Express, React's component hierarchy, and clear separation of concerns lead to cleaner, more testable codebases.

1.5 MERN Components

The MERN stack is comprised of four powerful technologies MongoDB, Express.js, React, and Node.js each playing a distinct role in the architecture of modern web applications.

Together, they form a cohesive stack that allows developers to build robust, scalable, and maintainable applications using a single programming language: JavaScript.

1. MongoDB - The Database Layer

Key features:

- **Schema Flexibility:** You can add or remove fields from documents without affecting the entire database structure.
- **High Performance:** Fast read/write operations, ideal for large-scale applications.
- **Horizontal Scalability:** Supports sharding to distribute data across multiple servers.
- **Aggregation Framework:** Allows complex data transformations, filtering, and summarization with ease.
- **Cloud Support:** With services like MongoDB Atlas, developers can easily deploy secure and scalable cloud-hosted databases.

In MERN, MongoDB is used as the primary data store, managing everything from user credentials to application-specific data like posts, transactions, or logs.

2. Express.js – The Backend Web Framework

Key features:

- **Routing system:** Simplifies the creation of RESTful APIs through clear and intuitive route handlers (e.g., `app.get()`, `app.post()`).
- **Middleware support:** Allows modular handling of request processing, such as authentication, error handling, logging, and more.
- **Performance-oriented:** Lightweight in design, ensuring fast request-response cycles.
- **Customizability:** Easy to extend and integrate with other Node.js libraries and middleware.

Express serves as the controller in the MERN architecture, processing user requests and orchestrating the application's business logic.

3. React – The Frontend Library

Key Features:

- **Virtual DOM:** React uses a virtual representation of the DOM to efficiently update only the parts of the page that change, improving rendering performance.
- **Component-based architecture:** Encourages breaking UI into modular, reusable components like buttons, forms, cards, and lists.
- **State and props management:** Helps manage data flow and dynamic interactions within the UI.
- **Hooks:** Functional components can leverage features like lifecycle methods and local state using built-in hooks (useState, useEffect, etc.).
- **Rich ecosystem:** Integrates easily with tools like Redux for state management and React Router for navigation.

React is the view layer of the MERN stack, enabling the creation of fast, responsive, and engaging user interfaces.

4. Node.js – The Runtime Environment

Key Features:

- **Asynchronous I/O:** Node.js processes multiple client requests concurrently without blocking the execution thread, making it efficient for I/O-heavy operations like reading from files or querying databases.
- **Fast execution:** Thanks to the V8 engine, JavaScript code is compiled into machine code, ensuring fast and optimized performance.
- **Modular ecosystem:** Comes with npm (Node Package Manager), the largest ecosystem of open-source libraries and tools.
- **Cross-platform:** Runs on various platforms including Windows, Linux, and macOS.

Node.js acts as the **runtime platform** that executes the backend logic, hosts the server, and integrates with the Express framework to form the backbone of the application.

1.6 MERN Architecture

The MERN stack architecture is a **full-stack JavaScript solution** that enables developers to build powerful, dynamic, and scalable web applications with a clear separation of concerns across the frontend, backend, and database layers. Each component of the stack **MongoDB**, **Express.js**, **React**, and **Node.js** has a well-defined role, and they work together seamlessly to deliver a complete application development lifecycle using only JavaScript.

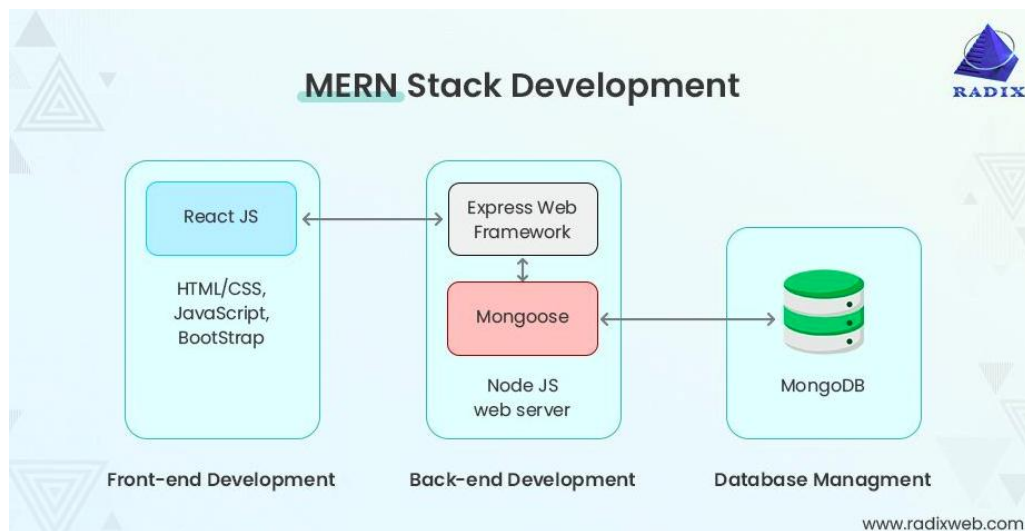


Figure 1.1: MERN stack architecture

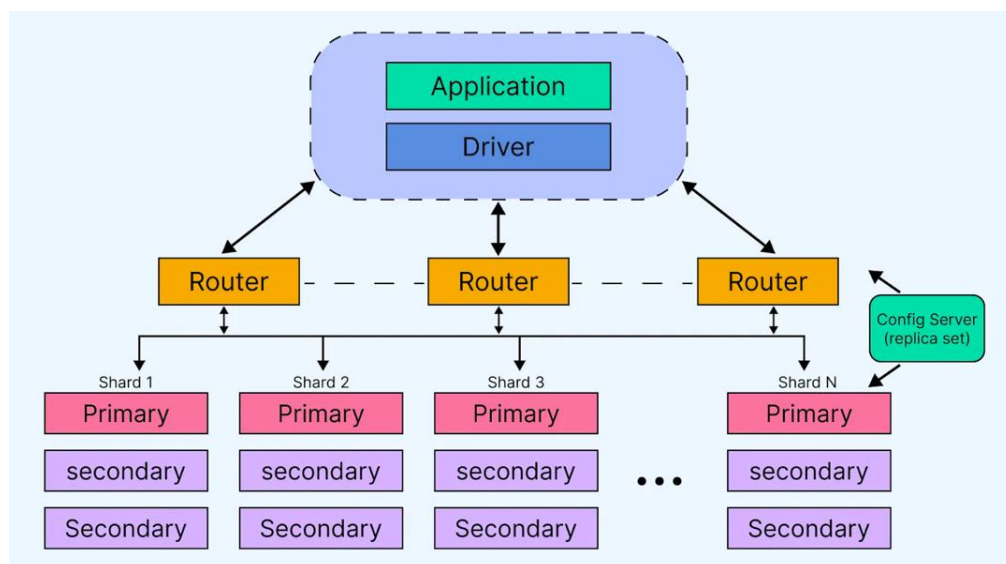


Figure 1.2: MongoDB architecture

Overview of Architecture Flow:

The MERN architecture follows a **three-tier model**:

1. **Client-side (React)**
2. **Server-side (Express.js + Node.js)**
3. **Database (MongoDB)**

Detailed Layer-wise Architecture:

1. React.js (Frontend/UI Layer)

- Built as a **Single Page Application (SPA)**.
- Utilizes components to structure the UI.
- Uses **React Hooks** (useState, useEffect, etc.) to manage data and lifecycle events.
- Calls backend APIs using **Axios or Fetch** to perform operations like login, register, fetch user data, or update portfolios.
- Manages routing (navigation) using **React Router** without reloading the page.

Example: A cryptocurrency dashboard page sends a GET request to fetch market data, which is then displayed as a chart using a library like Chart.js.

2. Express.js + Node.js (Backend/API Layer)

- Node.js provides the **runtime environment** for running JavaScript on the server.
- Express.js defines the **API endpoints** (e.g., /api/login, /api/portfolio, /api/marketdata).
- Handles **middleware**, such as JSON parsing, authentication (e.g., JWT verification), logging, and error handling.
- Communicates with MongoDB via **Mongoose**, a powerful ODM (Object Data Modeling) library.

Example: When a user logs in, the Express route validates credentials and generates a **JWT token**, which is sent back and used to access protected routes.

3. MongoDB (Database Layer)

- Stores data in flexible **document-oriented format**.
- Collections like users, portfolios, and transactions store application data.
- Interfaced via **Mongoose**, which allows the definition of schemas, models, and validations.
- Suitable for hierarchical and nested data like transaction histories or user settings.

Example: A user's portfolio data is stored in a document that includes coin names, investment values, quantities, and timestamps.

CHAPTER 2

SYSTEM REQUIREMENTS

2.1 Hardware Requirements

2.1.1 Development Environment (Local Machine)

- CPU: Dual-core processor (Intel i3 / AMD equivalent) or higher.
- RAM: Minimum 4 GB (8 GB+ recommended for smooth multitasking).
- Disk: At least 10 GB of free SSD space (for Node modules, local MongoDB data, logs).
- Network: Stable broadband connection (for pulling dependencies, hitting external APIs like CoinGecko).

2.1.2 Production Environment (Server)

- CPU: 2 vCPUs or more (for handling concurrent API requests and background tasks).
- RAM: 4 GB minimum (8 GB+ recommended if you cache price histories or run analytics)
- Disk: 20 GB SSD or more, with IOPS suited for database reads/writes.
- Network: Low-latency, high-throughput link (to serve clients and fetch real-time market data).
- Optional: GPU instance if you plan to add ML-based analytics or heavy data-processing workloads.

2.2 Software Requirements

- Node.js (v14.x or later) and npm (or Yarn) for package management and script execution.
- MongoDB (v4.4 or later) either the Community Edition installed locally or a managed instance on MongoDB Atlas.
- Express.js (v4.x) for building the backend API server.
- Mongoose (v6.x) as the ODM layer to define schemas and interact with MongoDB.
- bcryptjs (v2.x) for hashing and verifying user passwords.
- React (v18.x) and react-dom (v18.x) for building the frontend UI.
- react-chartjs-2 (v4.x) and chart.js (v3.x) for data visualization components.

CHAPTER 3

SYSTEM DESIGN

3.1.1 ER Diagram

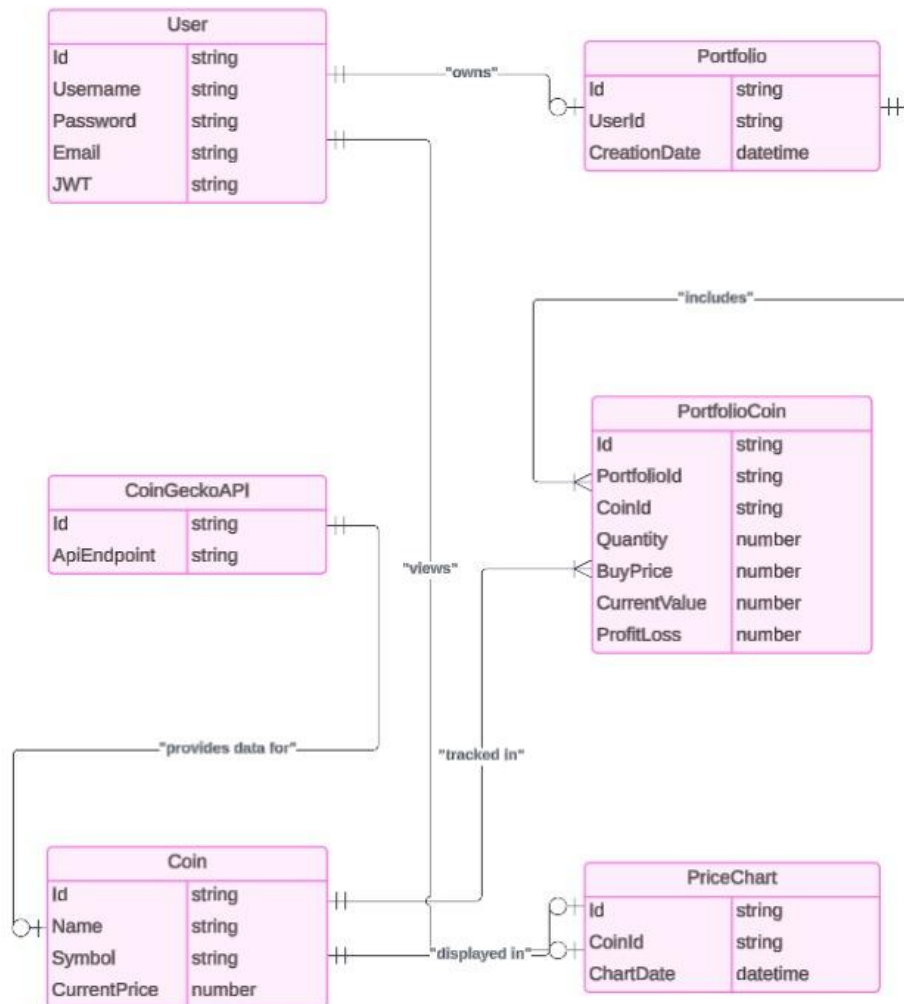


Figure 3.1 ER Diagram

The ER diagram for CoinlyCrypto captures how users track portfolios and market data. Each User may own multiple Portfolios (via UserId), and each portfolio contains many PortfolioCoin records detailing CoinId, Quantity, BuyPrice and computed CurrentValue/ProfitLoss. The Coin entity holds supported cryptocurrencies with fields like Name, Symbol, and live CurrentPrice—updated from CoinGeckoAPI endpoints. For historical charts, PriceChart stores time-stamped price snapshots (ChartDate) per coin. In summary, the key one-to-many links—User→Portfolio, Portfolio→PortfolioCoin, CoinGeckoAPI→Coin and Coin→PriceChart—tie user holdings seamlessly to real-time and historical market data.

3.1.1 Attributes and Relationship

User:

- **Attributes:** Id, Username, Password, Email
- **Relationships:**
 - owns Portfolios (1 User → *many* Portfolios)

Portfolio:

- **Attributes:** Id, UserId, CreationDate
- **Relationships:**
 - belongs to a User (many Portfolios → 1 User)
 - includes PortfolioCoins (1 Portfolio → *many* PortfolioCoins)

PortfolioCoin:

- **Attributes:** Id, PortfolioId, CoinId, Quantity, BuyPrice, CurrentValue, ProfitLoss
- **Relationships:**
 - belongs to a Portfolio (many PortfolioCoins → 1 Portfolio)
 - tracks a Coin (many PortfolioCoins → 1 Coin)

Coin:

- **Attributes:** Id, Name, Symbol, CurrentPrice
- **Relationships:**
 - provided by CoinGeckoAPI (many Coins → 1 CoinGeckoAPI)
 - displayed in PriceCharts (1 Coin → *many* PriceChart entries)
 - referenced by PortfolioCoins (1 Coin → *many* PortfolioCoins)

CoinGeckoAPI:

- **Attributes:** Id, ApiEndpoint
- **Relationships:**
 - provides data for Coins (1 CoinGeckoAPI → *many* Coins)

PriceChart:

- **Attributes:** Id, CoinId, ChartDate
- **Relationships:**
 - belongs to a Coin (many PriceChart entries → 1 Coin)

3.2 Detailed Design

3.2.1 Class Diagram

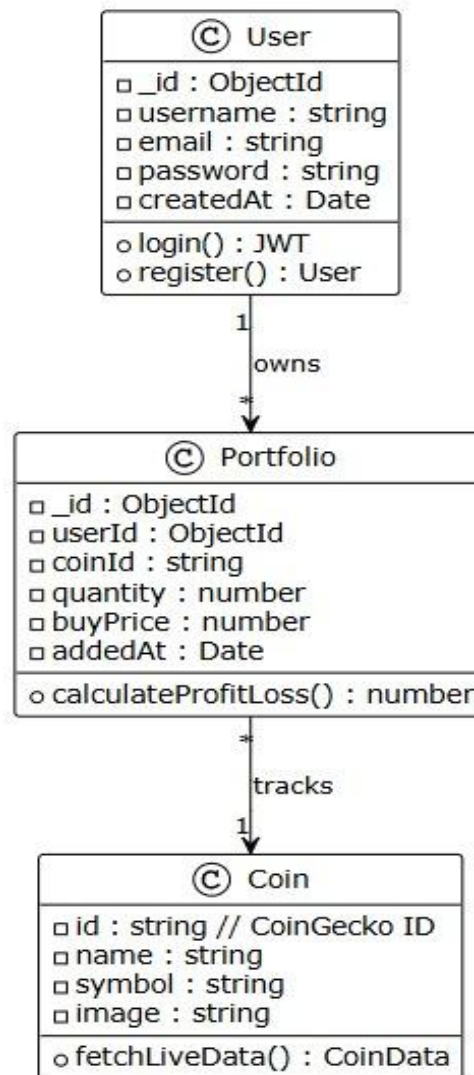


Figure 3.2 Class Diagram

The CoinlyCrypto class model centers on three classes: User, Portfolio and Coin. A User holds credentials (username, email, password), a unique `_id`, a `createdAt` timestamp, and methods `register()` and `login()` (returning a JWT). Each user owns multiple Portfolio entries, each of which records a specific `coinId`, `quantity`, `buyPrice` and `addedAt` date, plus a `calculateProfitLoss()` method to compute unrealized gains or losses. Every Portfolio item references a single Coin, whose class contains metadata (`id`, `name`, `symbol`, `image`) and a `fetchLiveData()` method that retrieves current price, volume and market-cap details as `CoinData`.

3.2.2 Activity Diagram

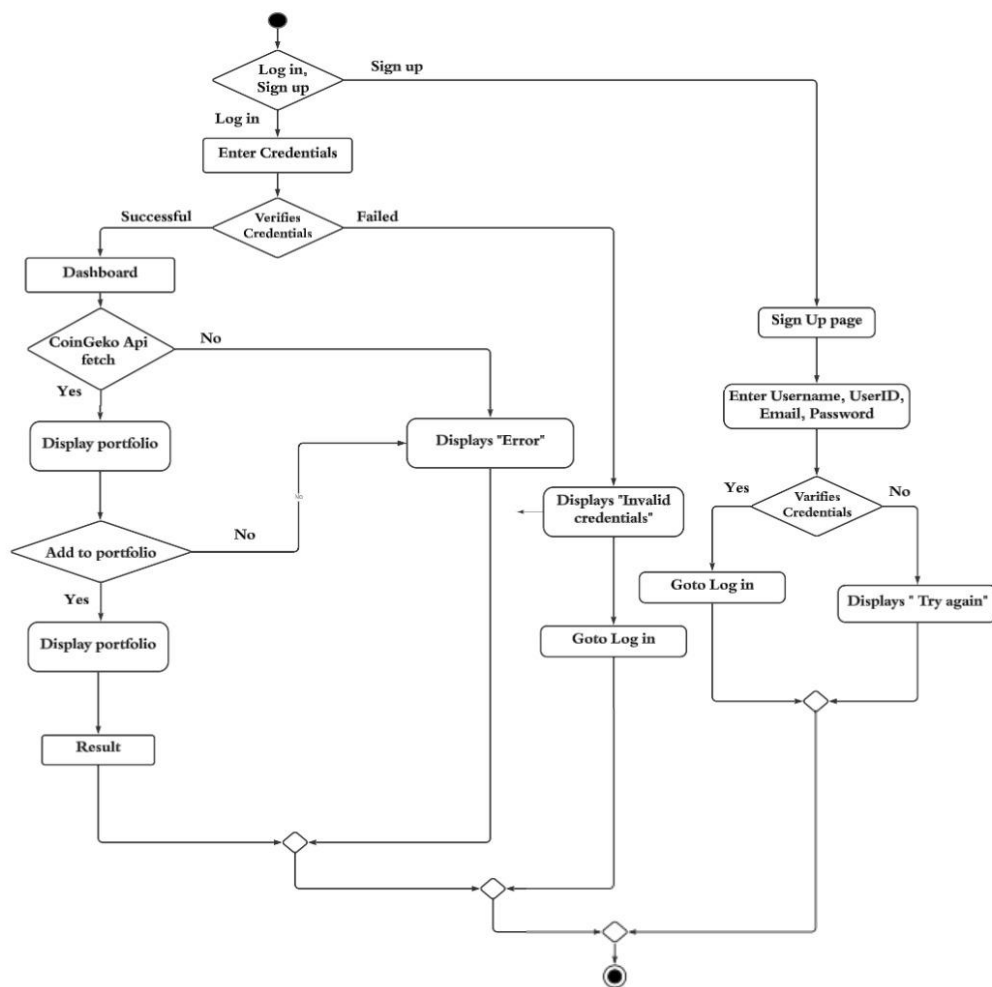


Figure 3.3 Activity Diagram

When a user first arrives at the CoinlyCrypto app, they're prompted to either log in or sign up. Once authenticated, they land on the Dashboard, where they choose one of several actions: add a new coin, remove an existing coin, view their performance charts, or log out. Each action triggers a straightforward sequence of steps—forms are displayed, inputs are collected, the backend verifies or updates data, and the Dashboard is refreshed to reflect changes. Finally, selecting “Logout” ends the session and returns the user to the start state.

Step-by-Step Breakdown

1. Start & Authentication Choice

- **Start:** Entry point of the application.
- **Login or Sign Up:** Decision node presenting the two authentication paths.

2. Login Path

- **Display Login:** The login form appears.
- **Enter Credentials:** User types in their email/username and password.
- **Verify Login:** Backend checks credentials.
- **Dashboard:** On success, the user is taken to the portfolio Dashboard.

3. Sign-Up Path

- **Display Sign-Up:** The registration form appears.
- **Enter Sign-Up Details:** User provides username, email, and password.
- **Submit Sign-Up:** Form data is sent to the backend.
- **Display Login:** After successful registration, the user is redirected to log in.

4. Dashboard & Action Selection

- **Dashboard:** Central hub showing current portfolio holdings.
- **Choose Action:** User sees four possible actions:
 - **Add Coin**
 - **Remove Coin**
 - **View Charts**
 - **Logout**

5. Add Coin Flow

- **Add Coin:** User clicks the “Add Coin” button.
- **Enter Coin Details:** A form prompts for coin symbol and quantity.
- **Submit Add:** Data is sent to the backend to create a new portfolio entry.
- **Dashboard:** After successful addition, the Dashboard refreshes to show the new holding.

3.2.3 Use Case Diagram

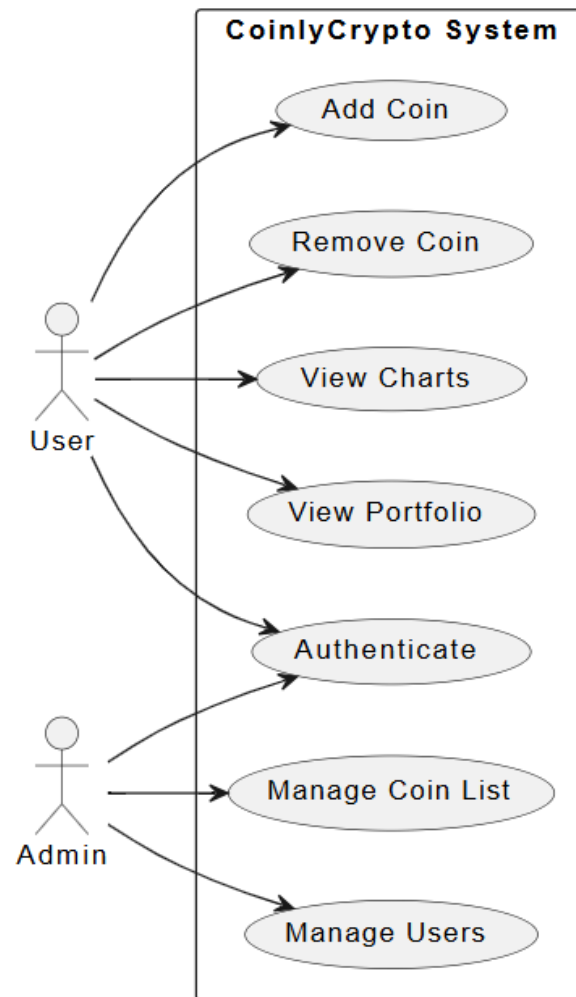


Figure 3.4 Use Case Diagram

The diagram models two main actors **User** and **Admin** interacting with the central “CoinlyCrypto System.” Both actors must first **Authenticate** (i.e. log in or sign up) before accessing any functionality. Once authenticated, the **User** can **View Portfolio**, **Add Coin**, **Remove Coin**, and **View Charts** to manage and visualize their personal cryptocurrency holdings. Meanwhile, the **Admin** has elevated privileges for **Manage Coin List** (adding or updating which currencies the platform supports) and **Manage Users** (e.g. deactivating accounts or resetting passwords). This clear separation of roles ensures that everyday portfolio actions are kept distinct from administrative maintenance tasks.

Use Case Diagram Breakdown

1. Actors

- **User:** A registered individual who tracks and visualizes their crypto investments.
- **Admin:** A system administrator responsible for maintaining the overall platform.

2. Shared Use Case

- **Authenticate:** Both User and Admin must log in (or sign up) to receive a session token before performing any other actions.

3. User-Specific Use Cases

- **View Portfolio:** Retrieve and display the list of coins the user holds, along with current values and P/L.
- **Add Coin:** Submit a new cryptocurrency purchase (symbol and quantity) to include in the portfolio.
- **Remove Coin:** Delete an existing holding from the portfolio.
- **View Charts:** Fetch historical price data and render visualizations (line and pie charts) of portfolio performance and allocation.

4. Admin-Specific Use Cases

- **Manage Coin List:** Add, update, or remove supported cryptocurrencies (e.g., metadata or available symbols).
- **Manage Users:** Perform administrative tasks on user accounts, such as deactivation, role changes, or password resets.

This use-case diagram succinctly captures both day-to-day user interactions with their portfolio and the background administrative operations that keep the platform running smoothly.

CHAPTER 4

IMPLEMENTATION

4.1 Modules

4.1.1 Backend Modules

The backend of CoinlyCrypto is built using Node.js and is powered by a carefully selected set of open-source libraries. These modules provide essential functionalities such as routing, authentication, database interaction, and API communication. Below is a brief overview of the key backend dependencies:

- **express**
A minimal, unopinionated web framework for Node.js.
Used to define REST endpoints (e.g. /api/auth, /api/portfolio) and wire up middleware for JSON parsing, CORS, and error handling.
- **mongoose**
An ODM (Object Data Modeling) library for MongoDB.
Provides schema definitions, validation, and easy querying via models (e.g. User, PortfolioCoin).
- **bcryptjs**
A password-hashing library.
Secures user passwords by hashing on signup and comparing hashes during login.
- **jsonwebtoken**
Implements JSON Web Tokens (JWT).
Issues signed tokens on login (jwt.sign) and verifies them in auth middleware to protect routes.
- **axios**
A promise-based HTTP client for Node.js.
Fetches live price and historical data from the CoinGecko API in utils/coinGecko.js.
- **cors**
Enables Cross-Origin Resource Sharing.
Allows the React frontend (running on a different port) to call Express endpoints without browser blocks.
- **dotenv**
Loads environment variables from a .env file.
Keeps sensitive values (e.g. MONGO_URI, JWT_SECRET) out of source code.

These modules work together to ensure that the backend remains modular, secure, and easy to maintain. They also simplify communication between the client and server, while enforcing good security practices. The result is a clean, robust backend foundation that supports all core functionalities of CoinlyCrypto.

4.1.2 Frontend Modules

The frontend of CoinlyCrypto is built using React, a popular JavaScript library for building dynamic and component-driven user interfaces. To enhance functionality, routing, API integration, and data visualization, several third-party modules are incorporated. Below is an overview of the major frontend dependencies:

- **react & react-dom**

The core libraries for building user interfaces.

Breaks the UI into reusable components and renders them into the DOM.

- **react-router-dom**

Client-side routing for React.

Manages navigation between /login, /signup, and /dashboard without full page reloads.

- **axios**

Same HTTP client in the browser.

Attaches the JWT to each request (Authorization: Bearer <token>) via an interceptor.

- **react-chartjs-2 & chart.js**

Wrapper and underlying library for charts.

Renders line charts (portfolio value over time) and pie charts (asset allocation).

- **(optional) Tailwind CSS**

A utility-first CSS framework.

Provides rapid styling for forms, tables, buttons, and layout.

These tools work cohesively to deliver a smooth and responsive user experience.

They also simplify frontend development by encouraging modularity, reusability, and consistency in UI design.

Overall, the chosen modules ensure the frontend is lightweight, fast, and highly maintainable.

4.2 Tables

Sl.NO	Action	Frontend Behavior	Backend Behavior
1	User Sign Up / Login	Renders SignupForm or LoginForm. On submit: POST /api/auth/signup or POST /api/auth/login via axios. Stores JWT in localStorage and navigates to Dashboard.	POST /api/auth/signup: validate input, hash password (bcrypt), create User. POST /api/auth/login: lookup User, compare password, issue JWT (jsonwebtoken). Returns { token }.
2	Dashboard Load	In PortfolioDashboard's useEffect, fires GET /api/portfolio with JWT header. Receives portfolio entries + current prices. Updates state and renders list, totals.	GET /api/portfolio (protected): auth middleware verifies JWT → req.userId. PortfolioCoin.find({ user: req.userId }).populate('coin'). Optionally fetch current price via CoinGecko util. Returns JSON array of holdings.
3	Add Coin to Portfolio	User clicks "Add Coin": shows AddCoinForm. User enters symbol & quantity. axios POST /api/portfolio with { coinId, quantity, buyPrice }. On success: close form & re-load Dashboard.	POST /api/portfolio (protected): verify JWT → validate payload → create PortfolioCoin with { user, coinId, quantity, buyPrice, addedAt }. Optionally fetch currentPrice via CoinGecko util. Saves document and returns the created entry.
4	Remove Coin from Portfolio	User clicks "Remove" next to an entry. Confirms prompt. Sends axios DELETE /api/portfolio/:id. On success, re-loads Dashboard.	DELETE /api/portfolio/:id (protected): verify JWT → ensure entry's user matches. Calls PortfolioCoin.findByIdAndDelete(id) and returns status 200 on success.
5	View Charts	User clicks "View Charts." Sends GET /api/portfolio/history?days=30. Receives { history: [...], allocation: [...] }. Renders line and pie charts via react-chartjs-2.	GET /api/portfolio/history (protected): verify JWT → fetch user's portfolio. For each coin, call CoinGecko /market_chart?days=... endpoint. Aggregate daily quantity × price_t into time-series. Build allocation percentages from current prices. Return JSON for charting.

Table 4.1: Implementation

CHAPTER 5

SNAPSHOTS

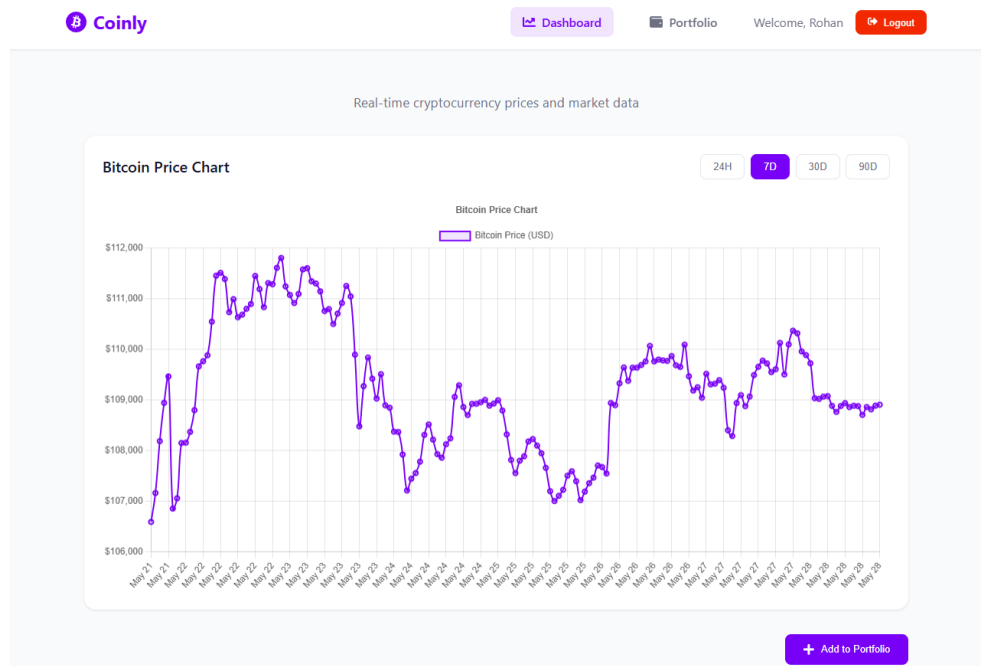


Figure 5.1: Dashboard

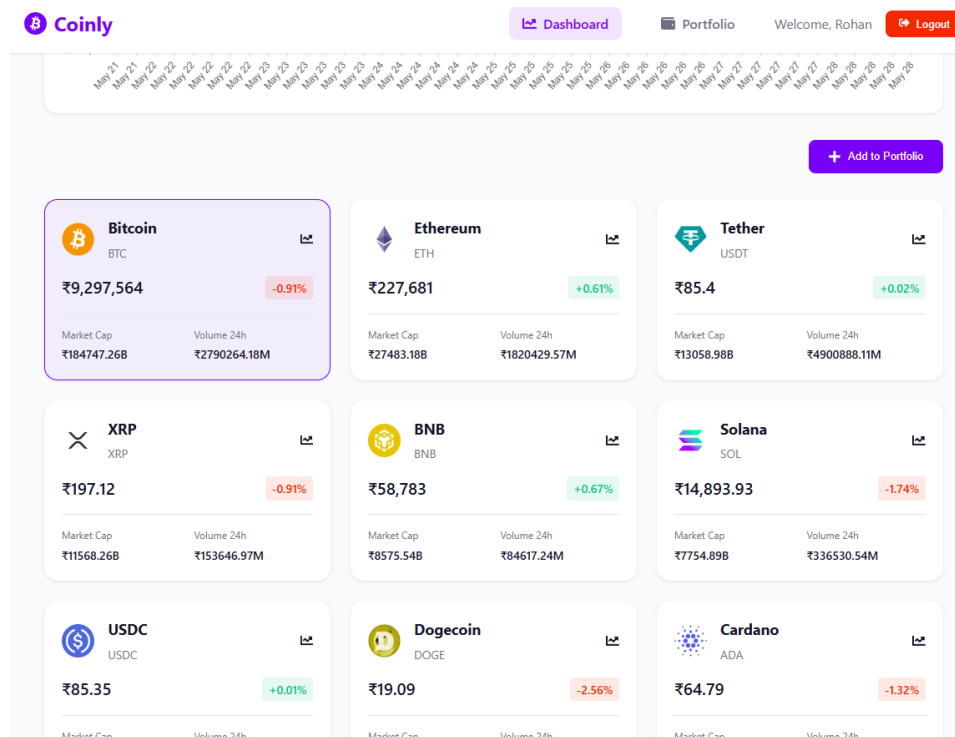


Figure 5.2: Live coins

Add to Portfolio

Select Coin

Select a coin

Amount

Enter amount

Purchase Price (USD)

Enter purchase price

Add to Portfolio

Cancel

Figure 5.3: Add Coin

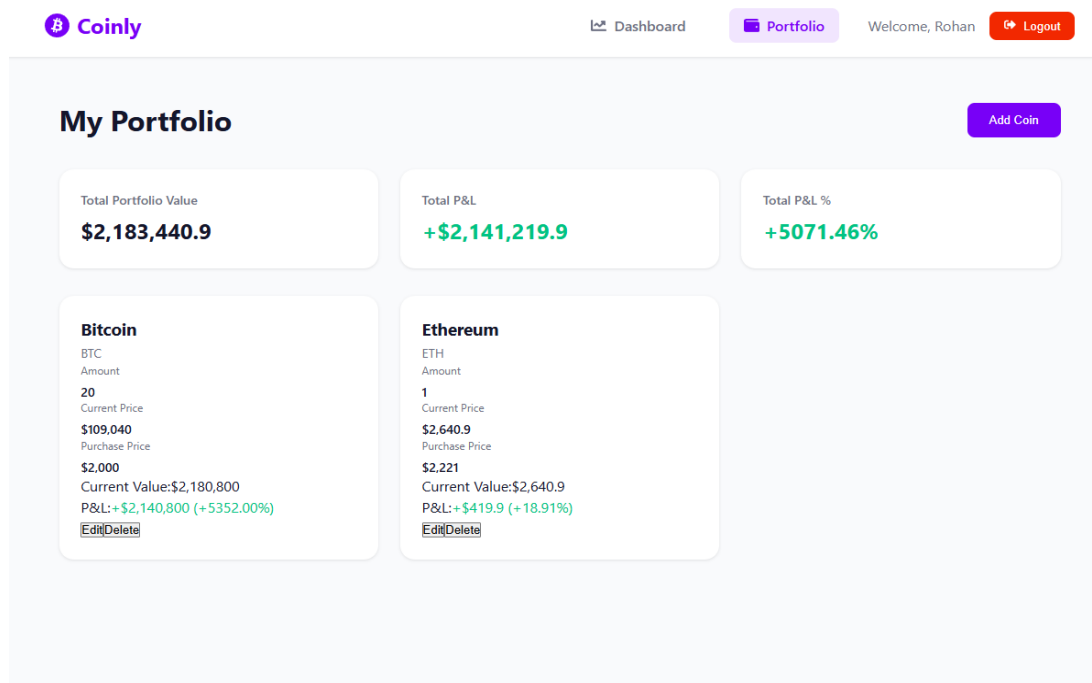


Figure 5.4: Portfolio

CONCLUSION

In closing, CoinlyCrypto exemplifies a modern, full-stack MERN application that brings together a unified JavaScript codebase, secure user management, real-time market data integration, and rich, interactive visualizations into a single cohesive platform. Beginning with MongoDB's adaptable document model, the backend effectively stores user profiles and portfolio entries, while Express and Node.js enforce robust RESTful endpoints, JWT-based authentication, and seamless communication with the external CoinGecko API. On the client side, React's component-driven architecture, state management, and hooks deliver a responsive user experience, enabling investors to add or remove positions with a click and immediately see up-to-date valuations. The integration of Chart.js empowers users to track their performance and asset allocation through dynamic line and pie charts, translating raw historical price feeds into actionable insights. Architecturally, the clear separation of concerns—database, API, business logic, and presentation layer—ensures maintainability, scalability, and rapid iteration, making CoinlyCrypto a strong foundation for future enhancements such as real-time WebSocket updates, portfolio sharing, or automated rebalancing features. By leveraging well-supported open-source modules and adhering to best practices in security and code organization, CoinlyCrypto not only serves as a valuable tool for cryptocurrency enthusiasts but also stands as a practical blueprint for building sophisticated web applications in today's JavaScript-centric ecosystem.

BIBLIOGRAPHY

Web References:

- React Official Documentation:
<https://react.dev/>
- Express.js Guide:
<https://expressjs.com/>
- Mongoose Documentation:
<https://mongoosejs.com/docs/>
- Axios GitHub Repository:
<https://github.com/axios/axios>
- MongoDB Manual:
<https://www.mongodb.com/docs/manual/>

Book References:

- **Pro MERN Stack:** *Full Stack Web App Development with Mongo, Express, React, and Node* – Vasani Subramanian, Apress, 2019.
- **Beginning MERN Stack:** *Build and Deploy a Full Stack MongoDB, Express, React, Node.js App* – Greg Lim, Independently Published, 2020.
- **Full-Stack React Projects** – Shama Hoque, Packt Publishing, 2020.
- **Node.js Design Patterns** – Mario Casciaro & Luciano Mammino, Packt Publishing, 2020.
- **JavaScript: The Definitive Guide** – David Flanagan, O'Reilly Media, 2020.

APPENDIX A: LIST OF FIGURES

1.1	MERN Stack Architecture	6
3.1	ER Diagram	9
3.2	Class Diagram	11
3.3	Activity Diagram	12
3.4	Use Case Diagram	14
5.1	Dashboard	19
5.2	Live Coins	19
5.3	Add Coin	20
5.4	Portfolio	20

APPENDIX B: LIST OF TABLES

4.1	Implementation Table	18
-----	----------------------	----