Hermetik Portfolio Management System - Complete System Specification

System Overview

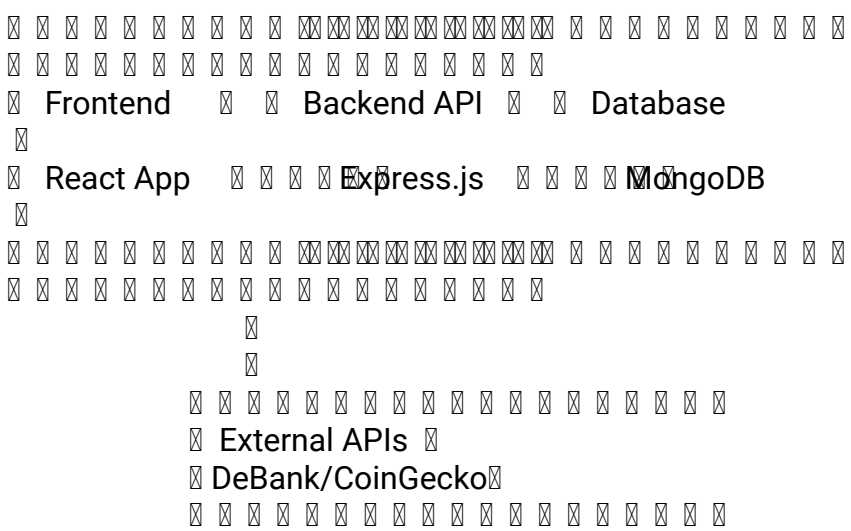
Hermetik is a comprehensive DeFi portfolio management platform that enables users to track, analyze, and manage their cryptocurrency investments across multiple wallets and protocols. The system provides real-time portfolio valuation, performance analytics, APY calculations, and administrative tools for fund management.

Architecture Overview

Technology Stack

- Frontend: React + TypeScript + Vite
- Backend: Node.js + Express.js
- Database: MongoDB
- External APIs: DeBank Pro API, CoinGecko API
- Authentication: JWT-based authentication
- Real-time Updates: HTTP polling with React Query

System Architecture



Core Modules

1. Authentication & User Management

User Roles

- Admin: Full system access, can view all users, export reports
- User: Personal portfolio access, wallet management

Authentication Flow

```
// JWT-based authentication
POST /api/auth/login
{
  "email": "user@example.com",
  "password": "password123"
}

Response:
{
  "token": "jwt_token",
  "user": {
    "id": "user_id",
    "email": "user@example.com",
    "role": "user|admin"
  }
}
```

User Schema

```
{
  _id: ObjectId,
  email: String, // unique
  password: String, // bcrypt hashed
  role: String, // 'admin' | 'user'
  wallets: [String], // Array of wallet addresses
  createdAt: Date,
  updatedAt: Date
}
```

2. Wallet Management System

Wallet Integration

- Multi-chain Support: Ethereum, BSC, Arbitrum, Polygon, Base, Optimism
- Real-time Data: Live portfolio valuation via DeBank Pro API

- Token Recognition: ERC-20 tokens, LP tokens, staking positions

Wallet Data Structure

```
{
  address: String,
  name: String, // Generated or user-defined
  tokens: [{
    symbol: String,
    name: String,
    amount: Number,
    price: Number,
    usd_value: Number,
    chain: String,
    logo_url: String,
    decimals: Number
  }],
  protocols: [{
    protocol_id: String,
    name: String, // "Uniswap V3", "Aave", etc.
    chain: String,
    net_usd_value: Number,
    positions: [{
      position_name: String,
      position_id: String,
      tokens: [TokenObject], // Supply tokens
      rewards: [TokenObject], // Reward tokens
      pool_id: String,
      health_rate: Number
    }]
  }],
  totalValue: Number,
  chainDistribution: Object,
  protocolDistribution: Object
}
```

Key Endpoints

```
// User's wallets
GET /api/wallet/wallets
// Admin viewing user wallets
GET /api/wallet/wallets?userId={userId}
// Individual wallet details
GET /api/wallet/{address}
// Protocol-specific data
GET /api/wallet/{address}/protocol/{protocolId}/{chainId}
```

3. Portfolio Analytics Engine

Real-time Valuation

- Token Valuation: DeBank API + CoinGecko price feeds
- Protocol Positions: DeFi position values from DeBank
- Cross-chain Aggregation: Unified portfolio view across all chains

Portfolio Metrics

```
{
  totalValue: Number, // Total portfolio USD value
  tokenValue: Number, // Simple token holdings
  protocolValue: Number, // DeFi positions value
  dailyReturn: Number, // 24h performance
  weeklyReturn: Number, // 7d performance
  monthlyReturn: Number, // 30d performance
  volatility: Number, // Price volatility
  sharpeRatio: Number, // Risk-adjusted returns
  maxDrawdown: Number // Maximum loss from peak
}
```

4. APY Calculation System

Snapshot-Based Tracking

- Daily Snapshots: Automatic portfolio snapshots on wallet access
- Historical Comparison: Position-by-position APY calculation
- Reward Token Tracking: Unclaimed rewards for yield calculation

APY Calculation Methods

1. New Positions (no history):
APY = (unclaimed_rewards / position_value) × 365 × 100%
2. Existing Positions (historical data):
APY = ((1 + period_return)^(365/days) - 1) × 100%

DailySnapshot Schema

```
{
  userId: ObjectId,
  walletAddress: String,
  date: Date,
  totalNavUsd: Number,
  tokensNavUsd: Number,
  positionsNavUsd: Number,
  tokens: [TokenSnapshot],
  positions: [{
    protocolId: String,
    protocolName: String,
    supplyTokens: [TokenObject],
    rewardTokens: [TokenObject], // Critical for APY
    totalUsdValue: Number
  }]
}
```

5. NAV Reporting System

Excel Export Functionality

- Admin Reports: Portfolio NAV reports for clients
- Historical Data: Time-series portfolio performance
- Breakdown Analysis: Token vs Protocol allocation

NAV Calculation Logic

```
// Hierarchical NAV calculation
const totalTokensValue = wallets.reduce((sum, wallet) => {
  return sum + wallet.tokens.reduce((tokenSum, token) =>
    tokenSum + token.usd_value, 0);
}, 0);

const totalPositionsValue = calculatePositionsValue(wallets);
const totalPortfolioNAV = totalTokensValue +
totalPositionsValue;
```

Export Endpoints

```
// Excel NAV report
GET /api/analytics/export/excel?userId={userId}
// Portfolio performance report
GET /api/analytics/portfolio/performance?period={days}
```

6. Administrative Dashboard

User Management

- User Overview: All registered users and their portfolios
- Portfolio Viewing: Admin can view any user's portfolio
- Report Generation: NAV reports for specific users/dates

Admin Features

```
// View all users
GET /api/admin/users
// View user's portfolio as admin
GET /api/wallet/wallets?userId={userId}
// Export user's NAV report
GET /api/analytics/export/excel?userId={userId}
// System analytics
GET /api/admin/analytics/summary
```

7. Data Standardization Service

Protocol Data Normalization

- Multi-source Integration: DeBank, protocol-specific APIs
- Data Validation: Price validation, outlier detection
- Error Handling: Fallback data sources, graceful degradation

Price Feed Management

```
// Price aggregation logic
const finalPrice = coinGeckoPrices[symbol] ||
                   debankPrice ||
                   fallbackPrice || 0;
```

Frontend Architecture

Component Structure

```
src/
   components/
      UI/                # Reusable UI components
```

```
├── components/
│   ├── Admin/              # Admin-specific components
│   ├── APY/                # APY display components
│   └── Portfolio/          # Portfolio components
├── pages/
│   ├── Dashboard.tsx       # Main dashboard
│   ├── Positions.tsx       # Detailed positions
│   ├── Analytics.tsx       # Performance analytics
│   └── Admin/              # Admin pages
├── services/
│   ├── api.ts              # API client
│   └── auth.ts             # Authentication
├── contexts/
│   ├── AuthContext.tsx     # Auth state management
│   └── UserViewContext.tsx # Admin user switching
└── types/
    └── index.ts            # TypeScript definitions
```

Key Frontend Features

Dashboard Components

- Portfolio Overview: Total NAV, daily performance
- Asset Breakdown: Tokens vs DeFi positions
- Performance Metrics: Returns, volatility, Sharpe ratio
- APY Display: Real-time yield calculations

Admin Interface

- User Switching: View any user's portfolio
- Export Controls: Generate NAV reports
- System Monitoring: User activity, system health

State Management

```
// React Query for server state
const { data: wallets } = useQuery({
  queryKey: ['wallets', userId],
  queryFn: () => walletApi.getWallets(userId),
  refetchInterval: 30000 // 30 second updates
});

// Context for admin user switching
const { viewedUser, switchToUser } = useUserView();
```

Database Schema

Core Collections

Users Collection

```
{
  _id: ObjectId,
  email: String,
  password: String, // bcrypt
  role: "admin" | "user",
  wallets: [String],
  createdAt: Date
}
```

DailySnapshots Collection

```
{
  _id: ObjectId,
  userId: ObjectId,
  walletAddress: String,
  date: Date,
  totalNavUsd: Number,
  tokensNavUsd: Number,
  positionsNavUsd: Number,
  tokens: [TokenSnapshot],
  positions: [PositionSnapshot],
  createdAt: Date
}
```

WalletData Collection (Cache)

```
{
  _id: ObjectId,
  userId: ObjectId,
  walletAddress: String,
  tokens: [TokenObject],
  protocols: [ProtocolObject],
  summary: SummaryObject,
  updatedAt: Date
}
```

Database Indexes

```
// Performance indexes
db.dailysnapshots.createIndex({ userId: 1, date: -1 });
db.dailysnapshots.createIndex({ userId: 1, walletAddress: 1,
date: 1 });
db.users.createIndex({ email: 1 });
db.walletdata.createIndex({ userId: 1, walletAddress: 1 });
```

External API Integration

DeBank Pro API

Purpose: Primary DeFi data source
Rate Limits: 1000 requests/minute
Key Endpoints:
- /user/token_list - User token holdings
- /user/protocol_list - DeFi protocol positions
- /user/protocol - Detailed protocol data

```
// DeBank API client
const debankClient = axios.create({
  baseURL: 'https://pro-openapi.debank.com/v1',
  headers: {
    'AccessKey': process.env.DEBANK_API_KEY,
    'Accept': 'application/json'
  }
});
```

CoinGecko API

Purpose: Token price validation and backup
Rate Limits: 100 requests/minute (free tier)
Key Endpoints:
- /simple/price - Current token prices
- /coins/markets - Market data

```
// Price aggregation with fallback
const getTokenPrice = (symbol) => {
  return coinGeckoPrice || debankPrice || 0;
};
```

Security Implementation

Authentication Security

- JWT Tokens: Stateless authentication
- Password Hashing: bcrypt with salt rounds
- Role-based Access: Admin vs User permissions

API Security

```
// Auth middleware
const auth = async (req, res, next) => {
  const token = req.headers.authorization?.split(' ')[1];
  if (!token) return res.status(401).json({ error: 'No token'
});


  try {
    const payload = jwt.verify(token, process.env.JWT_SECRET);
    req.user = await User.findById(payload.id);
    next();
  } catch {
    res.status(403).json({ error: 'Invalid token' });
  }
};
```

Data Validation

- Input Sanitization: All user inputs validated
- Rate Limiting: API endpoint rate limiting
- Error Handling: Graceful error responses without data leakage

Performance Optimization

Caching Strategy

- WalletData Cache: Stores recent wallet data
- React Query: Client-side API response caching
- Database Indexing: Optimized query performance

Data Processing

- Parallel API Calls: Concurrent wallet processing
- Background Tasks: Snapshot creation doesn't block responses
- Efficient Aggregation: MongoDB aggregation pipelines

Monitoring & Logging

```
// Structured logging
console.log(`🔄 Processing ${wallets.length} wallets...`);
console.log(`💰 Portfolio value: $${totalValue.toFixed(2)}`);
console.log(`📸 Created snapshot for ${walletAddress}`);
```

Deployment Configuration

Environment Variables

```
# Database
MONGO_URI=mongodb://127.0.0.1:27017/hermetikdb

# Authentication
JWT_SECRET=your_jwt_secret

# External APIs
DEBANK_API_KEY=your_debank_key
COINGECKO_API_KEY=your_coingecko_key

# Server
PORT=3001
NODE_ENV=production
```

Docker Configuration

```
# Backend Dockerfile
FROM node:18-alpine
WORKDIR /app
COPY package*.json ./
RUN npm ci --only=production
COPY . .
EXPOSE 3001
CMD ["node", "index.js"]
```

API Documentation

Authentication Endpoints

POST /api/auth/login

POST /api/auth/register
POST /api/auth/logout
GET /api/auth/verify

Wallet Endpoints

GET /api/wallet/wallets // User's wallets
GET /api/wallet/wallets?userId={id} // Admin view user wallets
GET /api/wallet/{address} // Specific wallet
GET /api/wallet/{address}/protocol/{protocolId}/{chainId}
GET /api/wallet/debug/snapshots // Debug endpoint

Analytics Endpoints

GET /api/analytics/positions/apy // Position APY data
GET /api/analytics/portfolio/performance // Portfolio performance
GET /api/analytics/portfolio/history // Historical data
GET /api/analytics/export/excel // NAV export

Admin Endpoints

GET /api/admin/users // All users
GET /api/admin/users/{id} // Specific user
PUT /api/admin/users/{id} // Update user
DELETE /api/admin/users/{id} // Delete user

Error Handling

Error Response Format

```
{
  "error": "Error description",
  "details": "Detailed error message",
  "code": "ERROR_CODE",
  "timestamp": "2025-08-19T17:30:00.000Z"
}
```

Common Error Scenarios

- Authentication Failures: 401/403 responses

- API Rate Limits: Exponential backoff retry
- Database Errors: Connection handling and retry logic
- External API Failures: Fallback data sources

Testing Strategy

Unit Testing

- Service Layer: Business logic testing
- API Endpoints: Request/response testing
- Data Models: Schema validation testing

Integration Testing

- Database Operations: CRUD operation testing
- External API Integration: Mock API responses
- Authentication Flow: Login/logout testing

Performance Testing

- Load Testing: Multiple concurrent users
- API Response Time: Sub-200ms response targets
- Database Query Performance: Index optimization

This comprehensive system specification covers all aspects of the Hermetik portfolio management platform, from authentication and wallet management to advanced APY calculations and administrative tools.