[역할 수정]

[박재성]: 프론트엔드
React 프로젝트 설정, 종속성 관리 및 핵심 페이지(Home, Login, Quiz 등) UI 구현

[배혜관]: 프론트엔드
메인 레이아웃(App.js), 거래 컴포넌트 및 핵심 기능 페이지(Market, Rank 등) UI 구현

[정송기]: 백엔드
백엔드(functions)시스템 설계 및 구현 (여러 함수, API/DB 로직), 인프라(Firebase) 설정

[로직 관련 내용]

1. 상단 바 및 홈페이지 (App.js, HomePage.js)

→ localhost:3000 접속 (npm start)

→ App.js가 상단 네비게이션 바를 표시

→ App.js의 onAuthStateChanged (Auth 리스너)가 실시간으로 로그인 상태를 감지

→
[로그아웃 상태] 상단 바 우측에 [로그인], [회원가입] 버튼이 표시
HomePage 본문에도 [로그인] 버튼이 표시

→
[로그인 상태] 상단 바 우측에 [이메일], [로그아웃] 버튼이 표시
HomePage 본문의 [로그인] 버튼은 자동으로 숨김

2. 회원가입 (RegisterPage.js)

→ [회원가입] 버튼 클릭.

→ 정보 입력 (이메일은 .ac.kr만, '학생 확인' 체크), (대학교 그룹 미구현)

→ [인증메일 발송] 버튼 클릭

→ createUserWithEmailAndPassword (Auth 계정 생성) + setDoc (DB 프로필 생성).

→ [Firestore] users 컬렉션에 문서가 생성

→ [Firestore] 상태: 인증대기 / 자산: 0 으로 저장

→ sendEmailVerification (인증 메일 발송).

→ "회원가입 요청이 완료되었습니다..." 알림이 뜹니다.

→ [확인] 버튼 클릭.

→ signOut(auth) (자동 로그인 강제 로그아웃).

→ Maps('/login') (로그인 페이지로 자동 이동).

3. 로그인 및 계정 활성화 (LoginPage.js)

→ [로그인] 버튼 클릭.

→ signInWithEmailAndPassword (Auth 로그인 시도).

→ (검증 1) user.emailVerified가 false (인증X) → "인증 필요" 알림 후 강제 로그아웃.

→ (사용자가 이메일 인증 링크를 클릭한 후) [로그인] 페이지에서 정보 입력.

→ (검증 2) user.emailVerified가 true (인증O) → users DB에서 status 조회.

→ status가 "pending_verification" (인증 대기) 상태인지 확인.

→ 백엔드 activateAccount 함수(user.ts) 호출.

→ [Firestore] activateAccount 함수가 상태: 활성화, 자산: 1,000만 원으로 업데이트.

→ "계정이 활성화되었습니다..." 알림

→ Maps('/') (홈페이지로 자동 이동).

4. 거래소(MarketPage.js + OrderForm.js) // 차트, 매수/매도

→ [거래소] 메뉴 클릭

→ 2단 레이아웃(차트 + 주문 폼) 표시.

→ (오른쪽 주문 폼) onSnapshot이 users DB를 실시간 감시

→ '주문 가능' 현금(예: 10,000,000 KRW) 표시.

→ (왼쪽 차트) 검색창에 'AAPL' 입력 → [검색] 버튼 클릭.

→ 백엔드 getMarketData 함수(market.ts) 호출.

→ getMarketData 함수가 Alpha Vantage API에서 'AAPL'의 100일치 캔들 차트 데이터를 가져옴

→ ApexCharts (차트 껍데기) 컴포넌트에 데이터를 전달하여 캔들 차트 표시.

→ 'AAPL' 종목 코드가 오른쪽 OrderForm 컴포넌트로 전달됨.

→ (오른쪽 주문 폼) 수량 '1' 입력 → [매수] 버튼 클릭.

→ 백엔드 buyAsset 함수(market.ts) 호출.

→ buyAsset 함수가 Finnhub API로 AAPL 현재가(예: $270) 조회.

→ 조회한 가격($270)에 고정 환율(1445)을 곱해 원화(KRW) 가치(351,000원)로 변환.

→ db.runTransaction (트랜잭션) 시작.

→ [Firestore] users DB의 virtual_asset (자산)이 351,000원보다 많은지 서버에서 검증.

→ (검증 통과) virtual_asset 차감 (→ 9,649,000원).

→ users/{userID}/holdings 하위 컬렉션에 AAPL 문서가 '평단가 351,000원'으로 기록.

→ (홈페이지 화면) onSnapshot이 DB 변경 감지 → '주문 가능' 현금이 9,649,000원으로 자동 갱신

5. 랭킹 (RankingPage.js)

→ [홈페이지 (포트폴리오 UC-6)] [자산] 메뉴 클릭.

→ 백엔드 getPortfolio 함수(portfolio.ts) 호출.

→
getPortfolio 함수가 [Firebase Console] (1) holdings (내 평단가), (2) Finnhub API (현재가) 조회.

→ 두 가격에 환율을 적용하여 (원화 현재가 - 원화 평단가) = '손익'을 계산.

→ 계산된 '총 자산', '총 손익'을 홈페이지 화면에 표시.

6. 자산 (PortfolioPage.js)

→ [홈페이지 (랭킹 UC-7)] [랭킹] 메뉴 클릭.

→ [Firebase Console] ranking/current_season 문서에 미리 계산된 랭킹 데이터를 즉시 조회.

→ 홈페이지 화면에 '개인 랭킹', '학교 랭킹' 테이블 표시.

→ (백엔드 자동화) calculateRankings 함수(portfolio.ts)가 5분마다 자동으로 실행

→ 이 함수가 모든 유저의 수익률을 (환율 적용하여) 계산하고, ranking/current_season 문서를 갱신

7. 게시판

→ [게시판 (UC-9)] [게시판] 클릭

→ [새 글 작성] UI에 입력

→ [글 등록] 버튼 클릭.

→ 백엔드 createPost 함수(community.ts) 호출.

→ [Firebase Console] posts 컬렉션에 user_id, created_at (서버 시간)과 함께 문서 생성.

→ CommunityPage.js가 posts 컬렉션 목록을 읽어와 화면에 표시.

8. 토론 배틀

→ [토론 배틀] 클릭

→ [Firebase Console] 관리자가 debates 컬렉션에 status: "progressing"으로 테스트 주제 문서를 수동 생성

→ DebatePage.js가 debates 컬렉션에서 status == "progressing"인 문서를 쿼리(조회).

→ 조회된 주제(예: "삼성전자 10만 원 간다 vs 못 간다")를 홈페이지 화면에 표시

→ (현재 찬반 투표 버튼은 UI만 구현되어 있으며, 상세한 로직은 미구현 상태임).

9. 퀴즈

→ [퀴즈] 클릭

→ 백엔드 checkQuizEligibility 함수(user.ts) 호출.

→ [Firebase Console] users DB에서 응시횟수가 2회 미만인지, 자산이 10% 미만인지(현재 true로 설정됨) 검증.

→ (검증 통과) quizzes DB에서 문제 로드

→ 정답 선택

→ [정답 제출] 버튼 클릭

→ 백엔드 submitQuizAnswer 함수(user.ts) 호출

→ [Firebase Console] quizzes DB의 정답과 비교

→ 정답이면 virtual_asset 200만 원 추가, quiz_try_cnt 1 증가. (오답이면 횟수 증가 안 함).

10. AI 챗봇
→ [AI 챗봇] 클릭

→ 질문 입력 후 [전송] 버튼 클릭

→ 백엔드 askChatbot 함수(community.ts) 호출.

→ askChatbot 함수가 Gemini API (gemini-2.5-flash)에 "투자 전문가처럼" 답변하라고 요청

→ AI가 생성한 답변을 받아와 홈페이지 채팅창에 표시.

11. 시즌 마감
→ [관리자] 메뉴 클릭.

→ [시즌 마감 실행] 버튼 클릭.

→ 백엔드 endSeason 함수(admin.ts) 호출.

→ [Firebase Console] endSeason 함수가 4가지 작업을 실행
(1) hall_of_fame (명예의 전당) 컬렉션에 최종 랭킹 저장
(2) holdings 하위 컬렉션(보유 종목)을 전부 삭제
(3) users 문서의 virtual_asset을 10,000,000으로, quiz_try_cnt를 0으로 리셋
(4) ranking/current_season 문서를 삭제하여 랭킹 페이지 초기화

12. 나머지 관리 기능
게시물 관리 (UC-14) → Firebase console에서 관리
토론 주제 관리 (UC-16) → Firebase console에서 관리
퀴즈 관리 (UC-17) → Firebase console에서 관리
공지사항 관리 (UC-18) → Firebase console에서 관리

13. 백엔드 코드
15개의 함수를 5개의 파일로 묶어서 분리.

[보완 해야 할 사항]

- CSS(스타일) 보완 작업
- 회원 가입 내용 추가/보완
- 대학교 데이터 그룹화
- 공지사항 페이지 추가
- 토론 배틀 로직 고민
- '총 투자 원금' 도입 (랭킹 왜곡 방지)
- 퀴즈 보상금 수익률에 미반영 하도록 수정
- ai 챗봇 개별 창으로 빼기
- 차트 기능 추가 (봉 종류, 인디케이터 등)
- 비밀번호 찾기(재설정) 기능
- 관리자 - 사용자 정지 기능 구현 (UC-13)

[토론 배틀 로직 방안]: 소속 학교 별 투표 수 증가 시키기

2. 토론 배틀 '투표' 로직 구현 (User's List #5 구체화)

"토론 배틀 로직 고민"을 구체화한 1단계입니다. 현재는 `debates` 컬렉션의 주제를 읽어오기만 할 뿐, 투표(쓰기)가 불가능합니다.

- **[보완 내용]**
  1. 사용자가 `DebatePage.js` 에서 [찬성] 또는 [반대] 버튼을 클릭합니다.
  2. `users` DB에서 사용자의 `school_name` (소속 학교)을 가져옵니다.
  3. `debates/{topicID}` 문서의 `votes` 맵(map) 필드에 `school_name` 기준으로 투표 수를 1 증가시키는 Cloud Function (`voteDebate`)을 구현합니다.

[랭킹 왜곡 방지]: 총 자산과 원금으로 분리

4. '총 투자 원금' 필드 도입 (랭킹 왜곡 방지)

위의 [정책 결정 1번]에서 이어지는 내용입니다. 퀴즈 보상금이 수익률을 왜곡하는 것을 막기 위한 기술적인 해결책입니다.

- **[보완 내용]**
  1. **[DB 수정]** `users` 컬렉션에 `total_investment` (총 투자 원금) 필드를 추가합니다. (`activateAccount` 함수가 `virtual_asset` 와 `total_investment` 모두 1000만 원을 지급)
  2. **[로직 수정]** `submitQuizAnswer` (퀴즈 정답) 함수가 `virtual_asset` (총 자산)은 200만 원 늘리지만, `total_investment` (원금)은 **건드리지 않도록** 수정합니다.
  3. **[로직 수정]** `getPortfolio` 및 `calculateRankings` 함수가 수익률을 계산하는 공식을 `(총 자산 - 총 투자 원금) / 총 투자 원금` 으로 변경합니다.