# 발표




💡 작성자 : 진실


작성 날짜 : 2025.03.27

작성 목표 : 주식 서비스 추가 기능 설명

## 🔧 기능 1: 플레이어 탈퇴 기능 추가

### 주요 변경 사항

- 외부에서 플레이어 id로 탈퇴 요청

```
const removePlayer = async (playerId) => {
  const player = props.players.find(p => p.playerId === playerId);
  if (!player) return;

  const ownedStocks = player.playerStocks?.filter(stock => stock.stockQua
```

```
  if (ownedStocks.length > 0) {
    alert(`보유한 주식이 남아 있으면 탈퇴할 수 없습니다.\n\n남아 있는 주식: ${own
    return;
  }

  if (confirm(`${playerId} 플레이어를 정말 탈퇴시키겠습니까?`)) {
    try {
      await axios.post('http://localhost:8080/api/removePlayer', null, {
        params: { player: playerId }
      });
      window.location.reload(); // 전체 페이지 새로고침
    } catch (error) {
      alert('플레이어 탈퇴 중 오류가 발생했습니다.');
    }
  }
};
```

- `SkalaStockMarket` 클래스에 `removePlayer(String id)` 메서드 추가

  → 이 메서드는 **자체적으로 삭제 로직을 구현하지 않고**, 역할을 `playerRepository`에 **넘깁니다.**

  ( **Wrapping** : `playerRepository.removePlayer(id)` 호출)

```
java
복사편집
public boolean removePlayer(String id) {
    return playerRepository.removePlayer(id);
}
```

- `PlayerRepository` 클래스에 `removePlayer(String id)` 메서드 추가

  → **플레이어 탈퇴(삭제)** 기능 구현

```
    // 7. 플레이어 탈퇴 (삭제) 후 파일에 저장
    public boolean removePlayer(String id) {
        Player playerToRemove = findPlayer(id);
        if (playerToRemove != null) {
```

```
            playerList.remove(playerToRemove);
            savePlayerList();
            return true; // 삭제 성공
        } else {
            return false; // 해당 ID 없음
        }
    }
```

### ⚠️ 추가 고려 사항

- API 요청 시, 해당 플레이어가 **보유 주식이 있는 경우**
  →
  **Alert(팝업)** 안내 후, **탈퇴 중단**
- `WebServer.java`에 **POST 요청 처리 API** 추가

  → 외부에서 **플레이어 탈퇴 요청 가능**

---

## 🔧 기능 2: 종목 추가 기능 구현

### 주요 변경 사항

- 외부에서 새로운 주식 이름과 주가 입력 및 추가 요청

```
const addStock = async () => {
  if (!newStockName.value || newStockPrice.value <= 0) {
    alert('종목명과 유효한 가격을 입력하세요.')
    return
  }

  try {
    await axios.post('http://localhost:8080/api/addStock', null, {
      params: {
        name: newStockName.value,
        price: newStockPrice.value
      }
    })
    alert('주식이 추가되었습니다!')
    newStockName.value = ''
    newStockPrice.value = 0
```

```
    emit('refresh') // 🔄 상위 컴포넌트에 목록 갱신 요청
  } catch (error) {
    alert('주식 추가 실패: 이미 존재하는 종목일 수 있어요.')
  }
}
```

- `SkalaStockMarket` 클래스는 `StockRepository.addStock(String name, int price)`에게 위임
  - 이 클래스는 **"서비스 계층"** 역할 → 클라이언트 요청을 받아서 처리만 하고, 중복 검사/저장 같은 **구체적인 비즈니스 로직은 레포지토리에 위임**

```
// 11. 주식 종목 추가
    public void addStock(String name, int price) {
        stockRepository.loadStockList();
        stockRepository.addStock(name, price);
    }
```

- `StockRepository.addStock(String name, int price)`

  → 중복 검사나 저장 등의 로직은 `stockRepository` 내부에서 처리
  - `StockRepository`는 **데이터 저장소 역할**에 충실
  - "주식이 중복되는지 확인 → 저장 리스트에 추가 → 파일 저장"까지의 **데이터 관리 책임**을 가짐

```
// 8. 주식 종목 추가 (중복 방지 포함)
    public void addStock(String name, int price) {
        Stock existing = findStock(name);
        if (existing == null) {
            stockList.add(new Stock(name, price));
            saveStockList(); // 파일에도 저장
        } else {
            System.out.println("이미 존재하는 주식입니다.");
        }
    }
```

## ⚠️ 추가 고려 사항

- `WebServer.java`에 **POST 요청 처리 API** 추가

⇒ **유지보수, 테스트, 확장성** 측면에서 더 유리

# 🔧 기능 3: 투자금 추가 기능 구현

## 주요 변경 사항

- `Player` 클래스에 투자금 추가 기능을 위한 `playerMoney`에 금액을 더하는 메서드를 새로 추가

```
// 투자금 추가 메서드
public void addMoney(int amount) {
    if (amount > 0) {
        this.playerMoney += amount;
    }
}
```

- SkalaStockMarket 클래스에서 API가 사용할 메서드 보완

```
// 특정 플레이어 반환
public Player getPlayerById(String playerId) {
    playerRepository.loadPlayerList();
    return playerRepository.findPlayer(playerId);
}

// 저장 메서드 노출
public void savePlayers() {
    playerRepository.savePlayerList();
}
```

## ⚠️ 추가 고려 사항

- `WebServer.java`에 **POST 요청 처리 API** 추가