



광석 게임

프론트 엔드 : https://gitlab.com/beethu1234/gathering_game

제작기간: 2023-04-24 ~ 2023.04.28

버전:v0.0.1

백 엔드 : https://gitlab.com/beethu1234/gathering_game_api

김영찬

1. 기획 의도
2. 소프트 웨어 구성
3. 목업 디자인
4. 기능 정의서 / ERD
5. Back-End
6. Front-End



Part 1

기획 의도



기획 의도



Wondershare
PDFelement

”

현재 앱 스토어 게임 부문에 가장 많은 수를 차지하고 있는 게임은 아무래도 방치형 혹은 오토 클리커 게임이라고 생각합니다.

바쁜 현대 사회에서 짧은 시간으로 언제 어디서나 즐길 수 있다는 장점이 굉장히 크기 때문일 것입니다.

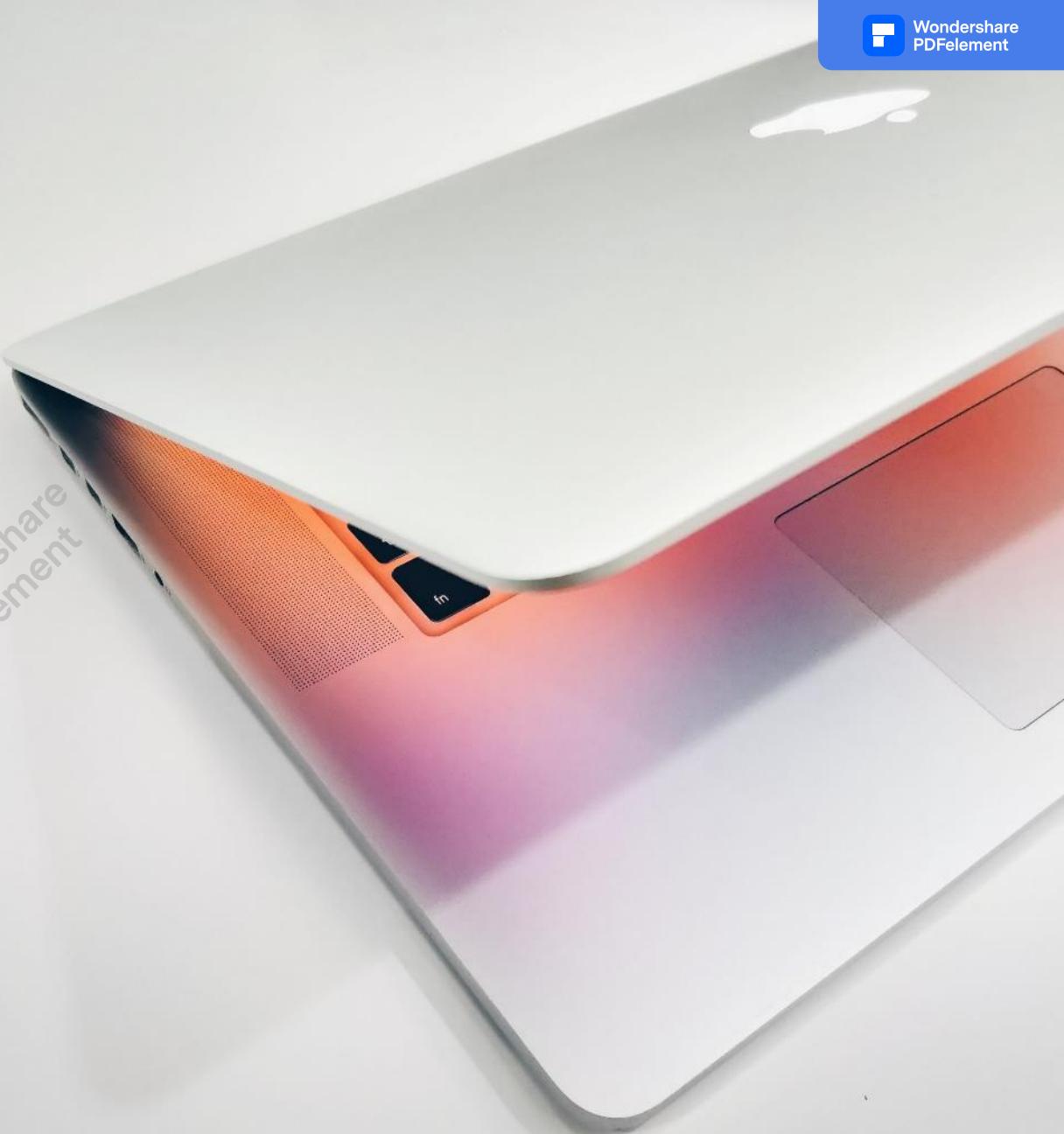
실제로 저도 방치형 게임을 즐기고 있고, 나름의 재미를 느끼다 보니, 어떤 방식으로 만들어질까 궁금해졌고 최소한의 게임 구동 방식을 구현하고 싶어져 만들게 되었습니다.

Part 2

소프트웨어 구성



Wondershare
PDFelement



Part 2 소프트웨어 구성



- Java
- Spring Boot
 - JPA
- Postgres



- Javascript
- vue.js
- Nuxt

Part 3

목업 디자인



Part 3 목업디자인

돈 벌기



소지 금액

5000G

광물 뽑기



나무 상자 : 5000G

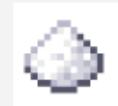


좋아 보이는 상자 : 15000G

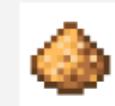
컬렉션



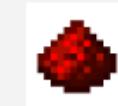
수량:



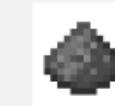
수량:



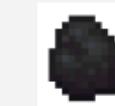
수량:



수량:



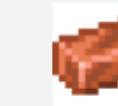
수량:



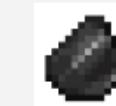
수량:



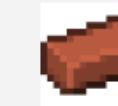
수량:



수량:



수량:



수량:



수량:



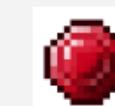
수량:



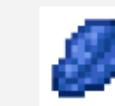
수량:



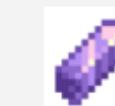
수량:



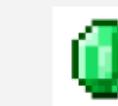
수량:



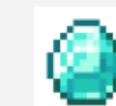
수량:



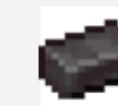
수량:



수량:



수량:



수량:

Part 4

기능 정의서 / ERD



Part 4 기능정의서 / ERD

기능 정의서

No	구분	기능명	기능 상세 설명	비고
1	백엔드	광석 캐기	한 번 클릭 시 100G 지급	
2	백엔드	소지 금액	"광석 캐기" 기능 사용 시 100G씩 증가	
3	백엔드	보석 뽑기 - 나무 상자	"소지 금액"에서 5,000G를 차감하고 "컬렉션" 중 하나를 확률에 따라 획득	뽑기 확률은 확률표 참고
4	백엔드	보석 뽑기 - 좋아 보이는 상자	"소지 금액"에서 15,000G를 차감하고 "컬렉션" 중 하나를 확률에 따라 획득	뽑기 확률은 확률표 참고
5	백엔드	컬렉션	"보석 뽑기"에서 얻은 모든 컬렉션을 보여준다.	컬렉션은 중복으로 획득 가능하며 수량이 늘어남
6	백엔드	다시하기	갖고 있는 컬렉션과 돈을 0으로 초기화한다.	

ERD

Wondershare
PDFelement

유저 스텝					
UserStat					
PK	AI	FK Null	Logical Name	Name	Type
✓	✓	✗		id	시퀀스
				moneyCount	보유 금액

보유 아이템					
UserCollection					
PK	AI	FK Null	Logical Name	Name	Type
✓	✓	✗		id	시퀀스
				gameItemStatId	게임 아이템 ID
				itemCount	보유 수량

게임 아이템					
GameItemStat					
PK	AI	FK Null	Logical Name	Name	Type
✓	✓	✗		id	시퀀스
				itemGrade	아이템 등급
				itemName	아이템 명
				woodBoxPercent	나무 상자 확률
				goodBoxPercent	좋아 보이는 상자 확률
				imageFileName	이미지 파일 명

광석 캐기 (수정 / 기본 값 0에서 100씩 추가)
-> 소지 금액 (불러오기)

게임 아이템 정보(뽑기 시스템 / 결과 아이템을 등급 별로 나누고 두 개의 상자 별로 확률을 다르게 설정)

컬렉션 (결과 아이템을 저장)

Part 5

Back-end



Part 5 Back-End (Entity)

```
@Entity
@Getter
@NoArgsConstructor(access = AccessLevel.PROTECTED)
public class UserStat {
    @ApiModelProperty(notes = "시퀀스")
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @ApiModelProperty(notes = "보유 금액")
    @Column(nullable = false)
    private Long moneyCount;

    public void plusMoneyCount() {
        this.moneyCount += 100;
    }

    public void minusMoneyCount(long boxPay) {
        this.moneyCount -= boxPay;
    }

    public void resetMoneyCount() {
        this.moneyCount = 0L;
    }

    private UserStat(userStatBuilder builder) {
        this.moneyCount = builder.moneyCount;
    }

    public static class userStatBuilder implements CommonModelBuilder<UserStat> {
        private final Long moneyCount;

        public userStatBuilder() {
            this.moneyCount = 0L;
        }

        @Override
        public UserStat build() {
            return new UserStat(this);
        }
    }
}
```

UserStat Entity (유저 정보)

Table 설정 :
Id (시퀀스 / GeneratedValue로 숫자 순서대로 알아서 설정)

moneyCount (보유 금액 / null값 불가 설정)

수정 기능(Builder) :
plusMoneyCount (광물 캐기 / moneyCount 값에 100씩 추가)
minusMoneyCount (상자 구매 / moneyCount 값에 상자 가격만큼 차감)
resetMoneyCount (게임 리셋 시 / moneyCount 값을 0으로 설정)

Builder 패턴을 이용한 값 설정 :
API 실행 시 moneyCount 값을 0으로 설정

Part 5 Back-End (Entity)

```
@Entity
@Getter
@NoArgsConstructor(access = AccessLevel.PROTECTED)
public class GameItemStat {
    @ApiModelProperty(notes = "시퀀스")
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @ApiModelProperty(notes = "등급 명")
    @Column(nullable = false, length = 10)
    @Enumerated(value = EnumType.STRING)
    private ItemGrade itemGrade;
    @ApiModelProperty(notes = "아이템 명")
    @Column(nullable = false, length = 20)
    private String itemName;
    @ApiModelProperty(notes = "나무 상자 확률")
    @Column(nullable = false)
    private Double woodBoxPercent;
    @ApiModelProperty(notes = "좋아 보이는 상자 확률")
    @Column(nullable = false)
    private Double goodBoxPercent;
    @ApiModelProperty(notes = "이미지 파일 명")
    @Column(nullable = false, length = 50)
    private String imageFileName;
```

GameItemStat Entity 1/2
(상자에서 나오는 아이템 정보)

Table 설정:

Id (시퀀스 / GeneratedValue로 숫자 순서대로 알아서 설정)

itemGrade (등급 명 / enum에 설정한 값 / Enumerated를 통해 String값으로 가져오기)

itemName (아이템 명 / 상자에서 나오는 아이템 이름)

woodBoxPercent (나무 상자 확률 / 나무 상자에서 나오는 아이템들의 각 확률)

goodBoxPercent (좋아 보이는 상자 확률 / 좋아 보이는 상자에서 나오는 아이템들의 각 확률)

imageFileName (이미지 파일 명 / 프론트 엔드에서 사용할 이미지 파일 명 ex: h123.png)

Part 5 Back-End (Entity)

```
private GameItemStat(gameItemStatBuilder builder) {  
    this.itemGrade = builder.itemGrade;  
    this.itemName = builder.itemName;  
    this.woodBoxPercent = builder.woodBoxPercent;  
    this.goodBoxPercent = builder.goodBoxPercent;  
    this.imageFileName = builder.imageFileName;  
}  
  
public static class gameItemStatBuilder implements CommonModelBuilder<GameItemStat> {  
  
    private final ItemGrade itemGrade;  
    private final String itemName;  
    private final Double woodBoxPercent;  
    private final Double goodBoxPercent;  
    private final String imageFileName;  
  
    public gameItemStatBuilder(GameItemStatRequest request) {  
        this.itemGrade = request.getItemGrade();  
        this.itemName = request.getItemName();  
        this.woodBoxPercent = request.getWoodBoxPercent();  
        this.goodBoxPercent = request.getGoodBoxPercent();  
        this.imageFileName = request.getImageFileName();  
    }  
    @Override  
    public GameItemStat build() {  
        return new GameItemStat(this);  
    }  
}
```

GameltemStat Entity 2/2
(상자에서 나오는 아이템 정보)

Builder 패턴을 이용한 값 설정:
Request를 통해 값을 받아와 초기 값을 설정 한다.

Part 5 Back-End (Entity)

```
@Entity
@Getter
@NoArgsConstructor(access = AccessLevel.PROTECTED)
public class UserCollection {
    @ApiModelProperty(notes = "시퀀스")
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @ApiModelProperty(notes = "게임 아이템 ID")
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(nullable = false, name = "gameItemStatId")
    private GameItemStat gameItemStat;
    @ApiModelProperty(notes = "보유 수량")
    @Column(nullable = false)
    private Integer itemCount;

    public void plusItemCount() {
        this.itemCount += 1;
    }
    public void resetItemCount() {
        this.itemCount = 0;
    }

    private UserCollection(UserCollectionBuilder builder) {
        this.gameItemStat = builder.gameItemStat;
        this.itemCount = builder.itemCount;
    }

    public static class UserCollectionBuilder implements CommonModelBuilder<UserCollection> {
        private final GameItemStat gameItemStat;
        private final Integer itemCount;

        public UserCollectionBuilder(GameItemStat gameItemStat) {
            this.gameItemStat = gameItemStat;
            this.itemCount = 0;
        }

        @Override
        public UserCollection build() {
            return new UserCollection(this);
        }
    }
}
```

UserCollection Entity (보유 아이템 정보 / 컬렉션)

Table 설정:

Id (시퀀스 / GeneratedValue로 숫자 순서대로 알아서 설정)

gameItemStat (게임 아이템 entity / ManyToOne으로 FK 참조)

itemCount (보유 수량 / 보유 중인 아이템의 수 표시)

plusItemCount (아이템 중복 획득 시 / 아이템의 수 1씩 증가)

resetItemCount (게임 리셋 시 / 아이템 보유 수량 0으로 초기화)

Builder 패턴을 이용한 초기값 설정

Part 5 Back-End (Service)

```
@Service
@RequiredArgsConstructor
public class UserStatService {
    private final UserStatRepository userStatRepository;

    public UserStatResponse putMoneyCountPlus() {
        UserStat money = userStatRepository.findById(1L).orElseThrow(CMissingDataException::new);
        money.plusMoneyCount();
        UserStat result = userStatRepository.save(money);

        return new UserStatResponse.UserStatResponseBuilder(result).build();
    }
}
```

UserStatService (광물 캐기)

UserStatRepository를 참조

Repository에서 저장된 UserStat 값을 가져와
plusMoneyCount 실행
(moneyCount에 100씩 추가)

가공된 값을 다시 Repository에 저장한 뒤
Builder 패턴을 통해 값을 저장.

Part 5 Back-End (Service)

```
@Service
@RequiredArgsConstructor
public class InitDataService {
    private final UserStatRepository userStatRepository;
    private final UserCollectionRepository userCollectionRepository;
    private final GameItemStatRepository gameItemStatRepository;

    public void setFirstMoney() {
        Optional<UserStat> originData = userStatRepository.findById(1L);

        if (originData.isEmpty()) {
            UserStat addData = new UserStat.userStatBuilder().build();

            userStatRepository.save(addData);
        }
    }

    public void setFirstGameItem() {
        List<GameItemStat> originList = gameItemStatRepository.findAll();

        if (originList.size() == 0) {
            List<GameItemStatRequest> result = new LinkedList<>();

            GameItemStatRequest request1 = new GameItemStatRequest.GameItemStatRequestBuilder(
                ItemGrade.COMMON, "뼈디귀", 10D, 5D, "bone.png").build();
            result.add(request1);
            GameItemStatRequest request2 = new GameItemStatRequest.GameItemStatRequestBuilder(
                ItemGrade.COMMON, "정체 모를 가루", 10D, 5D, "sugar.png").build();
            result.add(request2);
            GameItemStatRequest request3 = new GameItemStatRequest.GameItemStatRequestBuilder(
                ItemGrade.COMMON, "발광석 가루", 10D, 5D, "light.png").build();
            result.add(request3);
        }
    }
}
```

InitDataService 1/2
(API 첫 실행 시 초기 값 설정)

API를 처음으로 실행할 때 (값이 비어있을 때)

setFirstMoney: moneyCount 값을 0으로 설정

setFirstGameItem: 상자에서 나오는 아이템 정보를 설정

GameItemStat Entity에 들어가는 request 값을 Request의 Builder를 통해 초기 설정 값을 넣는다.

Part 5 Back-End (Service)

```
GameItemStatRequest request19 = new GameItemStatRequest.GameItemStatRequestBuilder(  
    ItemGrade.UNIQUE, "다이아몬드", 1.2, 2.25, "diamond.png").build();  
result.add(request19);  
GameItemStatRequest request20 = new GameItemStatRequest.GameItemStatRequestBuilder(  
    ItemGrade.LEGENDARY, "전설의 광석", 0.2, 10, "legendary.png").build();  
result.add(request20);  
  
result.forEach(item -> {  
    GameItemStat addData = new GameItemStat.gameItemStatBuilder(item).build();  
  
    gameItemStatRepository.save(addData);  
});  
}  
  
public void setFirstUserCollection() {  
    List<UserCollection> userCollections = userCollectionRepository.findAll();  
  
    if (userCollections.size() == 0) {  
        List<GameItemStat> gameItemStats = gameItemStatRepository.findAll();  
  
        gameItemStats.forEach(item -> {  
            UserCollection addData = new UserCollection.UserCollectionBuilder(item).build();  
  
            userCollectionRepository.save(addData);  
        });  
    }  
}
```

InitDataService 2/2
(API 첫 실행 시 초기 값 설정)

setFirstUserCollection:
userCollection 데이터를 List형태로 전부 가져왔을 때 데이터가 하나도 들어 있지 않다면 /
FK인 GameItemStat의 데이터 리스트를 가져와 Builder패턴을 통해 userCollection Entity에 값을 넣는다.

Part 5 Back-End (Service)

```
@Component  
@RequiredArgsConstructor  
public class WebRunner implements ApplicationRunner {  
    private final InitDataService initDataService;  
  
    @Override  
    public void run(ApplicationArguments args) throws Exception {  
        initDataService.setFirstMoney();  
        initDataService.setFirstGameItem();  
        initDataService.setFirstUserCollection();  
    }  
}
```

WebRunner
(API 첫 실행 시 InitDataService를 실행)

InitDataService를 참조해 ApplicationRunner
를 구현

보유 금액,
상자에서 나오는 아이템 정보,
컬렉션의 보유 수량을 초기 값으로 설정

Part 5 Back-End (Service)

```

public List<UserCollectionItem> getMyItems() {
    List<UserCollection> originList =
    userCollectionRepository.findAllByIdGreaterThanOrEqualTo(1L);

    List<UserCollectionItem> result = new LinkedList<>();
    originList.forEach(item -> result.add(new
    UserCollectionItem.UserCollectionItemBuilder(item).build()));

    return result;
}

private boolean isEnoughMoneyCheck(boolean isWoodBox) {
    UserStat userStat = userStatRepository.findById(1L).orElseThrow(CMissingDataException::new);

    boolean result = false;
    if (userStat.getMoneyCount() >= openBoxPay) result = true;
    return result;
}

private long openBoxResult(boolean isWoodBox) {
    List<GameItemStat> gameItemStats = gameItemStatRepository.findAll();
    List<OpenBoxPercentItem> percentBar = new LinkedList<>();

    double oldPercent = 0;
    for (GameItemStat gameItemStat : gameItemStats) {
        OpenBoxPercentItem addItem = new OpenBoxPercentItem();

        addItem.setId(gameItemStat.getId());
        addItem.setPercentMin(oldPercent);
        if (isWoodBox) {
            addItem.setPercentMax(oldPercent + gameItemStat.getWoodBoxPercent());
        } else {
            addItem.setPercentMax(oldPercent + gameItemStat.getGoodBoxPercent());
        }

        percentBar.add(addItem);
        if (isWoodBox) {
            oldPercent += gameItemStat.getWoodBoxPercent();
        } else {
            oldPercent += gameItemStat.getGoodBoxPercent();
        }
    }
    double result = (Math.random() * 100);

    long resultId = 0;
    for (OpenBoxPercentItem item : percentBar) {
        if (result >= item.getPercentMin() && result <= item.getPercentMax()) {
            resultId = item.getId();
            break;
        }
    }
    return resultId;
}

```

OpenBoxService 1/2 (상자 구매)

아이템 획득 후 UserCollection 보유 수량을 증가 시키기 위해 List를 불러온다.

상자를 열기 위해 필요한 금액이 충분한지 UserStat(보유 금액) 데이터를 가져온다. 기본적으로 false (구매 불가) 값을 설정하고 보유 금액이 openBoxPay (상자 구매 금액) 보다 같거나 높을 경우 true (구매 가능)로 변환한다.

상자 구매 시 확률에 따라 아이템 획득 gameItemStat (상자에서 나오는 아이템 정보) 을 불러와 아이템을 100이라는 숫자에 확률에 따라 할당한다.
(ex : A와 B 아이템이 나무 상자에서 나오는 확률이 10%라면 A = 0 ~ 10, B = 10 ~ 20)
0 ~ 100의 실수 중 랜덤으로 하나를 Math.random 을 통해 가져오고 반복 문을 통해 위에서 할당한 범위에 해당하는 아이템 id를 결과 값에 넣는다.

Part 5 Back-End (Service)

```

@Service
@RequiredArgsConstructor
public class OpenBoxService {
    private final GameItemStatRepository gameItemStatRepository;
    private final UserCollectionRepository userCollectionRepository;
    private final UserStatRepository userStatRepository;

    long openBoxPay = 0L;

    private void init(boolean isWoodBox) {
        this.openBoxPay = isWoodBox ? 5000 : 15000;
    }

    public OpenBoxResultResponse getResult(boolean isWoodBox) {
        this.init(isWoodBox);

        boolean isEnoughMoney = this.isEnoughMoneyCheck(isWoodBox);

        if (!isEnoughMoney) throw new CNotEnoughMoneyException();

        UserStat userStat = userStatRepository.findById(1L).orElseThrow(CMissingDataException::new);
        userStat.minusMoneyCount(this.openBoxPay);

        userStatRepository.save(userStat);

        long openBoxResultId = this.openBoxResult(isWoodBox);

        boolean isNewItem = false;
        Optional<UserCollection> userCollection =
        userCollectionRepository.findByGameItemStat_Id(openBoxResultId);
        if (userCollection.isEmpty()) throw new CMissingDataException();
        UserCollection userCollectionData = userCollection.get();

        if (userCollectionData.getItemCount() == 0) isNewItem = true;

        userCollectionData.plusItemCount();
        userCollectionRepository.save(userCollectionData);

        OpenBoxResultResponse result = new OpenBoxResultResponse();
        result.setUserStatResponse(new UserStatResponse.UserStatResponseBuilder(userStat).build());
        result.setNewItem(new
        OpenBoxResultItem.OpenBoxResultItemBuilder(userCollectionData.getGameItemStat(), isNewItem).build());
        result.setUserCollectionItems(this.getMyItems());

        return result;
    }
}

```

OpenBoxService 2/2 (상자 구매)

Boolean형태의 isWoodBox(상자 종류 여부)를 요청한 뒤 맞다면 5000, 아니라면 15000을 openBoxPay(상자 를 열 때 필요한 금액)에 넣는다.

위에 init을 불러와 상자를 열 금액을 가져온다.
돈이 충분한지 확인한 후에 아니라면 돈이 부족하다는 메시지 출력.
충분하다면 minusMoneyCount(보유 금액에서 상자 가격 만큼 차감)후 원본에 저장.

상자에서 나온 아이템 결과 (id) 를 가져온 뒤 userCollection 데이터에서 그 결과 (id) 에 해당하는 데이터를 가져와 수량을 1 증가 시킨다.

/혹시라도 initDataService에 오류가 생기거나 하는 이유로 userCollection에 데이터가 없다면 에러 메시지를 출력

new 생성자를 이용해 결과값을 반환

Part 5 Back-End (Service)

```
@Service
@RequiredArgsConstructor
public class GameDataService {
    private final UserStatRepository userStatRepository;
    private final UserCollectionRepository userCollectionRepository;

    public FirstConnectDataResponse getFirstData() {
        FirstConnectDataResponse result = new FirstConnectDataResponse();

        UserStat userStat = userStatRepository.findById(1L).orElseThrow(CMissingDataException::new);
        result.setUserStatResponse(new UserStatResponse.UserStatResponseBuilder(userStat).build());
        result.setUserCollectionItems(getMyItems());

        return result;
    }

    public List<UserCollectionItem> getMyItems() {
        List<UserCollection> originList =
        userCollectionRepository.findAllByIdGreaterThanOrEqualTo(1L);

        List<UserCollectionItem> result = new LinkedList<>();
        originList.forEach(item -> result.add(new UserCollectionItem.UserCollectionItemBuilder(item).build()));

        return result;
    }

    public void gameReset() {
        UserStat userStat = userStatRepository.findById(1L).orElseThrow(CMissingDataException::new);
        userStat.resetMoneyCount();
        userStatRepository.save(userStat);

        List<UserCollection> originList = userCollectionRepository.findAll();
        for (UserCollection item : originList) {
            item.resetItemCount();
            userCollectionRepository.save(item);
        }
    }
}
```

GameDataService (게임 데이터 관리)

getFisrtData:

접속 시 이전 데이터 (보유 금액, 보유 아이템) 를 가져온다. //id칸에 1L인 이유는 사용자가 한 명 뿐이기 때문

getMyItems:

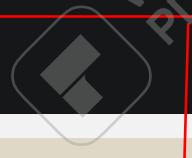
위에서 사용할 원본 데이터를 가져오기 위해 repository를 참조해 List형태의 보유 아이템 데이터를 가져온다.

gameReset:

게임 정보를 초기화 시킨다.
resetMoneyCount와 resetItemCount (각각 보유 금액과 아이템 보유 수량을 0으로 만든다) 를 실행 시킨 후 각각의 repository에 저장한다.

Part 5 Back-End (Controller)

```
@Api(tags = "보유 금액 관리")
@RestController
@RequiredArgsConstructor
@RequestMapping("/v1/user-stat")
public class UserStatController {
    private final UserStatService userStatService;
    @ApiOperation(value = "클릭 시 돈 증가")
    @PutMapping("/money-plus")
    public SingleResult<UserStatResponse> putMoneyCountPlus() {
        return ResponseService.getSingleResult(userStatService.putMoneyCountPlus());
    }
}
```



UserStatController
(보유 금액 관리)

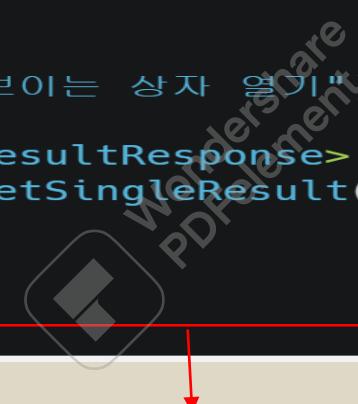
userStatService의 putMoneyCountPlus 메서드를
(보유 금액 100원씩 증가)
실행시키고 ResponseService를 통해 성공 메시지를 출력한다. / 기
능 구현 실패 시 실패 메시지 출력

Part 5 Back-End (Controller)

```
@Api(tags = "상자 열기")
@RestController
@RequiredArgsConstructor
@RequestMapping("/v1/open-box")
public class OpenBoxController {
    private final OpenBoxService openBoxService;

    @ApiOperation(value = "나무 상자 열기")
    @PostMapping("/wood")
    public SingleResult<OpenBoxResultResponse> openWoodBox() {
        return ResponseService.getSingleResult(openBoxService.getResult(true));
    }

    @ApiOperation(value = "좋아 보이는 상자 열기")
    @PostMapping("/good")
    public SingleResult<OpenBoxResultResponse> openGoodBox() {
        return ResponseService.getSingleResult(openBoxService.getResult(false));
    }
}
```



OpenBoxController
(상자 열기)

OpenBoxService의 getResult (보유 금액 차감, 아이템 획득 정보, 컬렉션 보유 수량 증가) 메서드를 실행한 후 ResponseService를 통해 메시지를 출력한다. (getResult의 요청 값(boolean isWoodBox) : true (나무 상자), false (좋아 보이는 상자))

Part 5 Back-End (Controller)

```
@Api(tags = "게임 데이터 관리")
@RestController
@RequiredArgsConstructor
@RequestMapping("/v1/game-data")
public class GameDataController {
    private final GameDataService gameDataService;

    @ApiOperation(value = "첫 접속 시 데이터 불러오기")
    @GetMapping("/init")
    public SingleResult<FirstConnectDataResponse> getFirstData() {
        return ResponseService.getSingleResult(gameDataService.getFirstData());
    }

    @ApiOperation(value = "게임 리셋")
    @DeleteMapping("/reset")
    public CommonResult gameReset() {
        gameDataService.gameReset();

        return ResponseService.getSuccessResult();
    }
}
```

GameDataController
(게임 데이터 관리)

getFirstData:
첫 접속 시 데이터 불러오기
gameDataService의 getFirstData (이전 보유 금액, 보유 아이템 정보 데이터를 가져온다) 메서드를 실행시키고 ResponseService를 통해 메시지를 출력한다.

gameReset:
게임을 초기화 시킨다.
gameDataService의 gameReset (보유 금액과 보유 아이템 수량을 0으로 만든다) 메서드를 실행시키고 ResponseService를 통해 메시지를 출력한다.

Part 6

Front-end



광석 게임 x pages/index.vue - main - moon x Swagger UI x Carbon | Create and share beau x Google x +

localhost:3000

돈 벌기

click ↓



광물 뽑기



보유 금액

0G

나무 상자 : 5000G

좋아 보이는 상자 : 15000G

컬렉션

?	?	?	?	?	?	?	?
???	???	???	???	???	???	???	???
?	?	?	?	?	?	?	?
???	???	???	???	???	???	???	???
?	?	?	?	?			
???	???	???	???	???			

초기화

기능 구현

Part 6 Front-End

돈 벌기

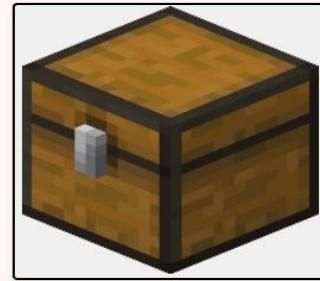
click ↓



보유 금액

3,100G

광물 뽑기



나무 상자 : 5000G

좋아 보이는 상자 : 15000G

컬렉션

???

정체 모를 가루
보유 : 1

???

???

화약 가루
보유 : 2석탄
보유 : 1숯
보유 : 1구리
보유 : 1

???

???

???

석영
보유 : 1

???

???

???

???

???

???

???

???

전체 화면

Part 6 Front-End

```
<div style="background-color: #ffff7f7">
  <div>
    <el-row :gutter="20" v-if="moneyCount != null">
      <el-col :span="12">
        <div class="header">
          돈 벌기
        </div>
        <div class="click-text">click ↓</div>
        <div style="text-align: center; margin-top: 30px">
          <button @click="gatheringMoney()" class="image-button">
            
          </button>
        </div>
        <div class="text">보유 금액</div>
        <div class="money">{{ moneyCount | currency }}G</div>
      </el-col>
    </el-row>
  </div>
</div>
```

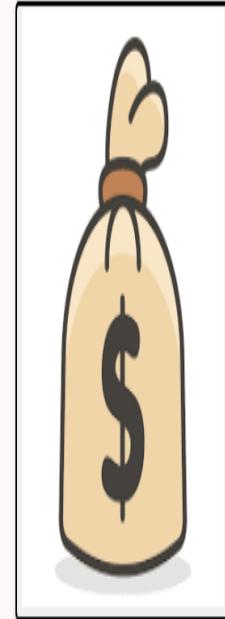
Front-End 코드

```
.header {
  border: 2px solid #BEBABA;
  border-radius: 5px;
  padding: 7px;
  font-size: 35px;
  font-weight: 700;
  text-align: center;
}
.money {
  border: 2px solid #BEBABA;
  border-radius: 5px;
  padding: 7px;
  font-size: 25px;
  font-weight: 600;
  text-align: center;
}
.text {
  margin-top: 30px;
  margin-bottom: 20px;
  font-size: 30px;
  font-weight: 600;
  text-align: center;
}
.click-text {
  font-size: 25px;
  text-align: center;
  font-weight: bold;
  margin-top: 5px;
  margin-bottom: -25px;
}
```

CSS

돈 벌기

click ↓



보유 금액

3,100G

구현 화면

Part 6 Front-End

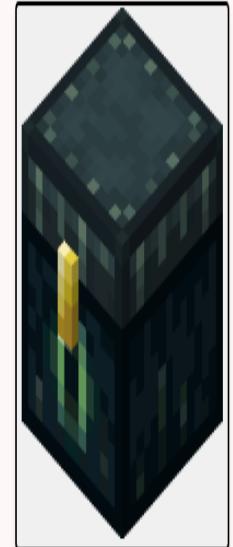
```
</el-col>
  <el-col :span="12">
    <div class="header">
      광물 뽑기
    </div>
    <div style="text-align: center">
      <div class="box">
        <div>
          <button @click="openWoodBox()" class="image-button">
        </button>
        </div>
        <div class="box-text">나무 상자 : 5000G</div>
      </div>
      <div class="box">
        <div>
          <button @click="openGoodBox()" class="image-button">
        </button>
        </div>
        <div class="box-text">좋아 보이는 상자 : 15000G</div>
      </div>
    </div>
  </el-col>
</el-row>
```

Front-End 코드

```
.header {
  border: 2px solid #BEBABA;
  border-radius: 5px;
  padding: 7px;
  font-size: 35px;
  font-weight: 700;
  text-align: center;
}
.box-text {
  font-size: 25px;
  font-weight: 600;
  text-align: center;
  margin-top: 84px;
  border: 2px solid #BEBABA;
  border-radius: 5px;
  padding: 7px;
}
.box {
  margin-top: 30px;
  margin-bottom: 20px;
  width: 50%;
  float: left;
}
```

CSS

광물 뽑기



나무 상자: 5000G

좋아 보이는 상자: 15000G

구현 화면

Part 6 Front-End

```


<div class="header">컬렉션</div>
  <el-row :gutter="20">
    <el-col :span="3" class="collection-center" v-for="(item, index) in collectionList"
      v-bind:key="index">
      <div v-if="item.itemCount > 0">
        <div>
          
        </div>
        <div>
          <el-badge :value="item.itemGradeName" class="item collection-text" :type="gradeValueType(item.itemGrade)">
            {{ item.itemName }}&ampnbsp&ampnbsp
          </el-badge>
        </div>
        <div class="collection-text">
          보유 : {{ item.itemCount }}
        </div>
      </div>
      <div v-else>
        <div>
          </div>
        <div>
          <el-badge :value="''??''" class="item collection-text">
            ???&ampnbsp&ampnbsp
          </el-badge>
        </div>
        <div class="collection-text">
          ???
        </div>
      </div>
    </el-col>
  </el-row>
</div>


```



구현 화면

Front-End 코드

```

.collection-center {
  text-align: center;
  border: 2px solid #BEBABA;
  margin-top: 10px;
}
.collection-text {
  font-size: 18px;
  font-weight: bold;
}

```

CSS

Part 6 Front-End

```
<el-dialog title="뽑기 결과" :visible.sync="dialogVisible" width="30%>
  <div class="box-result" v-if="resultItem != null">
    <div>
      
    </div>
    <div style="margin-bottom: 5px">{{ resultItem.itemGradeName }}</div>
    <div style="margin-bottom: 5px">{{ resultItem.itemName }}</div>
    <div v-if="resultItem.isNew">새로운 컬렉션 !</div>
  </div>
  <span slot="footer" class="dialog-footer">
    <el-button type="primary" @click="dialogVisible = false">확인</el-
button>
  </span>
</el-dialog>
<el-popover
  placement="top"
  width="160"
  v-model="resetVisible">
  <p style="font-size: 20px">초기화 하시겠습니까?</p>
  <div style="text-align: right; margin: 0; padding: 5px">
    <button class="reset-agree-button" @click="gameReset(resetVisible =
false)">예</button>
    <button class="reset-no-button" @click="resetVisible = false">취소
    </button>
  </div>
  <button slot="reference" class="reset-button">초기화</button>
</el-popover>
</div>
```

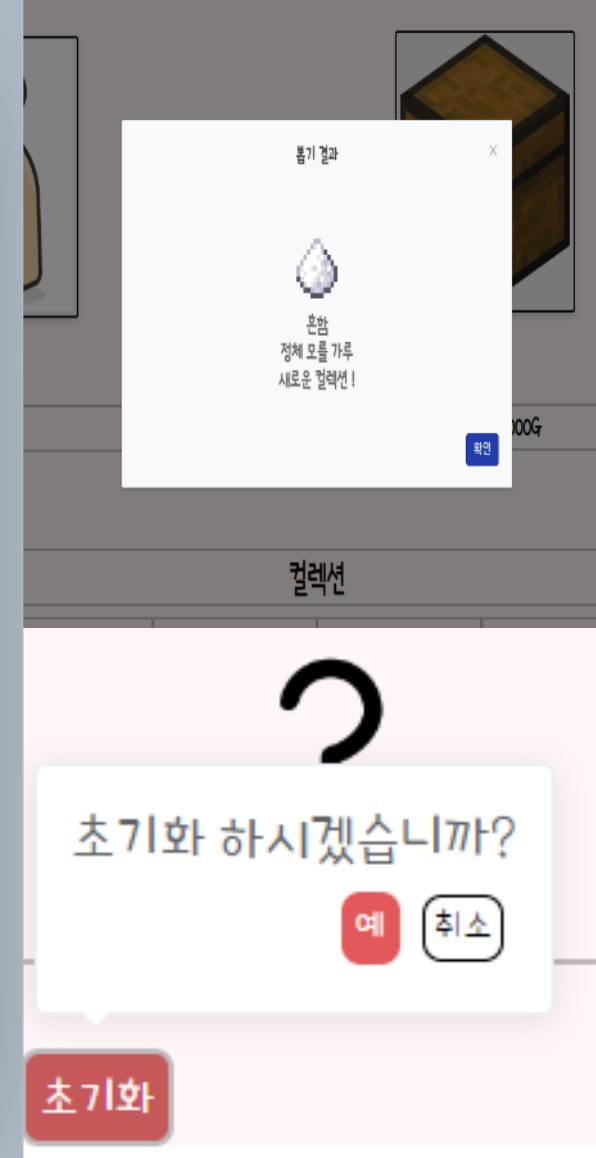
Front-End 코드

```
.reset-agree-button {
  font-size: 13px;
  color: white;
  background-color: #e35858;
  border: 2px solid #e35858;
  border-radius: 7px;
  cursor: pointer;
  text-align: right;
  font-family: 'Poor Story', cursive;
  padding: 3px;
  margin-right: 5px;
}

.reset-no-button {
  font-size: 13px;
  color: black;
  background-color: white;
  border: 1px solid black;
  border-radius: 7px;
  cursor: pointer;
  text-align: right;
  font-family: 'Poor Story', cursive;
  padding: 3px;
}

.box-result {
  font-size: 20px;
  font-weight: bold;
  text-align: center;
  margin-top: 10px;
}
```

CSS

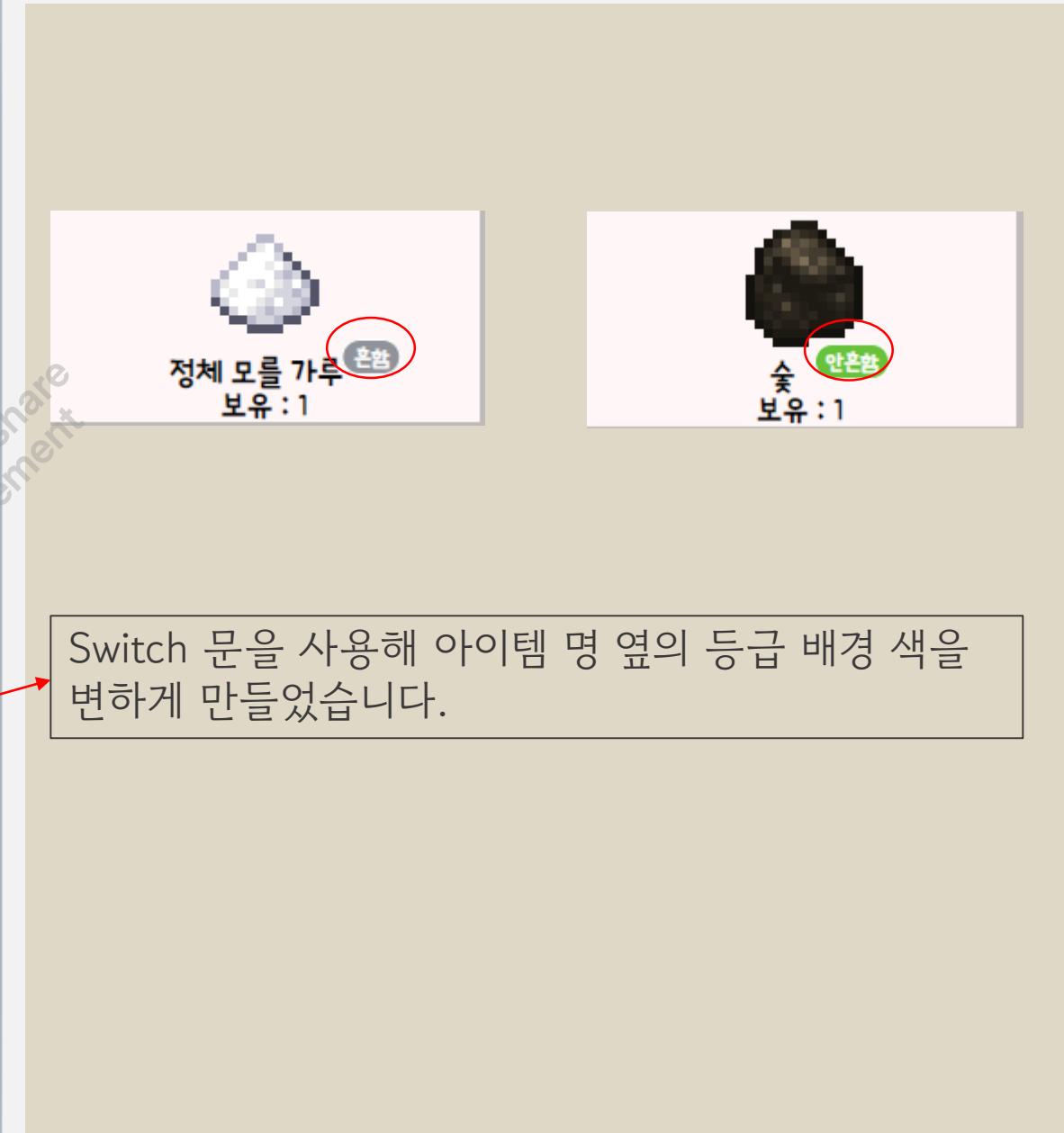


구현 화면

Part 6 Front-End

```
<script>
export default {
  data() {
    return {
      moneyCount: null,
      collectionList: [],
      dialogVisible: false,
      resultItem: null,
      resetVisible: false
    }
  },
  mounted() {
    this.$store.commit(this.$menuConstants.FETCH_SELECTED_MENU, 'DASH_BOARD')
  },
  methods: {
    gradeValueType(itemGrade) {
      let result = ''

      switch (itemGrade) {
        case 'COMMON':
          result = 'info'
          break
        case 'UNCOMMON':
          result = 'success'
          break
        case 'SPECIAL':
          result = 'warning'
          break
        case 'UNIQUE':
          result = 'danger'
          break
        case 'LEGENDARY':
          result = 'primary'
          break
        default:
          result = ''
      }
      return result
    }
  }
}
```



Part 6 Front-End

```
getFirstData() {
    this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, true)
    this.$store.dispatch(this.$getFirstDataConstants.GET_FIRST_DATA)
    .then((res) => {
        this.moneyCount = res.data.data.userStatResponse.moneyCount
        this.collectionList = res.data.data.userCollectionItems
        this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW,
false)
    })
    .catch((err) => {
        this.$toast.error(err.response.data.msg)
        this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW,
false)
    })
},
```

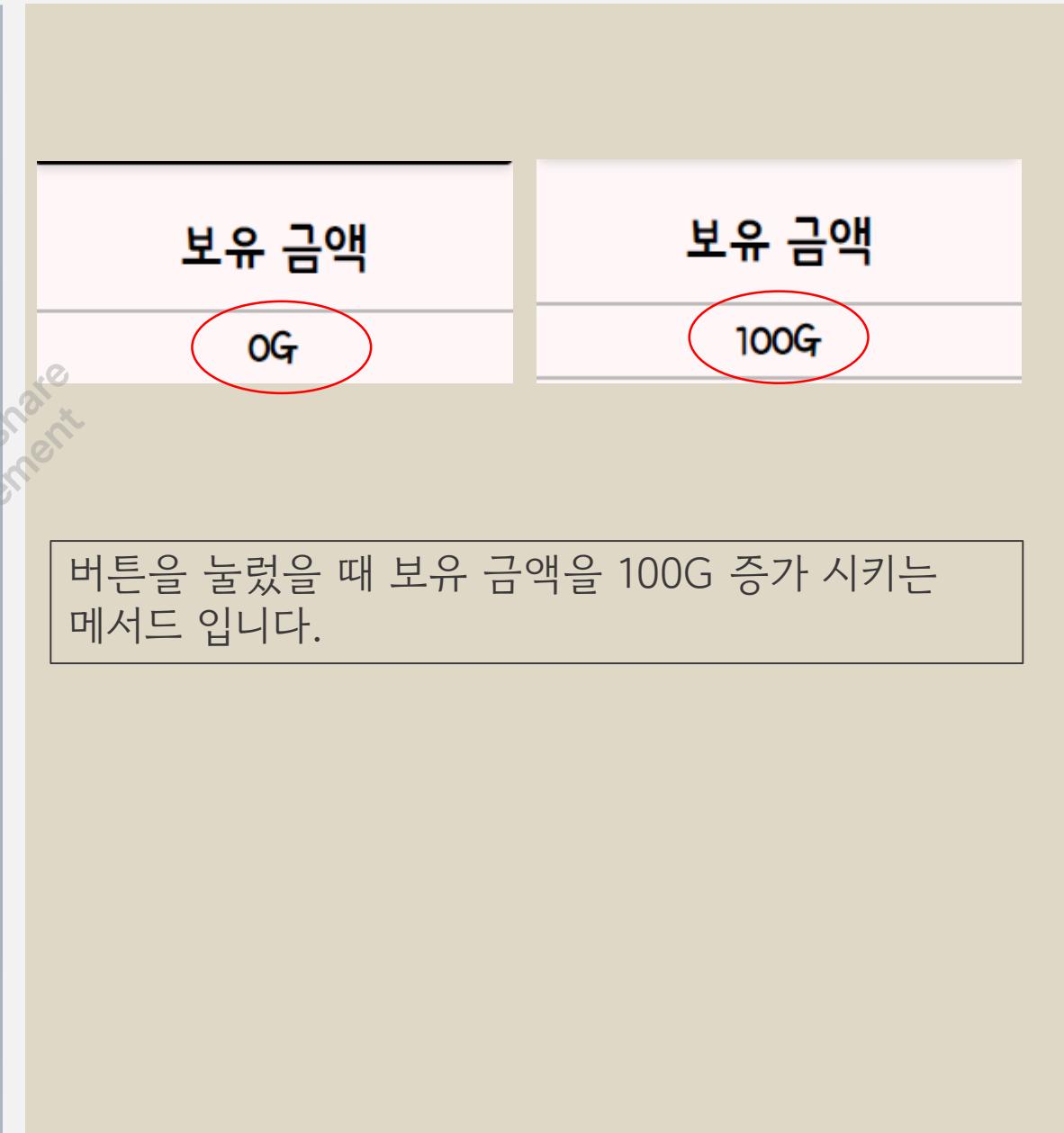
```
created() {
    this.getFirstData()
},
```

페이지를 열었을 때 FirstData (이전 보유 금액, 보유 아이템 컬렉션 정보)를 가져오게 하는 메서드입니다.

created (라이프 사이클) 를 이용해 처음 페이지가 열렸을 때 가장 먼저 getFirstData 메서드가 실행되게 합니다.

Part 6 Front-End

```
gatheringMoney() {
    this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, true)
    this.$store.dispatch(this.$gatheringMoneyConstants.GATHERING_MONEY)
    .then((res) => {
        this.moneyCount = res.data.data.moneyCount
        this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW,
false)
    })
    .catch((err) => {
        this.$toast.error(err.response.data.msg)
        this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW,
false)
    })
},
```



Part 6 Front-End

```
openWoodBox() {
    this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, true)
    this.$store.dispatch(this.$openWoodBoxConstants.OPEN_WOOD_BOX)
    .then((res) => {
        this.moneyCount = res.data.data.userStatResponse.moneyCount
        this.resultItem = res.data.data.resultItem
        this.collectionList = res.data.data.userCollectionItems
        this.dialogVisible = true
        this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW,
false)
    })
    .catch((err) => {
        this.$toast.error(err.response.data.msg)
        this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW,
false)
    })
},
openGoodBox() {
    this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, true)
    this.$store.dispatch(this.$openGoodBoxConstants.OPEN_GOOD_BOX)
    .then((res) => {
        this.moneyCount = res.data.data.userStatResponse.moneyCount
        this.resultItem = res.data.data.resultItem
        this.collectionList = res.data.data.userCollectionItems
        this.dialogVisible = true
        this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW,
false)
    })
    .catch((err) => {
        this.$toast.error(err.response.data.msg)
        this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW,
false)
    })
},
}
```



버튼을 눌렀을 때 보유 금액이 차감 되며 특정 아이템을 획득했다는 창이 뜨게 하고, 컬렉션에 추가 되게 하는 메서드입니다.

Part 6 Front-End

```
gameReset() {
  this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, true)
  this.$store.dispatch(this.$gameResetConstants.GAME_RESET)
  .then((res) => {
    this.getFirstData()
  })
  .catch((err) => {
    this.$toast.error(err.response.data.msg)
    this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW,
  false)
  })
},
```

