유물 모으기 게임

황상길

목차

Part 1, 기획 의도

기획 의도

평소에 게임을 좋아하다 보니 뽑기가 있는 게임을 많이 접해 보았습니다. 그러다 어떻게 뽑기프로그램이 돌아가는지 궁금하다 생각이 들어 간단하게 확률을 정해 직접 뽑기게임을 만들어 보자 생각이 들었습니다.

Part 2, 사용 기술

사용 기술

Back End

1	Java
2	Spring Boot
3	Postgres
4	JPA

Front End

1	JavaScript
2	Vue.js
3	Nuxt

Part 3, 요구 기능 정의서

요구 기능 정의서

	A	B	C	D	E
1	no	구분	기능명	기능 상세 설명	비고
2	1	백엔드	광석캐기	1요청당 500원을 지급	
3	2	백엔드	유물뽑기 은상자	5000원을 차감하고 노말~전설 유물중 하나를 랜덤으로 뽑는다.	확률표 참고
4	3	백엔드	유물뽑기 금상자	35000원을 차감하고 레어~신화 유물중 하나를 랜덤으로 뽑는다.	확률표 참고
5	4	백엔드	유물 컬렉션	내가 보유하고 있는 유물들을 보여준다.	유물은 중복획득이 가능하며 유물의 수량이 늘어난다.
6	5	백엔드	새로하기	광석과 유물을 0으로 초기화한다	
7					
8					

1. 광석 캐기 : 1요청(클릭)당 500원을 지급한다.

2. 유물 뽑기 은 상자: 5000원을 차감하고 노말~전설 유물중 하나를 랜덤으로 뽑는다.
   (확률 표 참고)
3. 유물 뽑기 금 상자: 35000원을 차감하고 노말~전설 유물중 하나를 랜덤으로 뽑는다.
   (확률 표 참고)
4. 유물 컬렉션: 내가 보유하고 있는 유물들은 보여준다.
   (유물은 중복 획득이 가능하며 유물의 수량이 증가한다.)
5. 새로 하기: 돈과 유물을 수량:0 으로 초기화한다.

확률 표

은 상자 확률 표

	A	B	C	D	E
1	no	등급	유물명	확률(%)	
2	1	노말	가죽 모자	17.5	
3	2	노말	가죽 갑옷	17.5	
4	3	노말	가죽 신발	17.5	
5	4	노말	몽둥이	17.5	
6	5	레어	철	5	
7	6	레어	철	5	
8	7	레어	철	5	
9	8	레어	철	5	
10	9	에픽	은	2	
11	10	에픽	은	2	
12	11	에픽	은	2	
13	12	에픽	은	2	
14	13	전설	금	0.5	
15	14	전설	금	0.5	
16	15	전설	금	0.5	
17	16	전설	금	0.5	
18					
19		합계		100	
20					
21	노말	레어	에픽	전설	
22	70	20	8	2	
23					

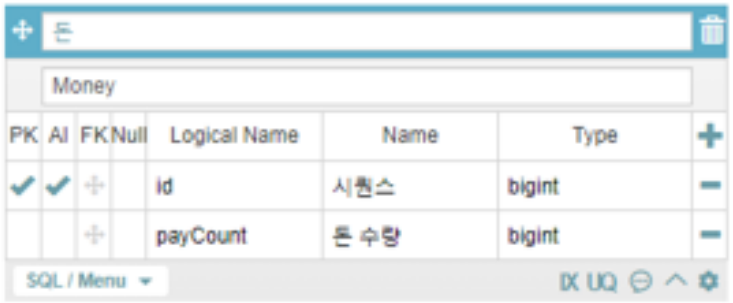
금 상자 확률 표

	A	B	C	D	E
1	no	등급	유물명	확률(%)	
2	1	레어	철 투구	15	
3	2	레어	철 갑옷	15	
4	3	레어	철 신발	15	
5	4	레어	철 검	15	
6	5	에픽	은	8.75	
7	6	에픽	은	8.75	
8	7	에픽	은	8.75	
9	8	에픽	은	8.75	
10	9	전설	금	1	
11	10	전설	금	1	
12	11	전설	금	1	
13	12	전설	금	1	
14	13	신화	다이아	0.25	
15	14	신화	다이아	0.25	
16	15	신화	다이아	0.25	
17	16	신화	다이아	0.25	
18					
19		합계		100	
20					
21	레어	에픽	전설	신화	
22	60	35	4	1	
23					

Part 4, E.R.D

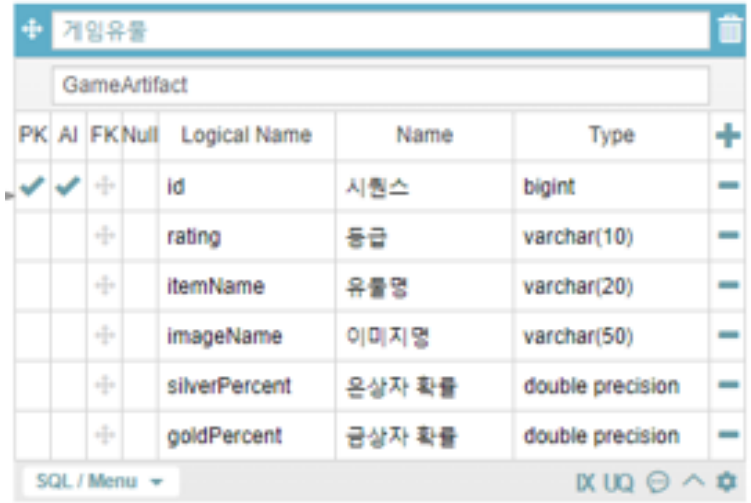
E.R.D

001. Money Entity



돈 수량에 관한 Entity

002. GameArtifact Entity



등급과 유물명, 이미지명,
은상자 확률, 금상자에 관한 Entity

003. UseArtifact Entity

(FK)

유물 id와 유물 수량에 관한 Entity

Part 5, Back End

Back End (Entity)

```java
@Getter
@Entity
@NoArgsConstructor(access = AccessLevel.PROTECTED)
public class Money {
    no usages
    @ApiModelProperty(notes = "시퀀스")
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    4 usages
    @ApiModelProperty(notes = "돈 수량")
    @Column(nullable = false)
    private Long payCount;

    1 usage  황상길
    public void resetMoney() { this.payCount = 0L; }

    1 usage  황상길
    public void plusMoney() { this.payCount += 500; }

    1 usage  황상길
    public void minusMoney(long money) { this.payCount -= money; }

    1 usage  황상길
    private Money(MoneyBuilder builder) { this.payCount = builder.payCount; }

    2 usages  황상길
    public static class MoneyBuilder implements CommonModelBuilder<Money> {
        2 usages
        private final Long payCount;

        1 usage  황상길
        public MoneyBuilder() { this.payCount = 0L; }

        황상길
        @Override
        public Money build() { return new Money( builder: this); }
    }
}
```

MoneyEntity

돈 수량을 나타내는 Entity입니다.
@ApiModelProperty는 swagger에서
알아보기 쉽게 하기 위함이고
Builder를 사용했습니다.
돈 수량의 값은 0L로 지정하였습니다.

Back End (Entity)

```java
@Entity
@Getter
@NoArgsConstructor(access = AccessLevel.PROTECTED)
public class GameArtifact {
    no usages
    @ApiModelProperty(notes = "시퀀스")
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    1 usage
    @ApiModelProperty(notes = "등급")
    @Enumerated(value = EnumType.STRING)
    @Column(nullable = false, length = 10)
    private Rating rating;

    1 usage
    @ApiModelProperty(notes = "유물명")
    @Column(nullable = false, length = 20)
    private String itemName;

    1 usage
    @ApiModelProperty(notes = "이미지명")
    @Column(nullable = false, length = 50)
    private String imageName;

    1 usage
    @ApiModelProperty(notes = "은상자 확률")
    @Column(nullable = false)
    private Double silverPercent;

    1 usage
    @ApiModelProperty(notes = "금상자 확률")
    @Column(nullable = false)
    private Double goldPercent;
```

```java
@Getter
@AllArgsConstructor
public enum Rating {
    4 usages
    NORMAL( name: "노말"),
    4 usages
    RARE( name: "레어"),
    4 usages
    EPIC( name: "에픽"),
    4 usages
    LEGEND( name: "전설"),
    4 usages
    GOD( name: "신화")
    ;
    no usages
    private final String name;
}
```

GameArtifactEntity, Enum

유물에 관한 Entity과 Enum입니다.
등급 EnumType은 문자열이고
글자수는 10글자까지 하였습니다.
유물명, 이미지명, 은상자 확률,
금상자 확률로 Column을 만들었습니다.

Back End (Entity)

```
1 usage  황상길
private GameArtifact(GameArtifactBuilder builder) {
    this.rating = builder.rating;
    this.itemName = builder.itemName;
    this.imageName = builder.imageName;
    this.silverPercent = builder.silverPercent;
    this.goldPercent = builder.goldPercent;
}

2 usages  황상길
public static class GameArtifactBuilder implements CommonModelBuilder<GameArtifact> {
    2 usages
    private final Rating rating;
    2 usages
    private final String itemName;
    2 usages
    private final String imageName;
    2 usages
    private final Double silverPercent;
    2 usages
    private final Double goldPercent;

    1 usage  황상길
    public GameArtifactBuilder(GameArtifactCreateRequest createRequest) {
        this.rating = createRequest.getRating();
        this.itemName = createRequest.getItemName();
        this.imageName = createRequest.getImageName();
        this.silverPercent = createRequest.getSilverPercent();
        this.goldPercent = createRequest.getGoldPercent();
    }

    황상길
    @Override
    public GameArtifact build() { return new GameArtifact( builder: this); }
}
}
```

GameArtifactEntity

유물에 관한 Entity의 Builder부분입니다.

Back End (Entity)

```java
@Entity
@Getter
@NoArgsConstructor(access = AccessLevel.PROTECTED)
public class UseArtifact {
    no usages
    @ApiModelProperty(notes = "시퀀스")
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    1 usage
    @ApiModelProperty(notes = "게임유물")
    @JoinColumn(name = "gameArtifactId", nullable = false)
    @ManyToOne(fetch = FetchType.LAZY)
    private GameArtifact gameArtifact;

    3 usages
    @ApiModelProperty(notes = "수량")
    @Column(nullable = false)
    private Integer artifactCount;
```

UseArtifactEntity

보유 유물에 관한 Entity입니다.
GameArtifactEntity와 @JoinColumn 으로 서로 알 수 있게(Fk) 하였습니다.
게임 유물과 유물 수량을 Column을 만들었습니다.

Back End (Entity)

```
1 usage  황상길
public void resetCount() { this.artifactCount = 0; }

1 usage  황상길
public void putCountPius() { this.artifactCount += 1; }

1 usage  황상길
private UseArtifact(UseArtifactBuilder builder) {
    this.gameArtifact = builder.gameArtifact;
    this.artifactCount = builder.artifactCount;
}

2 usages  황상길
public static class UseArtifactBuilder implements CommonModelBuilder<UseArtifact> {
    2 usages
    private final GameArtifact gameArtifact;
    2 usages
    private final Integer artifactCount;

    1 usage  황상길
    public UseArtifactBuilder(GameArtifact gameArtifact) {
        this.gameArtifact = gameArtifact;
        this.artifactCount = 0;
    }

    황상길
    @Override
    public UseArtifact build() { return new UseArtifact( builder: this); }
}
}
```

UseArtifactEntity

보유 유물에 관한 Entity의 Builder

Back End (Service)

```java
@Service
@RequiredArgsConstructor
public class MoneyService {
    2 usages
    private final MoneyRepository moneyRepository;

    1 usage  황상길
    public MoneyResponse putMoneyPlus() {
        Money money = moneyRepository.findById(1L).orElseThrow(ClassCastException::new);
        money.plusMoney();

        Money result = moneyRepository.save(money);

        return new MoneyResponse.MoneyResponseBuilder(result).build();
    }
}
```

MoneyService

MoneyRepository와 같이 일한다(@ReAr
)
돈을 추가 할 거기 때문에 putMoneyPlus
MoneyRepository에서 id가 1인 것을
가져온 후 money에 저장하고 새로운 Bui
lder에 result를 돌려줍니다.

Back End (Service)

```java
@Service
@RequiredArgsConstructor
public class InitDataService {
    2 usages
    private final MoneyRepository moneyRepository;
    3 usages
    private final GameArtifactRepository gameArtifactRepository;
    2 usages
    private final UseArtifactRepository useArtifactRepository;

    1 usage  황상길
    public void setFirstMoney() {
        Optional<Money> originData = moneyRepository.findById(1L);

        if (originData.isEmpty()) {
            Money addData = new Money.MoneyBuilder().build();
            moneyRepository.save(addData);
        }
    }
```

InitDataService

Repository 전부와 같이 (@ReAr) 일합니다.
MoneyRepository에서 id가 1인 것을 가져옵니다. 만약 데이터가 비어 있으면 새롭게 Build하고 MoneyRepository에 addData를 저장합니다.

Back End (Service)

```
List<GameArtifactCreateRequest> result = new LinkedList<>();
GameArtifactCreateRequest result1 = new GameArtifactCreateRequest.GameArtifactCreateRequestBuilder
(Rating.NORMAL,  itemName: "가죽 모자",  imageName: "leather_helmet.jpg",  silverPercent: 17.5,  goldPercent: 0D).build();
result.forEach(item -> {
    GameArtifact addData = new GameArtifact.GameArtifactBuilder(item).build();
    gameArtifactRepository.save(addData);
});
```

```
1 usage  황상길
public void setFirstArtifact() {
    List<GameArtifact> originList = gameArtifactRepository.findAll();

    if (originList.size() == 0) {...}
}

1 usage  황상길
public void setFirstUseArtifact() {
    List<UseArtifact> useArtifacts = useArtifactRepository.findAll();

    if (useArtifacts.size() == 0) {
        List<GameArtifact> gameArtifacts = gameArtifactRepository.findAll();

        gameArtifacts.forEach(item -> {
            UseArtifact addData = new UseArtifact.UseArtifactBuilder(item).build();
            useArtifactRepository.save(addData);
        });
    }
}

}
```

InitDataService

gameArtifactRepository에서 전부 가져오고 만약 원본의 size가 0이면 값을 세팅합니다. 그리고 각각 item을 하나씩 던져주면서 Build합니다.
gameArtifactRepository에 addData를 저장합니다.

Back End (Service)

```java
@Service
@RequiredArgsConstructor
public class GameDataService {
    3 usages
    private final MoneyRepository moneyRepository;
    3 usages
    private final UseArtifactRepository useArtifactRepository;

    /**
     * 게임 접속하면 맨 처음 받아올 데이터
     *
     * @return
     */
    1 usage  황상길
    public FirstConnectDataResponse getFirstData() {
        FirstConnectDataResponse result = new FirstConnectDataResponse();
        // 결과값을 담을 빈 그릇 생성.. 여기는 빌더 쓰지 않아서 new FirstConnectDataResponse() 빈 그릇 만들수 있다

        Money money = moneyRepository.findById(1L).orElseThrow(CMissingDataException::new); // 돈 원본 데이터 가져오기
        result.setMoneyResponse(new MoneyResponse.MoneyResponseBuilder(money).build()); // 돈 정보 가공해서 넣기
        result.setUseArtifactItems(this.getMyArtifacts()); // 돈 정보 가공해서 넣기

        return result;
    }

    1 usage  황상길
    public void gameReset() {
        Money money = moneyRepository.findById(1L).orElseThrow(CMissingDataException::new);
        money.resetMoney();
        moneyRepository.save(money);

        List<UseArtifact> originList = useArtifactRepository.findAll();
        for (UseArtifact item : originList) {
            item.resetCount();
            useArtifactRepository.save(item);
        }
    }
```

GameDataService

게임 접속하면 처음 받아올 데이터
빈 그릇을 생성한 후
moneyRepository에서 id가 1인 것을 가져와서 돈 정보와 유물 정보를 가공해서 빈 그릇에 채워 넣고 result를 돌려줍니다.

gameReset은 moneyRepository에서 id가 1인 것을 가져와서 돈 정보를 reset(0으로) 시키고 gameArtifactRepository에서 모두 가져온 후 reset(보유 유물 개수를 모두 0으로) 합니다.

Back End (Service)

```java
/**
 * 내가 보유한 유물 컬렉션을 가져온다.
 *
 * @return 보유한 유물 컬렉션
 */
1 usage  황상길 *
public List<UseArtifactItem> getMyArtifacts() {
    List<UseArtifact> originList = useArtifactRepository.findAllByIdGreaterThanEqualOrderByIdAsc(1L);

    List<UseArtifactItem> result = new LinkedList<>(); // 결과값을 담을 빈 리스트 생성

    originList.forEach(item -> result.add(new UseArtifactItem.UseArtifactItemBuilder(item).build()));

    return result;
}
```

GameDataService

useArtifactRepository에서
가지고있는 유물리스트 전체를
오름차순으로 가져온 후
결과값을 담을 빈 리스트에
Item을 던져주면서 하나씩 반복합니다.
Result에 새롭게 가공하고 result를
돌려줍니다.

Back End (Service)

```java
@Service
@RequiredArgsConstructor
public class ArtifactChooseService {
    3 usages
    private final MoneyRepository moneyRepository;
    1 usage
    private final GameArtifactRepository gameArtifactRepository;
    3 usages
    private final UseArtifactRepository useArtifactRepository;

    3 usages
    long choosePay = 0L;

    1 usage  황상길
    private void init(boolean isGoldBox) {
        this.choosePay = isGoldBox ? 35000L : 5000L; // 뽑기비용...
    }
```

```java
2 usages  황상길
public ChooseArtifactResponse getResult(boolean isGoldBox) {
    this.init(isGoldBox); // 초기화 시킨다.

    boolean isEnoughMoney = this.isStartChooseByMoneyCheck(isGoldBox); // 돈 충분한지 확인한다.

    if (!isEnoughMoney) throw new CNoMoneyException(); // 돈이 충분하지 않으면 뽑기 진행 불가능

    Money money = moneyRepository.findById(1L).orElseThrow(CMissingDataException::new);
    money.minusMoney(this.choosePay); // 뽑기 비용만큼 돈 소모시키기
    moneyRepository.save(money); // 돈 마이너스 시킨거 저장해서 확정하기

    long artifactResultId = this.getChooseArtifact(isGoldBox); // 랜덤으로 뽑은 유물 id 받아오기

    boolean isNewArtifact = false; // 새로 뽑은 유물인지 검사하기 위해 변수 추가하는데 기본으로 아니라고(false) 함
    Optional<UseArtifact> useArtifact = useArtifactRepository.findByGameArtifact_Id(artifactResultId);
    // 뽑힌 유물 id와 일치하는 보유 유물 데이터를 가져온다.
    if (useArtifact.isEmpty()) throw new CMissingDataException();
    // 만약 보유유물 데이터가 없으면 게임진행이 안되므로 오류메세지 출력..
    UseArtifact useArtifactData = useArtifact.get();
    // 명확성을 위해 UseArtifact 모양으로 상자 하나 만들어서 거기에 원본 옮겨놓기

    if (useArtifactData.getArtifactCount() == 0) isNewArtifact = true;
    //만약 원본데이터에서 보유유물 갯수가 0개라면 이건 없던걸 뽑은거니 isNewArtifact를 true로 바꿔주기

    useArtifactData.putCountPius(); // 보유유물 갯수 +1 해주기
    useArtifactRepository.save(useArtifactData); // 갯수 +1 해준거.. (수정된거) 반영 확정하기

    // 위에서 처리는 다 되었고 이제 정보 돌려줄 준비하기
    ChooseArtifactResponse result = new ChooseArtifactResponse();
    result.setMoneyResponse(new MoneyResponse.MoneyResponseBuilder(money).build());
    result.setCurrentItem(new ChooseArtifactCurrentItem.ChooseArtifactCurrentItemBuilder(useArtifactData.getGameArtifact(), isNewArtifact).build());
    result.setUseArtifactItems(getMyArtifacts());

    return result;
}
```

ArtifactChooseService

Repository전부와 같이 일 합니다.
뽑기 비용은 금박스면35000원 은박스는 5000원 돈이 충분한지 확인후 충분하지 않으면 돈이 부족한 메시지를 띄운다.
돈 Id가 1인 것을 가져온 후 뽑기 비용만큼 돈 수량을 빼고 저장한 후 확정, 뽑은 유물id를 받아오고 새로 뽑은 유물 검사하기 위해 변수 추가 기본으로 false 뽑힌 유물 id와 일치하는 유물 데이터를 가져옵니다. 없으면 오류메세지 유물갯수가 0이면 새로 뽑은 것이니 isNewArtifact를 true로 보유 유물 +1해주고 저장 후 결과를 돌려줍니다.

Back End (Service)

```java
/**
 * 내가 보유한 유물 컬렉션을 가져온다.
 *
 * @return 보유한 유물 컬렉션
 */
1 usage  황상길 *
public List<UseArtifactItem> getMyArtifacts() {
    List<UseArtifact> originList = useArtifactRepository.findAllByIdGreaterThanEqualOrderByIdAsc(1L); /

    List<UseArtifactItem> result = new LinkedList<>();

    originList.forEach(item -> result.add(new UseArtifactItem.UseArtifactItemBuilder(item).build()));

    return result;
}


/**
 * 뽑기를 진행함에 있어 돈이 충분한지 확인해주는 메서드
 *
 * @param isGoldBox 금박스 뽑기 여부
 * @return true : 충분하다 / false : 불충분하다
 */
1 usage  황상길
private boolean isStartChooseByMoneyCheck(boolean isGoldBox) {
    Money money = moneyRepository.findById(1L).orElseThrow(CMissingDataException::new); // 돈 데이터 가져

    boolean result = false; // 기본으로 일단 막아두기.

    if (money.getPayCount() >= choosePay) result = true; // 내가 가진돈이 뽑기비용보다 많거나 같으면 체크 결과를

    return result;
}
```

ArtifactChooseService

useArtifactRepository에서 가지고있는 유물리스트 전체를 오름차순으로 가져온 후 결과값을 담을 빈 리스트에 Item을 던져주면서 하나씩 반복합니다. Result에 새롭게 가공하고 result를 돌려줍니다. 뽑기를 진행하는데 돈이 충분한지 확인하려면 먼저 moneyRepository에서 id가 1인 데이터를 가져옵니다. 기본적으로 false 만약 내가 가진 돈이 뽑기 비용보다 많거나 같으면 true 바꾼 후 result를 돌려준다.

Back End (Service)

```
private long getChooseArtifact(boolean isGoldBox) {
    List<GameArtifact> artifacts = gameArtifactRepository.findAll();

    List<ChooseArtifactItem> percentBar = new LinkedList<>(); // 확률bar 결과 넣는 리스트

    double oldPercent = 0D; // 확률 누적값 담을 변수 생성
    for (GameArtifact artifact : artifacts) { // 유물 리스트에서 유물을 하나씩 던져주면서 반복시작한다.
        ChooseArtifactItem addItem = new ChooseArtifactItem(); // 확률bar를 채우기 위한 새 그릇을 만든
        addItem.setId(artifact.getId()); // 확률bar용 그릇에 유물id값을 넣는다.
        addItem.setPercentMin(oldPercent); // 확률bar용 그릇에 확률 시작값 세팅

        if (isGoldBox) { // 만약에 금상자를 여는것이라면
            addItem.setPercentMax(oldPercent + artifact.getGoldPercent()); // 확률bar용 그릇에 확률
        } else { // 그게 아니라면.. (= 은상자 여는거)
            addItem.setPercentMax(oldPercent + artifact.getSilverPercent()); // 확률bar용 그릇에 확
        }

        percentBar.add(addItem); // 확률bar 결과 넣는 리스트에 위에서 만든 그릇.. 세팅 다 된 그릇 추가

        if (isGoldBox) {
            oldPercent += artifact.getGoldPercent(); // 확률 누적값에 확률 누적 (금 확률값)
        } else {
            oldPercent += artifact.getSilverPercent(); // 확률 누적값에 확률 누적 (은 확률값)
        }

    }

    double percentResult = Math.random() * 100; // 랜덤으로 0~100 사이 수 하나 뽑아오기

    long resultId = 0; // 뽑기 결과 유물id 담을 변수 만들기
    for (ChooseArtifactItem item : percentBar) { // 확률bar 안에 아이템검사하면서 랜덤으로 뽑은 수치가 어
        if (percentResult >= item.getPercentMin() && percentResult <= item.getPercentMax()) {
            // 만약 랜덤으로 뽑은 수가 현재 구간의 최소값보다 크거나 같고 랜덤으로 뽑은 수가 현재 구간의 최대값보
            resultId = item.getId(); // 뽑기 결과 유물id에 현재 구간의 id를 뽑아다가 넣고
            break; // 반복 종료하기
        }
    }

    return resultId;
}
```

ArtifactChooseService

확률 값을 담을 변수를 생성합니다. 유물리스트에서 유물을 하나씩 던져주면서 반복합니다. 새 그릇을 만든 후 유물id
확률 시작 값을 세팅합니다. 금 상자를 여는 것이면 그릇에 금상자 확률 값 세팅
아니면 은상자 확률 값 세팅
랜덤으로 0~100 사이 수 하나를 뽑아옵니다. 유물id 담을 변수를 만들고 반복하면서 검사 시작 랜덤으로 뽑은 수가 최소값보다 크거나 같고 최대값보다 작거나 같으면 뽑기 결과 유물id에 현재 id를 넣고 반복 종료하고 resultId를 돌려줍니다.

Back End (Controller)

```java
@Api(tags = "돈 관리")
@RestController
@RequiredArgsConstructor
@RequestMapping("/v1/money")
public class MoneyController {
    1 usage
    private final MoneyService moneyService;

    no usages  황상길
    @ApiOperation(value = "돈 증가(광석캐기)")
    @PutMapping("/plus")
    public SingleResult<MoneyResponse> plusMoney() {
        return ResponseService.getSingleResult(moneyService.putMoneyPlus());
    }
}
```

MoneyController

moneyService와 같이 일 하고 @Api를 사용하여 swagger에서 가독성을 올렸습니다. 500원씩 추가(수정)하는 거기 때문에 putMapping을 썼습니다.

Back End (Controller)

```java
@Api(tags = "게임 데이터 관리")
@RestController
@RequiredArgsConstructor
@RequestMapping("/v1/first-connect")
public class GameDataController {
    2 usages
    private final GameDataService gameDataService;

    no usages  황상길
    @ApiOperation(value = "첫 접속시 현황 데이터")
    @GetMapping("/init")
    public SingleResult<FirstConnectDataResponse> getFirstData() {
        return ResponseService.getSingleResult(gameDataService.getFirstData());
    }

    no usages  황상길
    @ApiOperation(value = "게임 정보 리셋")
    @DeleteMapping("/reset")
    public CommonResult resetGameData() {
        gameDataService.gameReset();

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

GameDataController

첫 접속 시 현황 데이터를 보여주는 것이여서 getMapping을 사용하였고 게임데이터를 리셋하기 때문에 delMapping을 사용했습니다.

Back End (Controller)

```java
@Api(tags = "유물 뽑기")
@RestController
@RequiredArgsConstructor
@RequestMapping("/v1/choose-artifact")
public class ChooseArtifactController {
    2 usages
    private final ArtifactChooseService artifactChooseService;

    no usages  황상길
    @ApiOperation(value = "은상자 뽑기")
    @PostMapping("/silver")
    public SingleResult<ChooseArtifactResponse> chooseSilverBox() {
        return ResponseService.getSingleResult(artifactChooseService.getResult( isGoldBox: false));
    }

    no usages  황상길
    @ApiOperation(value = "금상자 뽑기")
    @PostMapping("/gold")
    public SingleResult<ChooseArtifactResponse> chooseGoldBox() {
        return ResponseService.getSingleResult(artifactChooseService.getResult( isGoldBox: true));
    }
}
```

ChooseArtifactController

유물 뽑기 Controller입니다.
은상자 뽑기 확률과 금상자 뽑기 확률에 관한 PostMapping입니다.

Part 5, Front End

Front End (Template)

```
<template>
    <div>
        <el-row :gutter="20" class="mb-4" v-if="payCount != null">
            <el-col :span="12" class="contents-center">
                <div>
                    <div>
                        <h1 class="custom-title">광석 캐기</h1>
                        <h2 style="font-size: 20px">돈 : {{ payCount | currency }}원</h2>
                    </div>
                    <div>
                        <div class="img-padding"><img src="/images/mineral.jpg" alt="광석" width="369" height="306"></div>
                        <div>
                            <el-button class="my-btn" @click.native="getMoney()">캐기</el-button>
                        </div>
                    </div>
                </div>
            </el-col>
            <el-col :span="12">
                    <div class="contents-center">
                        <h1 class="custom-title">유물 뽑기</h1>
                    </div>
                    <div>
                        <el-row :gutter="20">
                            <el-col :span="12" class="contents-center">
                                <div class="img-padding"><img src="/images/silver_box.PNG" alt="은상자" width="350" height="290"></div>
                                <div>
                                    <el-button class="silver-btn" @click.native="chooseBoxSilver()">열기 (5,000)</el-button>
                                </div>
                            </el-col>
                            <el-col :span="12" class="contents-center">
                                <div class="img-padding"><img src="/images/gold_box.png" alt="금상자" width="350" height="290"></div>
                                <div>
                                    <el-button class="gold-btn" @click.native="chooseBoxGold()">열기 (35,000)</el-button>
                                </div>
                            </el-col>
                        </el-row>
                    </div>
            </el-col>
        </el-row>
```

ChooseArtifactController

Index페이지의 Template 영역입니다.

Front End (Template)

```
<div class="artifact-list-box" v-if="artifactList.length > 0">
    <h1 class="contents-center custom-title">유물 컬렉션</h1>
    <el-row :gutter="20">
        <el-col :span="3" class="contents-center" v-for="(item, index) in artifactList" v-bind:key="index">
            <div v-if="item.artifactCount > 0">
                <div>
                    <img :src="'/images/' + item.imageName" :alt="item.itemName" width="90" height="90" />
                </div>
                <div>
                    <el-badge :value="item.ratingName" class="item" :type="getArtifactBadgeType(item.rating)">
                        {{ item.itemName }}   
                    </el-badge>
                </div>
                <div>{{ item.artifactCount }}개</div>
            </div>
            <div v-else>
                <div>
                    <img src="/images/no_artifact.png" alt="유물정보없음" width="90" height="90"/>
                </div>
                <div>
                    <el-badge :value="'모름'" class="item">
                        유물정보없음
                    </el-badge>
                </div>
                <div>개</div>
            </div>
        </el-col>
    </el-row>
</div>

<div class="contents-center">
    <el-button type="danger" round @click.native="gameReset()">게임리셋</el-button>
</div>
```

Index페이지의 Template 영역입니다.

Front End (Template)

```
    <div class="contents-center">
        <el-button type="danger" round @click.native="gameReset()">게임리셋</el-button>
    </div>

    <el-dialog title="유물 뽑기 결과" :visible.sync="artifactDialogVisible" width="30%" center>
        <div v-if="currentChooseArtifact != null">
            <div class="contents-center" style="margin-bottom: 10px">축하합니다!!</div>
            <div class="contents-center">
                <div class="img-style">
                    <img :src="'/images/' + currentChooseArtifact.imageName" :alt="currentChooseArtifact.itemName" width="100" height="100"/>
                </div>
                <div style="padding-top: 10px; font-size: 20px">
                    {{ currentChooseArtifact.ratingName }}
                </div>
                <div style="padding-top: 15px; font-size: 15px">{{ currentChooseArtifact.itemName }}</div>
                <div style="color: #c90c10; padding-top: 20px" v-if="currentChooseArtifact.isNew">NEW!!</div>
            </div>
        </div>

        <span slot="footer" class="dialog-footer">
            <el-button type="primary" @click="artifactDialogVisible = false">확인</el-button>
        </span>

    </el-dialog>
  </div>
/template>
```

Index페이지의 Template 영역입니다.

Front End (Script)

```vue
<script>
no usages
export default {
    data() {
        return {
            payCount: null,
            artifactList: [],
            artifactDialogVisible: false,
            currentChooseArtifact: null
        }
    },
    methods: {
        getArtifactBadgeType(ratingValue) {
            let result = ''

            switch (ratingValue) {
                case 'NORMAL':
                    result = 'info'
                    break
                case 'RARE':
                    result = 'success'
                    break
                case 'EPIC':
                    result = 'primary'
                    break
                case 'LEGEND':
                    result = 'warning'
                    break
                case 'GOD':
                    result = 'danger'
                    break
                default:
                    result = ''
            }

            return result
```

Index페이지의 Script 영역입니다.
If else if문 대신 switch를 사용해 가독성을 높였습니다.

Front End (Script)

```
getFirstData() {
    this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, payload: true)
    this.$store.dispatch(this.$gameApiConstants.DO_FIRST_DATA)
        .then((res) => {
            this.payCount = res.data.data.moneyResponse.payCount
            this.artifactList = res.data.data.useArtifactItems
            this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, payload: false)
        })
        .catch((err) => {
            this.$toast.error(err.response.data.msg)
            this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, payload: false)
        })
},
getMoney() {
    this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, payload: true)
    this.$store.dispatch(this.$gameApiConstants.DO_SHOW_ME_THE_MONEY)
        .then((res) => {
            this.payCount = res.data.data.payCount
            this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, payload: false)
        })
        .catch((err) => {
            this.$toast.error(err.response.data.msg)
            this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, payload: false)
        })
},
chooseBoxSilver() {
    this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, payload: true)
    this.$store.dispatch(this.$gameApiConstants.DO_CHOOSE_BOX_SILVER)
        .then((res) => {
            this.payCount = res.data.data.moneyResponse.payCount
            this.artifactList = res.data.data.useArtifactItems
            this.currentChooseArtifact = res.data.data.currentItem
            this.artifactDialogVisible = true
            this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, payload: false)
        })
        .catch((err) => {
            this.$toast.error(err.response.data.msg)
            this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, payload: false)
        })
},
```

Index페이지의 Script 영역입니다.

Front End (Script)

```
    },
    chooseBoxGold() {
        this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, payload: true)
        this.$store.dispatch(this.$gameApiConstants.DO_CHOOSE_BOX_GOLD)
            .then((res) => {
                this.payCount = res.data.data.moneyResponse.payCount
                this.artifactList = res.data.data.useArtifactItems
                this.currentChooseArtifact = res.data.data.currentItem
                this.artifactDialogVisible = true
                this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, payload: false)
            })
            .catch((err) => {
                this.$toast.error(err.response.data.msg)
                this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, payload: false)
            })
    },
    gameReset() {
        this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, payload: true)
        this.$store.dispatch(this.$gameApiConstants.DO_GAME_RESET)
            .then((res) => {
                this.getFirstData()
            })
            .catch((err) => {
                this.$toast.error(err.response.data.msg)
                this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, payload: false)
            })
    }
},
created() {
    this.getFirstData()
},
mounted() {
    this.$store.commit(this.$menuConstants.FETCH_SELECTED_MENU, payload: 'DASH_BOARD')
},

/script>
```

Index페이지의 Script 영역입니다.

목업 디자인

광석 캐기

나의 광물: 111111111

유물 뽑기

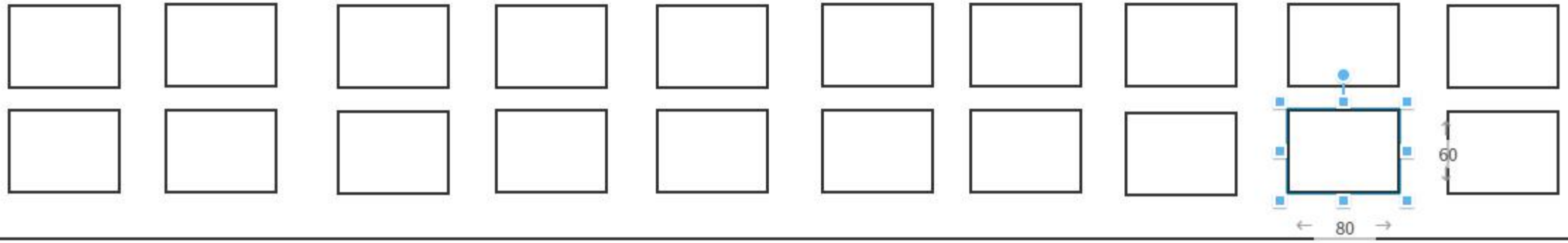
은 상자

5000

금 상자

30000

컬렉션



실행 화면

광석 캐기

돈 : 0원

캐기

유물 뽑기

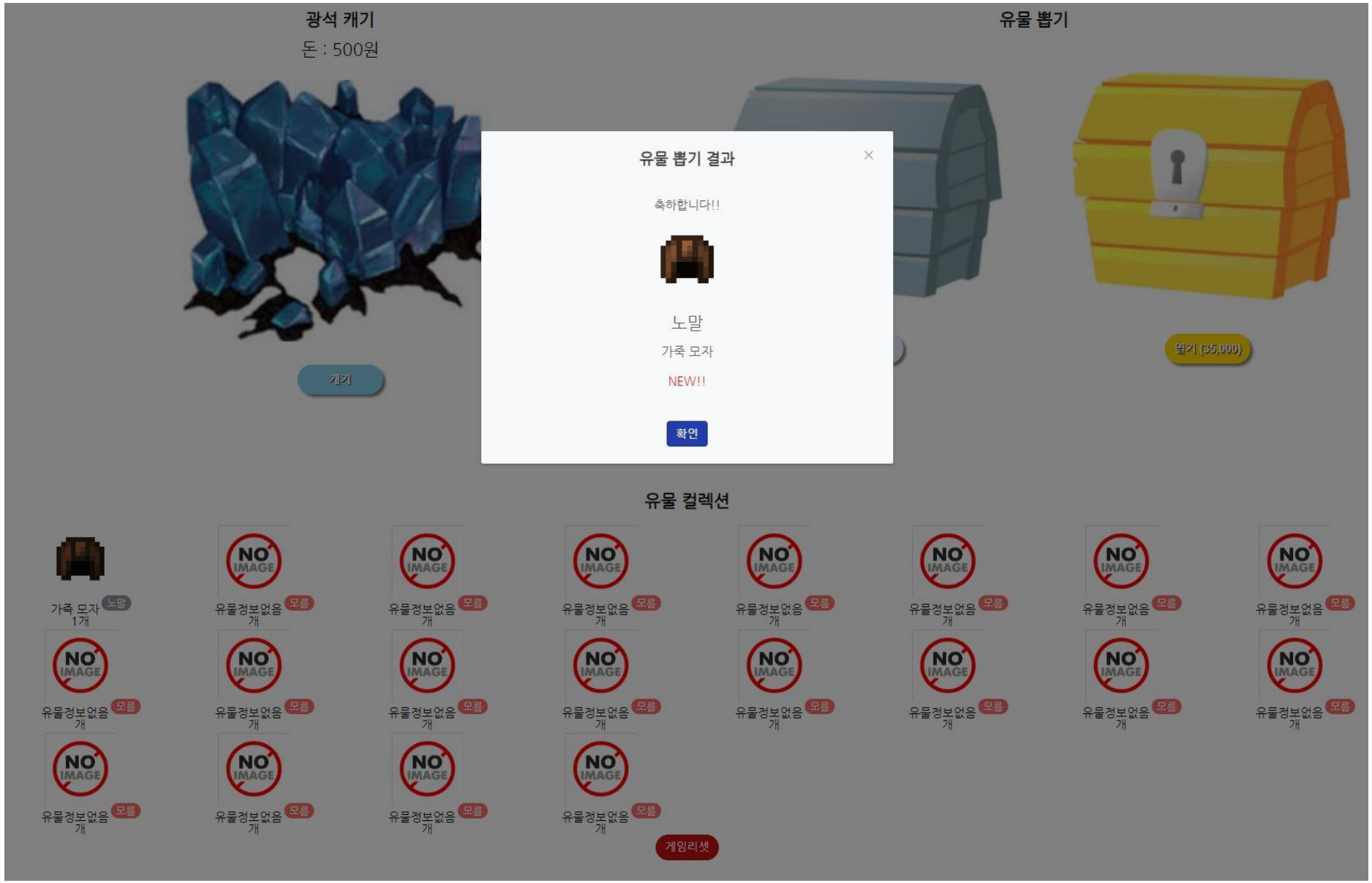
뽑기 (5,000)

뽑기 (35,000)

유물 컬렉션

게임리셋

실행 화면



실행 화면

광석 캐기

돈 : 500원

캐기

유물 뽑기

열기 (5,000)

열기 (35,000)

유물 컬렉션

가족 모자 노말
1개

유물정보없음 모름
개

유물정보없음 모름
개

유물정보없음 모름
개

유물정보없음 모름
개

유물정보없음 모름
개

유물정보없음 모름
개

유물정보없음 모름
개

유물정보없음 모름
개

유물정보없음 모름
개

유물정보없음 모름
개

유물정보없음 모름
개

유물정보없음 모름
개

유물정보없음 모름
개

유물정보없음 모름
개

유물정보없음 모름
개

유물정보없음 모름
개

유물정보없음 모름
개

유물정보없음 모름
개

유물정보없음 모름
개

게임리셋

감사합니다