POTION CREATE GAME

확률형 뽑기 게임 API & FRONT DESIGN

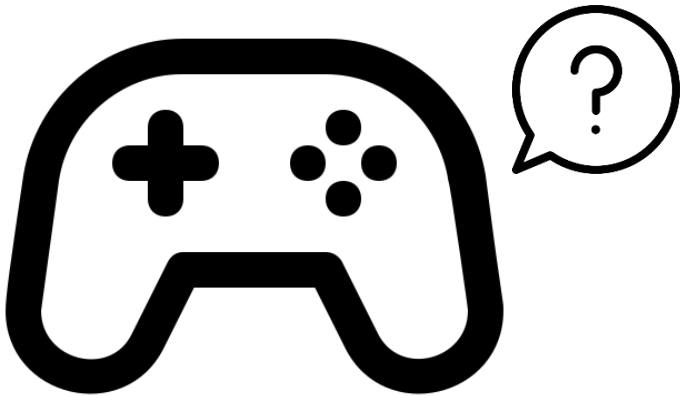
제작자 : 이 와이

기간 : 23.02.21 ~ 23.02.24

CONTENTS

기획의도

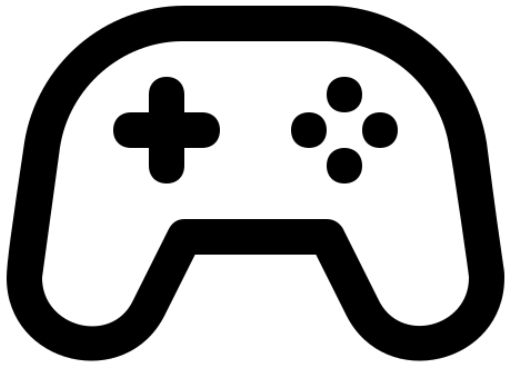
확률형 뽑기 시스템이 있는 게임을 이용하는 유저로서,
직접 이 게임을 만들어보며 로직을 이해하고, 설계해보고자 하여
이 게임을 기획하게 되었습니다.



굳이 포션 게임인 이유는?

게임에서 대표적으로 사용되는 아이템이자
상징성을 보일 요소라고 생각했기 때문입니다.
모티브를 얻은 요소는 포션크래프트라는 게임이며,
RPG 게임 요소를 부각시키기 위해 무료로 배포된
포션 도트 이미지를 사용하였습니다.

구성
로직과 구조



처음 이용하는 유저라면 초기값을 세팅

↓

뽑기를 이용하기 위한 재화(약초) 수집

↓

약초 수집 이후, 아이템 구성에 따른 상자(마력솥) 이용

Ex) 500개를 소모해서 일반 / 3500개를 소모해서 고급...

↓

아이템(포션)의 공개, 이후 인벤토리 리스트에 저장됨

아이템 획득 후 이미지의 스타일이 바뀌도록 구성

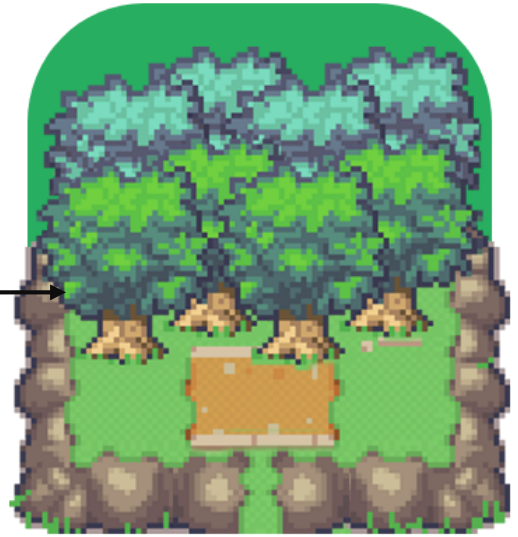
리셋 버튼을 클릭하면 모든 재화(약초)와 인벤토리가 0으로 변함

구성

러프 이미지

약초 캐기

클릭 당 뽑기에 필요한 재화(약초)를
50개씩 수령하도록 구성함



약초캐기

평범한/마녀의마력솥

약초캐기에서 얻은 재화(약초)를 소모하여
(일반 : 500개 / 마녀 : 3500개) 뽑기를 이용할 수 있음

평범한 마법솥
1성 ~ 5성

마녀의 마법솥
3성 ~ 5성

뽑기에서 얻은 아이템들을 리스트 형식으로
저장하여 리스트에 보여준다

컬렉션박스

완성


약초 : 0개
캐기
약초캐기


일반 마력솥 (1성~5성)
약초 : 500개
뽑기
마녀의 마력솥 (3성~5성)
약초 : 3500개
뽑기
포션만들기

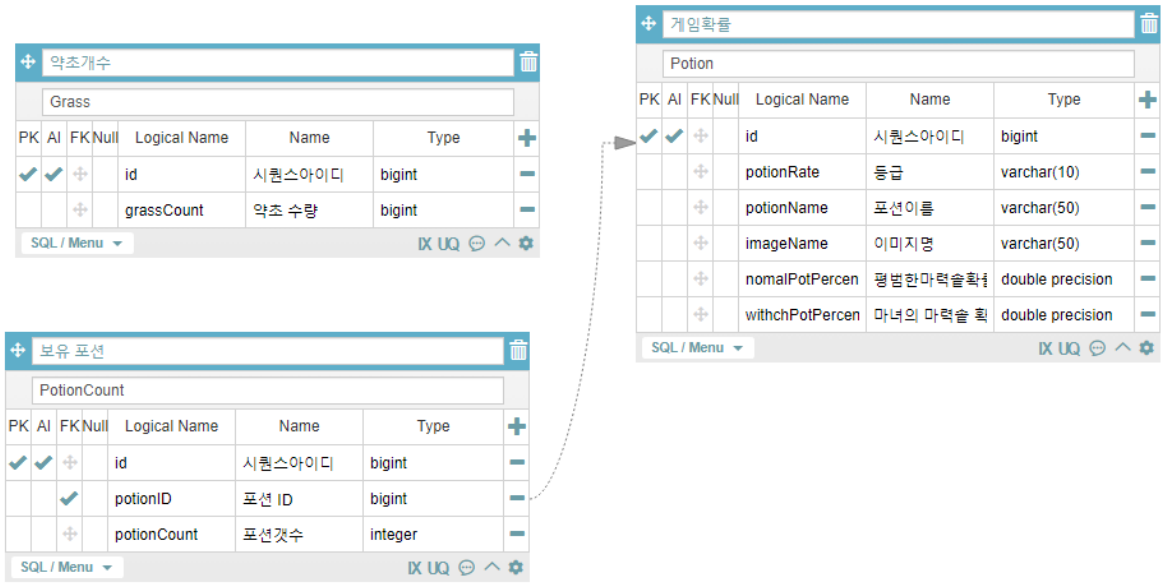

??? 0개
컬렉션박스

리셋

소프트웨어 구성

Java와 Vue.js를 이용하였습니다.

ERD & 기능정의서

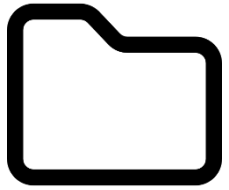


ID와 PK값의 구성을 ERD로 표현함

기능에 관한 정의서

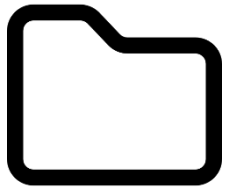
	A	B	C	D	E
1	no	구분	기능명	기능상세	비고
2	1	백엔드	약초캐기	1요청 당 50개의 약초를 지급한다	
3	2	백엔드	포션제작 - 평범한 마력솥	500개의 약초를 차감하고, 1성~3성의 포션을 랜덤지급한다.	확률은 확률표 참고
4	3	백엔드	포션제작 - 마녀의 마력솥	3000개의 약초를 차람하고, 3성~5성의 포션을 랜덤지급한다.	확률은 확률표 참고
5	4	백엔드	포션 컬렉션	포션들의 리스트를 보여준다	포션은 중복으로 제작이 가능하며, 중복 제작시, 해당 포션의 개수가 증가한다.
6	5	백엔드	새로하기	모든 아이템과 약초의 개수를 초기화한다	

BACKEND 구성

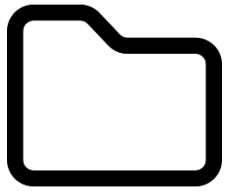


Entity

재화 (약초) 데이터
아이템(포션) 정보 및 확률 데이터
아이템(포션) 인벤토리 데이터



Repository



Service

초기값 세팅 및 확률 저장
클릭 당 재화(약초) 증가 및 리셋
아이템 뽑기 결과값 출력 및 개수 반영

BACKEND CODE

재화(약초) ENTITY

Builder() 패턴을 이용하여
가독성과 유지보수가 편하게끔 제작

Setter를 이용하여 일어나는 실수를
방지할 수 있게끔 하는 기능도 있음

```java
@Entity
@Getter
@NoArgsConstructor(access = AccessLevel.PROTECTED)
public class Grass {
    @ApiModelProperty(notes = "시퀀스값")
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ApiModelProperty(notes = "약초 갯수")
    @Column(nullable = false)
    private Long grassCount;

    public void plusGrass() {
        this.grassCount += 50;
    }

    public void resetGrass() {
        this.grassCount = 0L;
    }
    public void minusGrass(long grass) {
        this.grassCount -= grass;
    }
    private Grass(GrassBuilder builder) {
        this.grassCount = builder.grassCount;

    }
    public static class GrassBuilder implements CommonModelBuilder<Grass> {
        private final Long grassCount;

        public GrassBuilder() {
            this.grassCount = 0L;
        }
        @Override
        public Grass build() {
            return new Grass(this);
        }
    }

}
```

BACKEND CODE

아이템(포션) ENTITY

```java
@Entity
@Getter
@NoArgsConstructor(access = AccessLevel.PROTECTED)
public class Potion {

    @ApiModelProperty(notes = "시퀀스값")
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ApiModelProperty(notes = "포션등급")
    @Enumerated(value = EnumType.STRING)
    @Column(nullable = false, length = 10)
    private PotionRate potionRate;

    @ApiModelProperty(notes = "포션명")
    @Column(nullable = false, length = 50)
    private String potionName;

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

    @ApiModelProperty(notes = "평범한 마력솥 확률")
    @Column(nullable = false)
    private Double nomalPotPercent;

    @ApiModelProperty(notes = "미녀의 마력솥 확률")
    @Column(nullable = false)
    private Double witchPotPercent;

    private Potion( PotionBuilder builder ) {
        this.potionRate = builder.potionRate;
        this.potionName = builder.potionName;
        this.imageName = builder.imageName;
        this.nomalPotPercent = builder.nomalPotPercent;
        this.witchPotPercent = builder.witchPotPercent;

    }

    public static class PotionBuilder implements CommonModelBuilder<Potion>{
        private final PotionRate potionRate;
        private final String potionName;
        private final String imageName;
        private final Double nomalPotPercent;
        private final Double witchPotPercent;


        public PotionBuilder(PotionCreateRequest createRequest) {
            this.potionRate = createRequest.getPotionRate();
            this.potionName = createRequest.getPotionName();
            this.imageName = createRequest.getImageName();
            this.nomalPotPercent = createRequest.getNomalPotPercent();
            this.witchPotPercent = createRequest.getWitchPotPercent();

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

BACKEND CODE

아이템(포션) 인벤토리 ENTITY

```
@Entity
@Getter
@NoArgsConstructor(access = AccessLevel.PROTECTED)
public class PotionInventory {
    @ApiModelProperty(notes = "시퀀스값")
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @ApiModelProperty(notes = "포션 ID 시퀀스")
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "potionID", nullable = false)
    private Potion potion;

    @ApiModelProperty(notes = "포션 갯수")
    @Column(nullable = false)
    private Integer potionCount;

    public void putCountPlus() {
        this.potionCount += 1;
    }
    public void resetCount() {
        this.potionCount = 0;
    }
    private PotionInventory(PotionInventoryBuilder builder) {
        this.potionCount = builder.potionCount;
        this.potion = builder.potion;
    }
    public static class PotionInventoryBuilder implements
CommonModelBuilder<PotionInventory> {
        private final Potion potion;
        private final Integer potionCount;

        public PotionInventoryBuilder(Potion potion) {
            this.potion = potion;
            this.potionCount = 0;

        }

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

}
```

BACKEND CODE

초기값 (확률 및 초기 설정값) 세팅 서비스

```java
@Service
@RequiredArgsConstructor
public class InitDataService {
    private final GrassRepository grassRepository;
    private final PotionRepository potionRepository;
    private final PotionInventoryRepository potionInventoryRepository;

    /**
     * 약초 세팅
    // 데이터 첫 세팅 = 약초 갯수 0개로 설정
     **/

    public void setFirstGrass() {
        Optional<Grass> originData = grassRepository.findById(1L);

        if(originData.isEmpty()) {
            Grass addData = new Grass.GrassBuilder().build();
            grassRepository.save(addData);
        }
    }
```

데이터 세팅

**약초 Entity에서 id값(long)을 바탕으로 origindata를 만들고, 이 값이 비어있다면,
Grass.Builder().build()를 이용하여 값을 생성한다**

```java
// 데이터 첫 세팅 = 포션 확률 세팅

    public void setFirstPotionSetting() {
        List<Potion> originList = potionRepository.findAll();

        if(originList.size() == 0){
        List<PotionCreateRequest> result = new LinkedList<>();
            PotionCreateRequest request1 = new
              /**
              PotionCreateRequest.PotionCreateRequestBuilder(등급 ENUM, "포션명", "이미지명","일반 마력솥 확률","마녀의 마력솥 확률").build();
              result.add(request1);
                (이와 같은 구조로 반복)
              */

            result.forEach(item -> {
                Potion addData = new Potion.PotionBuilder(item).build();
                potionRepository.save(addData);
            });

        }

    }

    public void setFirstUserPotionInventory() {
        List<PotionInventory> potionInventories = potionInventoryRepository.findAll();

        if(potionInventories.size() == 0 ) {
            List<Potion> potions = potionRepository.findAll();
            potions.forEach(item -> {
                PotionInventory addData = new PotionInventory.PotionInventoryBuilder(item).build();
                potionInventoryRepository.save(addData);
            });
        }
    }
```

BACKEND CODE

첫 시작 데이터 설정 및 리셋 서비스

데이터 세팅

처음 게임을 켰을 때, 원본 데이터를 id값을 바탕으로
오름차순으로 배치하고, 가공한 데이터를 담을 빈 리스트를 만든다.

데이터 리셋

게임의 모든 데이터를 리셋 (= 확률을 제외한 모든 값을 0으로 변환)하기 위해,
포션의 인벤토리와 약초 항목의 아이디값과 리스트값을 가져온다.

```java
@Service
@RequiredArgsConstructor
public class GameDataService {
    private final GrassRepository grassRepository;
    private final PotionInventoryRepository potionInventoryRepository;


    /**
     * 게임 접속하면 맨 처음 받아올 데이터
     *
     * @return
     */
    public GameDataResponse getFirstData() {
        GameDataResponse result = new GameDataResponse();
      // 결과값을 담을 빈 그릇 생성. 여기는 빌더를 쓰지 않아서
      //new FirstConnectDataResponse()해서 빈 그릇 만들어 줄 수 있다.

        Grass grass = grassRepository.findById(1L).orElseThrow(CMissingDataException::new);
      // 약초의 원본 데이터 가져오기
        result.setGrassResponse(new GrassResponse.GrassResponseBuilder(grass).build());
      // 약초 정보 가공해서 넣기
        result.setUsePotionItems(this.getMyPotions()); // 보유 포션 컬렉션 넣기

        return result;
    }

    /**
     * 내가 보유한 포션 컬렉션을 가져온다.
     *
     * @return 보유한 포션 컬렉션
     */
    public List<UsePotionItem> getMyPotions() {
        List<PotionInventory> originList =
          potionInventoryRepository.findAllByIdIsGreaterThanEqualOrderByIdAsc(1L);
      // 내가 가지고있는 유물 리스트 전체를 불러온다.

        List<UsePotionItem> result = new LinkedList<>(); // 결과값을 담을 빈 리스트를 생성한다.

        originList.forEach(item -> result.add(new UsePotionItem.UsePotionItemBuilder(item).build()));

        return result;
    }
    /**
     * 게임 정보 리셋.
     *
     */

    public void gameReset() {
        Grass grass = grassRepository.findById(1L).orElseThrow(CMissingDataException::new);
        grass.resetGrass();
        grassRepository.save(grass);

        List<PotionInventory> originList = potionInventoryRepository.findAll();
        for (PotionInventory item : originList) {
            item.resetCount();
            potionInventoryRepository.save(item);
        }

    }
}
```

BACKEND CODE

재화(약초) 증가 및 소모 서비스

데이터 세팅

**소모되는 약초의 양을 마력솥의 종류에 따라 판별하며,
이에 맞추어 약초의 개수가 충분한지 체크한다.
돈이 충분하지 않다면 toast로 창을 띄우고,
충분하다면 비용만큼 약초를 소모시키고(이 때의 데이터도 저장)
랜덤으로 뽑은 포션을 보여준 뒤,
내가 뽑은 아이템의 인벤토리에 +1을 합니다.**

```java
public class GrassService {
    private final GrassRepository grassRepository;

    public GrassResponse putGrassPlus() {
        Grass grass = grassRepository.findById(1L).orElseThrow(CMissingDataException::new);
        grass.plusGrass();

       Grass result = grassRepository.save(grass);
       return new GrassResponse.GrassResponseBuilder(result).build();
    }
}
```

```java
private final GrassRepository grassRepository;
    private final PotionRepository potionRepository;
    private final PotionInventoryRepository potionInventoryRepository;
    long choosePay = 0L;

    private void init(boolean isWitchPot) {
        this.choosePay = isWitchPot? 3500L : 500L; //마녀의 마력솥은 3000개, 일반 마력솥은 500개
    }
    public PickUpPotionItemResponse getResult(boolean isWitchPot) {
        this.init(isWitchPot); // 초기화
        boolean isEnoughGrass = this.isStartBeforeCheckGrass(isWitchPot); // 약초 갯수가 충분한지 체크
        if (!isEnoughGrass)  throw new CNotEnoughGrassException(); // 돈이 충분치 않으면 뽑기 진행 불가
        Grass grass = grassRepository.findById(1L).orElseThrow(CNotEnoughGrassException::new);
        grass.minusGrass(this.choosePay); // 뽑기 비용만큼 약초 소모
        grassRepository.save(grass); // 약초 소모시킨 데이터 저장

        long potionResult = this.getPickedPotion(isWitchPot); // 랜덤으로 뽑은 포션 보이기

        boolean isNewPotion = false; // 랜덤으로 뽑힌 아이템이 새로운 것인지 검사, 기본값으로 false를 줌

       //내가 뽑은 아이디에 일치하는 아이템의 인벤토리에 +1 하기
        Optional<PotionInventory> potionInventory = potionInventoryRepository.findByPotion_Id(potionResult);
        if(potionInventory.isEmpty()) throw new CMissingDataException(); //에러창 띄우기. 만약 비어있는 것이 나온다면 데이터를 찾을 수 없다는 에러창 출력
        PotionInventory potionInventoryData = potionInventory.get(); // 명확성을 위하여, PotionInventory 모양으로 된 상자를 하나 만들어서 원본을 옮겨놓는다.

        if(potionInventoryData.getPotionCount() == 0 ) isNewPotion = true; // 인벤토리의 데이터의 갯수가 0이면,isNewPotion 의 값을 참으로 바꾼다.

        potionInventoryData.putCountPlus(); // 보유 포션 갯수에 +1
        potionInventoryRepository.save(potionInventoryData); // 갯수 +1 한 것에 대한 반영(수정) 확정하기

        PickUpPotionItemResponse result = new PickUpPotionItemResponse();
        result.setGrassResponse(new GrassResponse.GrassResponseBuilder(grass).build());
        result.setPickUpPotionCurrentItem(new
                                          PickUpPotionCurrentItem.PickUpPotionCurrentItemBuilder(potionInventoryData.getPotion(),isNewPotion).build());
        result.setUsePotionItems(this.getMyPotions());

        return result;
    }
```

BACKEND CODE

아이템 확률 추가 및 확률 출력 서비스

확률 카운트 메서드

두 개의 마력솥 중에서 선택된 마력솥이 기준(마녀의 마력솥)인지 판별 후, 각 아이템에 맞는 확률을 저장하도록 함.

확률을 저장하는 방식
oldPercent = 0 (확률이 0인 빈 변수)
각 확률의 Max(setPerCentMax)는
아이템의 확률을 oldPerCent와 더한 값과 일치하게끔 설정함

```java
private long getPickedPotion(boolean isWitchPot) {
        List<Potion> potions = potionRepository.findAll();

        double oldPercent = 0D;
        int index = 1;
        List<PickUpPotionPercent> percentBar = new LinkedList<>();

        for(Potion potion : potions){
            PickUpPotionPercent addItem = new PickUpPotionPercent();
            addItem.setId(potion.getId());
            addItem.setPercentMin(oldPercent);

            if (isWitchPot) {
                addItem.setPercentMax(oldPercent + potion.getWitchPotPercent());
            }else{
                addItem.setPercentMax(oldPercent + potion.getNomalPotPercent());
            }

            percentBar.add(addItem);
            if(isWitchPot) {
                oldPercent+= potion.getWitchPotPercent();
            }else{
                oldPercent+= potion.getNomalPotPercent();
            }
        }

        double percentValue = Math.random()*100;

        long resultId = 0; // 뽑기 결과 포션id 담을 변수 만들기

        for (PickUpPotionPercent item : percentBar) {
          // 확률bar 안에 아이템 검사하면서 랜덤으로 뽑은 수치가 어느 구간에 해당되는지 검사하기위해 반복 시작
            if (percentValue ≥ item.getPercentMin() && percentValue ≤ item.getPercentMax()) {
              // 만약 랜덤으로 뽑은 수가 현재 구간의 최소값보다 크거나 같고
              //랜덤으로 뽑은 수가 현재 구간의 최대값보다 작거나 같으면
                resultId = item.getId();
              // 뽑기 결과 포션 id에 현재 구간의 id를 뽑아다가 넣고
                break; // 반복 종료하기
            }
        }
        return resultId;
    }
```

BACKEND CODE

컨트롤러

```java
@Api(tags = "첫 접속 데이터 관리")
@RestController
@RequiredArgsConstructor
@RequestMapping("/v1/first-connect")

public class GameDataController {
    private final GameDataService gameDataService;

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

    @ApiOperation(value = "데이터 리셋")
    @DeleteMapping("/reset")
    public CommonResult resetData() {
        gameDataService.gameReset();
        return ResponseService.getSuccessResult();
    }
}
```

```java
@Api(tags = "클릭시 약초의 갯수가 50씩 증가")
@RestController
@RequiredArgsConstructor
@RequestMapping("/v1/grass")
public class GrassController {
    private final GrassService grassService;
    @ApiOperation(value = "클릭시 약초 갯수 50씩 증가")
    @PutMapping("/plus")
    public SingleResult<GrassResponse> plusGrass() {
        return ResponseService.getSingleResult(grassService.putGrassPlus());
    }
}
```

```java
@Api(tags ="포션뽑기")
@RestController
@RequiredArgsConstructor
@RequestMapping("/v1/pickup-potion")
public class PickUpPotionController {
    public final PotionPickUpService potionPickUpService;

    @ApiOperation(value = "일반 마력솥 뽑기")
    @PostMapping("/normal")
    public SingleResult<PickUpPotionItemResponse> pickupNormalPotion() {
        return ResponseService.getSingleResult(potionPickUpService.getResult(false));
    }

    @ApiOperation( value = "마녀의 마력솥 뽑기")
    @PostMapping("/witch")
    public SingleResult<PickUpPotionItemResponse> pickupWitchPotion() {
        return ResponseService.getSingleResult(potionPickUpService.getResult(true));
    }
}
```

FRONTEND CODE

<templet>영역

```
<!-템플릿과 div 영역 생략 // 약초 채집 및 포션 뽑기 영역->
            <div id="grassBox" v-if = "grassCount ≠ null" >
                <h2>약초 : <span>{{ grassCount | currency }}</span>개</h2>
                <img src="../static/image/forest.png" alt="숲 이미지">
                <el-button type="primary" round class="pick-grass" @click.native="getGrass()">캐기</el-button>
                <div class ="title-box grass">
                    <p>약초캐기</p>
                </div>
            </div>
            <div id="potBoxClick">
                <div class="pot-box normal">
                    <img src="../static/image/nomal-pot.png" @click="pickNormalPot()" alt="평범한 마력솥">
                    <p>일반 마력솥 (1성~5성)<br>
                        약초  : 500개
                    </p>
                    <el-button type="primary" round class="round-pick" @click.native="pickNormalPot()">뽑기</el-button>
                </div>
                <div class="pot-box witch">
                    <img src="../static/image/witch-pot.png" @click="pickWitchPot()" alt="마녀의 마력솥">
                    <p>마녀의 마력솥 (3성~5성)<br>
                        약초  : 3500개
                    </p>
                    <el-button type="primary" round class="round-pick" @click.native="pickWitchPot()">뽑기</el-button>
                </div>
                <div class ="title-box pot">
                    <p>포션만들기</p>
                </div>
            </div>
```

grassBox, potBoxClick

클릭시 이벤트가 발생하는 영역

@click.native="getGrass()",
@click.native="getNomalPot()",
@click.native="getWitchPot()"
클릭이벤트를 발생하고, 해당하는 메서드가 작동하도록 함

FRONTEND CODE

<templet>영역

collectionBox, el-dialog

이벤트가 발생되는 영역

획득한 아이템의 데이터를 받아와 반영하도록 하였으며,
메서드를 통해 받아온 값들을 {{ item.potionName }} 과 같은 형식으로 하여 값을 받아올 수 있도록 함.

이미지 영역은 :src와 같이 함수를 반영할 수 있도록 함

```
<!-템플릿과 div 영역 생략 // 포션 컬렉션 영역 및 팝업 영역->
<div id = "collectionBox">
                <ul id = "potion-list" v-if="potionCountList.length >0" >
                    <li class="potion"  v-for="(item, index) in potionCountList" >
                        <div v-if="item.potionCount>0">
                            <el-badge :value="item.potionRateName" class="item" :type="getPotionLevelType(item.potionRate)"></el-badge>
                            <img :src="'/image/' + item.imageName" :alt="item.itemName">
                            <p class="potion-info">
                                {{ item.potionName }}<br>
                                <span>{{ item.potionCount }}</span>개
                            </p>
                        </div>
                        <div class=" not-have" v-else>
                            <img :src="'/image/' + item.imageName" :alt="item.itemName">
                            <p class="potion-info">
                                ???<br>
                                <span>0</span>개
                            </p>
                        </div>
                    </li>
                </ul>
                <div class ="title-box collection">
                    <p>컬렉션박스</p>
                </div>

            </div>
        </div>

        <el-dialog title="아이템 획득" :visible.sync="dialogVisible" >
            <div v-if="pickedPotion != null">
                <h1 class="get-title">축하합니다!</h1>
                <img :src="'/image/'+ pickedPotion.imageName" :alt="pickedPotion.itemName">
                <p v-if="pickedPotion.isNew">NEW!</p>
                <p>아이템을 획득하셨습니다.</p>
                <p>{{ pickedPotion.potionName }}</p>
                <p>{{ pickedPotion.potionRateName }}</p>

                <span slot="footer" class="dialog-footer">
                    <el-button type="primary" @click="dialogVisible = false" class="contents-center">확인</el-button>
                </span>
            </div>
        </el-dialog>
        <el-button class="clear-all" type="danger" round @click.native="clearAllData()">리셋</el-button>
    </div>
```

FRONTEND CODE

<script>
methods영역

```
data() {
        return {
            grassCount: null,
            potionCountList : [],
            dialogVisible: false,
            pickedPotion: null
        }
    },
    methods: {
        getPotionLevelType(potionRateName) {
            //등급 설정에 따라 색이 바뀌는 것
            let result = ''
            switch (potionRateName) {
                case 'NORMAL' :
                    result = 'info'
                    break
                case 'RARE' :
                    result = 'success'
                    break
                case 'UNIQUE' :
                    result = 'primary'
                    break
                case 'EPIC' :
                    result = 'warning'
                    break
                case 'LEGEND' :
                    result = 'danger'
                    break
                default:
                    result = ''

            }
            return result
        },
```

```
getFirstData() {
            this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, true)
            this.$store.dispatch(this.$gameApiConstants.DO_FIRST_DATA)
                .then((res) => {
                    this.grassCount = res.data.data.grassResponse.grassCount
                    this.potionCountList = res.data.data.usePotionItems
                    this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, false)
                })
                .catch((err) => {
                    this.$toast.error(err.response.data.msg)
                    this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, false)
                })
        },
        getGrass() {
            this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, true)
            this.$store.dispatch(this.$gameApiConstants.DO_PICK_THE_GRASS)
                .then((res) => {
                    this.grassCount = res.data.data.grassCount
                    this.$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)
                })
        },
```

getPotionLevelType, getFirstData, getGrass

이벤트 메서드

등급에 따라포션의 등급 배지의 색상을
바꾸도록 하였으며. switch, brake 문을 이용하여
코드의 가독성을 높이고, 효율적이게끔 작성하였습니다.

Api의 연동후, 받아온 데이터를 프론트에 반영하도록 하였습니다.

FRONTEND CODE

<script>영역

```
pickNormalPot() {
          this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, true)
          this.$store.dispatch(this.$gameApiConstants.DO_PICK_NORMAL_POT)
              .then((res) => {
                  this.grassCount = res.data.data.grassResponse.grassCount
                  this.potionCountList = res.data.data.usePotionItems
                  this.pickedPotion = res.data.data.pickUpPotionCurrentItem
                  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.$customLoadingConstants.FETCH_LOADING_SHOW, false)
              })
      },
        pickWitchPot() {
          this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, true)
          this.$store.dispatch(this.$gameApiConstants.DO_PICK_WITCH_POT)
              .then((res) => {
                  this.grassCount = res.data.data.grassResponse.grassCount
                  this.potionCountList = res.data.data.usePotionItems
                  this.pickedPotion = res.data.data.pickUpPotionCurrentItem
                  this.dialogVisible = true
                  this.$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)
              })
      },
```

FRONTEND CODE

<script>영역

```
clearAllData() {
          this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, true)
          this.$store.dispatch(this.$gameApiConstants.DO_CLEAR_ALL)
              .then((res) => {
                  this.getFirstData()
              })
              .catch((err) => {
                  this.$toast.error(err.response.data.msg)
                  this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, false)
              })
      }
  },
  created() {
      this.getFirstData()
  },
  mounted() {
      this.$store.commit(this.$menuConstants.FETCH_SELECTED_MENU, 'DASH_BOARD')
  },
```

감사합니다!