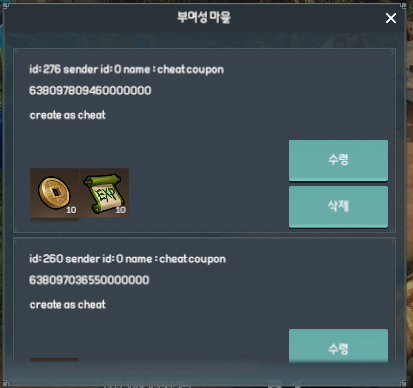
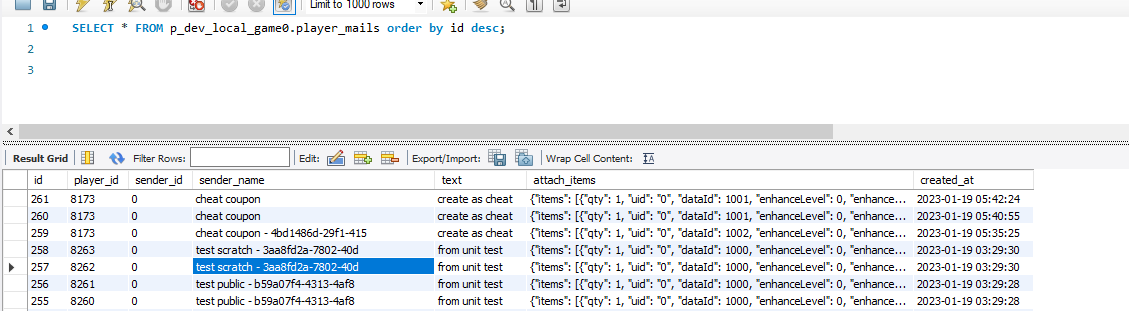
# 할일

* currency Type(금전, 경험치, 붉은 보석) 을 메일로 보낼 수 있게 한다.

| *// 화폐 관련*  public static readonly CheatCmd Gold = new(nameof(Gold), "금전", "//금전 {수량:변화량}");  public static readonly CheatCmd Ruby = new(nameof(Ruby), "붉은보석", "//붉은보석 {수량:변화량}");  public static readonly CheatCmd Exp = new(nameof(Exp), "경험치", "//경험치 {value}"); |
| --- |



# DB



# GamePlayer

| public static GamePlayer Of(XEntity entity, PlayerRow row, TAppearance appearance = null)  {  player.Mail = entity.AddComponent<PlayerMailComponent>(); |
| --- |

## PlayerMailComponent

## PlayerMailComponentTests

# 골드/붉은 보석 추가

* wallet 에서 관리

| public int Read(long mailUid, out List<TItem> addItems)  {  addItems = null;  if (0 >= mailUid)  return StatusCodeEx.**NotFoundMail**;  var db = GetSystem<DatabaseSystem>();  var mail = db.Game.PlayerMail.GetByIdAsync(mailUid, \_player.Psn.Uid).Result;  if (mail == null)  return StatusCodeEx.**NotFoundMail**;  *// TODO : 아이템을 넣을 수 있는지 체크*  *// TODO: 더 체크할 수있는것들 다 체크*  int affected = db.Game.PlayerMail.DeleteAsync(mailUid, \_player.Psn.Uid).Result;  if (0 >= affected)  return StatusCodeEx.**NotFoundMail**;  if (string.IsNullOrEmpty(mail.attach\_items))  return StatusCodeEx.**Success**;  var sz = mail.attach\_items.FromJson<TBagSlotSerializer>();  if (!sz.Items.Any())  {  Logger.Error(  $"mail attachItem is null - mailId({mail.id}), playerUid({\_player.Psn.Uid}), attach\_item({mail.attach\_items})");  return StatusCodeEx.**Error**;  }  addItems = new();  foreach (var item in sz.Items)  {  int status = \_player.Bag.TryPut(item, out var puttedItems);  if (!StatusCodeEx.IsSuccess(status))  {  Logger.Error(  $"mail attachItem push failed - mailId({mail.id}), playerUid({\_player.Psn.Uid}), attach\_item({item.ToJson()})");  continue;  }  addItems.AddRange(puttedItems);  }  return StatusCodeEx.**Success**;  } |
| --- |

골드 정산

| public int CheatGold(GamePlayer player, ImmutableArray<string> args)  {  return CheatWalletInternal(player.Wallet.Gold, args);  }  public int CheatRuby(GamePlayer player, ImmutableArray<string> args)  {  return CheatWalletInternal(player.Wallet.Ruby, args);  }  private int CheatWalletInternal(PlayerWallet wallet, ImmutableArray<string> args)  {  if (args.Length < 2) return StatusCodeEx.**InvalidCheatCommandParam**;  var amt = long.Parse(args[1]);  if (0 < amt)  wallet.Earn(ItemOriginFrom.Cheat, 0, amt);  else if (0 > amt)  wallet.Spend(ItemOriginFrom.Cheat, Math.Abs(amt));  return StatusCodeEx.**Success**;  } |
| --- |

| public Task<int> PlusAsync(long player\_id, int wallet\_type, long paid, long free)  {  var sql = @"UPDATE `player\_wallets`  SET `paid` = `paid` + @paid, `free` = `free` + @free, `updated\_at` = @updated\_at  WHERE `player\_id` = @player\_id AND `wallet\_type` = @wallet\_type";  var param = new  {  player\_id,  wallet\_type,  paid,  free,  updated\_at = DateTime.UtcNow,  };  return \_nodes.Node(player\_id).DoAsync(conn => conn.ExecuteAsync(sql, param));  } |
| --- |

# 경험치 추가

| public int CheatExp(GamePlayer player, ImmutableArray<string> args)  {  if (args.Length < 2) return StatusCodeEx.**InvalidCheatCommandParam**;  int exp = int.Parse(args[1]);  if (0 < exp)  return player.Exp.Gain(ItemOriginFrom.Cheat, exp);  else if (0 > exp)  return player.Exp.Loss(ItemOriginFrom.Cheat, exp \* -1);  return StatusCodeEx.**Success**;  } |
| --- |

| public class PlayerExpComponent : XComponent  {  public int TryGain(ItemOriginFrom originFrom, long point, out int outLevelUp)  {  outLevelUp = 0;  if (0 >= point)  return StatusCodeEx.**NegativeExp**;  var rSys = GetSystem<ResourceSystem>();  var rLevel = rSys.Level.GetByJob(\_player.Psn.Job);  if (null == rLevel)  return StatusCodeEx.**NotFoundExpTable**;  if (rLevel.Spec.MaxLevel <= GetLevel())  return StatusCodeEx.**ExpIsFull**;  var resLevel = rLevel.Table.GetLevel(GetLevel());  int levelUpCnt = 0;  long cappingPoint = 0;  lock (\_lock)  {  cappingPoint = Math.Max(0, Math.Min(rLevel.Spec.MaxExp - GetExp(), point));  if (0 >= cappingPoint)  return StatusCodeEx.**NegativeExp**;  \_exp.AddAndGet(cappingPoint);  \_addedExp.AddAndGet(cappingPoint);  while (null != resLevel && 0 < resLevel.NextAccumulatedExp && resLevel.NextAccumulatedExp <= GetExp())  {  levelUpCnt += 1;  resLevel = rLevel.Table.GetLevel(GetLevel() + levelUpCnt);  }  }  if (0 < levelUpCnt)  {  \_level.AddAndGet(levelUpCnt);  \_addedLevel.AddAndGet(levelUpCnt);  outLevelUp = levelUpCnt;  *// TODO : 레벨업 1단위로 보낼건지, 아니면 한번에 보내고, 클라에서 연출 제어 할건지 이야기 해야 함*  NotifyLevelChanged(levelUpCnt);  }  *// 경험치 획득 부터 전파*  var notify = new PlayerExpChangedNotify()  {  Exp = ToTExp(),  VarianceExp = new TExp()  {  Exp = cappingPoint,  Level = levelUpCnt,  },  OriginFrom = originFrom.Value  };  \_player.Session.Write(0, notify);  return StatusCodeEx.**Success**;  } |
| --- |