Call Of Duty App Game Management System and Database

Group 87
Gabriel Alacchi, Owen Lewis, Rebecca Jaubert, Yunus Can Cukran

REQUIREMENTS ANALYSIS

This application will describe the social aspects of a simpler version of the mobile game Call of Duty that was released in 2019. It is a shooting game in which players can either fight in teams in an online arena. The goal of the game is to kill as many enemies as possible in a fixed amount of time while not being killed. When a player registers for an account, they supply a unique username. Items include weapons, such as guns or melee weapons like knives, armor and attachments. Attachments are simply modifications to existing weapons like scopes, extended magazines and silencers. Attachments exist only for weapons, not for armor. A player can purchase items with coins, the in game currency that can be earned through increasing their level or via in game purchases with Canadian dollars. Players must purchase coins directly to buy new items, they can't directly purchase items with Canadian dollars. Initially players are given a set of starting items when they register for an account. Attachments are purchased on a per-item basis, meaning that if you have a certain type of scope for one gun, you don't own that same type of scope for another gun. Players can level up through gaining experience points by participating in games.

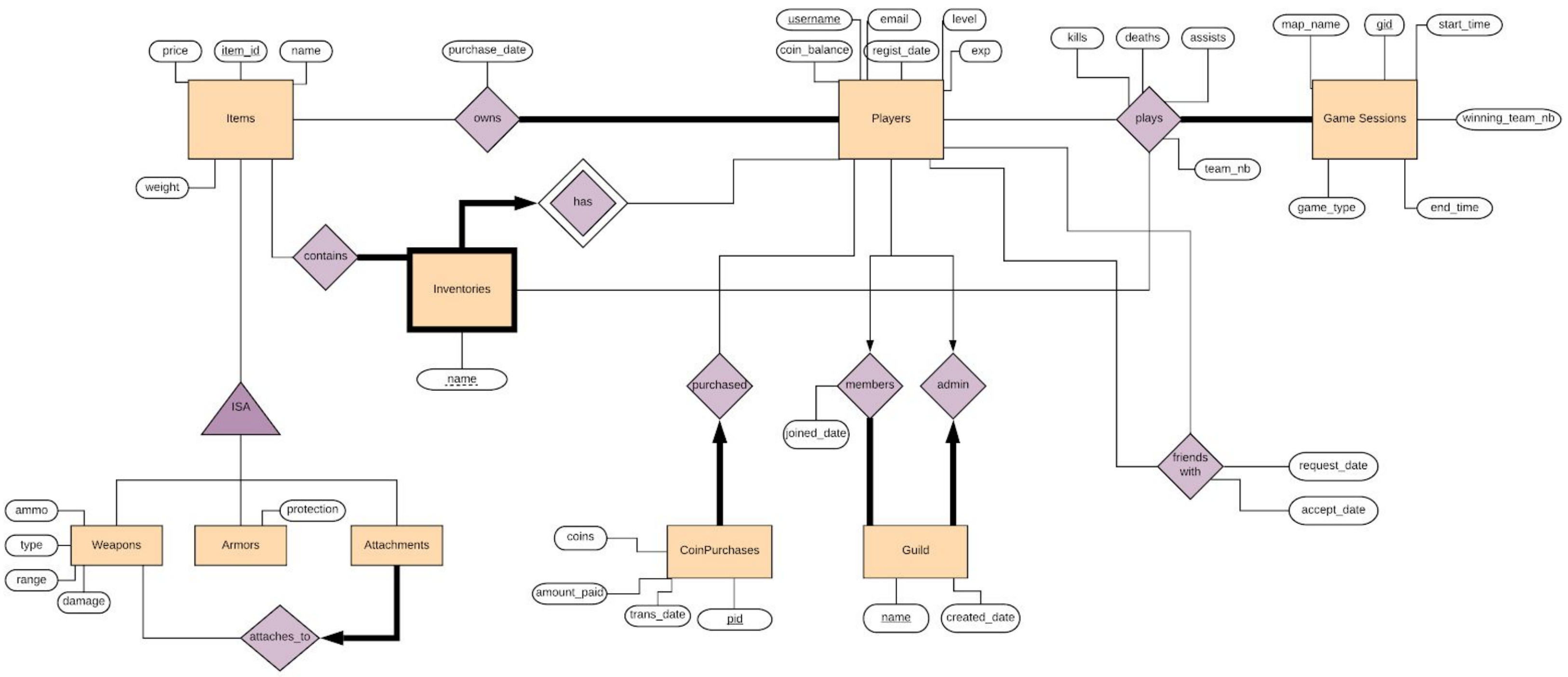
Players must prepare one or more inventories before they can start playing games. An inventory is a named collection of items consisting of a primary weapon, a sidearm (such as a pistol), a melee weapon like a knife, armor and attachments. An inventory, which is a specific item combination has a name, so that famous players can share their inventories.

Game types are Deathmatch and Free-For-All. A player enters matchmaking which pairs them with other players, in the case of a Team Deathmatch forming a game session of 2 teams each of 10 players in a randomly selected map. If Free-For-All is selected, every individual player is in their own team and everyone can kill everyone. Since each player selects their inventory to use in the game beforehand, once the game starts this can't be changed. Depending on the availability, players are assigned to game sessions with players of similar level. The game will track the number of kills, deaths and assists each player scores as well as the winning team.

On the social side of the game, players may add each other as friends. This is done by first requesting friendship, and having the other person accept the friend request. Lastly players may also join guilds, which are groups of players with a single administrator. Guild's also have a unique name so that players can easily identify them by name. Players may be a member of one guild at a time. Players may queue up to play games with friends, or play games within their guild. Players can also see view statistics about their kill to death and win to loss ratio across all their games over time, or by gun and map, so that they can determine points to improve as they compete. They may also compare these statistics with their friends and guildmates.

Call Of Duty App Game Management System and Database

Entity Relationship Diagram



Database Description

Entity Sets

- **Items:** Items are modelled as having a unique id, name and price. The price for an item is in coins. Additionally items may have a weight which may slow a player down. For instance a heavy machine gun will incur a movement penalty for the player to balance out the fact that it has incredible damage potential.
- **Players:** Players register with a unique username, we also track their email to keep in touch with them. We also store how many coins they have in their balance, their level, and the amount of experience they’ve accumulated, which defines their level.
- **Game Sessions:** A game session describes a single game, it has attributes for the start and end time, the winning team, along with the map and game type.
- **Coin Purchases:** Players can purchase coins with Canadian currency. We store the number of coins purchased and the amount of CAD paid as well as the transaction date for each purchase.
- **Guilds:** A guild must have at least 1 player and 1 admin. A guild can have multiple admins. A Player can belong to one guild at a time, and can administer one guild at a time.
- **Weapons:** Includes guns and melee weapons. Melee weapons have a range of 0 and an ammo of infinity.
- **Armors:** Armors decrease damage percentage through their protection attribute.

- **Attachments:** Attachments can be bought for weapons. Examples would include muzzlers, ACOG scopes e.t.c. Each attachment can be bought for one weapon at a time. An ACOG for a AK-47 and AR-15 are different items.

Relationships

- Owns
- Plays
- Contains
- Attaches To
- Purchased
- Members
- Admin
- Friends With
- Has

Weak Entities

- **Inventories:** Inventories have a unique name on a per-player basis, since it is a weak entity than depends on Players, and thus its name is a partial key.

Missing Constraints and Design Decisions

This database models the social aspect of the game, therefore game assets, such as 3D models associated with maps and items, along with game logic are not modeled.

We made several design decisions with regards to the ER model to handle certain constraints at the application level rather than build them into the database. First off, there is no mechanism to constrain the types and number of items in an inventory, as well as checking that the items in the inventory are owned by the player, it's up to the application backend to validate this at the application level. Our model also does not capture that the guild admin must also be a member of the guild, as this is generally easy to validate at the application level.

The one major decision that we made is the way we defined attachments as a subclass of items. This allows us to avoid duplicating the item ownership logic, since attachments are treated as items in this sense. The one issue with this is that ideally attachments would be a weak entity dependent on weapons, since attachments are tied to a particular weapon by the "attaches_to" relationship. Therefore we will have duplicate item names, as some attachments have the same name, yet are compatible with different guns. This is the reason we chose to make items have a unique id, because we can't make attachments a weak entity since that would violate the Liskov Substitution principle, as an attachment is an item, and items are not weak entities depending on weapons. We decided that the cost of having to duplicate the item ownership logic for attachments is worse than having duplicate item names without a weak entity.

Call Of Duty App Game Management System and Database

Application Description

Overview

Since the first-person shooter game requires multiplayer functionality, a backend with a database is necessary to reliably store information about game sessions, weapon ownership, player friendships and guild information etc. Our database implementation focuses on providing this functionality, along with allowing for computing statistics for players. The database captures and stores information pertaining to various aspects of the game which allows for computing statistics to be both viewed by the users for their own use to improve at the game.

Designing for Statistics

Our DB model is designed with computing statistics in mind. We keep track of what items and attachments are brought into a game by every player, along with tracking the kills deaths and assists each player scored in the game. We also track the winning team of each game, so players can track either their win/loss ratio and kill/death ratio to see how well they stack up against their friends and other players.

Typically in multiplayer online games, the developers try to ensure that there are no intrinsic advantages built in to the game, i.e. the game remains balanced. For example, we require in principle that no gun has a global win/loss ratio not equal to 50%. This is obviously infeasible in practice, but data in our database can help inform developers on which guns to weaken and strengthen in future patches to help nullify these global imbalances.

Personal Statistics Query Examples

1. For a given player, what is their win/loss or kill/death ratio with a particular inventory? We also may specifically track these ratios for a particular item that they use across different inventories.
2. How did a player’s win/loss or kill/death ratio improve over time (in buckets of 1-month).
3. Compute the leaderboard of win/loss and kill/death ratio for all the players in a guild.

Our application will allow for a dashboard for players to visualize such statistics so they can identify where they need to improve, and allow for filtering by game time, game type, the map, which team the player was on etc…

Game Balance Statistics Query Examples

1. Is the win/loss ratio for a particular team number 50% for a given map? Since team number affects which side of the map you start on, some maps may not be designed well and may have more cover or high ground on one side, offering an advantage to the team playing that side.
2. What is the global win/loss ratio for a particular gun, or particular set of items? This could help developers decide if a certain weapon is overpowered to weaken it in future patches to ensure the game is balanced.

Call Of Duty App Game Management System and Database

Relational Model

ENTITIES:

Items(item_id, name, price, weight)
Weapons(item_id, name, price, weight, ammo, type, range, damage) (item_id REF Items)
Armors(item_id, name, price, weight, protection) (item_id REF Items)
Attachments(item_id, name, price, weight, attaches_to_id)
    (item_id REF Items
    attaches_to_id REF Items)
Players(username, email, coin_balance, level, regist_date, exp, guild_id, guild_join_date)
    (guild_id REF Guilds)
GameSessions(gid, game_type, map_name, start_time, end_time, winning_team_nb)
CoinPurchases(pid, username, coins, amount_paid, trans_date) (username ref Players)
Guilds(name, created_date, admin_id) (admin_id REF Players)

RELATIONSHIPS:

Owns(username, item_id, purchase_date)
    (username REF Players,
    Item_id REF Items)
InventoryContains(name, item_id)
    (name REF Inventories,
    Item_id REF Items)
Plays(username, gid, inv_name, kills, deaths, assists, team_nb)
    (username REF Players,
    gid REF GameSessions,
    inv_name REF Inventories)
Friends(requester_id, requestee_id, request_date, accept_date)
    (requester_id REF Players,
    requestee_id REF Players)

WEAK ENTITIES:

Inventories(name, username) (username REF Players)