Developing a Conceptual Database for an MMORPG Item System using Object-Relational Concepts

Term Project for ACS 575 – Dr. Yoo

Purdue University, Fort Wayne

Alekzander Green | CS Department | greead04@pfw.edu

Abstract

The Massively Multiplayer Online Role-Playing Game (MMORPG) genre is a popular video game genre that mixes a large number of players (MMO) with many role-playing elements like combat, collecting items, and more (RPG). This mixture of high active player count and lots of player data (such as tracking inventories) leads to a massive data burden that needs to be handled. To accomplish this, a database system and backend API was developed using PostgreSQL and FastAPI. PostgreSQL allowed the use of Object Relational (OR) technologies, which more naturally represents much of the data that needed to be stored. This was an overall success, but future work will be necessary to tune the project and integrate it with a future MMORPG system.

Project Description

As an avid video game player with a soft spot for MMORPGs, I jumped at the opportunity to build a database that could stand the pressures of the genre. This is because it is well-known that MMORPGs are both difficult to develop and work with a lot of data. This, in addition to my want to develop my own MMORPG project, pushed me to develop a database for this term project.

Early in the conceptual process, it was apparent that relational database technology was not able to fully realize the needs of an MMORPG database. Therefore, OR technologies were researched and implemented to better the process.

When it came time to develop the necessary servers and applications for the database, I chose to learn new technologies rather than relying on technologies that I had used before. For this reason, the project was developed with PostgreSQL, FastAPI, and more.

In the end, a database, backend server, and client application were all successfully developed such that the project could be easily extended for future use.

Problem Statement

MMORPGs need to store a lot of data. A large part of the genre is collecting and finding new items to use in the game. This project will attempt to build a database that is capable of storing and retrieving different kinds of items in an MMORPG and allowing the game to store a user's inventory in a reliable manner. This will require the planning and implementation of a database as well as a backend system to support API calls to the database.

Objectives

Market Research – Several RPG inventory systems were investigated to see how they classify items and organize inventories. The games selected will be from an array of different RPGs from various subgenres. This will ensure that we have a good idea of how we might implement an inventory system.

Schema Design – Conceptual, logical, and physical schemas were designed to plan and demonstrate how the database should function and how it has been implemented.

Database Implementation – A database capable of storing and retrieving data concerning items, inventories, and players has been implemented. This database system mostly focuses on how items and inventories are stored in a wide array of tables, but limited player information is also stored to connect players to their inventories. The database implementation was done using PostgreSQL and pgAdmin4.

Backend Implementation – A backend API is implemented to allow for streamlined and controlled database querying. Despite my prior experience with Django and the Django REST Framework, I decided to implement the API using FastAPI. This API allows me to retrieve information for any item in the database, add new items to the database, add items to player inventories, remove items from player inventories, retrieve player states, and store player states. Further actions may be needed, but these requirements will be necessary without a doubt. In addition to the core FastAPI library, the Uvicorn library was used to host the server and a PostgreSQL database adapter called psycopg3 was used to make database queries.

Demo Application – After implementing the database and backend API, a demo application was developed to showcase the database working. A simple desktop application built in C# with .NET Core was built for these purposes. This application is very primitive, and only serves to showcase limited inventory interactions.

Testing and Refining – Testing will be done continuously as prototyping is being carried out. After the prototypes are all put together, final testing and refining will take place. Many tests were carried out using Postman.

Presentation – Once the prototype is properly refined, a presentation will be developed to highlight the technology used and the project results.

Unmet Goals – The demo application is very primitive and does not properly make use of all of the possible interactions. Additionally, many of the database tables remain largely unused. Finally, use of stored procedures and UDFs were in process, but could not be refined in time to implement correctly, so they were withheld.

Scope

Given that the system will be able to store custom data, there will not be much data available for the database. This system will focus on an MMORPG's item and inventory system. Other systems within an MMORPG may be implemented but are considered stretch goals. The item system will consider different types of items and will implement extensive tables to accommodate them. This project will:

- Implement a database system.
- Implement a backend API.
- Develop a basic demo.
- Include tables for items of different types.
- Include tables containing player inventories and/or state.

This project will not:

- Consider other aspects of an MMORPG.
- Include a game that uses the database.
- Contain extensive data within tables.
- Have extensive functionality beyond inventory and item interactions.

Market Research

World of Warcraft (WoW) [1]

Figure 1 - Example WoW Inventory

World of Warcraft (WoW) is an industry leading MMORPG where players take on the role of heroes in the mythical world of Azeroth. WoW allows players to collect all sorts of items each with their own purpose. We can broadly group items into five categories:

- Equipment (Weapons, armor, bags, and ammunition),
- Usable Items (Consumables and reusable items with effects),
- Quest Items (Items used for quests and no other purpose),
- Junk Items (Items with no specific use but that can be sold), and
- Ingredients (Items used in crafting)

However, despite the different types of items available, all items go into the same bags, and it is up to the player to organize their inventory. Player inventories in WoW consist of a Backpack that all players have equipped (and cannot remove) and up to four additional bags that must be equipped. Each of these bags has different sizes which determines how many items the player can place within the bag. For instance, the Backpack has 16 inventory slots. Additionally, some bags have restrictions on what kind of items can go into them e.g., only arrows can go into a quiver.

Multiple items will be grouped together into "stacks" of items. It should be noted that different items have different maximum stack sizes, with items like armor having a maximum stack size of 1 and items like potions having a maximum stack size of 20. Once a stack reaches its maximum stack size, excess items will begin a new stack in another inventory slot.

Additionally, players have access to a large bank where players can store items that they do not need immediate access to on their adventures. These banks are expandable by adding bags to them in the same way a player equips bags

WoW Research Summary

- There are five basic categories of items,
- Items are collected into stacks of like items with a limit depending on the item,
- Stacks are contained within equipped bags and take up one inventory slot each,
- Players have between 1 and 5 bags equipped with different sizes,
- Some bags have restrictions on the types of items that can go into them,
- Player inventories are not otherwise divided, and
- Players have access to a bank where they can store more items.

Dungeons and Dragons Online (DDO) [2]

Figure 2 - Example DDO Inventory

Dungeons and Dragons Online (DDO) is a relatively dated MMORPG set in the Dungeons and Dragons Eberron campaign setting. Players in DDO take up the mantle of questing adventures and inevitably collect all sorts of items in their journeys. Items in DDO are categorized very similarly to those in WoW, with two additional categories:

- Storage items (used to store items within), and
- Special items (items that themselves have entire systems devoted to their use)

Unlike WoW, DDO does not have an intricate bag system. Players all have access to three 20-slot bags and can purchase up to three additional bags for a total of six bags. Like WoW, items are placed into stacks within the bags and have different stack sizes depending on the type of item. Finally, DDO also has a bank system that allows players to store additional items. Due to some complexities unique to DDO, some items get very complex in their use, which is why the categorization of items can become difficult.

DDO Research Summary

- Many complex items make categorization difficult,
- Items are collected into stacks of like items with a limit depending on the item,
- Stacks are contained within bags and take up one inventory slot each,
- Players have between 3 and 6 bags that are all the same size,
- The inventory has no restrictions or divides, and
- • Players have access to a bank where they can store more items.

Guild Wars 2 (GW2) [3]

Figure 3 - Example GW2 Inventory

Guild Wars 2 (GW2) is another fantasy MMORPG like both WoW and DDO. The Inventory system in GW2 is nearly identical to that of WoW. The one key difference is that items are either stackable or not stackable, and all stackable items stack up to 250 items per inventory space. Additionally, players can equip many more bags (up to 13) but all items exist within a single scrollable inventory view (unlike in WoW and DDO where you have to open specific bags to view items within them).

GW2 Research Summary

Similar in most respects to WoW, but with the following differences:

- Simplified item stacking system, and
- A central inventory for all inventory items where bags increase the inventory size.

Final Fantasy XIV (FFXIV) [4]

Figure 4 - Example FFXIV Inventory

Final Fantasy XIV (FFXIV) is another fantasy RPG, and it allows us to begin seeing a trend. The inventory system is very similar to that of DDO, with stacking like GW2 (but up to 999 of an item).

FFXIV Research Summary

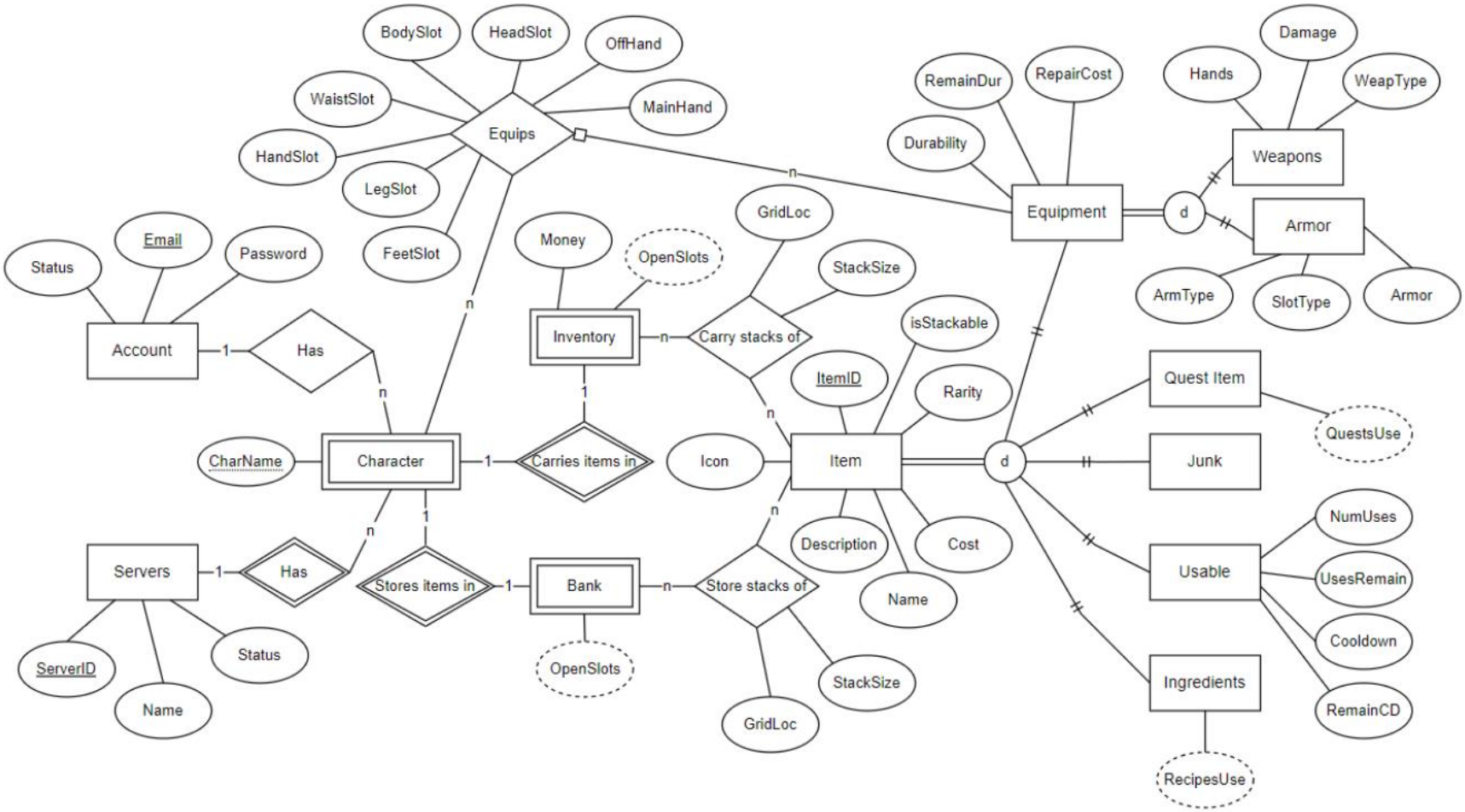
No additional information was gained besides reinforcement of inventory ideas from other games in different ways.

Overall Market Research Summary

With the addition of FFXIV, we can see that many MMORPG inventory systems follow a similar formula with the largest differences being how they handle bags (bags as equipment or bags as inventory tabs) and how they handle stacking (simple stacking with stackables/non-stackables or complex stacking with different stack sizes for different items). I will be putting into practice what I believe is the simplest inventory system. The inventory system will be very similar to FFXIV's inventory system, but with item categories more in line with WoW. UsaThis means that the inventory system that will be implemented has the following features:

- Five item categories with various subcategories as necessary,
- Items that are classified as stackable or non-stackable,
- Stackable items have the same maximum stack size regardless of the item,
- Item stacks take up exactly one inventory slot,
- Players have a set amount of bags all with the same size, and
- Players will have access to a personal bank to store additional items.

Conceptual Schema



Logical Schema

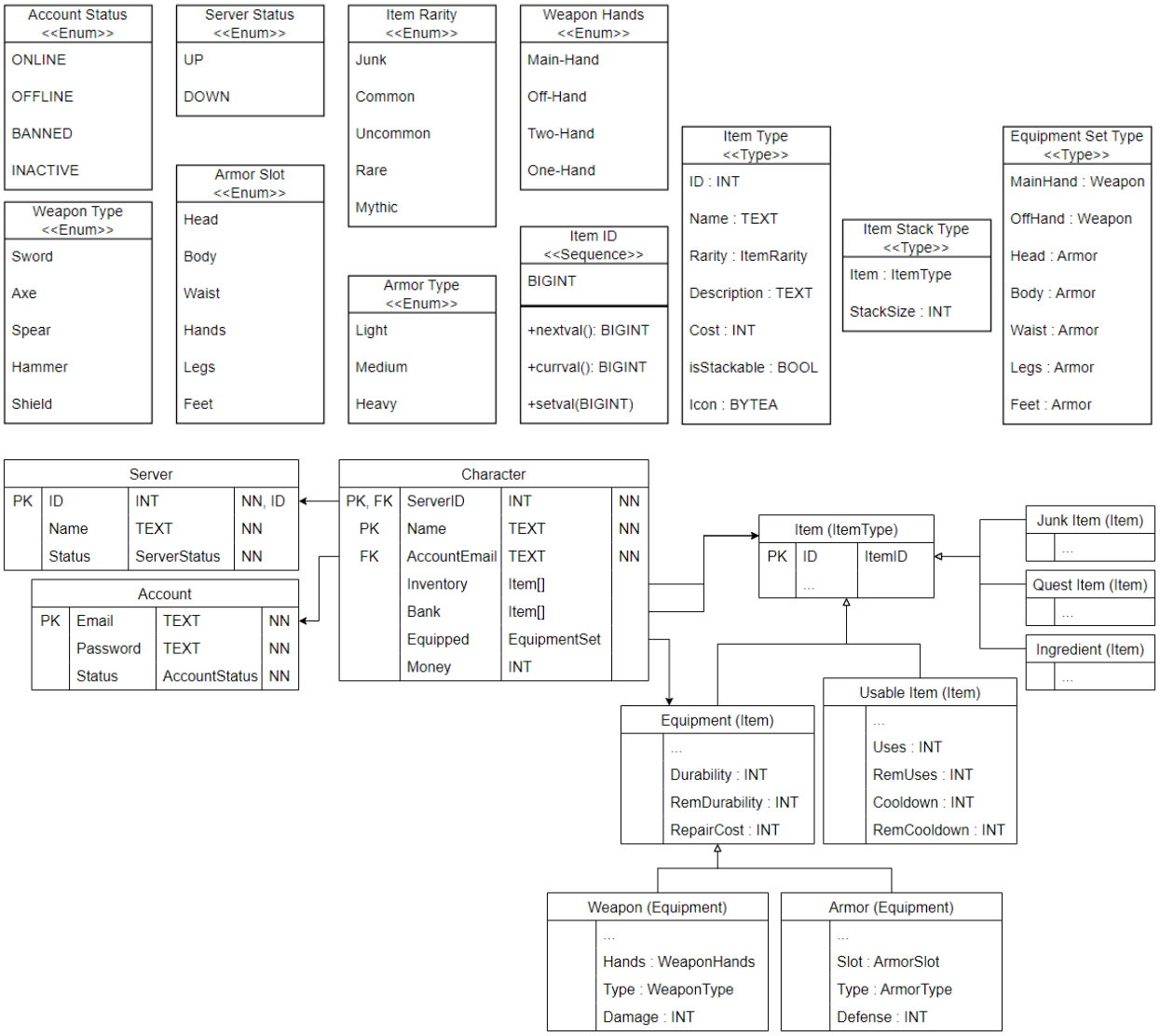
ENUMS
account_status_enum ('ONLINE', 'OFFLINE', 'BANNED', 'INACTIVE')
server_status_enum ('UP', 'DOWN')
item_rarity_enum ('Junk', 'Common', 'Uncommon', 'Rare', 'Mythic')
weapon_hands_enum ('Main-Hand', 'Off-Hand', 'Two-Hand', 'One-Hand')
weapon_type_enum ('Sword', 'Axe', 'Spear', 'Hammer', 'Shield')
armor_slot_enum ('Head', 'Body', 'Waist', 'Hands', 'Legs', 'Feet')
armor_type_enum ('Light', 'Medium', 'Heavy')

SEQUENCES
item_id_seq (BIGINT)

TYPES
item_type (id, name, rarity [item_rarity_enum], description, cost, is_stackable, icon)
item_stack_type (item [item_type], stackSize)
equipment_set_type (mainHand [weapon], offhand [weapon], head [armor], body [armor], waist [armor], hands [armor], legs [armor], feet [armor])

TABLES
item [item_type] (**id** ...)
equipment [item] (..., durability, remDurability, repairCost)
weapon [equipment] (..., hands [weapon_hands_enum], type [weapon_type_enum], damage)
armor [equipment] (..., slot [armor_slot_enum], type [armor_type_enum], defense)
quest_item [item] (...)
junk_item [item] (...)
usable_item [item] (..., uses, remUses, cooldown, remCooldown)
ingredient [item] (...)
account (**email**, password, status [account_status_enum])
server (**id**, name, status [server_status_enum])
character (***server_id [server(id)]*, name**, *account_email [account(email)]*, inventory[], bank[], equipped[equipment_set_type], money)

Physical Schema



System Architecture

The system uses a three-tiered architecture with a database server, application server, and client application.

Database Server – The database server was built using PostgreSQL as the DBMS of choice. Development occurred using pgAdmin4. The database server runs as a service on my local machine.

Application Server – The application server was built using the Python FastAPI library. The FastAPI app is served via Uvicorn. The FastAPI app connects to the local database server using the psycopg3 database adapter. All requests are made via HTTP with JSON data passed as inputs and outputs.

Client – The client application is a simple Windows Forms App built using C# and .NET Core. This is a GUI application that makes requests using the built-in HTTPClient and async structures.

DBMS Technology and Development Methods

The system was developed using various technologies and development methods.

PosgreSQL – PostgreSQL (Postgres) is an Object-Relational (OR) DBMS that was used to implement a database server. Several types and tables were developed in accordance with Postgres's standards of implementation. This DBMS was chosen as it is open-source and widely popular in addition to providing many OR features that were chosen for this project.

pgAdmin4 – This is the application that comes with the installation of Postgres. This application was used to write queries and develop the database.

FastAPI – This is a Python API library. This library was used to develop a simple backend API using various functions with decorators to enable efficient routing. This framework was chosen to develop the backend because it was a new technology that I had been wanting to explore.

Uvicorn – This is a Python server library. This library was used to host the FastAPI application. It was chosen for its simplicity in addition to FastAPI being built on top of Starlette, an extension of Uvicorn.

Psycopg3 – This is a Python-PostgreSQL database adapter library. This library was used to query the database server for the purpose of responding to the client. This was chosen over other technologies like SQLAlchemy because it is a popular database adapter rather than an ORM. The reason for this being a factor is because I was unsure of how difficult it would be to use an ORM for some of the Postgres-specific OR concepts that were implemented.

Visual Studio Code – This is my editor of choice, and is the editor I used to develop the backend API.

Windows Form App (C# .NET Core) – This is a template provided by Visual Studio to develop desktop applications using C# and the .NET Core framework. I chose this because it is simple to use, and I am relatively familiar with the technology.

Visual Studio – This is a project editor that I used to develop the desktop client.

Data and CRUD Options

The prototype system will need custom data to be created to develop each table in the database. Data will include basic information for the following:

- Player accounts,
- Game Servers
- Player characters,
- Character inventories & Inventory Items,
- Character banks & Bank Items, and
- Item data.

Each will require operations to be carried out on the data. These operations are detailed in the following sections.

Player Account Operations

Players and game systems should be able to do the following operations with the account information:

Create – Create a new account

Read – Get account status, get login info (should be done with an authentication service)

Update – Update account status

Delete – N/A (Accounts should not be easily deleted)

Game Server Operations

Players and game systems should be able to do the following operations with the server information:

Create – N/A (Servers should not be created frequently)

Read – Get server status

Update – Change server status

Delete – N/A (Servers should not be easily created)

Player Character Operations

Players and game systems should be able to do the following operations with the character information:

Create – Create new characters for a given account on a given server

Read – Get name, get equipped items list, get inventory, get bank

Update – N/A (Current character information does not require updating; however, expansion of this database would allow other character state information to be stored here long-term)

Delete – Delete characters (CASCADE delete when server or account is deleted)

Character Inventory Operations

Players and game systems should be able to do the following operations with the inventory information:

Create – Create a new inventory for a new character

Read – Get character inventory list

Update – Update the inventory list

Delete – N/A (CASCADE delete when character is deleted)

Item (Equipment, Quest, Junk, Usable, Ingredients) Operations

Developers and game systems should be able to do the following operations with the item information (note that these operations are similar for each item table, but there exists 7 item tables):

Create – Add new items to the database

Read – Retrieve item information given an item ID

Update – Tune item information and statistics, update dynamic stats (equipment & usable only)

Delete – Remove items from a database

Prototype Functionality

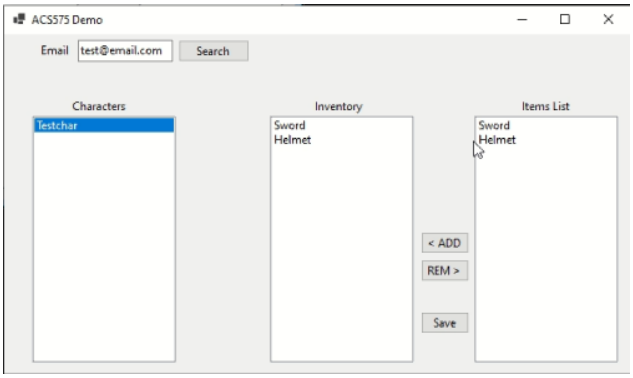


Figure 5 - Prototype System Functionality

The prototype system is very primitive in nature. It is restricted to getting a character list for a given account, showing the inventory of a selected character, adding and removing items from the character's inventory based on the item list, and saving the updated inventory to the database.

Much of this functionality is done through the HTTPClient system built in to C# .NET Core, as shown below.

```
1 reference
private async void btnSearch_Click(object sender, EventArgs e) {
    lstCharacters.Items.Clear();
    var query = txtEmail.Text.Trim();
    var request = new HttpRequestMessage {
        Method = HttpMethod.Get,
        RequestUri = new Uri("http://localhost:8000/account/characters"),
        Content = new StringContent(
            JsonSerializer.Serialize(new CharacterCredentials { email = query }),
            Encoding.UTF8,
            "application/json" ),
    };
    using HttpResponseMessage response = await client.SendAsync(request);
    var respStr = await response.Content.ReadAsStringAsync();
    Debug.WriteLine(respStr);
    JsonNode? respJson = JsonNode.Parse(respStr);
    foreach (var item in respJson.AsArray()) {
        lstCharacters.Items.Add(new CharacterItem(item["name"].ToString(), item["server_id"].ToString()));
    }
}
```

Figure 6 - Example Code Showing a Request for the Character List of a Given Account

Additional functionality is not available due to time constraints.

Validation

Much of the system was tested using pgAdmin4 to query the database tables after operations, debugging in Visual Studio, and sending various requests via Postman.

Discussion

Overall, the project was a success. A database was successfully implemented that is able to handle accounts, servers, characters, inventories, items, etc. for an MMORPG system. This system should be easily expandable to serve a full MMORPG project in the future. Additionally, this system makes use of many OR technologies. The use of OR technology was the main driving force for this project. Additionally, I have learned a lot when it comes to using Postgres and OR systems.

In addition to the database, a database API server was also successfully implemented. The server is fully functional and fully capable of interacting with the database in each and every intended manner. The API should be easily extendible if needs arise in the future. Otherwise, the backend can be migrated to Django or similar, more scalable backend frameworks. However, being able to use three new technologies (FastAPI, Uvicorn, and psycopg3) while developing the backend made this project invaluable to me. I've learned quite a lot regarding connecting to database servers and hosting API endpoints for them.

Finally, a primitive client was successfully implemented. This client does not do as much as I wish it did, but it does showcase the database and API in action, which is its intended purpose.

I had trouble working with the types and inherited tables, which has caused a few functions to be limited in nature. This is especially the case when trying to work with database adapters and clients that are not prepared to interface with these OR concepts. This wasn't so bad when working with Python on the backend server but became very apparent when working with C# on the client. This is because, while Python is much more forgiving, C# was very strict regarding datatypes and the structures required to make HTTP services work. C# was actually a very large roadblock to finishing the rest of the project, as I was not prepared to deal with HTTP services using C#. I feel more confident now, but it is still tiring to work with.

In the future, the API should be expanded to care for many more cases, and the database should be refined and filled with more items, tables, etc. This can then be integrated with an MMORPG project.

Conclusion

A three-tiered system with a database, backend, and frontend client was successfully developed. I learned to use PostgreSQL, pgAdmin4, FastAPI, psycopg3, Uvicorn, C#'s async system, C#'s HTTP system, OR concepts, and more. Each of these technologies were things that I knew nothing about before this project, but now I feel I have become much more acquainted with. Learning these technologies will help me to work with these and similar technologies in my future work. I feel much more confident with the idea of working with databases and backend APIs, whether it be for the purpose of personal projects or for future industry and research projects.

References

[1] Fanbyte, "Inventory (WoW)," 17 June 2008. [Online]. Available: https://wow.allakhazam.com/wiki/Inventory_(WoW). [Accessed March 2024].

[2] DDOWIKI, "Inventory System," 16 October 2023. [Online]. Available: https://ddowiki.com/page/Inventory_system. [Accessed March 2024].

[3] Guild Wars 2 Official Wiki, "Inventory," 4 March 2024. [Online]. Available: https://wiki.guildwars2.com/wiki/Inventory. [Accessed March 2024].

[4] Square Enix, "Gear and Inventory," [Online]. Available: https://na.finalfantasyxiv.com/uiguide/equipment/#category-equipment. [Accessed March 2024].