World of Gamecraft

In this project, we will develop a game system for a fictional RPG video game. In this game, players can create characters and venture out into the world where they can get gear and fight other players. Because the system needs to be scalable in order to support a large number of players, we have adopted a microservice architecture. During the analysis process, the following services were identified:

1. Account Service
2. Character Service
3. Combat Service

All of the services should be implemented using the standard microsoft stack (Java Technologies).Each of the services should maintain its own database and own its own schema. The schema should be defined by a set of migrations that should be run on each startup of the service.

To implement the project, the following infrastructure will be needed:

- Redis - for caching
- RabbitMQ - for inter process communication
- SQL Server - for storing data

All of these services can be provisioned using docker compose. There is no need to develop any UI for the system (although some example test scripts would be very much appreciated). Feel free to develop using either a single or multiple solutions.

Account Service

The Account Service allows users to register accounts into the system. After registration, the API exposes a login route which mints JWT tokens (via username/password). The JWT should include the role of the user which can be either User or GameMaster. This token should be included in all subsequent requests to the other services as a bearer token.

Character Service

The Character Services allows registered users to create and manage their characters in the world. This service is where most of the domain logic is implemented.

Entities

1. *Character* - represents a character in the game world
    - Id - Unique Id
    - Name - Unique String
    - Health - Int
    - Mana - Int
    - BaseStrength - Int
    - BaseAgility - Int
    - BaseIntelligence - Int
    - BaseFaith - Int

    - Class - The class of the character
    - Items - The items of the character
    - CreatedBy - Id - the id of the user that owns the character
2. *Class* - each character has an associated class (example 'Warrior', 'Rogue', 'Mage', 'Priest', etc)
    - Id - Unique Id
    - Name - Unique String
    - Description - String
3. *Item* - items which can be wielded by characters. Each item grants the character a boost to one of their statistics (str, agi, int, fth)
    - Id - Unique Id
    - Name - Not Null
    - Description - Not Null
    - BonusStrength - Int
    - BonusAgility - Int
    - BonusIntelligence - Int
    - BonusFaith - Int

Strength, agility, intelligence and faith are called stats. Players can have mutliple copies of the same item.

What to implement:

1. During startup, after applying all the migrations, seed the database with character, item and item ownership data.
2. Implement the following API endpoints:
    - /api/character - GET - lists the characters with their names, health and mana. Only accessible to Game Masters.
    - /api/character/{id} - GET - gets all the information for a given character by their Id. The stats are calculated as the **sum of the base stats with with all the bonus stats** from the items. Lists all of the associated items in the response as well. Accessible to Game Masters and Owners of the character. This request needs to be cached. The cache needs to be invalidated when the query result changes.
    - /api/character - POST - creates a new character with the specified name, class and stats. Accessible to all users.
    - /api/items - GET - lists all items in the system. Only accessible to Game Masters.
    - /api/items - POST - creates an item with the provided data.
    - /api/items/{id} - GET - gets all the details of the given item. The item name is defined in the following way:
        - if BonusStrength is the largest stat, the name gets the suffix ' Of The Bear' ('Sword' -> 'Sword Of The Bear')
        - if BonusAgility is the largest stat, the name gets the suffix ' Of The Cobra' ('Sword' -> 'Sword Of The Cobra')
        - if BonusIntelligence is the largest stat, the name gets the suffix ' Of The Owl' ('Sword' -> 'Sword Of The Owl')
        - if BonusFaith is the largest stat, the name gets the suffix ' Of The Unicorn' ('Sword' -> 'Sword Of The Unicorn')
        - brake ties in any way that seems simplest
    - /api/items/grant - POST - grants a character a specific item

    - /api/items/gift - POST - moves an item from one player to another
3. Implement unit tests for all the operations specified
4. (Bonus) During character creation, check if the user id in the jwt token exists (is valid).
5. (Bonus) Add logging

Combat Service

The Combat Service deals with the way combat happens in the system. It lets characters (which are created in the characters service) participate in combat and earn items (if victorious). All of the communication with other services happens via RabbitMQ (both incoming and outgoing communication). The Combat Service should also expose an API which lets players use it directly.

Description

The Combat Service allows characters to challenge other characters to duels. Duels cannot be denied. During a duel, players can do one of three actions:

- attack - deals `strength + agility` in damage - once every second
- cast - deals `2 * intelligence` in damage - once every 2 seconds
- heal - restores `faith` in health - once every 2 seconds

Dealing damange in this context means reducing the health of the other player by the damage value. Restoring health means increasing the players health by the health value. Combat lasts until one of the players health reaches zero. If a duel lasts longer than 5 minutes, declare it a draw (there is no winner or loser).

The player who's health has reached zero during the duel is the loser, while the other player is the winner. The winner gets one random item from the loser.

What to implement

1. During startup, after applying all the migrations, seed the database with characters so as to be synced with the characters service
2. Implements the following API endpoints:
    - /api/challenge - POST - the body contains the challenger and challangee fields (both are character ids). Returns the duel id. Only users that own the challenger character can issue challenges.
    - /api/{duel_id}/attack - POST - one character attacks the other. Only users participating in the duel can issue combat commands.
    - /api/{duel_id}/cast - POST - one character casts a spell on the other. Only users participating in the duel can issue combat commands.
    - /api/{duel_id}/heal - POST - one character heals himself. Only users participating in the duel can issue combat commands.
3. Implement a mechanism to sync the database state with the state of the character service (i.e during player registration). Tables don't need to be replicated entirely, so long as it makes sense in the domain.
4. After settling a duel, notify the Character Service in order to swap the actual items.
5. (Bonus) Add logging