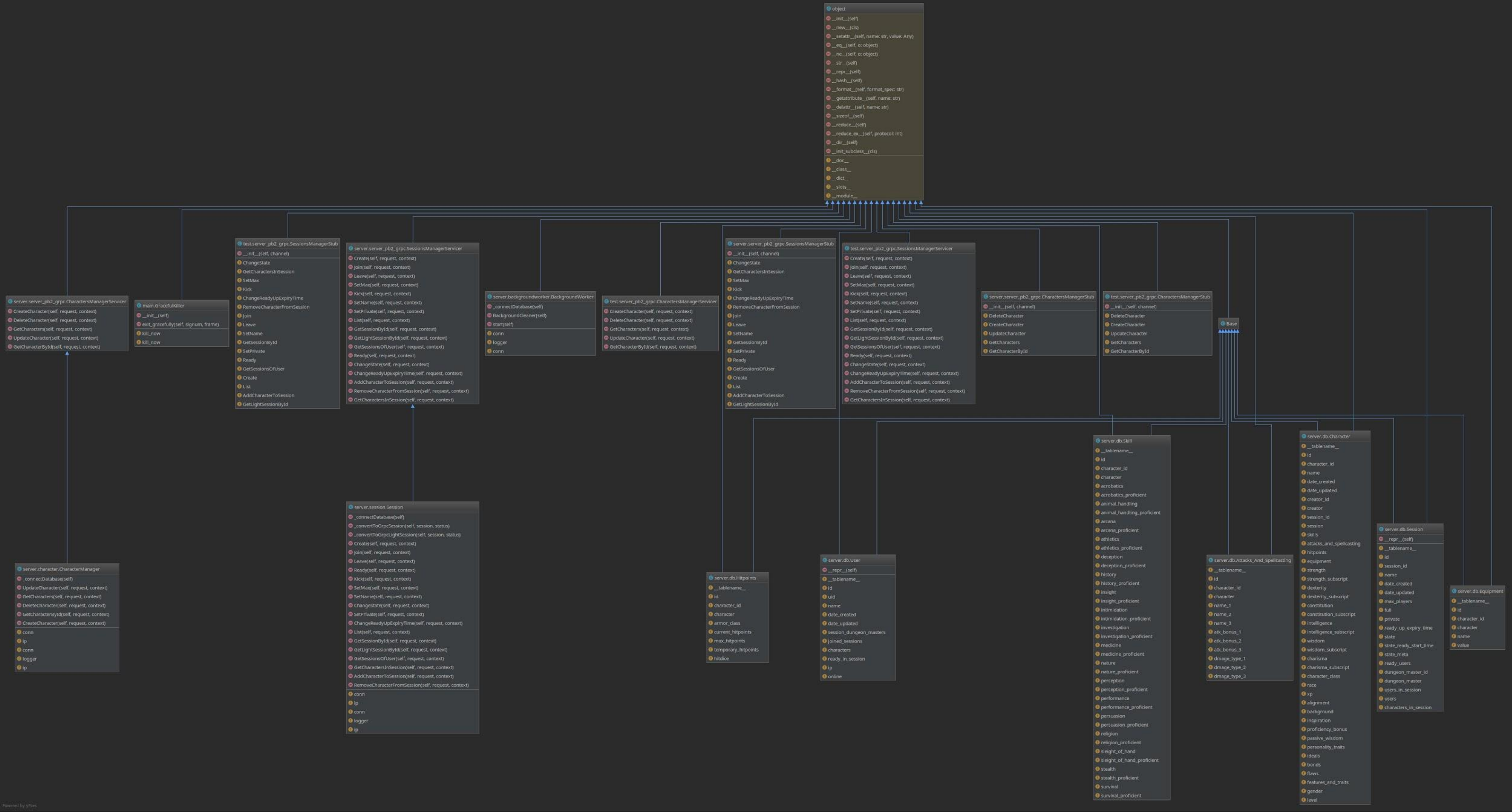
Dungeons & Dragons Game State Manager

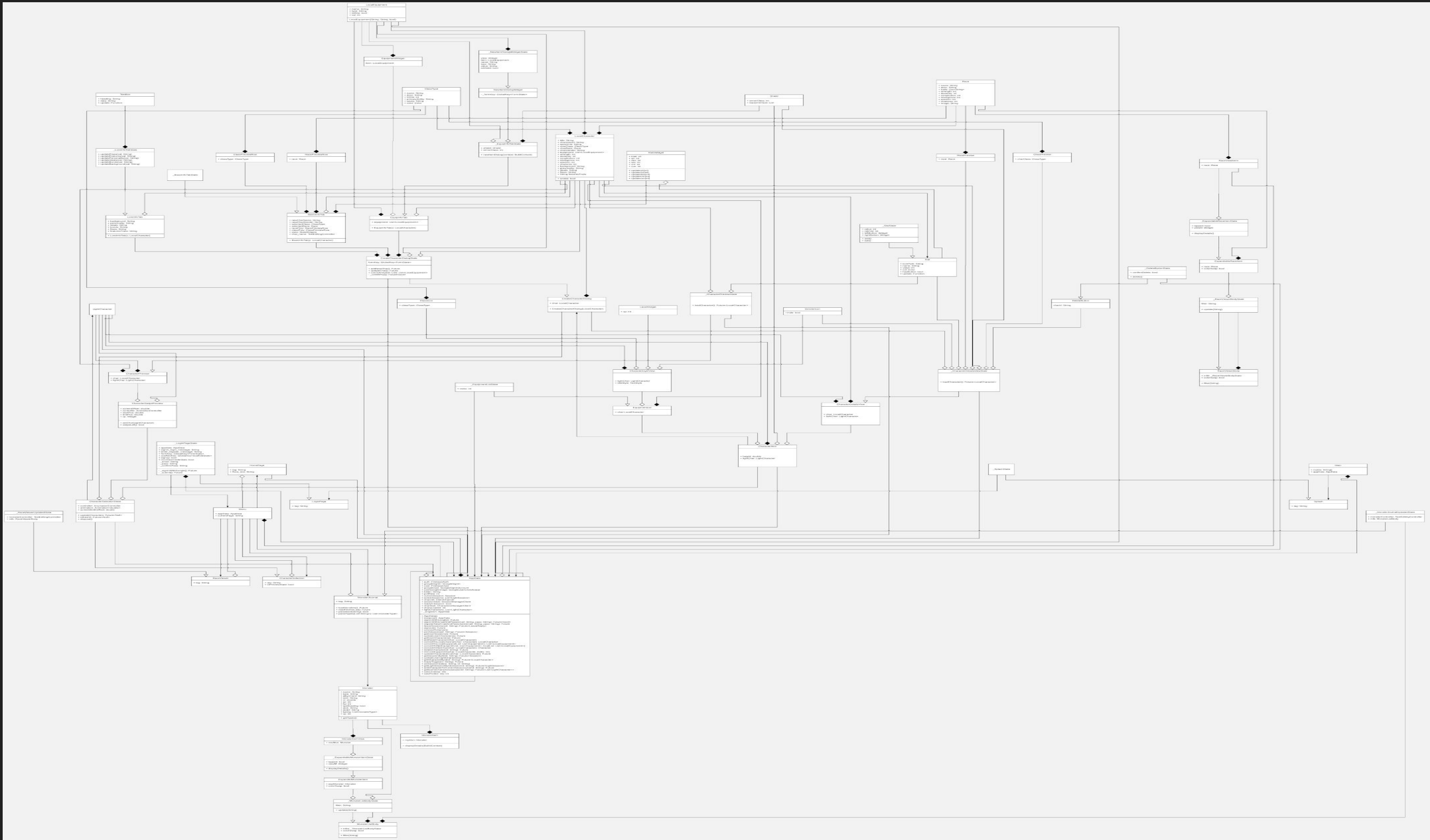
SRS

Developed for Retro Rabbit by Optimize Prime

Overview

The application is intended to act as a helper and assistant when users are playing a game of Dungeons & Dragons. The application tracks information and events of the current game session being played, to provide players with an easier way to see the current state of the game, compared to keeping track manually on paper. It will provide users with an easier way to keep track of their current characters, as well as creating new characters or updating information on existing characters. Users will also have the ability to look at a monster journal, which shows all monsters that have been encountered so far, and allows users to add monsters that they encounter later.


object
__init__(self)
__new__(cls)
__setattr__(self, name: str, value: Any)
__eq__(self, o: object)
__ne__(self, o: object)
__str__(self)
__repr__(self)
__hash__(self)
__format__(self, format_spec: str)
__getattribute__(self, name: str)
__delattr__(self, name: str)
__sizeof__(self)
__reduce__(self)
__reduce_ex__(self, protocol: int)
__dir__(self)
__init_subclass__(cls)
__doc__
__class__
__dict__
__slots__
__module__
server.server_pb2_grpc.CharactersManagerServicer
CreateCharacter(self, request, context)
DeleteCharacter(self, request, context)
GetCharacters(self, request, context)
UpdateCharacter(self, request, context)
GetCharacterById(self, request, context)
main.GracefulKiller
__init__(self)
exit_gracefully(self, signum, frame)
kill_now
kill_now
test.server_pb2_grpc.SessionsManagerStub
__init__(self, channel)
ChangeState
GetCharactersInSession
SetMax
Kick
ChangeReadyUpExpiryTime
RemoveCharacterFromSession
Join
Leave
SetName
GetSessionById
SetPrivate
Ready
GetSessionsOfUser
Create
List
AddCharacterToSession
GetLightSessionById
server.server_pb2_grpc.SessionsManagerServicer
Create(self, request, context)
Join(self, request, context)
Leave(self, request, context)
SetMax(self, request, context)
Kick(self, request, context)
SetName(self, request, context)
SetPrivate(self, request, context)
List(self, request, context)
GetSessionById(self, request, context)
GetLightSessionById(self, request, context)
GetSessionsOfUser(self, request, context)
Ready(self, request, context)
ChangeState(self, request, context)
ChangeReadyUpExpiryTime(self, request, context)
AddCharacterToSession(self, request, context)
RemoveCharacterFromSession(self, request, context)
GetCharactersInSession(self, request, context)
server.backgroundworker.BackgroundWorker
_connectDatabase(self)
BackgroundCleaner(self)
start(self)
conn
logger
conn
test.server_pb2_grpc.CharactersManagerServicer
CreateCharacter(self, request, context)
DeleteCharacter(self, request, context)
GetCharacters(self, request, context)
UpdateCharacter(self, request, context)
GetCharacterById(self, request, context)
server.server_pb2_grpc.SessionsManagerStub
__init__(self, channel)
ChangeState
GetCharactersInSession
SetMax
Kick
ChangeReadyUpExpiryTime
RemoveCharacterFromSession
Join
Leave
SetName
GetSessionById
SetPrivate
Ready
GetSessionsOfUser
Create
List
AddCharacterToSession
GetLightSessionById
test.server_pb2_grpc.SessionsManagerServicer
Create(self, request, context)
Join(self, request, context)
Leave(self, request, context)
SetMax(self, request, context)
Kick(self, request, context)
SetName(self, request, context)
SetPrivate(self, request, context)
List(self, request, context)
GetSessionById(self, request, context)
GetLightSessionById(self, request, context)
GetSessionsOfUser(self, request, context)
Ready(self, request, context)
ChangeState(self, request, context)
ChangeReadyUpExpiryTime(self, request, context)
AddCharacterToSession(self, request, context)
RemoveCharacterFromSession(self, request, context)
GetCharactersInSession(self, request, context)
server.server_pb2_grpc.CharactersManagerStub
__init__(self, channel)
DeleteCharacter
CreateCharacter
UpdateCharacter
GetCharacters
GetCharacterById
test.server_pb2_grpc.CharactersManagerStub
__init__(self, channel)
DeleteCharacter
CreateCharacter
UpdateCharacter
GetCharacters
GetCharacterById
Base
server.character.CharacterManager
_connectDatabase(self)
UpdateCharacter(self, request, context)
GetCharacters(self, request, context)
DeleteCharacter(self, request, context)
GetCharacterById(self, request, context)
CreateCharacter(self, request, context)
conn
ip
conn
logger
ip
server.session.Session
_connectDatabase(self)
_convertToGrpcSession(self, session, status)
_convertToGrpcLightSession(self, session, status)
Create(self, request, context)
Join(self, request, context)
Leave(self, request, context)
Ready(self, request, context)
Kick(self, request, context)
SetMax(self, request, context)
SetName(self, request, context)
ChangeState(self, request, context)
SetPrivate(self, request, context)
ChangeReadyUpExpiryTime(self, request, context)
List(self, request, context)
GetSessionById(self, request, context)
GetLightSessionById(self, request, context)
GetSessionsOfUser(self, request, context)
GetCharactersInSession(self, request, context)
AddCharacterToSession(self, request, context)
RemoveCharacterFromSession(self, request, context)
conn
ip
conn
logger
ip
server.db.Hitpoints
__tablename__
id
character_id
character
armor_class
current_hitpoints
max_hitpoints
temporary_hitpoints
hitdice
server.db.User
__repr__(self)
__tablename__
id
uid
name
date_created
date_updated
session_dungeon_masters
joined_sessions
characters
ready_in_session
ip
online
server.db.Skill
__tablename__
id
character_id
character
acrobatics
acrobatics_proficient
animal_handling
animal_handling_proficient
arcana
arcana_proficient
athletics
athletics_proficient
deception
deception_proficient
history
history_proficient
insight
insight_proficient
intimidation
intimidation_proficient
investigation
investigation_proficient
medicine
medicine_proficient
nature
nature_proficient
perception
perception_proficient
performance
performance_proficient
persuasion
persuasion_proficient
religion
religion_proficient
sleight_of_hand
sleight_of_hand_proficient
stealth
stealth_proficient
survival
survival_proficient
server.db.Attacks_And_Spellcasting
__tablename__
id
character_id
character
name_1
name_2
name_3
atk_bonus_1
atk_bonus_2
atk_bonus_3
dmage_type_1
dmage_type_2
dmage_type_3
server.db.Character
__tablename__
id
character_id
name
date_created
date_updated
creator_id
creator
session_id
session
skills
attacks_and_spellcasting
hitpoints
equipment
strength
strength_subscript
dexterity
dexterity_subscript
constitution
constitution_subscript
intelligence
intelligence_subscript
wisdom
wisdom_subscript
charisma
charisma_subscript
character_class
race
xp
alignment
background
inspiration
proficiency_bonus
passive_wisdom
personality_traits
ideals
bonds
flaws
features_and_traits
gender
level
server.db.Session
__repr__(self)
__tablename__
id
session_id
name
date_created
date_updated
max_players
full
private
ready_up_expiry_time
state
state_ready_start_time
state_meta
ready_users
dungeon_master_id
dungeon_master
users_in_session
users
characters_in_session
server.db.Equipment
__tablename__
id
character_id
character
name
value
Powered by yFiles


AppData
+ auth: FirebaseAuth
+ googleSignIn: GoogleSignIn
+ user: FirebaseUser
+ googleUser: GoogleSignInAccount
+ userGoogleImage: GoogleUserCircleAvatar
+ token: String
+ port: int
+ currentSession: Session
+ activeSessions: List<LightSession>
+ channel: ClientChannel
+ sessionStub: SessionManagerClient
+ readyInSession: bool
+ charStub: CharacterManagerClient
+ charsLoaded: int
+ lightCharacters: List<LightCharacter>
- _singleton: AppData
+ AppData()
+ instance(): AppData
+ signInWithGoogle(): Future
+ signOut(): Future
+ connectToServer()
+ joinSession(id: String): Future<Session>
+ getUserSessions(): Future
+ updateUserCharacters(): Future
+ getUserCharacters(): Future
+ addNewCharacter(char: LocalCharacter)
+ convertToLocalChar(netChar: Character): LocalCharacter
+ convertToNetChar(char: LocalCharacter): Character
+ deleteCharacter(id: String): Future
+ getSessionById(id: String): Future<Session>
+ updateUserLightCharacters()
+ getCharacterById(id: String): Future<LocalCharacter>
+ readyToggled(id: String): Future
+ setSessionSide(s: String, id: String)
+ getLightSessionById(sessionId: String): Future<LightSession>
+ addCharacterToCurrentSession(charId: String): Future
+ getGameCharacters(sessionId: String): Future<List<LightCharacter>>

If you wish to see the full UML diagrams, here are the links to the 2 images, which are on our GitHub repo

1. Front-end:

   https://github.com/COS301-OptimizePrime/COS301-DnD/blob/master/Documentation/dnd_front_end_UML.png

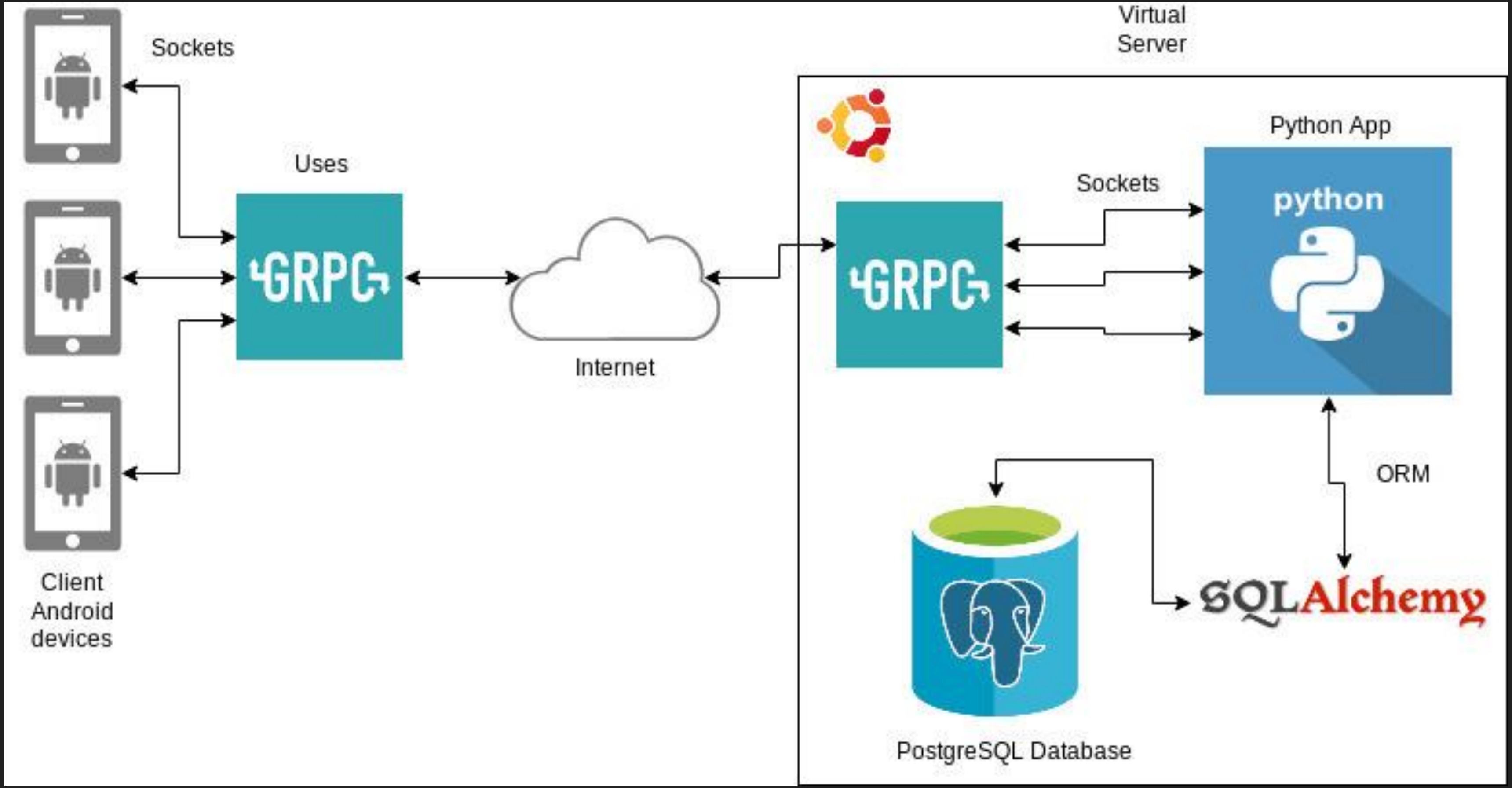
2. Back-end:

   https://github.com/COS301-OptimizePrime/COS301_DND_Backend/blob/master/COS301_DND_Backend.svg

Functional Requirements

1. Users should be able to create a game session in which they are either the player or game master.
2. Users should be able to view all their characters and create new ones or delete old ones.
3. Users should be able to view all monsters previously encountered and add to them in their monster journal.
4. Users should be notified of events that are triggered by the game master.
5. Users that are game masters should be able to set and trigger events.
6. Users should be able to view and manage their accounts.
7. Users should only be able to join a session that is near them.
8. Users should be able to add friends on the application and filter game sessions by friends.
9. The Application should provide users with informative displays about the current state of the game, such as their characters health.
10. The Application should allow the game master to track the condition of all elements of the game session, such as monsters and their stats.
11. Users should be able to create accounts and login with them or a social account.
12. Users should be able to sign out of their accounts.

Deployment Diagram:



Architectural Design Explanation:

Our system uses a custom architecture.
The application serves as the Actor at all times and is controlled by a human.
The application interacts with an external authentication service, Firebase, using an Interactive ("client-server") architecture to authenticate existing and register new users.
After this authentication process and the user has been verified, the application interacts with the game server.
The application then only interacts with the authentication server again once a sign-out has been requested by the user. The application then communicates with the game server in regards to session requests and server responses.
In this way the game server is similar to an Event-Driven architecture in that it maintains a virtual state for each game session and responses are pushed to clients when an event is triggered.

A more detailed approach of each subsystem of our custom architecture follows.

Explanation Cont:

D&D Application:

- The D&D application performs a basic authentication request with the users credentials via http to the Firebase Server.
- The Firebase Server then responds with an Authentication Token (AuthToken) or with an empty response, indicating incorrect credentials.
- The D&D application also communicates with the Game Server to create,join or leave a game session.
- The D&D application also requests the users Monsters and Characters from the Firebase Database once the user is verified.
- The D&D application is “made up” of user interactive Screens populated with Widgets.
- The D&D application makes queries to the Firebase Database to be able to populate itself with Characters and Monsters specific of the current User.
- The D&D application receives pushed events when they occur during the game, and updates the game Screen as necessary.

Explanation Cont:

Firebase:

- Firebase is Google's free authentication and database management system, complete with an API for communicating securely with the service.
- Firebase interactions take place in a "Client-Server" manner and therefore this subsystem follows an Interactive architecture with the D&D application.
- Firebase is used by the D&D application to authenticate users with received credentials via http requests.
- Firebase then attempts to find a matching user in its user base and return that Users unique Authentication Token (AuthToken).
- Firebase also holds the databases for Monsters and Characters created by Users of the D&D application.
- These databases contain data on the item (JSON files) as well as the Authentication Token of the User who created it. This applies to Monsters and Characters.
- This Firebase Database service returns these items to the D&D application when they are requested.

Explanation Cont:

Firebase Database:

- The Firebase Database holds a database of Users and their credentials as well as their Authentication Token (AuthToken) which is handled internally in Firebase systems.
- The Firebase Database holds two important databases; one being for Characters and the other for Monsters.
- Each entry into either database consists of the Authentication Token of a User and a JSON file which encapsulates the data of the actual item.
- This User-Data pairing allows only select Characters and Monsters to be available to a specific User.
- The Firebase Database interacts with the D&D application using the Firebase API to query the database.

Explanation Cont:

Game Server:

- The Game Server does not respond in typical interactive “Client-Server” style and instead follows a virtualized Event-Driven architectural approach. That is to say that each Game Session is treated as a virtual server that pushes information to Users D&D applications when an event is triggered or a criteria met.
- The Game Server responds directly only on select requests, such as a list of available Game Sessions, and this response depends on the current state of the communication between the D&D application and the Game Server.
- The Game Server creates, clears and manages Game Sessions as well as the Users that are connected to them.
- The Game Server must ensure only a specific amount of Users may enter a specific game session.
- The Game server must also enforce joining policies if they are in place for each session (like a password).

Architectural Model Postface:

The architectural diagram is completed with detail to allow one to follow the workings and reasoning for each subsystem in the architecture. This is because one can always identify the higher level concept from a lower level diagram but the opposite is not always the case. In this mindset we have provided a diagram that allows one a glimpse into the basic workings of our custom architecture in the hopes that it will help one to follow the flow of information and the order of interaction between each subsystem whilst the user traverses and utilizes the application.*