Preliminary Detailed Design

T305 – Mouthpiece

11/30/2022

I. Introduction

a. Problem Statement

i. Develop an iOS mobile device-driven social media platform whose sole purpose is to seamlessly share posts, pictures, and videos for a community of athletes.

b. Motivation

i. Having a centralized application aimed towards sports, more specifically basketball to help bring the community of athletes, sports fans, team members and recruiters together.

c. Requirements

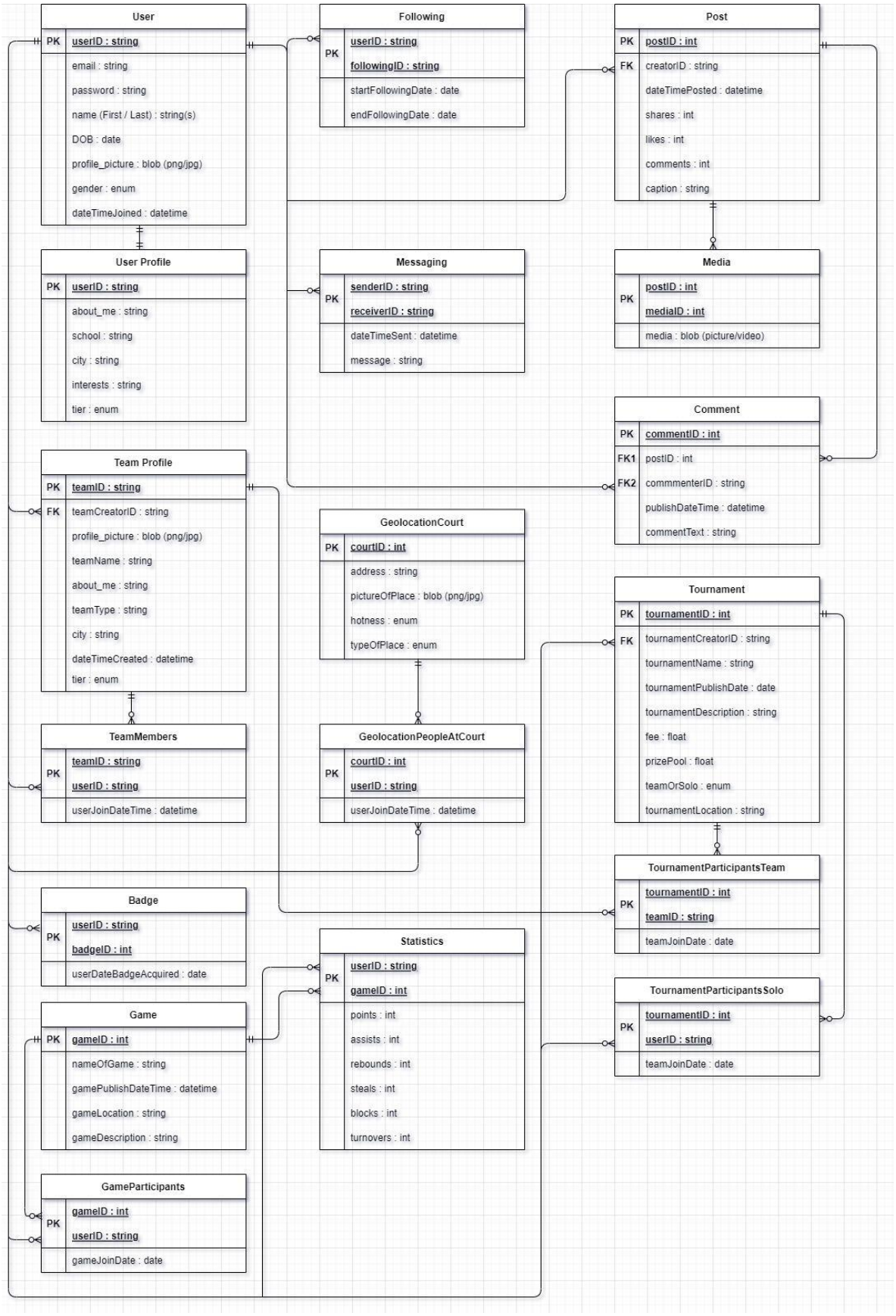
i. Have basic social-media application functionality such as sharing posts, pictures and videos as well as reacting to said content, following/friend system, messaging system
ii. Have specialized user profiles with basic user profile data, tiers, and statistics about the user.
iii. Geolocation functionality – Court/Field locator using Google Maps API, showing nearest/popular courts in the area.
iv. Team/Group profiles – For various types of teams (street/recreational/content-creator/school/group of friends), teams can perform social sharing functionality like individuals.
v. Competitive – In-game tournaments, tier panels/ranking system.

II. Selected Concept

a. For our selected design concept, we have shown the full database schematic of the application. Going from basic user profile data to tournament data, this design concept aims to show the full scope that this project hopes to implement.

III. Preliminary Design

a. Database ER (Entity-Relationship diagram)


User
PK userID : string
email : string
password : string
name (First / Last) : string(s)
DOB : date
profile_picture : blob (png/jpg)
gender : enum
dateTimeJoined : datetime
Following
PK userID : string
PK followingID : string
startFollowingDate : date
endFollowingDate : date
Post
PK postID : int
FK creatorID : string
dateTimePosted : datetime
shares : int
likes : int
comments : int
caption : string
User Profile
PK userID : string
about_me : string
school : string
city : string
interests : string
tier : enum
Messaging
PK senderID : string
PK receiverID : string
dateTimeSent : datetime
message : string
Media
PK postID : int
PK mediaID : int
media : blob (picture/video)
Comment
PK commentID : int
FK1 postID : int
FK2 commmenterID : string
publishDateTime : datetime
commentText : string
Team Profile
PK teamID : string
FK teamCreatorID : string
profile_picture : blob (png/jpg)
teamName : string
about_me : string
teamType : string
city : string
dateTimeCreated : datetime
tier : enum
GeolocationCourt
PK courtID : int
address : string
pictureOfPlace : blob (png/jpg)
hotness : enum
typeOfPlace : enum
Tournament
PK tournamentID : int
FK tournamentCreatorID : string
tournamentName : string
tournamentPublishDate : date
tournamentDescription : string
fee : float
prizePool : float
teamOrSolo : enum
tournamentLocation : string
TeamMembers
PK teamID : string
PK userID : string
userJoinDateTime : datetime
GeolocationPeopleAtCourt
PK courtID : int
PK userID : string
userJoinDateTime : datetime
TournamentParticipantsTeam
PK tournamentID : int
PK teamID : string
teamJoinDate : date
Badge
PK userID : string
PK badgeID : int
userDateBadgeAcquired : date
Statistics
PK userID : string
PK gameID : int
points : int
assists : int
rebounds : int
steals : int
blocks : int
turnovers : int
TournamentParticipantsSolo
PK tournamentID : int
PK userID : string
teamJoinDate : date
Game
PK gameID : int
nameOfGame : string
gamePublishDateTime : datetime
gameLocation : string
gameDescription : string
GameParticipants
PK gameID : int
PK userID : string
gameJoinDate : date

i. Using our choice of database, Amazon Web Services, we will create each of these entities as models represented as the core back-end of our project schema. With these models we will then store all the data/table entries using an AWS SQL (structured query language) server where we can flexibly communicate with the database back-end whether that be updating ,deleting, or retrieving any data.

b. Component Requirements

i. For our core system component, we have our user entity, here we can store user data. Most importantly we can reference a user using a unique primary key, userID. This key can be used in all major aspects of the project database. Using the userID, we can gain access to user profile/badge and follower data. For our post block, we have a unique postID key that references a creatorID to the userID mentioned above. This postID key is used to retrieve any media associated with the post, whether that be a gif, video, or picture. In the comment block, it also uses the postID to fetch all comments associated with the post. Moving over to the message block, the userID is used for both the sender/receiverID properties. Switching to the team functionality, like a user profile, we also have a block dedicated to team profile data, represented by a unique identifier, teamID. From teamID, we can reference all the team members associated with the team, again using the userIDs. For our competitive requirement, we have both a tournament and game block. Both blocks have their own unique identifier, tournament/gameID with all the data associated with the two. In a tournament we have both team and solo play functionality, so we use the teamID/userID for tournament participants. Likewise with the game functionality, we use userID to reference all the game participants. With the game participants there is also the statistics associated with the game, per each participant. In the Statistics block, we use both the userID and gameID to reference all basketball statistics for a user in a particular game. Lastly with our geolocation requirement/functionality, we need to store court information, identified with a courtID, we can store all the data associated with a court. Moreover, with court information, we also store any people at the court, again using the userID input.

c. Function Implementation

i. As said in the introduction, in order to implement each and every one of these blocks/entities we first need to make a model (code classes) with each of the properties listed in the diagram for each model as well as their data typing. With each of the model/class implemented in our codebase, we can use AWS to reference these models to connect the two

and store any instances of the models. With the database structured this way we can then easily communicate with the database whether that be with debugging or pulling manipulated data on the front-end.

d. Functional Decomposition Changes

i. No changes made between the functional decomposition and preliminary detailed design. All services/functionality are the same and implemented as such.

IV. Summary

a. With the database fully plotted out, that gives the back-end developers a good understanding of the class/model implementation that we will use in our back-end code. This entity-relation diagram does just that. Each block in this ER diagram demonstrates functionality that is to be fully implemented in development. Starting from basic user/user-profile data we can gain access to other blocks like post/comment data, with their respective media embeds, messaging/following functionality to connect with other users of the app directly. Providing a more competitive nature of the app by implementing tournament/game functionality with the statistics of each player/team. Lastly by connecting users together in-person, having built-in geolocation functionality, we can store court data as well as all the people in said court. All in all, the preliminary design discussed will give us developers/engineers a better time on both the back/front-end when it comes to development of the application in order to implement project requirements.