AmicaBet – Betting-Oriented Social Network

Taras Zherebetskyy
Graduate School of Arts and Sciences
Fordham University
New York, NY
tzherebetskyy@fordham.edu

***Abstract* — Software Engineering of a betting-oriented social network web-app which enables its users to engage in custom gambling activities.**

Keywords — Social Network, RESTful APIs, Distributed Architecture, Python, Flask

I. Introduction

This paper delves into how to build a social network, specifically the creation of AmicaBet, a betting social network built with Flask. It allows users to bet on customizable bets and invite friends to accept their challenges.

With the rapid evolution of technologies and the digitalization of different types of entertainment, the idea to digitalize also “betting between friends” came along quickly. The development of a social network involves many fields of computer science:

- **Web Development**: fundamental field for building user interfaces, functionalities, and interactions.
- **Database Management**: to store users, posts, connections, and interactions within the web application efficiently.
- **Networking and Distributed Systems**: use different protocols to allow data transmission and, optionally, distributed systems to modularize into microservices the web application.
- **Cybersecurity**: using prebuilt security features to protect user data and accounts.
- **Software Engineering**: provides an engineering-based approach to simplifying various stages of SDLC from gathering requirements to deployment and maintenance.
- **Cloud Computing**: provides a scalable and reliable infrastructure for managing and hosting web applications.

These domains are used by Facebook, LinkedIn, and Pinterest as the three most popular social networking [1] websites with unique features designed for different user needs. To use web content and its attributes, it’s essential to understand data structure, semantics, data relationships, and application programming interfaces (APIs) for data retrieval.

The standard processes for utilizing content in online social networks involve data requests (in this case via APIs), data format reorganization (querying and joining tables to achieve desired shape), and data analysis [2]. This paper also explores the complexities of managing data within a social network instance and discusses the integration of Flask, distributed architecture, and deployment on Google Cloud Kubernetes Engine. Managing data in a social network requires handling large volumes of user-generated content, profiles, connections, and interactions. With Flask, a versatile and lightweight web framework, building the network’s front-end and back-end components is more accessible, allowing for easy content delivery and user interactions. The distributed systems principles are employed to handle the scale and complexity of modern social networks, ensuring seamless data replication, fault tolerance, and load balancing, which allows for the processing and storing of user data using multiple instances to manage different loads. Google Cloud Kubernetes Engine orchestrates these containers to enhance scalability and reliability by automating application deployment, scaling, and management of the distributed architecture.

A website must be reliable and have visually appealing user interfaces, which is crucial to the success of a social network platform. Bootstrap, a popular front-end framework, is a valuable resource for achieving this. Using Bootstrap’s pre-designed responsive components, typography, and layout utilities, developers can ensure uniform and aesthetically pleasing interfaces across various devices and screen sizes [3]. This improves user experience and speeds development by providing a solid foundation for UI elements. Bootstrap’s grid system assists in creating flexible and well-organized layouts, while its vast library of customizable styles enables seamless integration with the network’s branding. Combined with Flask and the broader technological ecosystem, Bootstrap empowers developers to create visually engaging and user-friendly interfaces that interact seamlessly with the distributed data and system architecture beneath the surface.

Storing connections, posts, and other data in a social network’s relational database is an efficient and structured way to manage complex interactions. This ensures data integrity and allows seamless querying and retrieval through tables with well-defined relationships, attributes, and keys. This method enables features such as user profiles, friend connections, post histories, and more. Because these databases are good at managing structured data and maintaining referential integrity, they make a strong foundation for social networks’ dynamic and interconnected nature. To demonstrate how this works in practice, let's explore the history and development of AmicaBet - a social networking platform that fosters meaningful connections and interactions while using a relational database.

II. Background and Related work

The idea of social networks has existed for a long time, even before the digital age. Traditional social networks have always consisted of family, friends, and acquaintances and can be traced back to the beginning of human civilization. However, the way we connect with others has drastically changed with the introduction of online platforms. Websites such as Friendster, MySpace, and LinkedIn that emerged in the early 2000s led to the creation of popular social media platforms like Facebook, Twitter, Instagram, and TikTok. These platforms enable users to generate and share content, engage with others, and establish new connections, ultimately transforming how people interact with each other and the world.

The development of social networks has significantly influenced how people connect, communicate, and share information. From their historical origins to the complexities of modern digital platforms, social networks continue to shape human interactions in various ways. Researchers from diverse fields actively study these networks to understand their societal implications better, inform policy decisions, and tackle emerging challenges. As the digital environment continues to evolve, ongoing research is necessary to navigate the opportunities and challenges social networks present in the digital age.

III. Procedures of building an entity relationship diagram for social Networks

An Entity-Relationship (ER) diagram is a visual representation used in database design to illustrate the structure of a database system and the connections between its different components. It is a conceptual tool that models the objects (entities), properties (attributes), and associations (relationships) within a particular domain. ER diagrams consist of various essential elements:

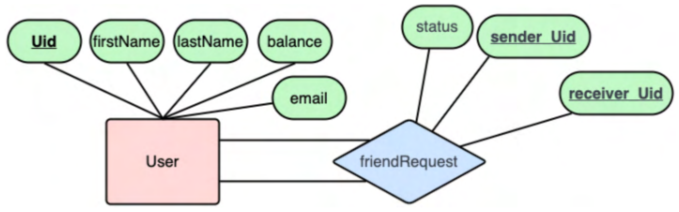
- **Entities**: Entities are real-world objects or concepts in the database. For example, in a social network database, entities could include "User," "Post," "Comment," etc.
- **Attributes**: Attributes are properties or characteristics that describe entities. Each entity has a set of attributes that provide details about it. For a "User" entity, attributes might include "Username," "Email," and "Date of Birth."
- **Relationships**: Relationships define how entities are connected. They represent associations and interactions between entities. Examples of relationships in a social network context are "User follows User," "User creates Post," and "Comment belongs to Post."
- **Keys**: Keys are attributes that uniquely identify an entity within a given context. For instance, a "User ID" might serve as a primary key to uniquely identify users in a social network.

In ER diagrams, graphical symbols and notations represent different elements. Entities are represented by rectangles, attributes by ovals, and relationships by diamonds. These symbols are connected by lines and arrows to show the connections between entities and relationships.

Creating ER diagrams is an essential step in database design as it helps stakeholders and designers better understand the system's structure and behavior even before its implementation. ER diagrams serve as a common visual language that facilitates the communication of database concepts and assists in refining the schema to suit the application's needs. This crucial process lays the foundation for developing a well-organized and efficient relational database structure.

A. *Storing Users and friend requests*

Managing user information and friendship requests in social networking platforms can be made easier by creating two tables in the database schema. The first table should store essential user information, such as User ID, Username, Email, and Date of Birth, while the second table should serve as a relationship table to establish user connections. It is important to incorporate mechanisms that account for the dynamic nature of these relationships. A flexible system where friendship requests can be categorized as accepted, rejected, or blocked, giving users more control over their social interactions. This structured approach aims to optimize social networking platforms and enhance user engagement.



- **Uid**: serves as a distinctive identifier associated with an individual user. Notably, employing first and last names as identifiers might be unsuitable due to the potential presence of homonyms, coupled with the dynamic nature of users being able to alter their terms.
- **balance**: users can place wagers on bets, thereby necessitating the inclusion of an attribute capable of retaining the user's present balance.
- **firstName** and **lastName**: full name of the user
- **status**: an enum attribute which can be only accepted, rejected, blocked, pending. The default value is pending.
- **sender_Uid** and **receiver_Uid**: reference to Uid of the user who sent the connection request and the Uid of the user who received it. N.B.: The primary key of a relationship is always the combination of the primary keys of the entities

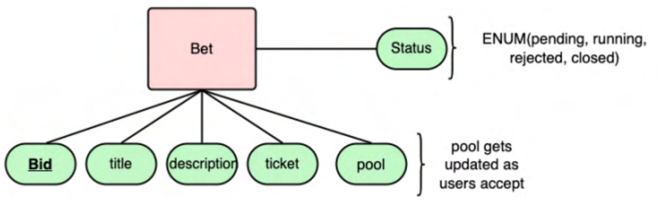
related. In this case the unique identifier of *friendRequest* table is a tuple: *(sender_Uid, receiver_Uid).*

With the provided ER diagram, it is possible to utilize the following queries for storing, retrieving, and updating data. Sample queries (assume the symbol ? is replaced by some data):

- **Add new user**: *INSERT INTO user (email, password, firstName, lastName) VALUES (?, ?, ?, ?)*
- **Add new connection between users**: *INSERT INTO friendRequest (sender_Uid, receiver_Uid, status) values (1, ?, 'pending'),(2, ?, 'pending'),(3, ?, 'pending')*
- **Retrieve a friends information**: *SELECT * FROM friendRequest as f, user as u WHERE status="accepted" AND (f.sender_Uid=? AND f.receiver_Uid=? AND u.Uid=?) OR (f.sender_Uid=? AND f.receiver_Uid=? AND u.Uid=?)*

B. *Storing Bets*

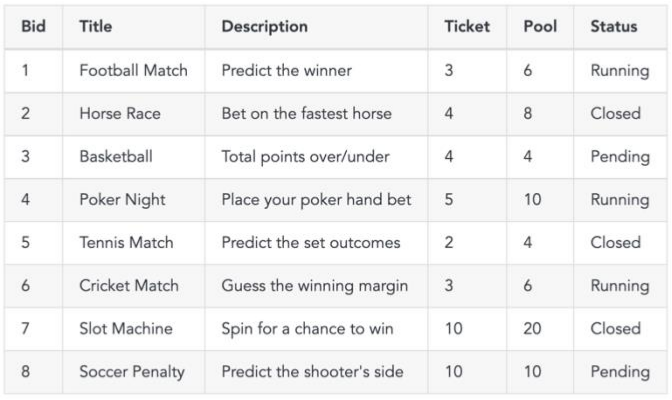
The structure of the bet table is presented as follows:


Bet
Status
ENUM(pending, running, rejected, closed)
Bid
title
description
ticket
pool
pool gets updated as users accept

The discussed table has a unique identifier called *Bid*, just like the user table. Additionally, each bet has a status reference that allows the system to recognize changes in the bet's status, such as closures or rejections, based on user actions.

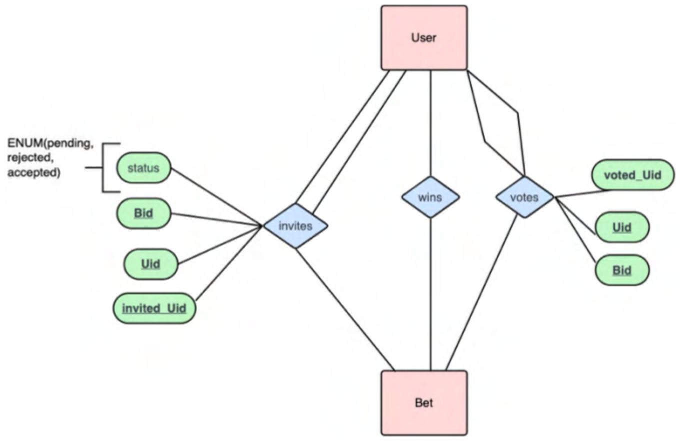
C. *Creating bets and inviting friends*

The connection between users and bets is made possible through the use of *invite*, *win*, and *vote* tables. This schema is designed to maintain database normalization while also allowing for the collection of various data sets. Its scalability allows for multiple invitations and votes for bets involving many users. Note that the *wins* table refers to the primary keys of both the user and bet as (*Uid*, *Bid*) to ensure a single winner in each bet.



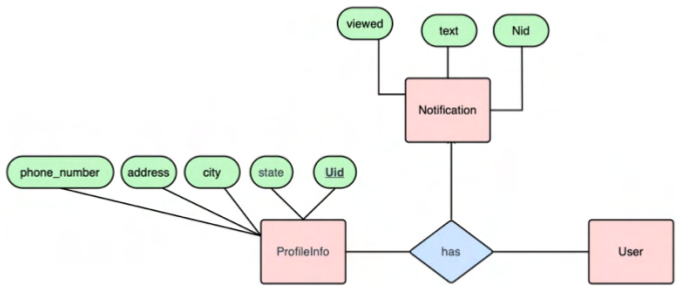
Bid	Title	Description	Ticket	Pool	Status
1	Football Match	Predict the winner	3	6	Running
2	Horse Race	Bet on the fastest horse	4	8	Closed
3	Basketball	Total points over/under	4	4	Pending
4	Poker Night	Place your poker hand bet	5	10	Running
5	Tennis Match	Predict the set outcomes	2	4	Closed
6	Cricket Match	Guess the winning margin	3	6	Running
7	Slot Machine	Spin for a chance to win	10	20	Closed
8	Soccer Penalty	Predict the shooter's side	10	10	Pending

**Sample Bet table visualization with data*



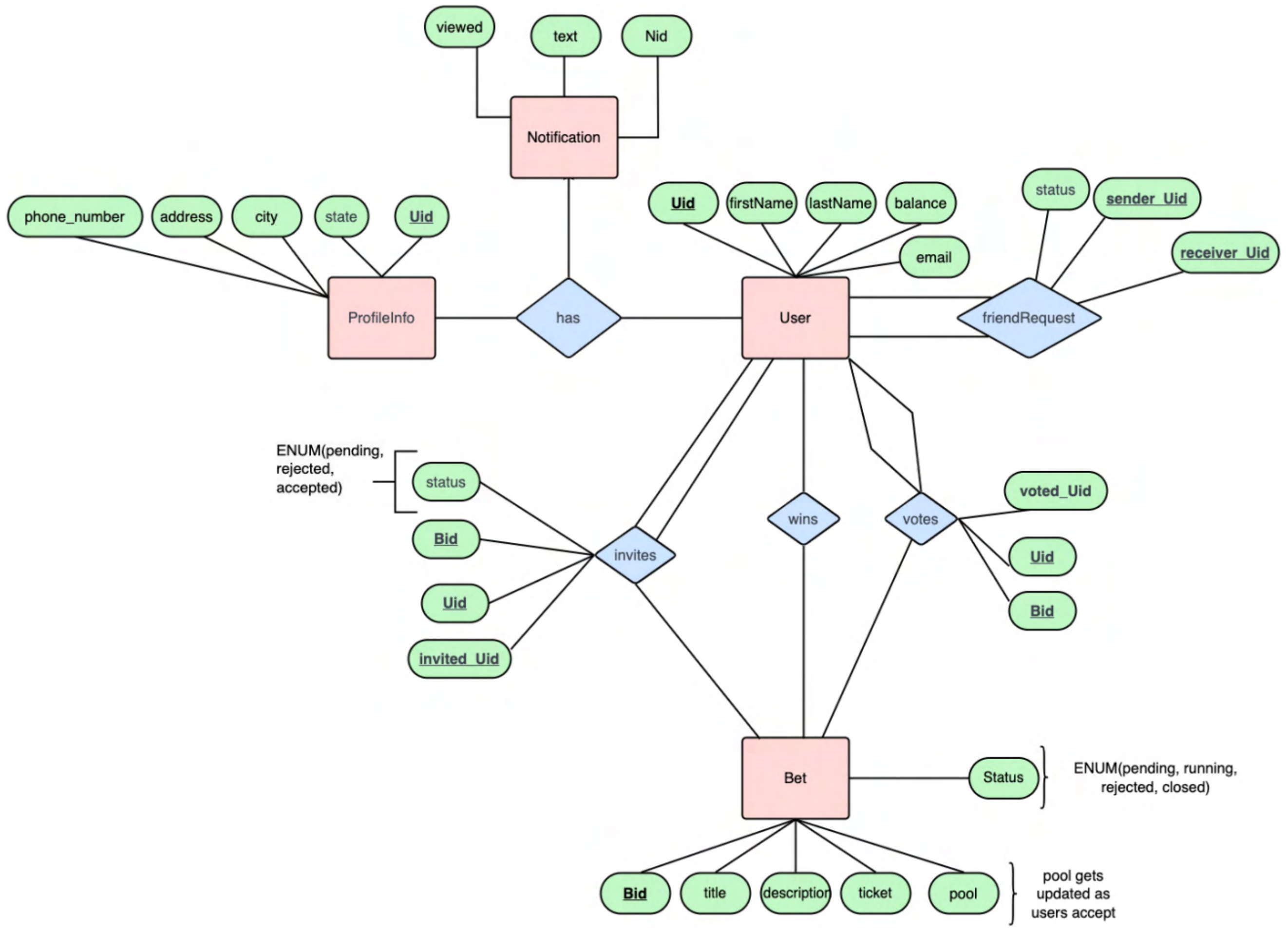
D. *Notifications of activities and other informations*

Constructing a distinct table where these details can be stored for supplementary data and information is standard. This approach involves creating a relationship that utilizes the *User*'s unique identifier to associate the information with the corresponding account correctly. In a similar vein, notifications can be linked through the Uid, and the inclusion of a Boolean variable termed "viewed" guarantees that notifications are presented only once. This strategy also ensures the retention of all notifications, even after their being viewed.



E. *Complete ER diagram*

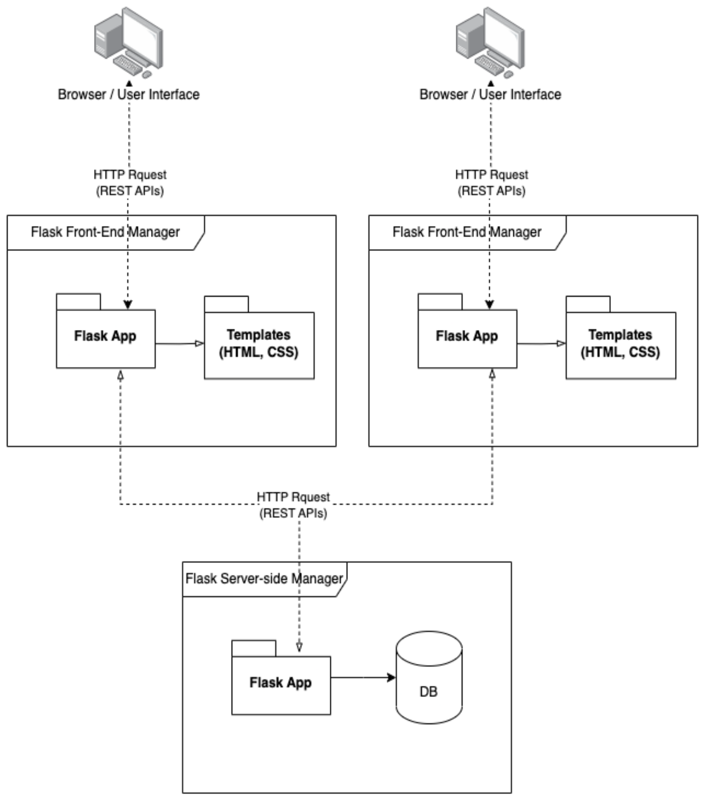
Creating an Entity-Relationship (ER) diagram involves several steps: Identity entities, define their attributes, and establish relationships. Drawing the diagram is necessary using standard shapes, specifying how entities relate, assigning primary keys for identification, refining the diagram, and validating it. ER diagrams visually represent a system's structure and connections, aiding in designing databases effectively. The complete Entity Relationship diagram is as following:



IV. Distributed Architecture

Distributed architecture is a design method in which a system's components or processes are spread across multiple computers or nodes. These computers communicate and cooperate to achieve a common goal. This approach enhances scalability, fault tolerance, and performance by distributing the workload and data across the network. Consequently, this enables better resource utilization and responsiveness.

In AmicaBet's architecture, the system is designed as distributed with two distinct services: server-side and front-end manager. The server-side rendering service manages the database, including data reads, writes, structuring, and error handling. This component operates behind the scenes and is not directly accessible to users. One of its primary functions is to process incoming API requests from the front-end manager. These requests seek specific data to be retrieved, restructured, and prepared for user interface handling. This division of labor allows AmicaBet to leverage the advantages of distributed systems, enhancing scalability and performance while ensuring efficient communication between server-side and front-end components. The front-end manager can be replicated with *Kubernetes* and *Docker* to produce many instances that communicate with the same *server-side manager* to improve user experience responsiveness further as shown in the representation on the right.



V. Security

Flask has a built-in session object for storing data specific to a user's session on the server. This feature helps maintain a user's state across different HTTP requests without relying solely on cookies or URL parameters. The session object is based on signed cookies, which prevent tampering with the data stored in the session. Using the session object in Flask is a helpful way to maintain user states in web applications without manually passing information between views or routes. It simplifies managing user-specific data and makes it convenient for users to keep context across multiple requests, enhancing their overall experience.

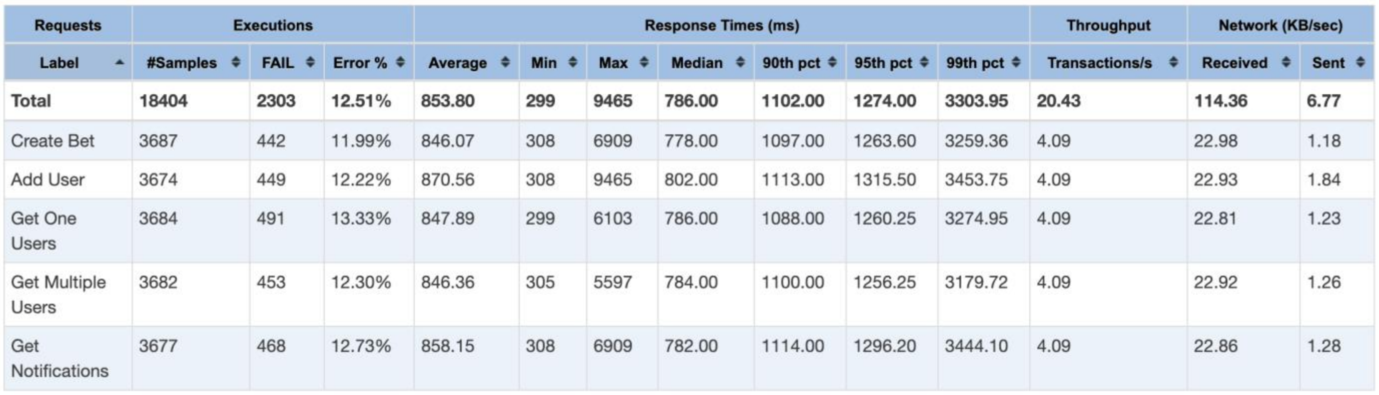
Moreover, a *wrapper function* is implemented in conjunction with each user's interaction with the front-end manager. This function, classified as an operation that activates upon receiving any incoming HTTP request, guarantees that the user is authorized to undertake specific actions. This is achieved by utilizing a security system founded on token-based authentication. The last feature that ensures another base of security is input validation. Passwords are hashed and must follow constraint such as:

- length must be 8 characters
- must have a lower case
- must have upper case
- must have a number

The user is notified via notification systems if the constraints are not fulfilled.

VI. Deployment end performance testing

As of August 2023, AmicaBet is deployed on amicabet.pythonanywhere.com as free service deployment is provided by the PaaS. However, a successful deployment on Google Cloud Platform has been attempted using GKE, Kubernetes Engine service provided by google. Reference this git repository *deploy.sh* file for complete description and commands of deployment. While the web application was deployed a performance engineering test was run on some of the APIs. The objective was to validate the functionality and performance of AmicaBet platform, by creating a test scenario to evaluate API endpoints, authentication mechanisms, data retrieval, and overall system responsiveness. the simulation involved a load of 20 threads (per API) to mimic user interactions with the application. The test was designed with a ramp-up period of 240 seconds, gradually increasing the user load to reach the entire 20 threads. The total duration of the trial was set to 900 seconds (15 min), allowing sufficient time to observe the application's behavior under sustained load. By carefully configuring the test plan, I could evaluate the system's performance as **adequate** for 20 concurrent users. For a total of 18404 API requests the error percentage was 12.51, and the 90th percentile settled at 1102 milliseconds. The tabulated data below furnishes a comprehensive report detailing the outcomes derived from the conducted test [5].



Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**18404**	**2303**	**12.51%**	**853.80**	**299**	**9465**	**786.00**	**1102.00**	**1274.00**	**3303.95**	**20.43**	**114.36**	**6.77**
Create Bet	3687	442	11.99%	846.07	308	6909	778.00	1097.00	1263.60	3259.36	4.09	22.98	1.18
Add User	3674	449	12.22%	870.56	308	9465	802.00	1113.00	1315.50	3453.75	4.09	22.93	1.84
Get One Users	3684	491	13.33%	847.89	299	6103	786.00	1088.00	1260.25	3274.95	4.09	22.81	1.23
Get Multiple Users	3682	453	12.30%	846.36	305	5597	784.00	1100.00	1256.25	3179.72	4.09	22.92	1.26
Get Notifications	3677	468	12.73%	858.15	308	6909	782.00	1114.00	1296.20	3444.10	4.09	22.86	1.28

VII. Conclusion

The report has delved into creating and developing AmicaBet, a unique betting-oriented social network web application built using Flask. The report emphasized the importance of understanding data structure, semantics, data relationships, and APIs for data retrieval in online social networks. It showcased the complexities of managing user-generated content, profiles, connections, and interactions within a social network and how Flask, distributed architecture, and Google Cloud Kubernetes Engine were employed to address these challenges. Using Flask and Bootstrap facilitated the creation of user-friendly interfaces that interact seamlessly with the underlying distributed architecture.

The paper explored various aspects of building the social network, including designing the Entity-Relationship (ER) diagram, storing user and bet information, managing friend connections, creating bets, handling notifications, and maintaining security through token-based authentication and input validation. It outlined the distributed architecture employed in AmicaBet, enabling scalability, fault tolerance, and performance enhancement through efficient communication between server-side and front-end components. The deployment of AmicaBet on the Google Cloud Platform and performance testing results demonstrate the system's capability to handle concurrent users effectively, highlighting response times under sustained load and showcasing the platform's responsiveness and robustness.

In conclusion, the creation of AmicaBet showcases the intricate relationship between various computer science disciplines. The platform's user-friendly interface, distributed architecture, security measures, and performance testing results illustrate the great implementation of the concepts discussed throughout this report.

Acknowledgment

I want to express my deepest gratitude to Gary Weiss, Ph.D., Graduate Director at Fordham University, for the guidance and feedback throughout the development process of AmicaBet.

References

[1] Duggan, M., Ellison, N. B., Lampe, C., Lenhart, A., & Madden, M. (2015). Social media update 2014. *Pew research center*, *19*, 1-2.

[2] Pongpradit P., Prompoon N., "Constructing initial design patterns for online social network-based applications," 2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS), Okayama, Japan, 2016, pp. 1-7, doi: 10.1109/ICIS.2016.7550802.

[3] Bootstrap. 2023. Bootstrap Version 5. [Online]. Available: https://getbootstrap.com/

[4] Song I., Evans M., Park EK. A comparative analysis of entity-relationship diagrams. Journal of Computer and Software Engineering. 1995;3(4):427-59.

[5] Molyneaux, I.. The art of application performance testing: from strategy to tools. " O'Reilly Media, Inc.", 2014.