1. INTRODUCTION

1.1 Purpose

The software design document describes the architecture and the design system of the Gam3ya system. This document will allow the software developers to fully understand what software components will be built and how such components relate to one another This document will provide the details of all parts of the software that, when integrated, constitute the Gam3ya software system.

1.2 Scope

Gam3ya is a for-profit project that aims to allow online peer-to-peer lending that implements the concept of Gam3ya securely. The Gam3ya, in the Egyptian culture, is some sort of a money sharing system, where each user pays a predetermined amount of money each month. For that month, only one user takes the whole amount of money collected. This process is repeated for many months where each month a different user receives the collected Money. “Gam3ya” project uses a model provides alternatives to the traditional Gam3ya format where money might not be distributed starting from the first month and more than one person might receive their payments on the same month; this is explored in detail in the SRS document. This document will provide an overview for the architecture and component designs of the system as well as the data description and dictionary.

1.3 Overview

This document consists of 4 parts. This first part is concerned with the overall architecture of the system , the environment in which the system is placed and the interaction of the system with external systems. The second part of the document discusses how the architecture of our system is built in detail, including the subsystems and all internal components of the system, their functionality and how these components interact with each other, including the interfaces between those components. The following part discusses the structural design in detail. That is the classes that are used in this software, their related attributes, methods and the relationships between those classes. Also, controller classes are discussed in detail. These controller classes are responsible for data maintaining relationships between certain classes. In the last part, the screenshots displaying the user interface are shown. A brief description is

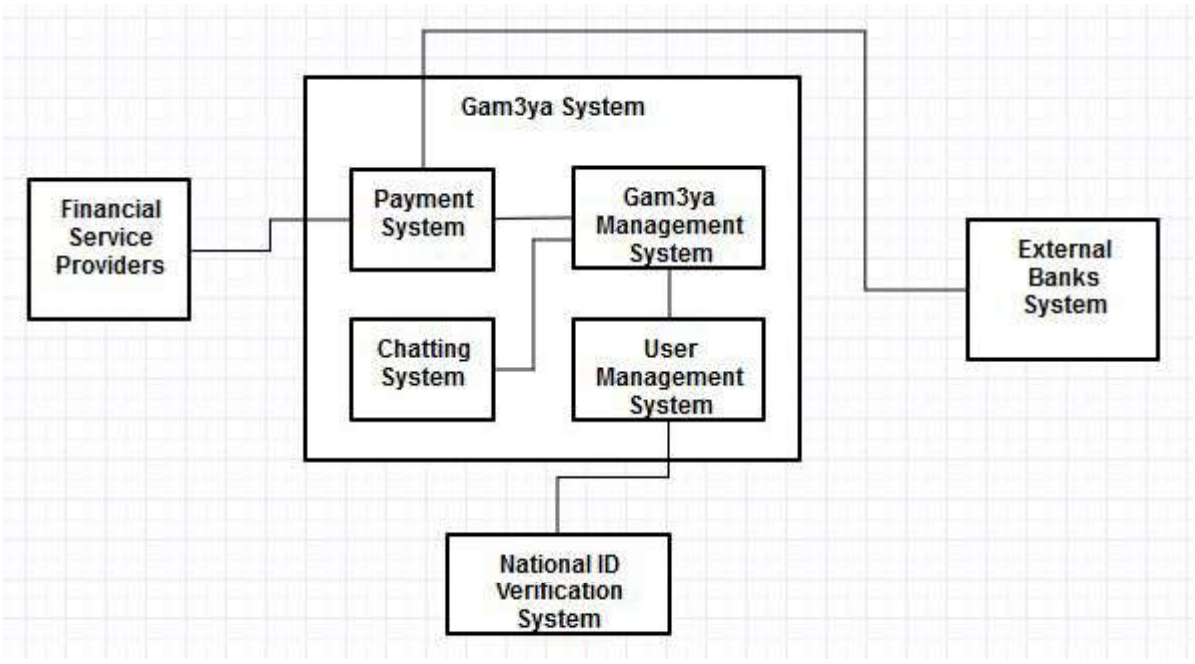
provided for how each feature of the user interface can be used and its corresponding functionality.

1.4 Definitions and Acronyms

This section contains the definitions of the technical terms used in this document

Term	Definition
TLS	Transport Layer Security protocol
API	Application programming interface
ASP.NET	A web development framework developed by Microsoft
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
Javascript	High-level programming language used to allow interactive web pages
C#	A general purpose programming language; within the scope of this document, it is used within Microsoft's ASP.NET framework for the server-side functionality

2. SYSTEM OVERVIEW

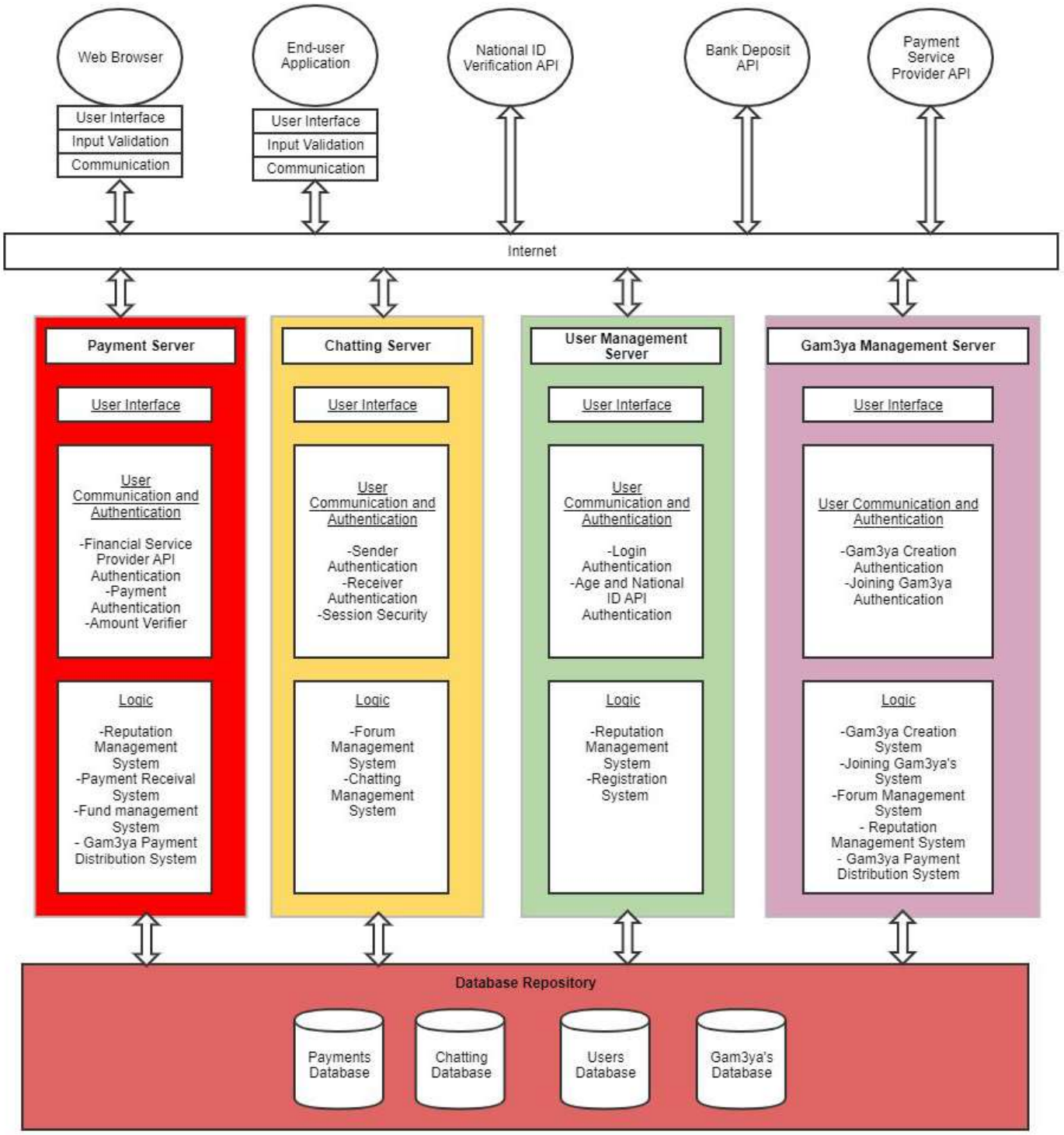


Gam3ya mainly interacts with 3 external systems. In order to facilitate payment services for customers, Gam3ya will be interacting with external payment service providers such as Fawry, Vodafone Cash and Coinbase. Any interaction will be done via the API's provided by the service providers. Moreover, to deposit the money received from the service providers into our bank account we will have to use a communication medium with the bank; this medium would be the Bank's API which allows the automation of depositing the money directly into the bank account. Finally, it was mentioned in the SRS document that only users with a valid national ID will be able to create an account. In order to verify the national IDs, we will use an online verification service provided by the government which takes the name and the ID and verifies if they match or not. Similar to the two other external systems, the interaction will take place through their API.

Internally, the Gam3ya management system has to interact with the Chatting system in order to be able to create a public forum once a Gam3ya is created. It has to be connected to the payment system as well to keep track of which users paid. Furthermore, it has to be connected to the user management system to monitor which users joined which Gam3ya.

3. SYSTEM ARCHITECTURE

3.1 Architectural Design



In this system, a combination of various architecture models is used to better display the proposed design. Essentially, a client-server architecture is used where both clients and servers are layered in such a way to show the decomposition of each system into smaller subsystems that could be implemented. The clients communicate with our system over the internet. We use a fat-client model where clients provide presentation, a very basic level of functionality (such as input validation) and a communication protocol to connect to the servers. The development of the web-application would be implemented using Microsoft ASP.NET; in such framework, the frontend is implemented using HTML, CSS and Javascript whereas the backend is implemented using CSharp (C#).

Since we have multiple clients who are to be handled differently, we split the types of requests and designed 3 different servers to handle the different requests of the system. Each server is layered into four layers: interface, communication and authentication, and the logic layer. The interface and communication layers are collectively responsible for facilitating the interaction between the clients and the servers while enforcing security; this would be done using the Transport Layers Security (TLS) cryptographic protocol to secure the communication between the clients and the servers. The final layer in the servers, the logic layer, consists of subsystems that simply handle the business logic associated with each server. Each server is responsible for handling different business operations and therefore, the load on each server is not distributed evenly; thus, the number and capacities of each server is going to be different.

The final component of the architecture design is a repository of databases with which all the servers communicate using requests to access, modify and delete data elements. The database repository would consist of multiple databases to keep track of the different types of data associated with the system. We would have 4 different databases to record data regarding payments, users, Gam3ya's and chatting. We will be using MySQL relational database management system (RDBMS). MySQL is open-source, easy to use, and it is compatible with Microsoft ASP.NET.

3.2 Decomposition Description

The end-users will interact with our system, 2 potential platforms are supported: web browser and application. As mentioned earlier, both of these will follow the fat-client model. Essentially, these will have a user interface, a basic logic layer where they handle input

validation and a communication layer through which communication will take place between them and our system.

Besides, our architecture consists of 4 main subsystem. Each of these is implemented as a layered server which consists of 3 layers: User Interface, User Communication and Authentication, and Logic Layer.

3.2.1. Payment Subsystem:

The payment subsystem is responsible for managing fund transfer between the users, the financial service providers and our bank account.

Payment Subsystem	
Financial Service Provider API Authentication	This authenticates the identity of the financial service providers APIs.
Payment Authentication	This authenticates the payment ticket received from any of the financial service providers.
Amount Verifier	It verifies the amount paid by the user and compares it to the value recorded in the payment ticket.
Reputation Management System	This is a common component between Payment and User Management Servers. It manages the users' reputations and updates them accordingly. Whenever a user finishes a Gam3ya their reputation improves and whenever they do not pay their monthly payments on time their reputation gets reduced.
Payment Receival System	This component is responsible for receiving the money from the Financial Service Providers. This will be represented by a digital ticket that the Financial Service Providers will send to us once a payment has been made.
Fund management System	This manages the flow of funds from the point the Financial Service Providers send us the ticket to the point we deposit this money in our bank account.
Gam3ya Payment Distribution System	This component is used by the Gam3ya Management Server as well. It is responsible for choosing which user will receive the

	payment, the payment amount and in which month.

3.2.2. Chatting Subsystem:

The chatting subsystem manages the flow of messages in forums as well as securing the medium through which the messages are sent.

Chatting Subsystem	
Sender Authentication	It is responsible for authenticating the identity of the user that is sending the message.
Receiver Authentication	Similar to Sender Authentication, this component is authenticates the identity of the receiver of the message.
Session Security	This component secures the session between the sender and the receiver through using end-to-end encryption.
Forum Management System	Forum Management System handles the creation, deletion and joining requests for both Gam3ya Forums and Private Forums.
Chatting Management System	Chatting Management System handles the messages sent by different users and broadcasts them accordingly.

3.2.3. User Management Subsystem:

The User Management Subsystem manages the creation of user accounts with all required authentications. It also handles login and logout requests.

User Management Subsystem	
Login Authentication	This component uses the username and password that the user inputs and validates them in the database to either grant or deny access to the user.
Age and National ID API Authentication	This is responsible for authenticating the National ID Verification API before sending any sensitive information through it.

Registration System	This handles the account creation process by firstly using the Authentication System and then creates an account and saves it in the database in the case of an authenticated ID.
Reputation Management System	This is the same system used in the Payment Subsystem

3.2.4. Gam3ya Management Subsystem

The Gam3ya Management Subsystem manages the creation and join request of Gam3ya's with all required authentications. It also manages the forums creation associated with the creation of Gam3ya's. In addition, it is responsible for the modification of user's reputation points based on their payments to the Gam3ya and their commitment (continuation of the Gam3ya till the end).

Gam3ya Management Subsystem	
Gam3ya Creation Authentication	This component is responsible for the authentication for the Gam3ya creation and ensures that the request is transmitted successfully.
Joining Gam3ya Authentication	This component is responsible for the authentication and transmission of a Gam3ya joining request to the Gam3ya members a vote could take place.
Gam3ya Creation System	This system is responsible for handling the Gam3ya creation requests. When a user sends a request to create a Gam3ya, their reputation points are checked and according to the predefined criteria, their request is either approved or rejected.
Joining Gam3ya System	This system is responsible for handling the Gam3ya joining requests. When a user sends a request to join a Gam3ya, their reputation points are checked and according to the vote of the majority of the members of the Gam3ya, the user's request to join is either approved or rejected.
Forum Management	Forum Management System is used here to manage the creation

System	and deletion for Gam3ya Forums.
Reputation Management System	This is the same component that's mentioned in earlier subsystems
Gam3ya Payment Distribution System	This component is used by the Payment Management Server as well. It is responsible for choosing which user will receive the payment, the payment amount and in which month.

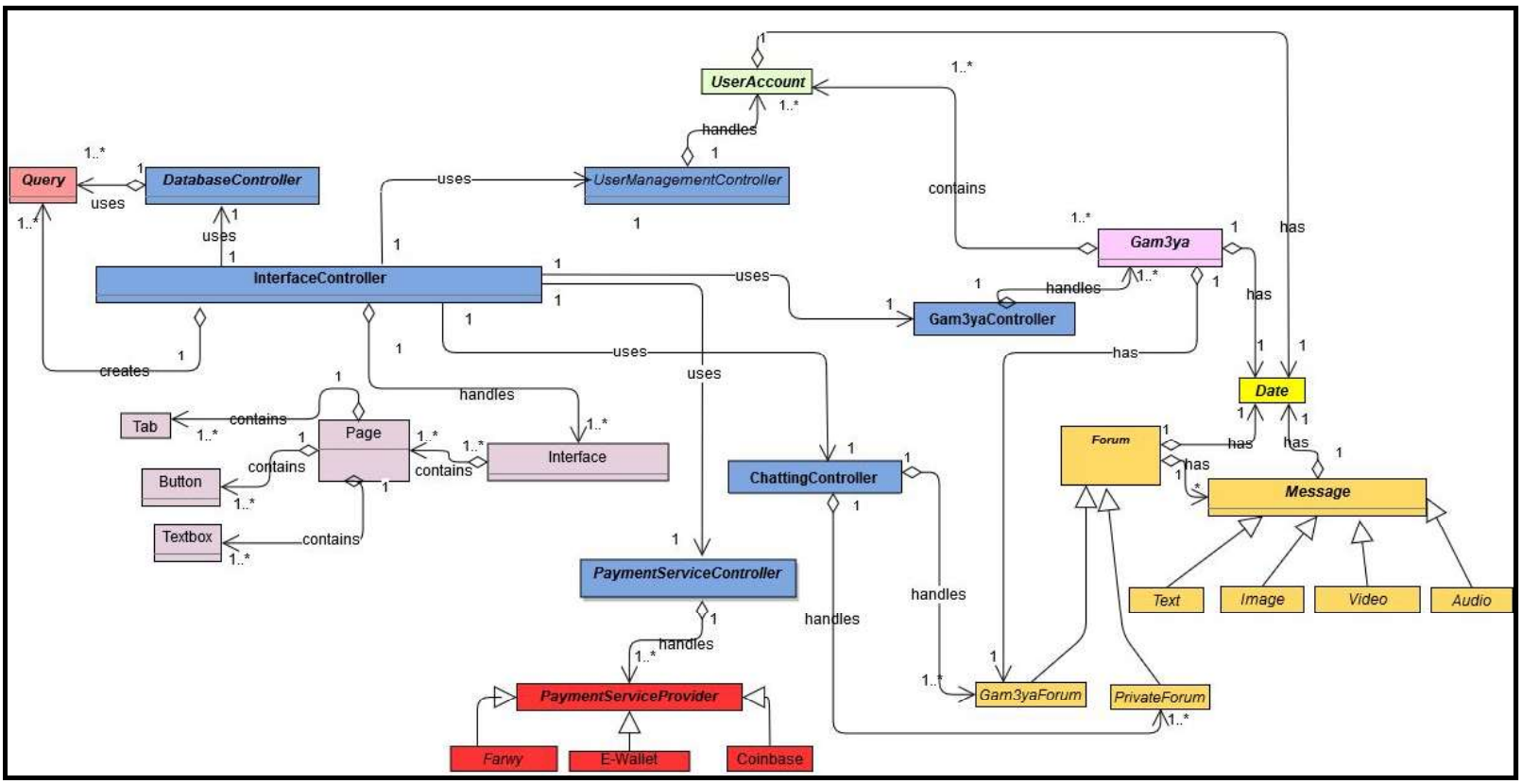
3.3 Design Rationale

We used a client server architecture which allows for servers to be distributed around the network. Additionally, we have a set of shared databases to be accessed from different places and by different clients. A fat-client model was implemented in order to reduce some of the load of the servers and thus, improve performance; this is also because different clients would interact with different servers based on their request. Furthermore, a layered structure was implemented to both the servers and the clients to divide the functionality into layers to accomplish different goals such as authentication, communication and logic. Finally, a repository of databases was shared between all clients to allow for the access, delete, modification of data entries in different databases per client requests.

4. DATA DESIGN

4.1 Data Description

A very basic Class Diagram can be found below. For the Full Diagram, Please look at the attached classDiagram.png file



4.2 Data Dictionary

4.2.1 Controller Classes

UserManagementController		
Attributes	- usersList: UserAccount[]	An array of all users.
Methods	+ UserManagementController()	A constructor that creates and initializes the class
	+ getUsersList(): UserAccount[]	A getter that returns the array of all UserAccounts
	+ authenticateNationalID(string): bool	A function that takes the national ID and calls the NationalIDVerification API to authenticate it
	+ authenticateAge(string): bool	An function that takes the national ID and authenticates that the user is above 21 years old.
	+ updateUserReputation(UserAccount, int)	A function that takes a UserAccount and an amount and adds that amount to the passed user's reputation points. The amount can be positive or negative
	+ createAccount(string, string): bool	A function that takes a username and a NationalID and creates an account for the user. It firstly authenticates the national ID and the user age. It returns true if the operation was successful and false otherwise.
	+ logout(UserAccount): void	A void function that logs a user out of the platform

	+ requestTutorialView(UserAccount): void	A function that sends a request to view the tutorials. This communicates with the Interface controller to change the viewed interface and replace it by the tutorials interface
	+ getUser(AccountID: string): UserAccount	A function that returns the user that is currently logged in.

DatabaseController		
Attributes	+ myQueries: Query [ ]	An array of all the queries that have been used; this is simply a log of all the modifications to the databases
	+ connectionStrings: string[4]	This string array carries the connection strings to the 4 databases in our system
Methods	+ queryChattingDB (myQuery: Query, returnMessage: &string): bool	Query the chatting Database using the chatting DB connection string. Any return message is stored in the passed string and if the operation was successful, return true
	+ queryUsersDB (myQuery: Query, returnMessage: &string): bool	Query the users Database using the User DB connection string. Any return message is stored in the passed string and if the operation was successful, return true
	+ queryPaymentDB (myQuery: Query, returnMessage: &string): bool	Query the payments Database using the Payment DB connection string. Any return message is stored in the passed string and if the operation was successful, return true

	+ queryGam3yasDB (myQuery: Query, returnMessage: &string): bool	Query the Gam3ya's Database using the Gam3ya's DB connection string. Any return message is stored in the passed string and if the operation was successful, return true

ChattingController		
Attributes	- myGam3yaForums: Gam3yaForum [ ]	An array of all Gam3yaForums
	- myPrivateForums: PrivateForum [ ]	An array of all PrivateForums
Methods	+ ChattingController()	A constructor that creates an initializes the chatting controller
	+ getGam3yaForum(myForumID: string): Gam3yaForum	A function that takes a forum ID and returns its corresponding Gam3yaForum
	+ getPrivateForum(myForumID: string): PrivateForum	A function that takes a forum ID and returns its corresponding PrivateForum
	+ getAllGam3yaForums(): Gam3yaForum [ ]	A function that returns an array of all Gam3yaForum objects
	+ getAlPrivateForums(): PrivateForum [ ]	A function that returns an array of all PrivateForum objects
	+ AuthenticateSender(myMessage: Message, UserAccountID: string): bool	A function that takes a Message and an AccountID and authenticates if that user sent the passed Message.
	+ AuthenticateReceiver(myMessage: Message, UserAccountID: string):bool	A function that takes a Message and an AccountID and authenticates if that user received the passed Message.

	+ ForwardPrivateMessage(myMessage: Message, myPrivateForumForum: PrivateForum): void	A function that takes a Message and A PrivateForum and then sends the passed Message to all UserAccounts associated with the passed PrivateForum
	+ ForwardGam3yaMessage(myMessage: Message, myGam3yaForum: Gam3yaForum): void	A function that takes a Message and A Gam3yaForum and then sends the passed Message to all UserAccounts associated with the passed Gam3yaForum
	+ CreatePrivateForum(UserAccountID: string): bool	A function that takes an AccountID and then creates a PrivateForum where the creator is the passed AccountID. It returns true if the PrivateForum was created successfully
	+ JoinPrivateForum(myPrivateForumID: string): bool	A function that takes a user ID and a PrivateForum ID and creates a join request. It returns true if the request was approved

Gam3yaController		
Attributes	+ TotalNumberofGam3yas: int	An integer corresponding to the total number of Gam3yas
	+ TotalNumberofForums: int	An integer corresponding to the total number of Gam3yaForums
	+ Gam3yas [ ]: Gam3ya	An array of all Gam3yas
Methods	+ Gam3yaController ()	A constructor that creates an instance of the Gam3ya Controller Class

	+ getGam3ya(Gam3yaID: string): Gam3ya	A function that receives a Gam3yaID and returns its corresponding Gam3ya
	+ getAllGam3yas(): Gam3ya [ ]	A function that returns the array of all Gam3yas
	+ AuthenticateCreateGam3ya(myGam3ya: Gam3ya, myUserAccount: UserAccount): bool	A function that authenticates the creation of a new Gam3ya. It takes a Gam3ya and a UserAccount and returns true if the user is allowed to create a Gam3ya.
	+ AuthenticateJoinGam3ya(myGam3ya: Gam3ya, myUserAccount: UserAccount): bool	A function that takes a UserAccount and a Gam3ya and initiates a join request. It returns true if a majority approve the request.
	+ createGam3ya (UserAccount, int moneyAmount, int paymentInterval, int receivalWaitTime, int minRepPoints, string startDate): bool;	A function that calls the AuthenticateCreateGam3ya and if it returns true, this function creates both the Gam3ya and its Gam3yaForum and returns true
	+ JoinGam3ya(myGam3ya: Gam3ya, myUserAccount: UserAccount):bool	A function that calls the AuthenticateJoinGam3ya function and if it returns true, it allows the user to join the Gam3ya as well as the Gam3yaForum
	+ DeleteGam3ya(myGam3ya: Gam3ya):void	A function that deletes a Gam3ya from the Gam3yas array
	+ setReceivalOrder(myGam3ya: Gam3ya): void	A function that takes a Gam3ya and sets the money receival date for each member randomly

	+ DistributePayment( ): void	A function that goes through each Gam3ya in the Gam3yas array and distributes the money to the members who are supposed to receive their payment that month.

PaymentServiceController		
Methods	+ PaymentServiceController()	A constructor that creates an instance of the PaymentServiceController class
	+ manageFunds( ): void	A function that takes the money from the Payment Providers and calls the Bank API to deposit this money in our bank account
	+ payMonthlyPayment(UserAccount) void	This function is called whenever the user who is to receive the money that month requests to get it from one of the Financial Service Providers. It basically records that they have withdrawn the money
	+ calculateLatePenalty(UserAccount, Gam3ya): int	A function that takes a UserAccount and a Gam3ya and it extracts how many days the user was late on their monthly payments and returns the appropriate penalty
	+ calculateReputationBonus(UserAccount, Gam3ya): int	A function that takes a UserAccount and a Gam3ya and it returns the number of bonus points the user should receive by the end of the Gam3ya

InterfaceController		
Attributes	- userController: UserManagementController	An instance of the UserManagementController class
	- chattingController: ChattingController	An instance of the ChattingController class
	- gam3yaController: Gam3yaController	An instance of the Gam3yaController class
	- databaseController: DatabaseController	An instance of the DatabaseController class
	- allInterfaces: Interface[]	An array of all the interfaces the platform uses.
Methods	+ InterfaceController()	A constructor that creates the InterfaceController class
	+ register(userAccount): void	A function that calls the registration-related functions from the UserManagementController class
	+ ExecuteGam3yaDBQuery(Query): bool	A function that takes a Query and executes it on the Gam3yaDB
	+ ExecuteChattingDBQuery(Query): bool	A function that takes a Query and executes it on the ChattingDB
	+ ExecuteUsersDBQuery(Query): bool	A function that takes a Query and executes it on the UsersDB
	+ ExecutePaymentDBQuery(Query): bool	A function that takes a Query and executes it on the PaymentDB

	+ login(username: string, password: string): bool	A function that takes a username and a password and tries to log in. It calls createQuery and then calls ExecuteUsersDBQuery function by passing the newly created Query to it. If authentication succeeds, it returns true.
	+ viewLoginPage( ): void	A function that redirects to the Login Interface
	+ viewLogoutPage( ): void	A function that redirects to the Logout Interface
	+ logout(UserAccount): void	A function that calls the logout-related function from the UserManagementController
	+ viewHomepage( ): void	A function that redirects to the Homepage Interface
	+ viewProfile( ): void	A function that redirects to the ViewProfile Interface
	+ createGam3ya (UserAccount, int moneyAmount, int paymentInterval, int receivalWaitTime, int minRepPoints, string startDate): bool;	A function that calls the Gam3ya-Creation-related functions from the Gam3yaController class
	+ displayGam3yas(UserAccount): void	A function that displays the Gam3yas that the passed UserAccount is allowed to join
	+ userCreatePrivateForum( ): void	A function that calls the PrivateForum-Creation-related functions from the ChattingController class
	+ displayMessageSent(Message,UserAccount): void	A function that views the Interface related to a sent message

	+ redirectToLiveChat(UserAccount): void	A function that redirects a user to the Facebook Page where they can initiate a Live Chat with the customer service
	+ redirectToTutorials(UserAccount): void	A function that redirects the user to the Tutorials Interface
	+ getUser(AccountID: string): UserAccount	A function that retrieves a User Account through its ID
	+ createQuery(string[]): Query[]	A function that takes an array of string, parses it and creates an array of Queries for all the required Databases.
	+ createPrivateForum(UserAccount): bool	A function that calls the Private-Forum-creation-related function from the ChattingController class. It returns true if the creation process was successful.

4.2.2 The Rest Of The Classes

UserAccount		
Attributes	- Username: string	A string that represents the username that the user uses to log in.
	- AccountID: string	A unique string identifier for each user account
	- Age: int	An integer to represent the day
	- NationalID: string	A string which represents the nationalID of the user

	- ReputationPoints: int	An integer that represents the reputation points of the user
	- JoinDate: Date	The date at which the user joined the platform.
	- NumberofGam3yasCurrentlyJoined: int	The number of Gam3yas this user is currently enrolled in.
	- TotalNumberofGam3yasJoined: int	The overall number of Gam3yas the user has joined in the past as well as in the present.
	- Gam3yasID: string[]	An array of all IDs of the Gam3yas the user is currently enrolled in.
	- NumberofForumsCreated: int	The number of forums this user has created throughout their history
	- NumberofForumsCurrentlyJoined: int	The number of active forums this user has joined.
	- JoinedForumsID: string[]	An array of all IDs of forums this used has joined.
	- CreatedForumsID: string[]	An array of all IDs of forums this used has created.
	- NumberofForumsJoined: int	The total number of Forums this user has joined that are either active or inactive.
	- MessagesSentToSupport : string[]	An array of all the messages the user has sent to the customer support.
Methods:	+ UserAccount ( )	A constructor that creates and initializes a user account. Initially, a user is created with a reputation of 500 points.

	+ sendErrorMsg( ): string	A function that reports an error message sends it to customer support
	+ storeMessageToSupport(string): void	A function that takes a text message and appends it to the array that contains the messages sent to the support.
	+ UpdateRepPoints(int): void	A setter function that updates the reputation points of a user
	+ getRepPoints( ): int	A getter function that returns the reputation points of a user.
	+ getJoinedGam3yasID( ): string[]	A getter function that returns the array of IDs of the Gam3yas the user has joined
	+ getNumberofForumsCreated( ): int	A getter function that returns the number of forums the user has created
	+ getNumberofForumsCurrentlyJoined( ): int	A getter function that returns the number of forums the user is currently enrolled in
	+ getForumsID( ): string[]	A getter function that returns the array of IDs of the Forums the user has joined
	+ getNumberofForumsJoinedBefore( ): int	A getter function that returns the number of forums the user has ever joined
	+ getMessagesSentToSupport( ): string[]	A getter function that returns an array of all the messages sent to the support
	+ setNumberofForumsCreated( ): int	A setter for the variable that contains the number of forums created

	+ setNumberofForumsCurrentlyJoined( ): int	A setter for the variable that contains the number of forums the user is currently enrolled in.
	+ setUserName(string): void	A setter for username
	+ getUserName (): string	A getter for username
	+ setAccountID (string): void	A setter for AccountID
	+ getAccountID (): string	A getter for AccountID
	+ setAge(int): void	A setter for Age
	+ getAge(): int	A getter for Age
	+ setNationalID(string): void	A setter for NationalID
	+ getNationalID(): string	A getter for NationaID

Gam3ya		
Attributes	- Gam3yaID: string	A unique string representing the ID of a Gam3ya
	- myGam3yaForum: Gam3yaForum	The forum that's associated with the Gam3ya
	+MaxNumberofParticipants: int	The maximum number of members that the Gam3ya allows
	+CurrentNumberofParticipants: int	The current number of members in the Gam3ya
	- Participants : UserAccount [ ]	An array of the users in the Gam3ya
	+Gam3yaAmount: double	The amount of the Gam3ya that is to be distributed to each member when it is their time to receive the payment

	+NextPaymentDistribution: Date	The date of the next time money would be distributed to Gam3ya members
	+NextPaymentDeadline: Date	The next deadline for all the members to pay the periodic paymentt
	+MinReputationPoints: int	The minimum reputation points that the any User wanting to join the Gam3ya must have
	+StartDate: Date	The start date of the Gam3ya
	+EndDate: Date	The end date of the Gam3ya
	+RecievalOrder : UserAccount [ ]	The array that represents all the members of the Gam3ya and their receiving order. For example, the first element in the array is the one to receive their payments in the first month, etc…
Methods	+ Gam3ya ()	The constructor that is used to instantiate objects and initialize the variables
	+ setGam3yaID(id: string): void	The setter function for the Gam3yaID variable
	+ getGam3yaID(): string	The getter function for the Gam3yaID variable
	+ setMyGam3yaForum(newForum: myGam3yaForum): void	The setter function for the Gam3yaForum variable
	+ getMyGam3yaForum(): Gam3yaForum	The getter function for the Gam3yaForum variable
	+ setParticipants(myParticipants: UserAccount [ ]): void	The setter function for the members of the Gam3ya (the UserAccount [ ])

	+ getParticipants( ): UserAccount [ ]	The getter function for the members of the Gam3ya (the UserAccount [ ])
	+ viewGam3yaDetails (): void	This function displays a formatted version of all the details of the Gam3ya
	+ createGam3yaForum(): void	A function that creates the Gam3yaForum for the Gam3ya class it is in.

Query		
Attributes	- myQuery: string	This string represents the sql query that would be used to update/remove/add entries to the databases
	- length: int	This integer determines the length (in characters of the query)
Methods	+ Query()	The constructor that is used to instantiate objects and initialize the variables
	+ setQuery(newQuery: string): void	The setter function for the myQuery variable
	+ setLength(newLength: int): void	The setter function for the length variable
	+ getQuery(): string	The getter function for the myQuery variable
	+ getLength(): int	The getter function for the length variable

Message		
Attributes	- senderID: string	A string that keeps track of the ID of the message sender
	- SendingTime: Date	Records the date this message was sent
	+ readBy: string [ ]	This string array keeps track of all the users (using their user ID) who have seen the message
	+ receiversID: string [ ]	This array gives an indication of all the users to whom the message must be sent
Methods	+ Message()	The constructor that is used to instantiate objects and initialize the variables
	+getSenderID( ): string	The getter function for the senderID variable
	+setSenderID(id: string): void	The setter function for the senderID variable
	+getSendingTime( ): Date	The getter function for the sendingTime variable
	+setSendingDate(date: Date): void	The setter function for the sendingTime variable
	+ sent ( ): bool	A function indicating whether a message is sent or not
	+ delivered ( ): bool	A function indicating whether a message is delivered or not

	+ EncryptMessage(): void	A function that encrypts a message
	+ DecryptMessage(): void	A function that decrypts a message

Text		
Attributes	- Length: int	An integer to represent the length of a text message (in characters)
	- messageContent: string	The string containing the actual content of a text message
Methods	+ Text()	The constructor that is used to instantiate objects and initialize the variables
	+ getLength(): int	The getter function for the Length variable
	+ setLength(newLength: int): void	The setter function for the Length variable
	+ getMessageContent(): string	The getter function for the messageContent variable
	+ setMessageContent(content: string): void	The setter function for the messageContent variable
	+ playMessage ( ): void	This function is overridden in every child class inheriting from the Message Class. This function displays a text message

Image		
Attributes	- width: int	The width of an image (in pixels)
	- length: int	The length of an image (in pixels) so that the total size is lengh*width

Methods	+ Image()	The constructor that is used to instantiate objects and initialize the variables
	+ getLength(): int	The getter function for the Length variable
	+ setLength(newLength: int): void	The setter function for the Length variable
	+ getWidth(): int	The getter function for the width variable
	+ setWidth(newWidth: int): void	The setter function for the width variable
	+ playMessage ( ): void	This function is overridden in every child class inheriting from the Message Class. This function displays an image

Video		
Attributes	- videoLength: int	The length of the video (in seconds)
Methods	+ Video()	The constructor that is used to instantiate objects and initialize the variables of the Video class
	+ setLength(newLength: int): void	The setter function for the videoLength variable
	+ getLength( ): int	This getter function for the videoLength variable
	+ played ( ): bool	A function that returns true if the video has been played by the users that received it.
	+ playMessage ( ): void	This function is overridden in every child class inheriting from the

		Message Class. This function plays the video.

Audio		
Attributes	- audioLength: int	The length of the audio (in seconds)
Methods	+ Audio()	The constructor that is used to instantiate objects and initialize the variables
	+ getAudioLength (): int	The getter function for the audioLength variable
	+ setAudioLength(newLength: int): void	The setter function for the audioLength variable
	+ played ( ): bool	A function that returns true if the audio has been played by the users that received it.
	+ playMessage ( ): void	This function is overridden in every child class inheriting from the Message Class. This function plays the audio recording

Date		
Attributes	+ year: int	An integer to represent the year
	+ month: int	An integer to represent the month
	+ day: int	An integer to represent the day
Methods	+ Date(year: int, month: int, day: int)	The constructor that is used to instantiate objects and initialize the variables

Forum		
Attributes	- ForumID: string	The Forum ID. Each forum has a unique ForumID
	- ForumTitle: string	The title for the forum
	- StartDate: Date	This represents the date on which the forum was created
	- ParticipantsID: string [ ]	This array keeps track of all the participants ID in the forum
	- ForumMessages: Message [ ]	This array records all the messages that have been sent in that forum
	- NumberofParticipants: int	This integer counts the number of participants in the forum
Methods	+ Forum ( )	The constructor that is used to instantiate objects and initialize the variables
	+ setForumID(id: int): void	The setter method for the ForumID
	+ getForum(): int	The getter method for the ForumID
	+ setForumTitle(title: string): void	The setter method for the ForumTitle
	+ getForumTitle(): string	The getter method for the ForumTitle
	+ setForumStartDate(date: Date): void	The setter method for the StartDate
	+ getForumStartDate(): Date	The getter method for the StartDate
	+ addForumParticipant(participantID: string): void	This function is used to add a participant to the array of participants

	+ getForumParticipants(): string [ ]	This function gets all the members in a forum
	+ addForumMessage(newMSG: Message): void	This function adds a message in a forum
	+ getForumMessages() Message [ ]	This function gets all the messages that have been sent in a forum
	+ SendTextMessage( )	This function is used to send a message to the members in a forum

Gam3yaForum		
Attributes	- myGam3yaID: string	A string that associates a Gam3yaForum to the Gam3ya it is created for
Methods	+Gam3yaForum ()	The constructor that is used to instantiate objects and initialize the variables
	+ setGam3yaID(id: string): void	The setter method for the myGam3yaID
	+ getGam3yaID(): string	The getter method for the myGam3yaID

PrivateForum		
Attributes	- OwnerAccountID: string	A string that associates a Gam3yaForum to the Gam3ya it is created for

Methods	+PrivateForum ()	The constructor that is used to instantiate objects and initialize the variables
	+ set OwnerAccountID (id: string): void	The setter method for the OwnerAccountID
	+ get OwnerAccountID (): string	The getter method for the OwnerAccountID

Interface		
Attributes	- pages: Page[]	An array of pages that represent the interface
Methods	+Interface ()	The constructor that is used to instantiate objects and initialize the variables
	+ setPages(Page[ ]): void	The setter function for page array
	+ getPages(): Page[ ]	The getter function for the pages array
	+ displayInterface(): void	This function displays the interface

Page		
Attributes	- tabs: Tab[]	An array of tabs to be displayed on the interface
	- buttons: Button[]	An array of buttons to be displayed and used on the interface
	- textboxes: Textbox[]	An array of texboxes to allow for user input on the interface

Methods	+ Page()	The constructor that is used to instantiate objects and initialize the variables
	+ setTabs(Tab[ ]): void	The setter function for the tabs array
	+ getTabs(): Tab [ ]	The getter function for the tabs array
	+ setButtons(Button [ ]): void	The setter function for the buttons array
	+ getButtons(): Button [ ]	The getter function for the buttons array
	+ setTextboxes(Textbox [ ]): void	The setter function for the textboxes array
	+ getTextboxes(): Textbox [ ]	The getter function for the textboxes array
	+ displayPage()	This function displays the whole page with all the elements in it

Tab		
Attributes		
Methods	+ Tab()	The constructor that is used to instantiate objects and initialize the variables
	+ displayTab(): void	This function is responsible for displaying a tab

Button		
Attributes	- redirectionPage: string	A string that carries the URL that the button redirects to

Methods	+ Button()	The constructor that is used to instantiate objects and initialize the variables
	+ setRedirectionPage(string): void	The setter function for the redirectionPage variable
	+ getRedirectionPage(): string	The getter function for the redirectionPage variable
	+ displayButton(): void	This function is used to make the button visible
	+ clickOnButton(): void	This function is invoked when a button is clicked

Textbox		
Attributes		
Methods	+ Textbox()	The constructor that is used to instantiate objects and initialize the variables
	+ displayTextbox(): Void	This function is responsible for displaying the textbox and making it available for the user to input a string
	+ setContent(newContent: string): void	The setter function for the content variable
	+ getContent(): string	The getter function for the content variable

PaymentServiceProvider		
Attributes	+ serviceAPIURL: string	The URL for the API for the payment service provider

Methods	+ PaymentServiceProvider ()	The constructor that is used to instantiate objects and initialize the variables

Fawry		
Attributes	- phoneNumber: string	The phone number associated with Fawry
	- merchantIDs: string [ ]	The list of IDs of the merchant that are associated Fawry and our system
Methods	+ Fawry( )	The constructor that is used to instantiate objects and initialize the variables
	+ getNumber(int);	The getter function for the phoneNumber variable
	+ getMerchantIDs(): string [ ]	The getter function for the merchantIDs array
	+ setNewNumber(int);	The setter function for the phoneNumber variable
	+ setMerchantIDs(string [ ]): void	The setter function for the merchantIDs array
	+ receivePayment(): string	The function to handle receiving a payment from payment service provider and returns the payment details
	+ sendPayment: void	The function to handle sending a payment to a user via Fawry
	+ validatePayment(): bool	A function that validates that a payment has been received successfully

EWallet		
Attributes	- walletID: string	The ID of the wallet that is going to receive or send an amount of money (payment)
Methods	+ EWallet( )	The constructor that is used to instantiate objects and initialize the variables
	+ getWalletID(): string	The getter function for the walletID variable
	+ setWalletID(string): void	The setter function for the walletID variable
	+ receivePayment(): string	The function to handle receiving a payment from payment service provider and returns the payment details
	+ sendPayment: void	The function to handle sending a payment to a user via an EWallet
	+ validatePayment(): bool	A function that validates that a payment has been received successfully

CoinBase		
Attributes	- email: string	The CoinBase email of the client using this payment service
Methods	+ CoinBase( )	The constructor that is used to instantiate objects and initialize the variables
	+ getEmail(): string	The getter function for the email variable
	+ setEmail(string): void	The setter function for the email variable

	+ receivePayment(): string	The function to handle receiving a payment from payment service provider and returns the payment details
	+ sendPayment: void	The function to handle sending a payment to a user via CoinBase
	+ validatePayment(): bool	A function that validates that a payment has been received successfully

5. COMPONENT DESIGN

In this section, some of the used algorithms in the program will be explained.

5.1 calculateLatePenalty

```
// A function that

newReputation = oldReputation - numberOfLateDays * 10
```

5.2 calculateReputationBonus

```
newReputation = oldReputation + numberOfGam3yaDays * 2
```

5.3 setReceivalOrder

```
//A function that orders which decided which user gets paid each month.

let maxNumber = NumberOfUsersInGam3ya
loop i from 1 to maxNumber:
        while(true):
        currentRandomNumber = randomNumberBetween(1, maxNumber)
        if(!usedBefore(currentRandomNumber)):
        Gam3yaUser[i] = currentRandomNumber
        break;
        endif
        else:
        //loop again
        endelse
        endloop
endloop
```

6. HUMAN INTERFACE DESIGN

6.1 Overview of User Interface

There user interface will allow interaction between end-users and the system which will allow the users to make use of the system functionalities. The actions that the user interface provides are going to be:

1. Sign up if the user does not have an account
2. Sign in if the user has an account
3. Search for an already existing Gam3ya
4. Creating a new Gam3ya if the user has enough reputation points
    - 4.1. Enter the start date of the Gam3ya
    - 4.2. Enter the minimum number of reputation points required to join
    - 4.3. Enter receival intervals
    - 4.4. Enter Gam3ya amount
5. Join an already existing Gam3ya
6. Display all Gam3yas the user has joined
7. Display all details about a specific Gam3ya the user has joined
    - 7.1. Time left for receiving money
    - 7.2. Deadline for payment
    - 7.3. Amount to be paid
    - 7.4. Amount to be received
    - 7.5. Members in the Gam3ya
8. Display all the forums the user has joined
9. Create private forums
10. Send message
    - 10.1. Send text
    - 10.2. Send Audio
    - 10.3. Send text
    - 10.4. Send image
11. Use help services
12. Watch tutorials

Action	System Feedback
1. Sign up if the user doesn't have an account.	The system displays a form where the user can enter their personal data. This data includes the birthdate and the national ID number. The system checks if the user's ID is verified and if the user has passed the appropriate age to join the Gam3ya.
2. Sign in if the user has an account.	If the user entered valid credentials, the system will display the home screen of the user. If the user entered invalid credentials, the system will display an error message.
3. Search for an already existing Gam3ya	The system has a search box, where the user can search for a Gam3ya using the amount to get paid, the receival intervals or the minimum number of reputation points. The system then displays search results with the join button activated only for the Gam3yas the user can join.
4. Creating a new Gam3ya if the user has enough reputation points	If the user has enough reputation points, the system allows the user to create a Gam3ya and enter 4 details about it: the minimum amount of reputation points required to join, the start date and the receival intervals and the Gam3ya's amount.
5. Join an already existing Gam3ya if the user has enough reputation points	The system displays a join button beside each each Gam3ya that the user is eligible for joining. The user can join by clicking this button.
6. Display all Gam3yas the user has joined	The system displays all the Gam3ya's the user has joined.
7. Display all details about a specific Gam3ya the user has joined	The system displays the time left for receiving money, the deadline for payment, the amount to be paid and the members in that Gam3ya.

8. Display all the forums the user has joined	The system displays all the forums that a user is a member of.
9. Create private forums	The system creates a forum where only chosen participants would be able to join the forum based on the forum creator's approval.
10. Send Message	The system displays a textbox, that allows users to attach photos, videos or a text message to send to other users in the forum.
11. Use help services	The system directs user to the Facebook Page where they can start a live-chat with a customer service representative
12. Watch tutorials	The system displays videos about how to use the platform.

6.2 Screen Images

The following are screenshots to show how the interface will look like for a user. The objects(including buttons, textboxes, etc.) are numbered so that they can be referenced in section 6.3 for further explanation.

6.2.1. Welcome Screen

This is the first screen that appears to the user when he/she tries to use the application. The user has one of two options:

1. The first is to press the login button if he/she has an already valid account.
2. The second is to press the sign up button if he/she does not have a valid account, so he/she has to create a new account.

6.2.2. Home Screen

This is the screen that will be displayed to the user after logging in. The reputation points the user has are displayed on the screen. There will be 3 options for the user to choose from:

1. To search for Gam3yas using the search bar at the top. The search criteria can be a Gam3ya ID or the amount of money to share with in a Gam3ya.
2. If the user has a certain amount of reputation points, the user can click on Create Gam3ya button which will redirect the user to the appropriate screen to create a new Gam3ya.
3. The system displays the best matches with the user below create a Gam3ya button. The user can click the Join button to join any of these Gam3yas.

There is a navigation bar at the very bottom of the screen that has 5 items:

1. Profile: The user can view and edit his personal information in this screen
2. Join: This is the screen currently displayed and the screen for creating a Gam3ya.
3. Current: This is for displaying the current Gam3ya the user is enrolled in and the details about each Gam3ya.
4. Chat: This moves to the screen of the forums
5. Help: Clicking this moves the user to the help and tutorials screen.

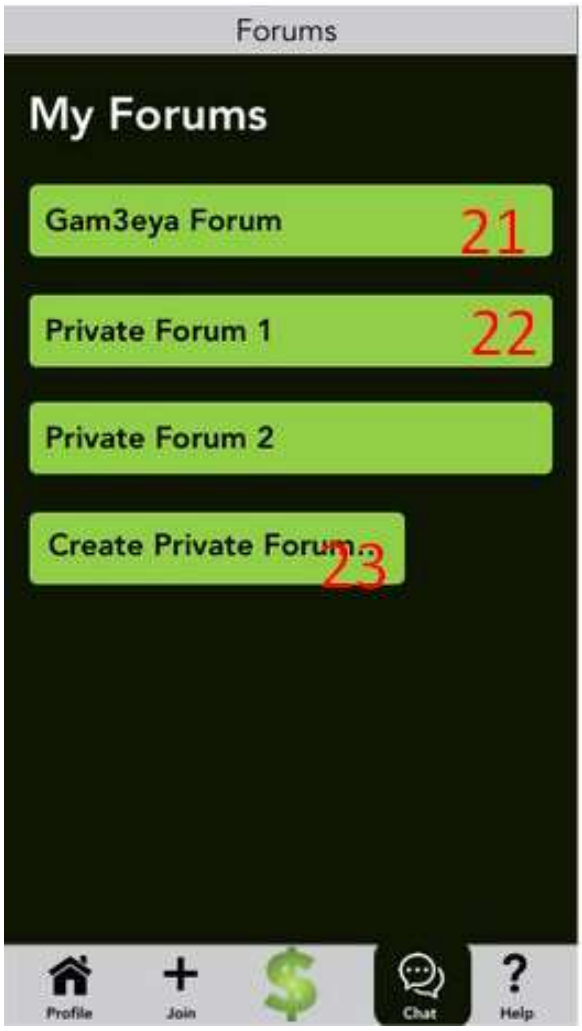
6.2.3. Create Gam3ya Screen

This screen is for creating a new Gam3ya from scratch. There are fields to enter the start date of the Gam3ya, the receival intervals, the minimum reputation points required to join the Gam3ya and the value of a share in a Gam3ya. The screen also displays the number of members in the Gam3ya so far. There is also a button at the top left corner to delete this Gam3ya before it starts.

6.2.4. Gam3ya Details Screen

This screen just displays info about the current Gam3ya the user is enrolled in. The details displayed are the time remaining to pay the monthly share, the time remaining to receive the money the user will get and the amount the user will get.

6.2.5. My Forums Screen



This screen is for displaying the forums the user is currently enrolled in. For entering each forum, there is a button for that forum. There is also a button for creating a new private forum.

6.2.6. Gam3ya Forum

Messages sent by all members in a Gam3ya are displayed here, in the Gam3ya Forum. There is a textbox in the bottom to enter text messages. The plus sign next to it is for attachments. There is also a camera sign for image/video attachments and a mic sign for sending voice notes.

6.2.7. Request Help Screen

There are two buttons, one for going into the live chat mode and the other for going into the tutorial mode.

6.3 Screen Objects and Actions

This following table shows each object in the User Interface with the action it is doing.

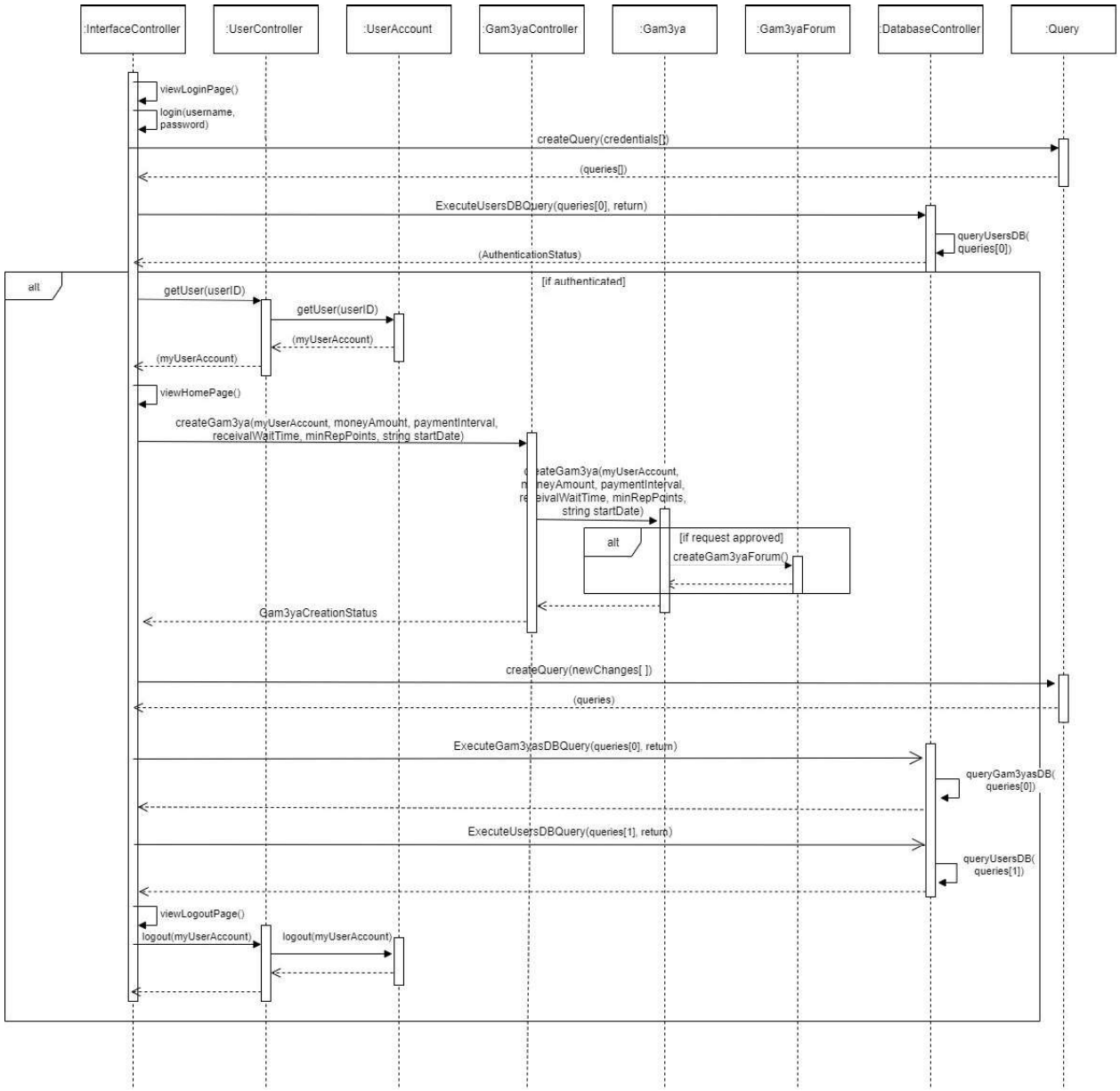
Object Number	Type	Description
1	Button	Login button
2	Button	Sign up button
3	Search bar	Gam3ya search bar
4	Label	Displaying reputation points of a user
5	Button	Join a Gam3ya button, this registers a user in that Gam3ya if he/she have the minimum number of reputation points.
6	Tab	This redirects the user to his/her profile so that the user can view and edit his personal information.
7	Tab	This redirects the user to his/her to screen where the user can search for a Gam3ya to join.
8	Tab	This redirects the user to a screen that shows all the Gam3yas the user has joined and can show the details of the Gam3yas the user has joined.
9	Tab	This tab redirects the users to the screen showing all the forums the user is enrolled in.

10	Tab	The tab redirects the users to the help screen.
11	Button	This button redirects the user to a screen where he/she can create a new Gam3ya if he/she has enough reputation points.
12	Button	This button deletes the Gam3ya that was about to be created (cancels the creation request).
13	Date field	This is to choose the starting date of the Gam3ya.
14	Date field	This is to choose length of the time interval between receiving each two payments.
15	Textbox	Textbox for entering the minimum number of reputation points.
16	Textbox	Monthly share to be paid amount.
17	Label	Time remaining for money receival label
18	Label	Deadline for next payment and the due amount
19	Button	This button redirects to a screen showing all members of a specific Gam3ya.
20	Label	Amount to be received
21	Button	This button redirects to the forum of a specific Gam3ya
22	Button	This button redirects to the

		forum of a private chat
23	Button	This button redirects to the screen of creating a private forum.
24	Textbox	Messaging textbox
25	Button	Uploading a picture or a video button
26	Button	Recording a voice note button
27	Button	Attaching a document to send in the chat button
28	Picture	This is a hyperlink image that redirects the user to the facebook page where the user can initiate a live-chat with a customer support representative
29	Picture	This is a hyperlink image that redirects the user to the youtube channel where the user can watch tutorials for using the platform
30	Button	Okay button to confirm the creation of the Gam3ya. If the creation is successful, a confirmation message is displayed.

7 Sequence Diagram

7.1 Scenario 1: Create a Gam3ya (request approved)



7.2 Scenario 2: Create a Private Forum