Bilkent University

Department of Computer Engineering

Senior Design Project

Project short-name: Charin

Low Level Design Report

Abdulkhaligli Mastan, Ahmadov Fuad, Mammadov Ismayil, Sadigova Shabnam

Supervisor: Prof. H. Altay Güvenir
Jury Members: Prof. Özcan Öztürk and Prof. Özgür Ulusoy

Feb 17, 2020

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS492

Contents

1. Introduction

On different continents of the world conditions of different societies, groups of people or even animals might be in poor conditions. Benevolent people help those who are in need or in a bad condition. Donating or in other words helping those who are in need can be a tiring process and would need a more organized, transparent way. Moreover, people can not always trust the 3rd person therefore we are proposing a mobile app called “Charin” to solve discussed issues. Charin is a mobile application where people can help others and socialize while supporting them. We tried to make the process of benefiting others more fun and appealing for users. While arranging events to help others, users will have a chance to meet with people from different backgrounds and make new friends. Nonetheless, the main purpose of Charin is finding a solution to problems we are able to solve such as providing necessary needs of people in need. Although there are some websites, foundations which serve the same purpose, we tried to reach a clearer approach where the users can clearly see where their donation goes. To be able to reach our goal, we used blockchain. Blockchain is becoming a famous technology nowadays. It was introduced in October 2008. First application of blockchain was bitcoin [1]. Blockchain technology allows you to monitor transaction of money according to hash codes of the transacted money. It gives the possibility of transparency at the payments. If we consider different social events that contain claims for donations to different people or places, transparency becomes crucial thing. Nowadays there are a lot of social awareness acts which people generally share these acts and give comments about these acts. However, in donation cases, there are a lot of frauds which you cannot know in which account your money is combined. In the Charin application, developers tried to solve this problem for society. The purpose of Charin application is to use Blockchain technology in social charity application. With the help of this application people can share the event and get support from their followers. In this application, people become famous with the help of their donations.

1.1 Object design trade-offs

1.1.1 Usability vs Functionality

Since Charin is an application which is aimed to be used by people of every age, the usability of the application is chosen over the functionality. For Charin , it is important that users from different backgrounds be able to use the application.

1.1.2 Security vs Cost

For security of the Charin, information about users will not be shared with third parties and for making the system even more secure blockchain is used. Blockchain might be costly, however; we chose security of the system over the cost.

1.1.3 Security vs Time

Charin is an application which uses blockchain. Blockchain is a technology which provides a secure platform for the users, it allows the users to see all the transactions that have been done but it is impossible for them to modify it. On the contrary, the blockchain is slow because the number of the ledgers might decrease the response time since for finding specific information all ledgers need to be visited.

1.2 Interface documentation guidelines

Classes which will be used in this project are documented in this report with some conventions. Every class follows the same design pattern such as the class name is demonstrated the first. Then the attributes and methods follow the class name.

1.3 Engineering standards (e.g., UML and IEEE)

In this report, IEEE citation guidelines are used for references. For visualizing design patterns, Unified Model Languages (UML) was used throughout these project reports.

1.4 Definitions, acronyms, and abbreviations

UI: User Interface

2. Packages

2.1 General Diagram

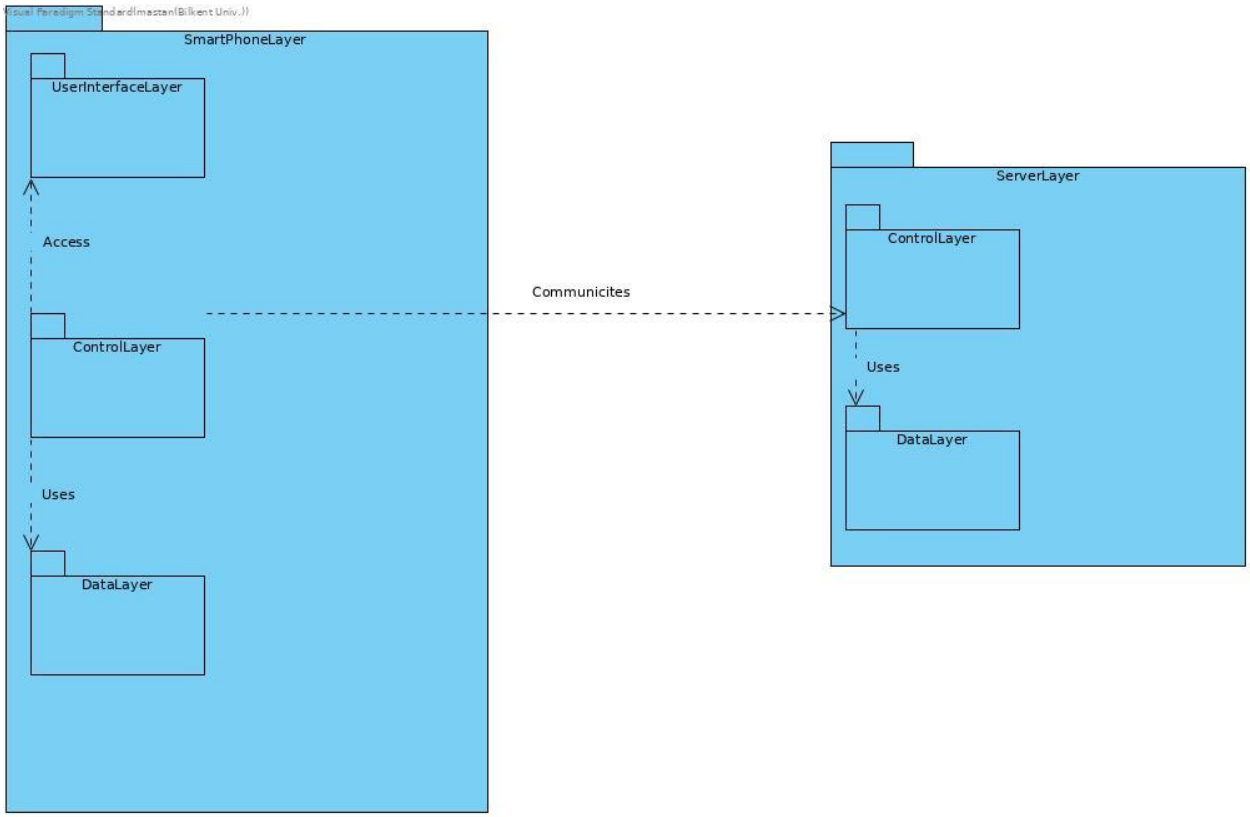

Figure 1. General Diagram

2.2 Smartphone Layer

2.2.1 UI Layer

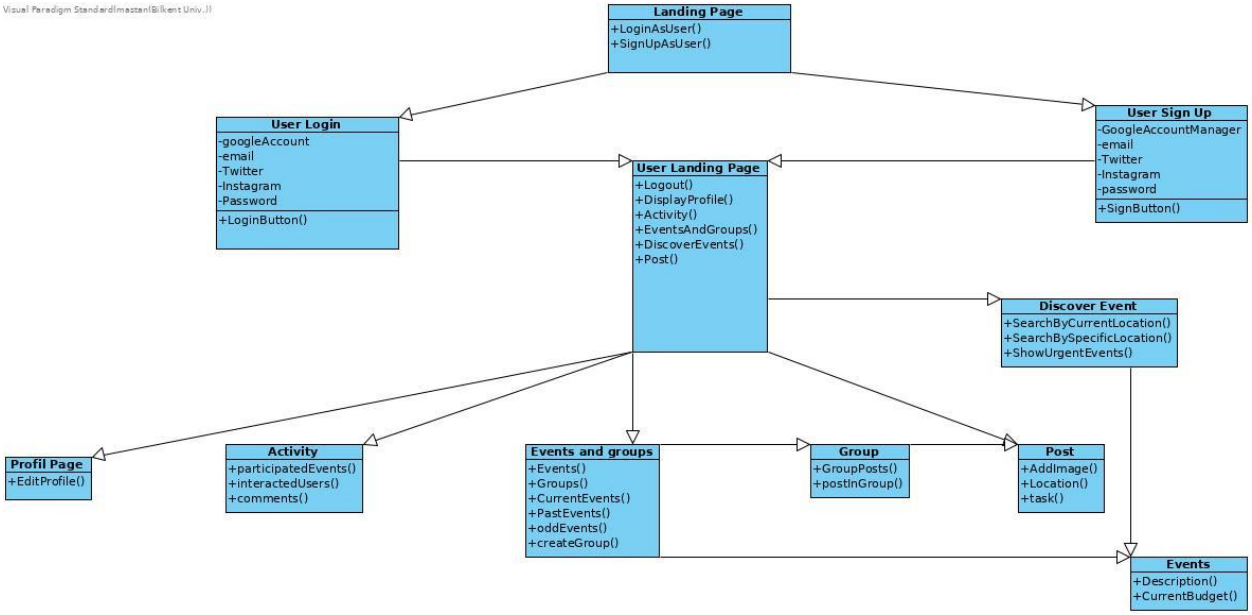

Figure 2. UI Layer Diagram

LandingPageActivity: This page is the entry screen of the program and connected to the authentication layer and signup, login pages. To use the application, users need to login or sign up to the program.

UserLoginActivity: User logins with Google, Instagram, Twitter or email accounts. Authentication manager of the account opens a pop up and the user enters credentials. The Control and Data Layer check info entered by the user. If a user exists, the User Landing Page activity appears.

UserSignupActivity: User signs into the program by using Google, Instagram, Twitter accounts or entering credentials with email address. The Control and Data Layer controls if this user exists. If it does not exist, the user's information will be stored on the database. After that, User Landing Page activity appears.

UserLandingPageActivity: User Landing Page is the home page of the program. On this page, user can share posts with Post Activity, and see shared posts by his/her friends. User can change his/her info from ProfilePageActivity. User can discover events from Discover Events Activity. User can add events/groups from Add Events Groups Activity. User can check interaction with other users/events from Activity.

ProfilePageActivity: On this page, user can update his/her profile information or change the photo.

DiscoverEventsActivity: On this page, user can discover events around him/her or at the specific location.

AddEventsGroupsActivity: On this page, user can control current/past events and also add groups or events.

Activity: On this page, user can control follow request of other users, interaction with followed users/events.

PostActivity: User can share posts with image and location information.

2.2.2 Control Layer

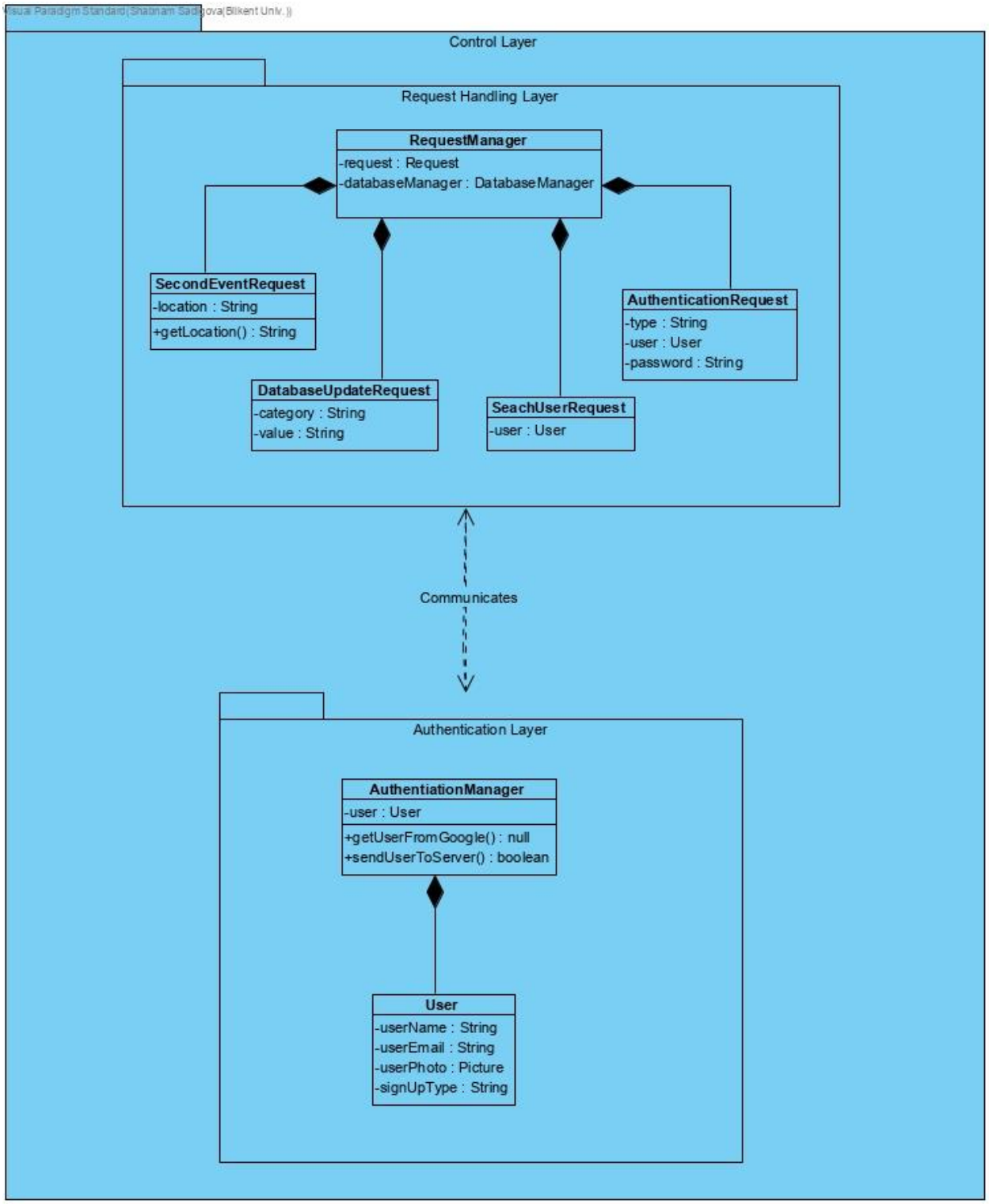

Figure 3. Control Layer Diagram

2.2.2.1 Authentication Layer

This layer is responsible for providing the authentication manager on the server-side with user information

User: User class represents any user in the system and holds user information.

AuthenticationManager: This class gets the user information and sends it to the server for authentication. It receives information from Google, Instagram, Twitter APIs or email accounts for regular users.

2.2.2.2 Request Handling Layer

Request handling layer evaluates and directs user requests.

RequestManager: This class is responsible for evaluation of requests and their responses. RequestManager is created by ServerManager and won't be terminated until the server is terminated. After incoming requests, RequestManager communicates with database layer and issue changes accordingly. If any response to a request is required RequestManager also sends the according response to the client.

DatabaseUpdateRequest: This entity class does not require any response to the client. Instances of this class are created by RequestManager and hold the related information of the corresponding request. They are terminated when the request is served. This class handles the process of updating the database on the client-side to synchronize it with the main database.

SearchEventRequest: Responds to the user with the output of current location or specific location related events.

2.2.3 Data Layer

This layer handles operations on the local database residing on the client device.

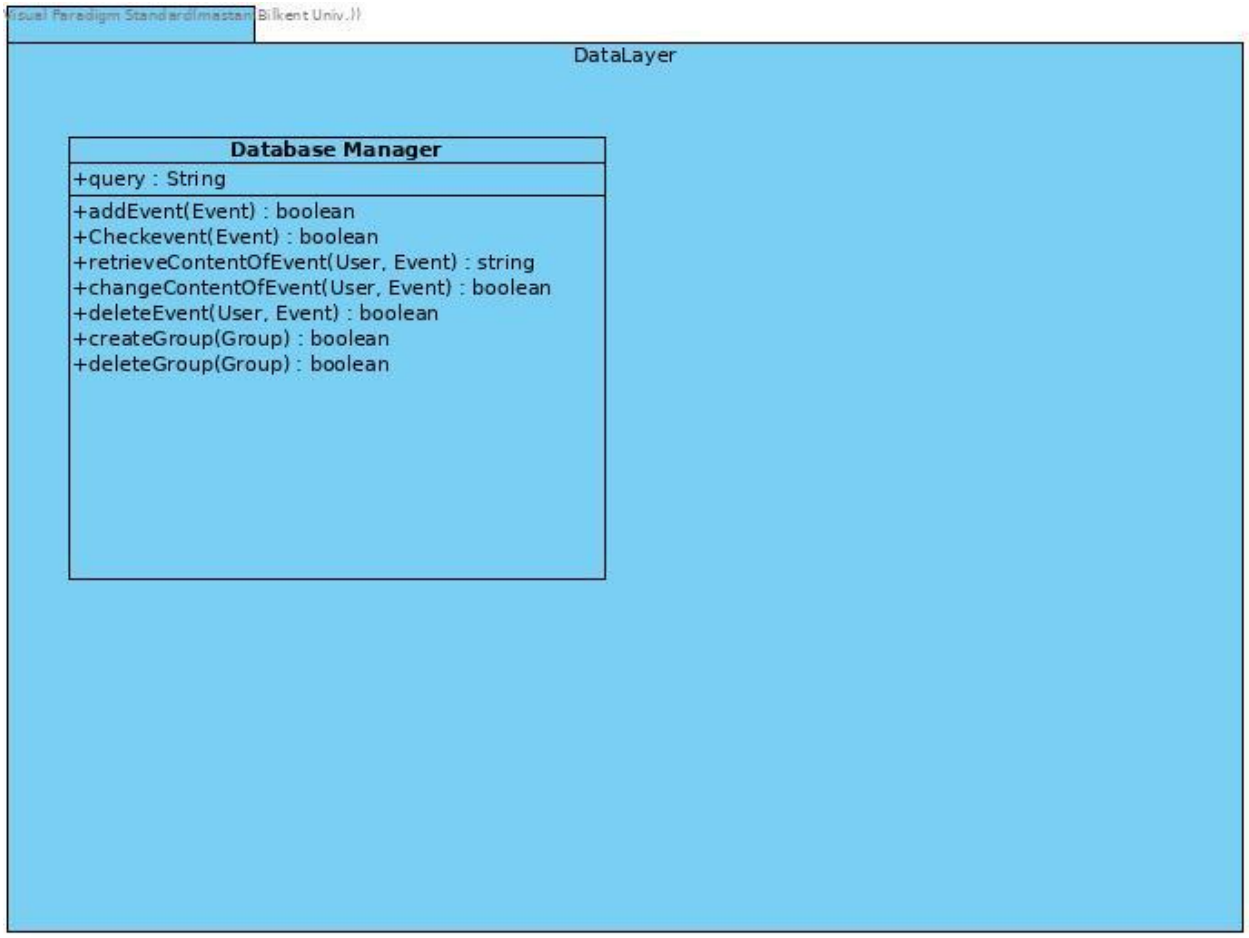

Figure 4. Data Layer

DatabaseManager: This class returns result to user search according to his/her search queries and handles requests which are coming from request handler.

2.3 Server Layer

2.3.1 Control Layer

Control Layer has authentication and request handling sublayers.

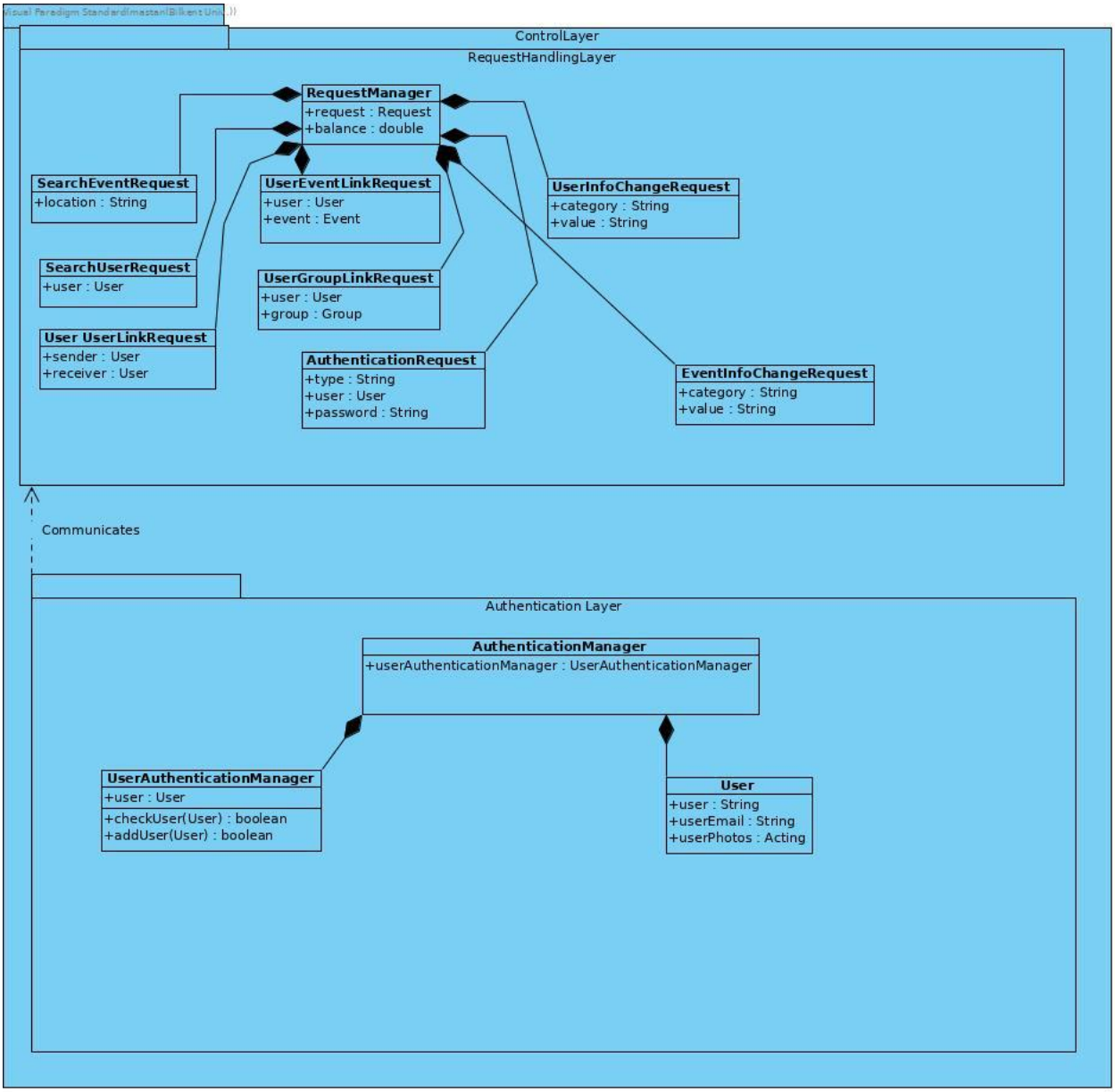

Figure 5. Control Layer Diagram

2.3.1.1 Request Handling Layer

User requests are evaluated and responded in Request Handling Layer.

Request Manager: Request manager class is responsible for evaluation of requests and their responses. RequestManager is created by ServerManager and won't be terminated until the server is terminated. After incoming requests, RequestManager communicates with database layer and issue changes accordingly. If any response to a request is required RequestManager also sends the according response to the client.

UserInfoChangeRequest-EventInfoChangeRequest: These classes do not respond to the client but change according to the values of User Info/Event Info. These classes are terminated when the request is served.

AuthenticationRequest: This request request authentication for according user.

UserUser/UserGroup/UserEventLinkingRequests: These classes are responsible for linking user with users, groups or events.

SearchEventRequest-SearchUserRequest: These classes are responsible for searching specific events in specific location or near the current location or user with the given username.

2.3.1.2 Authentication Layer

Authentication layer consists of classes that represent users and the control classes that are responsible for authenticating or registering a user.

User: User class represents user in the application and holds the information of the user who requested to login.

UserAuthenticationManager: This class is the controller class user authentication and sign up. It is instantiated by AuthenticationManager and won't be terminated until the server is terminated. It confirms the user authentication by communicating with the user database or sign up the user if he/she has no record in the user database.

AuthenticationManager: This class is the controller class for and UserAuthenticationManager. When an authentication request comes, it sends the request to UserAuthenticationManager.

2.3.2 Data Layer

Data Layer provides access to the database of Charin.

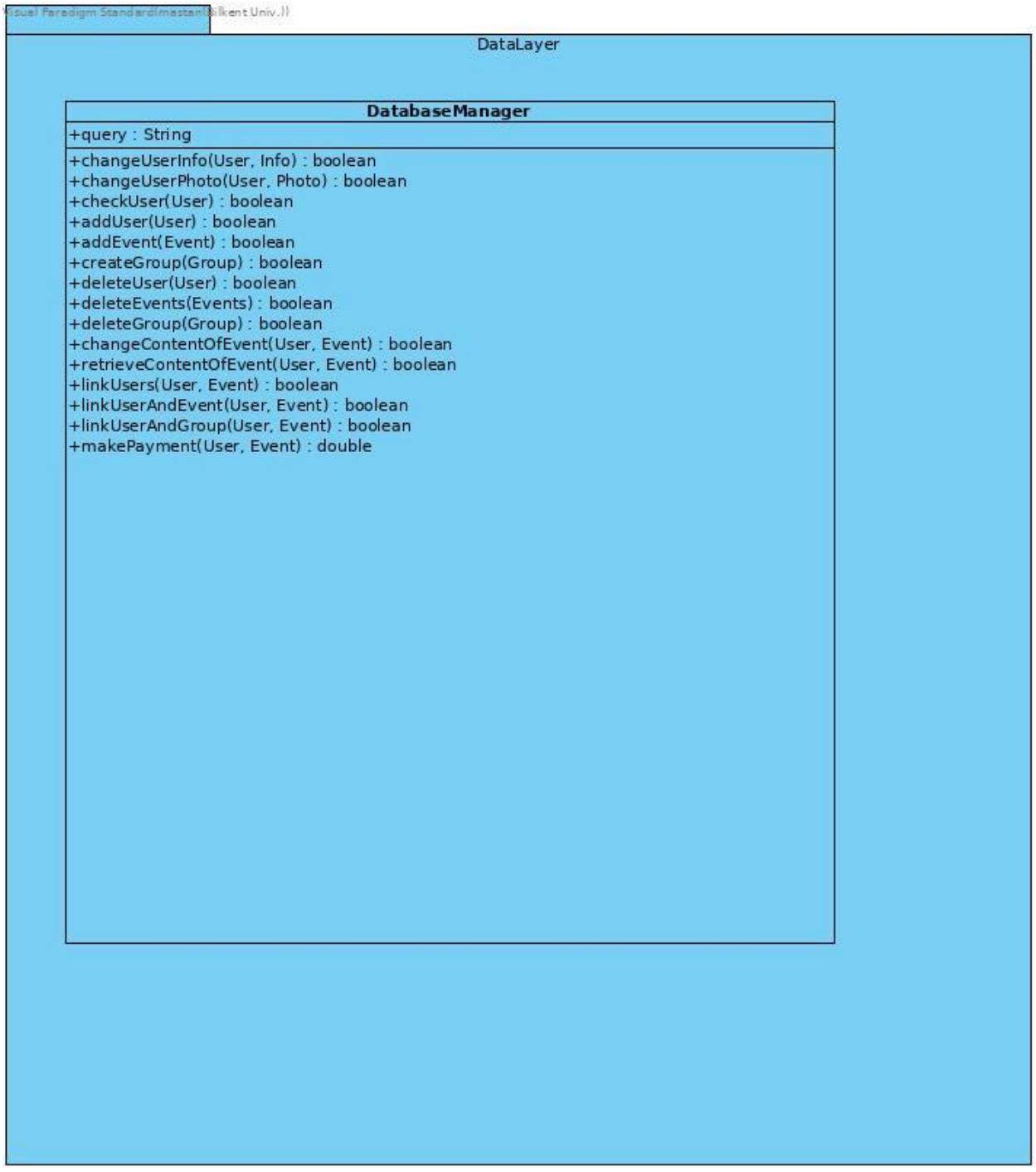

Figure 6. Data Layer Diagram

DatabaseManager: This class is running on the MongoDB system. According to given queries by the user, it changes user/group/event info, add/delete user/event/group, retrieve content of event, link user with user/group/event and update payment history of user.

3. Class Interfaces

3.1 User Interface Layer

3.1.1 Components

Class	SignIn
Class component for signing in to the system with email and password.	
Attributes	
email<String>	email to make registration to the app.
password<String>	password to logging in later
redirect<bool>	redirecting to the next page
error<String>	error message for displaying the erroneous state while logging in
Methods	
handleInputChange (ev <Object>)	changes the states of inputs while user is filling them
handleSubmit (e <Object>)	submitting the filled data
render()	renders the components of the SignIn

Class	NavigationBar
Class component for navigating among pages	
Methods	
NavigationBar(props <Object>)	Functoin to renders the link components

Class	App
Class component that contains every other component.	
Methods	
NavigationBar(props <Object>)	Function to renders the link components

Class	SignUp
Class component for signing up to the system with email.	
Attributes	
username<string>	Username to make registration to the app
email<String>	Email to make registration to the app
password<String>	Password to logging in later
confirmPassword	Confirmation for previously inserted password
firstname<String>	first name for registration
lastname<String>	last name for registration
profilePhoto<String>	link of the profile photo
error<string>	error message for displaying the erroneous state while signing up
redirect<bool>	state to redirect to next page or not
Methods	
handleInputChange (ev <Object>)	Changes the states of inputs while user is filling them

handleSubmit (e <Object>)	Submitting the filled data
renderImage()	Displays the selected image by the user
render()	Renders the components of the SignUpByEmail

Class	CreatePost
Class component for rendering and handling inputs for creating a post.	
Attributes	
postDescription<string>	description of created post
postImage<string>	directory string of the image
redirect<bool>	to redirect to new page
error<string>	error message for displaying the problem
Methods	
handleInputChange (ev <Object>)	changes the states of inputs while user is filling them
handleSubmit (e <Object>)	submitting the filled data
renderImage()	displays the selected image by the user
render()	renders the components of the CreatePost

Class	CreateEvent
Class component for rendering and handling inputs for creating an event.	
Attributes	

postDescription<string>	description of created event
eventLocation<List>	list of attributes that is essential for displaying location
eventPhotos<list>	list of photo link's that is uploaded
error<string>	error message for displaying the problem
Methods	
handleInputChange (ev <Object>)	changes the states of inputs while user is filling them
handleSubmit (e <Object>)	Submitting the filled data
renderImages()	displays the selected images by the user
renderLocation()	displays the selected location by the user
render()	renders the components of the CreateEvent

Class	CreateGroup
Class component for rendering and handling inputs for creating an group.	
Attributes	
groupName <string>	name of the group
groupDescription<string>	description of created event
groupUsers <list>	list of the users that is added to the group
error<string>	error message for displaying the problem
Methods	
handleInputChange (ev <Object>)	changes the states of inputs while user is filling them

handleSubmit (e <Object>)	Submitting the filled data
renderImages()	displays the selected images by the user
render()	renders the components of the CreateEvent

Class	UserProfile
Class component for displaying user profile.	
Attributes	
userData<Object>	obtained information about the user
isLoading<bool>	checking if the page is loading
error<string>	error message for displaying the problem
Methods	
componentWillMount()	React method to handle user requests
render()	renders the components of the UserProfile

Class	EditUserProfile
Class component for editing user profile.	
Attributes	
username<string>	Username to make registration to the app
email<String>	Email to make registration to the app.

password<String>	Password to logging in later
confirmPassword	Confirmation for previously inserted password.
firstname<String>	first name for registration
lastname<String>	last name for registration
profilePhoto<String>	link of the profile photo.
error<string>	error message for displaying the erroneous state while signing up
redirect<bool>	state to redirect to next page or not
Methods	
handleInputChange (ev <Object>)	Changes the states of inputs while user is filling them
handleSubmit (e <Object>)	Submitting the filled data
renderImage()	Displays the selected image by the user
render()	Renders the components of the SignUpByEmail
error<string>	error message for displaying the problem
Methods	
componentWillMount()	React method to handle user requests
render()	renders the components of the UserProfile

Class	UserProfile
Class component for displaying user profile.	

Attributes	
userData<Object>	obtained information about the user
isLoading<bool>	checking if the page is loading
error<string>	error message for displaying the problem
Methods	
componentWillMount()	React method to handle user requests
render()	renders the components of the UserProfile

Class	Activity
Class component for displaying activity of user and his/her follows.	
Attributes	
activityData<Object>	obtained activity data for the user
isLoading<bool>	checking if the page is loading
error<string>	error message for displaying the problem
Methods	
componentWillMount()	React method to handle user requests
render()	renders the components of the Activity

Class	Discover
Class component for user to discover.	
Attributes	
discoverData<Object>	filtered data for user to discover
isLoading<bool>	checking if the page is loading
error<string>	error message for displaying the problem
Methods	
componentWillMount()	React method to handle user requests
render()	renders the components of the Discover

Class	MainPage
Class component which will be displayed in the main page for the user.	
Attributes	
mainPageData<Object>	filtered data for user
isLoading<bool>	checking if the page is loading
error<string>	error message for displaying the problem
Methods	
componentWillMount()	React method to handle user requests
render()	renders the components of the MainPage

Class	Event
Class component for displaying details of an event.	
Attributes	
eventData<Object>	information about a specific event
isLoading<bool>	checking if the page is loading
error<string>	error message for displaying the problem
Methods	
componentWillMount()	React method to handle user requests
render()	renders the components of the Event

Class	Post
Class component for displaying the post	
Attributes	
postData<Object>	information about a post
isLoading<bool>	checking if the page is loading
error<string>	error message for displaying the problem
Methods	
componentWillMount()	React method to handle user requests
render()	renders the components of the Post

Class	Group
Class component for displaying details of a group.	
Attributes	
groupData<Object>	information about a specific group
isLoading<bool>	checking if the page is loading
error<string>	error message for displaying the problem
Methods	
componentWillMount()	React method to handle user requests
render()	renders the components of the Event

3.1.2 Services

Class	AuthServices
This module (script file) will export the auth REST request methods for being used by the auth components.	
Methods	
login(userData <Object>)	handles the sign in request of API
signUpEmail(userData<Object>)	handels the signup request with email of API
signUpTwitter(userData<Object>	handles the signup request with twitter of API

Class	EventServices
This module (script file) will export the REST request methods for being used by the Event components.	
Methods	
getEvent(eventId <Int>)	Gets data from API for the specific event with its specific id
getEvents(eventOption <Object>)	Gets a list of events from API for the specific options
uploadEvent(data <Object>)	Handles the POST request to upload an event to API
favEvent(eventId<Int>)	Sets the selected event as favourite event.

Class	PostServices
This module (script file) will export the REST request methods for being used by the Post components.	
Methods	
getPost(postId <Int>)	Gets data from API for the specific postt with its specific id
getPosts(postOption <Object>)	Gets a list of posts from API for the specific options
uploadPost(data <Object>)	Handles the POST request to upload an post to API

Class	UserServices
This module (script file) will export the REST request methods for being used by the User components.	

Methods	
getProfile(userEmail <String>)	Gets data from API for the specific user with its specific email
getProfile(userId <Int>)	Gets data from API for the specific user with its specific id
getPosts(userId <Int>)	Gets a list of posts from API from the specific user
getEvents(userId <Int>)	Gets a list of events from API from the specific user
followUser(userId <Int>)	Handles the POST request to follow a user to API

3.2 Logic Tier

3.2.1 Routers

Class	BaseRouter
This router will have all the router connected in a file.	
Attributes	
base_router <SimpleRouter>	the main router of the application

Class	UserRouter
This router will handle user route operations.	
Attributes	
router <SimpleRouter>	SimpleRouter object
Methods	
register()	registers the router

Class	AuthRouter
This router will handle authentication route operations.	
Attributes	
router <SimpleRouter>	SimpleRouter object
Methods	
register()	registers the router

Class	EventRouter
This router will handle event route operations.	
Attributes	
router <SimpleRouter>	SimpleRouter object
Methods	
register()	registers the router

Class	PostRouter
This router will handle post route operations.	
Attributes	
router <SimpleRouter>	SimpleRouter object
Methods	

register()	registers the router

Class	GroupRouter
This router will handle group route operations.	
Attributes	
router <SimpleRouter>	SimpleRouter object
Methods	
register()	registers the router

3.2.2 Controller

Class	UserController
This will hold the controller functions for the user	
Attributes	
-userModel <userViewModel>	accessor property to Data tier
Methods	
getProfile(userEmail <String>)	Gets profile by an email
getProfile(userId <Int>)	gets profile by user id
getPosts(userId <Int>)	gets shared posts shared by a user
getEvents(userId <Int>)	gets shared events shared by user
followUser(userId <Int>)	follows a specific user
getFollowers(userId <int>)	gets followers of the user

Class	AuthController
This will contain attributes and methods for user authentication.	
Attributes	
-authModel <authViewModel>	accessor property to data tier
Methods	
registerUserEmail(userObj <Object>)	registers user to system with email
registerUserTwitter(userObj <Object>)	registers user to system with twitter account
registerUserFacebook(userObj <Object>)	registers user to system with Facebook account
registerUserGoogle(userObj <Object>)	registers user to system with Google account
loginUser(userObj <Object>	logins user to the system

Class	EventController
This will contain attributes and methods for events.	
Attributes	
-eventModel <eventViewModel>	accessor property to data tier
Methods	
fetchEvent(userObj <Object>)	fetching the specific event with given settings
Class	PostController
This will contain attributes and methods for posts.	
Attributes	

-postModel <eventViewModel>	accessor property to data tier
Methods	
fetchPost(userObj <Object>)	fetching the specific post with given settings

3.3 Data Tier

Class	User
This will contain information about user.	
Attributes	
user_id <int>	id of the user
email <string>	email of the user
bio <string>	bio of the user
password <string>	password of the user
first_name <string>	first name of the user
last_name <string>	last name of the user
follows <list>	follows of the user
followers <list>	followers of the user
shared_events <list>	shared events by the user
shared_posts <list>	shared posts by the user
faved_events <list>	user’s favourite events
groups <list>	groups that user is in

Class	Event
This will contain information about event.	
Attributes	
event_id <int>	id of the event
description <string>	description of the event
images <lists>	links of images of an event
location <Object>	location of the event
donation <Object>	donation link for the event
enrolled_users <list>	users that wants to help to this event and enrolled
faved_users <list>	users that favored this event

Class	Post
This will contain information about post.	
Attributes	
post_id <int>	id of the post
description <string>	description of the post
images <lists>	links of images of an post
faved_users <list>	users that favored this event

Class	Group
This will contain information about group.	
Attributes	
group_id <int>	id of the group
description <string>	description of the group
enrolled_users <string>	users that are enrolled to the group
shareds <list>	list of the shared posts and events

4. References

[1] Iansiti Marco and Karim R. Lakhani. “The truth about blockchain” Harvard Business Review. https://enterprisersproject.com/sites/default/files/the_truth_about_blockchain.pdf. [Accessed: Oct 25, 2019]