

APUSTU APLIKAZIOA **SI PROIEKTUA** **DOKUMENTAZIOA**

Garikoitz Artola Obando
Ander Berruezo Aguirre
Beñat Burutaran Maiz

Aurkibidea

1. Sarrera.....	3
2. Proiektuaren enuntziatua.....	4
3. Eskakizunen bilketa.....	5
3.1 Domeinuaren eremua.....	5
3.2 Erabilpen kasuak.....	6
4. Diseinua.....	15
4.1 Sekuentzia diagramak.....	15
4.2 Klase diagrama.....	19
5. Izandako arazo nagusiak.....	20
6. Ondorioak.....	22

1. Sarrera

Proiektu honen taldeko kideak hasieran Beñat Burutaran, Garikoitz Artola, Ander Berruezo eta Eneko Ballesteros izan ziren. Lehenengo iterazioaren Scrum masterra Beñat Burutaran izan zen. Bigarren iterazioaren Scrum masterra Garikoitz Artola izan zen. Hirugarren iterazioan, Eneko Ballesteros proiektutik uko egin zuen. Orduan, taldean hiru kide gelditu ginen. Hirugarren eta azken iterazioaren Scrum masterra Ander Berruezo da.

Proiektuaren implementazioan, web zerbitzuak inplementatu egin ditugu. Gainera, internalizatuta dago, hau da, hizkuntza desberdinak inplementatu egin ditugu. 30 ordu eskini egin ditugu proiektu hau aurrera eramateko.

Bideo bat egin dugu proiektuaren analisi/diseinuarekin, garapena eta demoarekin. Lehendabizi komentatu starUML fitxategian Erabilpen Kasuen eta Domeinuaren Eredua egin ditugun moldaketak azpimarratuz. Jarraian, sortu dugu erabilpen kasuari buruzko sekuentzia diagrama aurkezten dugu. Amaitzeko, kodean egin ditugun aldaketak aipatzen ditugu eta proiektuaren demo bat azaltzen dugu.

Bideoa: <https://youtu.be/PiL0htz8VLY>

2. Proiektuaren enuntziatua

Proiektu honen helburua kirol apustueta oinarritutako sistema bat diseinatzea eta garatzea izan da. Horretarako, hiru mailako software arkitektura implementatu dugu programaren mantenimendua eta ulerterraztasuna bermatuz.

Sistemaren administratzaileak sistemaren gertaerak, gertaeren galderak eta galdera bakoitzaren pronostikoak bere irabaziarekin sortuko ditu. Erabiltzaile erregistratuek pronostiko batengan apustu bat egin dezakete.

Gertaera bukatzen denean, administratzaileak galderen emaitzak txertatuko ditu, eta sistemak bezeroek egin dituzten apustuen arabera bere kontuak eguneratzen ditu.

Edozein momentuan, erabiltzaileek egindako apustuak, beren egoera, eta bere etekina kontsultatzeko aukera izan beharko dute.

Proiektuko funtzionalitateak implementatzeko, 3 iteraziotan banatu dugu garapena:

0. Iterazioa:

Hasieran, bi erabilpen kasu inplementatuta zeuden: *Galdera sortu* eta *Galderak kontsultatu*. Biak *Admin* aktorearen erabilpen kasuak dira. Bestalde, *NLFacade*, *DataAccess* eta *EntityManager* klaseetaz gain, *Event* eta *Question* klaseak inplementatuta zeuden.

1. Iterazioa

Lau erabilpen kasu inplementatu behar dira: *Gertaera sortu*, *Kuota sortu*, *Hasi saioa* eta *Erregistratu*. Lehenengo biak *Admin* aktorearen erabilpen kasuak dira eta azken biak *Registered* aktoreenak.

2. Iterazioa

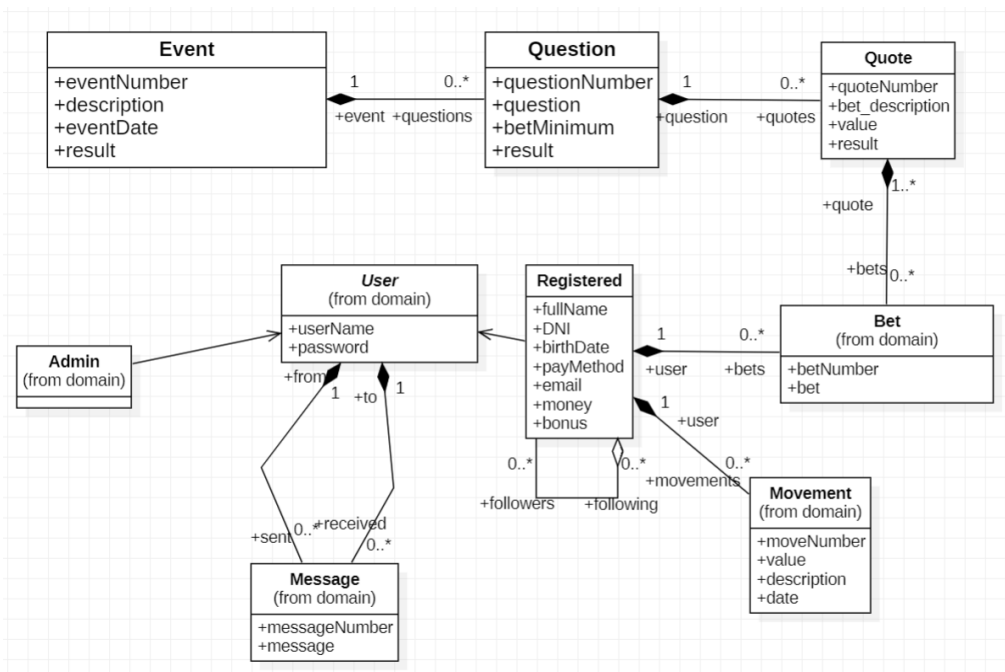
Bost erabilpen kasu inplementatu behar dira: *Dirua Sartu*, *Apustua Egin* eta *Mugimenduak ikusi*, *Registered aktorearentzako* eta *Emaitza Ipini* eta *Gertaera ezabatu*, *Admin* aktorearentzako.

3. Iterazioa

Apustu anitzak (kuota bat baino gehiagokoak), *Erabiltzailea jarraitu* (jarraitzaileek apustu berak egin), *Gertaera bikoiztu* eta *Mezuak bidali*. Gainera, gure aplikazioari balio berezi bat eman dezakeen erabilpen kasu bat inplementatu behar dugu.

3. Eskakizunen bilketa

3.1 Domeinuaren eremua



Hasieratik *Event* eta *Question* klaseak mantendu ditugu, *Event* bakoitza *Question* anitz izan dezake, eta *Question* bakoitza bakarrik *Event* bat dauka, bestela ezin da existitu.

Lehenengo iterazioan, *Quote* eta *User* klaseak gehitu genituen. *User* klasea abstraktua da, hau da, ezin dira *User* motako objekturik sortu, bakarrik bere klase-umekoak (*Admin* eta *Registered*). *Question* bakoitza *Quote* anitz izan dezake, eta *Quote* bakoitzak *Question* bat izan behar du.

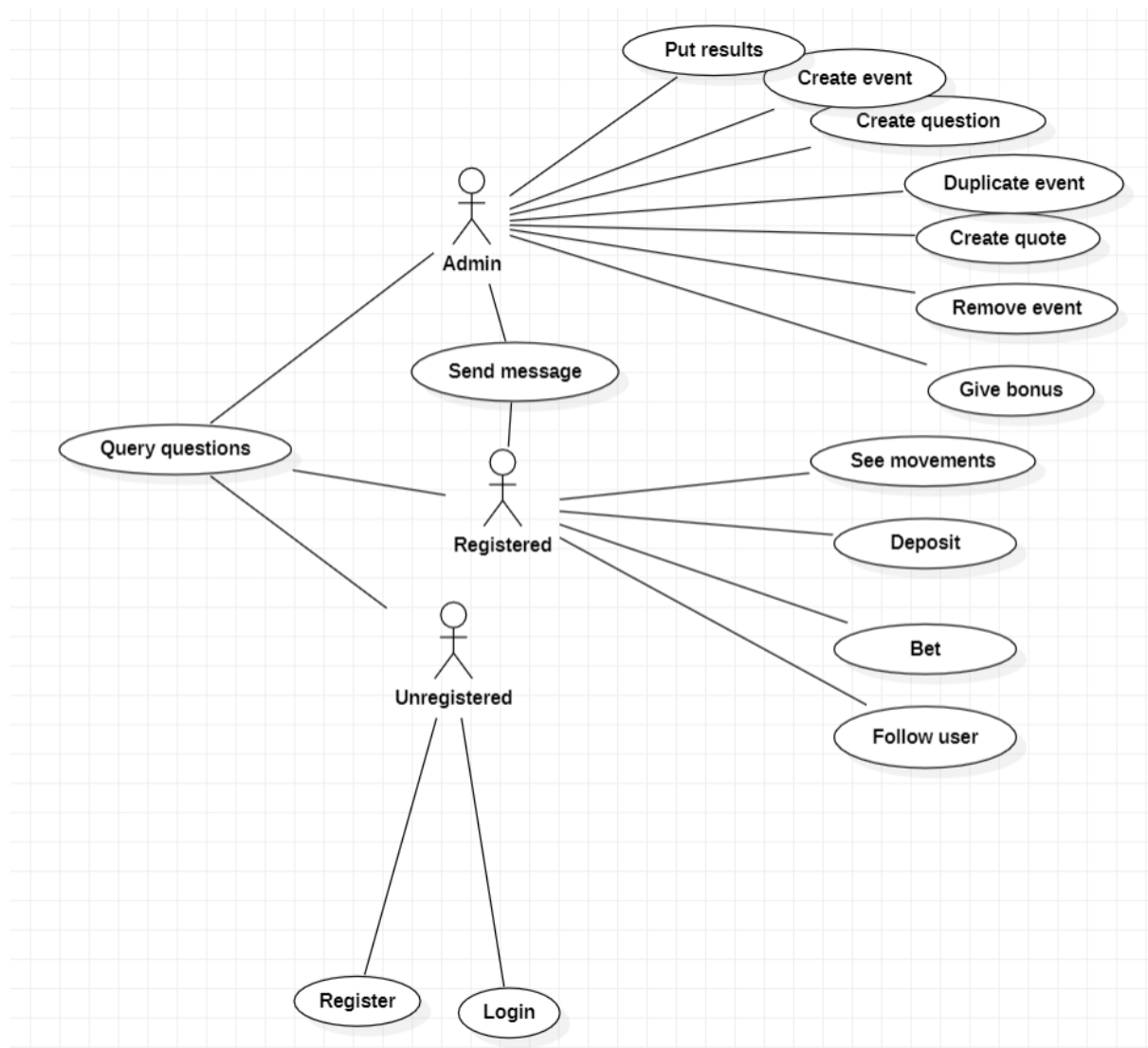
Bigarren iterazioan *Bet* eta *Movement* klaseak gehitu genituen. *Registered* bat edozein diru aldaketa egiten duenean, *Movement* berri bat sortzen da. Horregatik, *Registered* bakoitza *Movement* anitzak dauzka, eta *Movement* bakoitza *Registered* bat dauka soilik.

Bet klasea bi erlazio dauzka: alde batetik *Registered* bakoitza hainbat *Bet* egin ditzake, eta *Bet* bakoitza *Registered* bakar batekin erlazionatuta egon behar da. Beste aldetik, *Quote* bakoitza *Bet* anitz ditu, eta *Bet* bakoitza gutxienez *Quote* bat izan behar du, baina gehiago eduki ditzake.

Hirugarren iterazioan, *Message* klasea sortu genuen eta erlazioen aldaketa batzuk egin genituen. *Admin* eta *Registered* arteko mezuak *Message* klasean implementatu ditugu. *User* motako klase guztiek *sent* eta *received* atributuak dituzte, eta bertan gordetzen dira jasotako eta bidalitako mezu guztiak.

Message klasean *to* eta *from* atributuak daude, bakoitzak *User* bakar batekin erlazionatuta egon behar direnak. Erabiltzaileak jarraitzeko aukera implementatu dugunez, erlazio berri bat sortu dugu *Registered* klasean, non *Registered* bakoitza jarraitzaile anitz eta jarraitzen duten erabiltzaile anitz eduki dezaketen, ere *Registered* klasekoak.

3.2 Erabilpen kasuak



Aplikazioan hiru aktore daude:

- **Administratzailea:** Kontrol gehiena dauka, datu-basean atzipen maila handiena duen aktorea da. Gertaerak, galderak eta kuotak sortzeko gaitasuna dauka, eta gertaeren emaitzak ipintzekoa ere. Gertaerak ezabatu edo bikoiztu ahal ditu eta beste erabiltzaileekin komunikatzeko eta bere informazioa ikusteko aukera dauka.
- **Erregistratua:** Aplikazioa erabiltzen duen erabiltzaile normal bat da, eta eskubide mugatuak ditu. Apustuak egin, beste erabiltzaileak jarraitu, dirua sartu eta mugimenduak ikusteko gaitasuna dute.
- **Ez-erregistratua:** Aplikazioa ireki baino saioa hasi ez duen aktorea da. Gertaerak eta galderak ikus dezake, baina beste edozer gauza egiteko logeatu behar da. Konturik ez badauka, erregistratzeko aukera dauka.

Hauek dira erabilpen kasu guztien gertaera fluxuak eta interfaze-grafikoak:

-Query questions:

Flow of events

Basic Flow

1. *System* shows a Calendar where days with events are highlighted
2. *Admin* selects a **Date** in a Calendar
3. *System* displays the **events** of this **date**
4. *Admin* selects an **event**
5. *System* displays the **questions** associated to the selected **event**

Event date: June 2023

Events: June 17, 2023

Event#	Event
1	AtiÃtico-Athletic
2	Eibar-Barcelona
3	Getafe-Celta
4	AlavÃs-Deportivo
5	EspaÃol-Villareal
6	Las Palmas-Sevilla
7	Malaga-Valencia
8	Girona-LeganÃs

Questions for the event AtiÃtico-Athletic

Question#	Question
38	Zeinek irabaziko du partidua?
39	Zeinek sartuko du lehenengo gola?

Close

Alternative flow

1. There are no events on this date. Events cannot be shown.
2. There are no questions associated to the event. Questions cannot be shown.

-Create question:

Flow of events

Basic Flow

1. *System* shows a Calendar where days with events are highlighted
2. *Admin* selects a **Date** in a Calendar
3. *System* displays the **events** of this **date**
4. *Admin* selects an **event**
5. *Admin* introduces a **question** and a minimum **betting price**
5. *System* adds the new **question** with a minimum **betting price** to the selected **event**

Event date: June 2023

Events: June 17, 2023

2:Eibar-Barcelona

Question: Nork irabaziko du?

Min Bet: 5

Create Question Close

Alternative flow

1. There are no events on this date. Question cannot be added.
2. The question field is empty. Question cannot be added.
3. The minimum betting price is empty or it is not a number. Question cannot be added.
4. Event date has already finished (event day is before current day). Question cannot be added.

-Create event:

Basic Flow

1. *System* shows a Calendar where days with events are highlighted
2. *Admin* selects a **Date** in a Calendar
3. *System* displays the **events** of this **date**
5. *Admin* introduces a **event** and a **description**
6. *System* checks if the event already exists
7. *System* adds the new **event** with its **description** to the selected **Date**

Event date

June 2023

Description

Create Event Close

Alternative flow

1. The description field is empty. Event cannot be added.
2. Event day is before current day. Event cannot be added.
3. Event already exists. Event cannot be added.

-Create quote:

Basic Flow

1. *System* shows a Calendar where days with events are highlighted
2. *Admin* selects a **Date** in a Calendar
3. *System* displays the **events** of this **date**
4. *Admin* selects an **event**
5. *Admin* selects a **question**
6. *Admin* introduces a **description** and a **value**
7. *System* checks if quote already exists
8. *System* adds the new **description** and its **value** to the selected **question**

Event date

Events: June 17, 2023

1:Atlético-Athletic

List of questi..

38;Zeinek irabaziko du partidua?;1.0

Description

Value

Create quote Close

Alternative flow

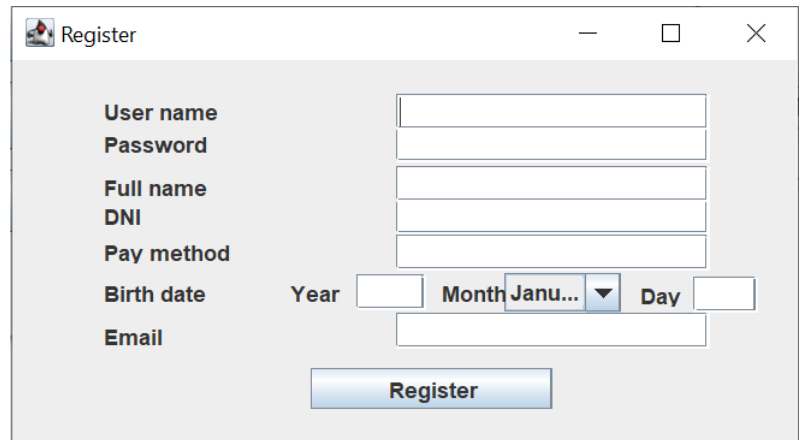
1. There are no events on this date. Quote cannot be added.
1. There are no questions on this event. Quote cannot be added.
2. The bet description field is empty. Quote cannot be added.
3. The value field is empty. Quote cannot be added.
4. The value field is less than 1. Quote cannot be added.
5. Event date has already finished (event day is before current day). Quote cannot be added.
6. Quote already exists. Quote cannot be added.

-Register:

Flow of events

Basic Flow

1. *System* shows three fields where the username, password and age can be written
2. *User* writes a **username**
3. *User* writes a **password**
4. *User* writes its **age**
5. *User* chooses **pay Method**
6. *User* writes a *email*
7. *System* checks if user is already registered
8. *System* registers new user



The Register form is a window titled 'Register' with standard window controls. It contains the following fields and controls:

- User name: text input field
- Password: text input field
- Full name: text input field
- DNI: text input field
- Pay method: text input field
- Birth date: composed of Year, Month, and Day fields. The Month field is a dropdown menu currently showing 'Janu...'. The Year and Day fields are text input boxes.
- Email: text input field
- Register: a blue button at the bottom.

Alternative flow

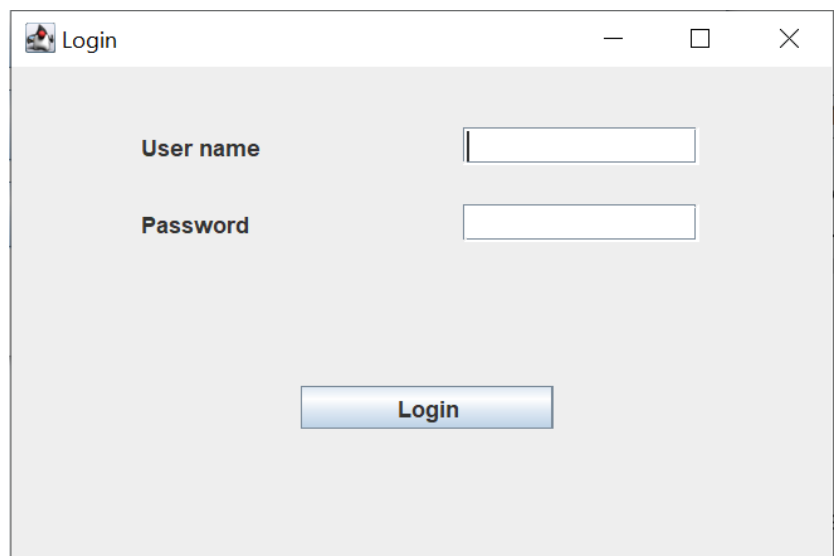
1. Username field is empty. Cannot register.
2. Password field is empty. Cannot register.
3. Username already exists. Cannot register.
4. Age field is empty. Cannot register.
5. Age is too low. Cannot register.
6. Pay method bottom is empty. Cannot register.
7. Email field is empty. Cannot register.
8. Email already exists. Cannot register.

-Login:

Flow of events

Basic Flow

1. *System* shows two fields where the username and password can be written
2. *User* writes a **username**
3. *User* writes a **password**
4. *System* checks if user exists and password is correct
5. *System* logs in



The Login form is a window titled 'Login' with standard window controls. It contains the following fields and controls:

- User name: text input field
- Password: text input field
- Login: a blue button at the bottom.

Alternative flow

1. Username field is empty. Cannot log in.
2. Password field is empty. Cannot log in.
3. Username does not exist. Cannot log in.
4. Password is incorrect. Cannot log in.

-Bet:

Flow of events

Basic Flow

1. *System* shows a Calendar where days with events are highlighted
2. *User* selects a **Date** in a Calendar
3. *System* displays the **events** of this **date**
4. *User* selects an **event**
5. *User* selects a **question**
6. *User* selects multiple **quotes**
7. *User* introduces a **bet amount**
5. *System* adds the new **bet** with a **valuet** to the selected **quote**
6. *System* checks if *user* has *followers*
7. *System* adds the same **bet** to the *followers*

Quote/#	Quote	Value
1	quote1	1.5
2	quote2	2.0
3	quote3	3.0

Alternative flow

1. There are no events on this date. Bet cannot be added.
2. There are no questions on this event. Bet cannot be added.
3. There are no questions on this event. Bet cannot be added.
4. The value field is empty. Bet cannot be added.
5. The value field is less than bet minimum . Bet cannot be added.
6. Event date has already finished or has a result (event day is before current day). Bet cannot be added.
7. User does not have enough money. Bet cannot be added.
8. Follower does not have enough money. Bet cannot be added.

-Deposit:

Flow of events

Basic Flow

1. *System* shows current deposit money and one field where the amount of money can be written
2. *User* writes a **amount money**
3. *System* deposit money

Alternative flow

1. Amount money field is empty. Cannot deposit in.
2. Amount money is negative. Cannot deposit in.
- 3.Amount money is string. Cannot deposit in.

Current money:	1250.0
Insert money	<input type="text"/>

-See movements:

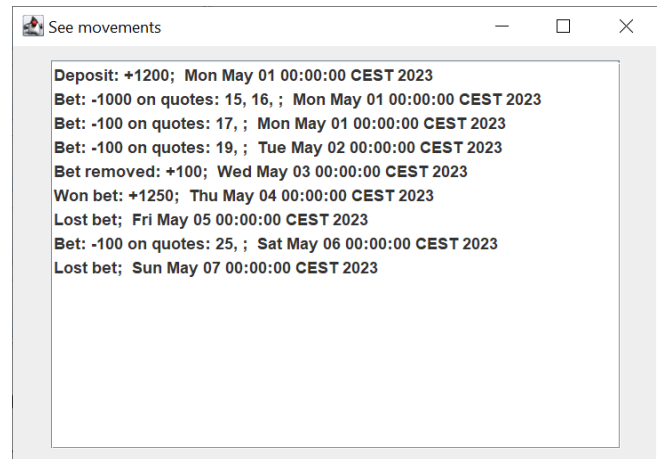
Flow of events

Basic Flow

1. *System* shows current deposit money and label where all the operation can query

Alternative flow

1. Movements are not exists. Cannot see movements.

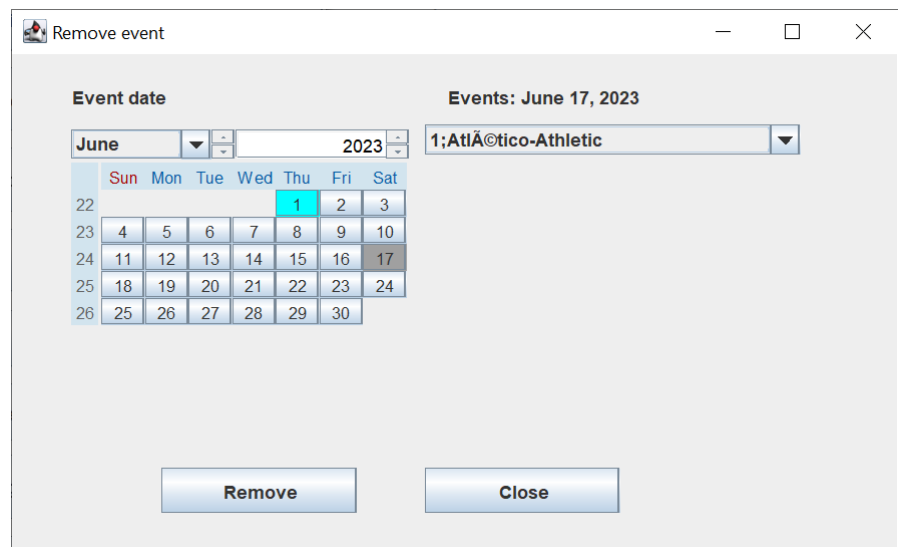


-Remove event:

Flow of events

Basic Flow

1. *System* shows a Calendar where days with events are highlighted
2. *Admin* selects a **Date** in a Calendar
3. *System* displays the **events** of this **date**
4. *Admin* selects an **event**
5. *Admin* delete an **event**
6. *System* checks if the event already exists
7. *System* return all amount money have bet.
8. *System* remove all bet have relation with this **event**
9. *System* remove all question have relation with this **event**
10. *System* remove **event** from Calender



Alternative flow

1. Event does not exist. Event cannot be removed.

-Put results:

Flow of events

Basic Flow

1. *System* shows a Calendar where days with events are highlighted
2. *Admin* selects a **Date** in a Calendar
3. *System* displays the **events** of this **date**
5. *Admin* selects a **event**
6. *Admin* introduces a **result**
7. *Admin* selects winning **quotes**
8. *Admin* clicks putResults button
9. *System* checks if the **event** exists (it has not been deleted while introducing results)
11. *System* checks if the **result** have been previously saved
12. *System* gets event's all **quote**
13. *System* gets **quote**'s all **bet**
14. if bet is selected *System* calculate $\text{bet.value} \times \text{quote value}$
15. *System* return result of bet to **User** by creating a **movement**
16. **bet** is removed from user

Put Result

Event date: June 2023

Events: June 17, 2023

1:AtlÁtico-Athletic

Event result:

List of questions: 38;Zeinek irabaziko du partidua?;1.0

List of quotes. Select winning quotes. Use CTRL to select multiple. Zeinek irabaziko du

Quote#	Quote	Value
1	quote1	1.5
2	quote2	2.0
3	quote3	3.0

Put results

Alternative flow

1. The result field is empty. Results cannot be saved.
2. Event has not finished yet. Results cannot be saved.
3. Event has been removed while introducing results. Results cannot be saved.
4. Result already exists. Results cannot be saved.

-Send message:

Basic Flow

1. *System* shows current message between registered and admin.
2. *Registered* write message to Admin or Admin write message to Registered.
3. System add new message.

Alternative flow

1. Messages are not exists. Cannot see messages
2. Registered are not exists. Cannot see messages.

Send Message

Received messages

Sat May 20 00:00:00 CEST 2023, From: admin, To: r1; kaixo

Sent messages

Sat May 20 00:00:00 CEST 2023, From: r1, To: admin; zer moduz?

Send to: admin

Send Message: zer moduz?

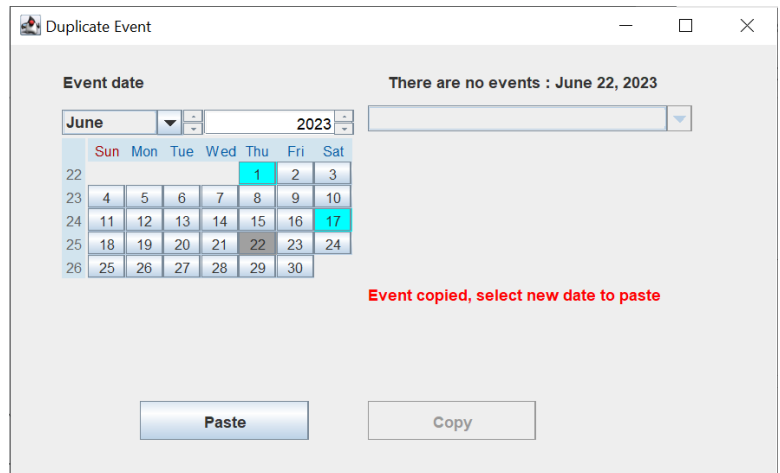
Send Message

-Duplicate event:

Flow of events

Basic Flow

1. *System* shows a Calendar where days with events are highlighted
2. *Admin* selects a **Date** in a Calendar
3. *System* displays the **events** of this **date**
4. *Admin* selects an **event**
5. *Admin* inserts a **date**
6. *System* gets **event**
7. *System* sets new **eventNumber** and **date**
8. *System* saves new **event**



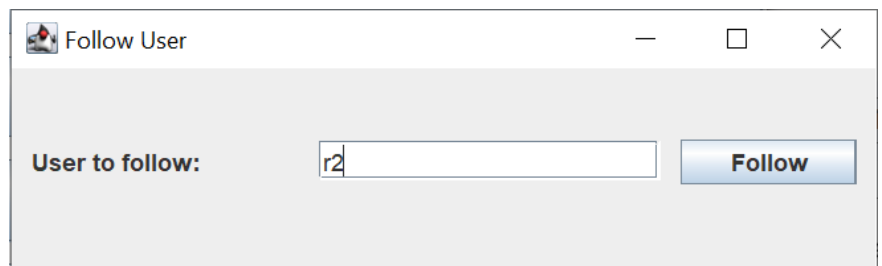
Alternative flow

1. **date** field is empty. Event cannot be duplicated.
2. **date** is incorrect. Event cannot be duplicated.
3. **date** is past. Event cannot be duplicated.

-Follow user:

Basic Flow

1. *User* introduces **username** to follow
2. *System* finds **username**
3. *System* verifies **username** is assigned to registered
4. *System* verifies *User* does not already follow **username**
5. *System* assigns *User* id as **follower** and **username** id as **following**



Alternative flow

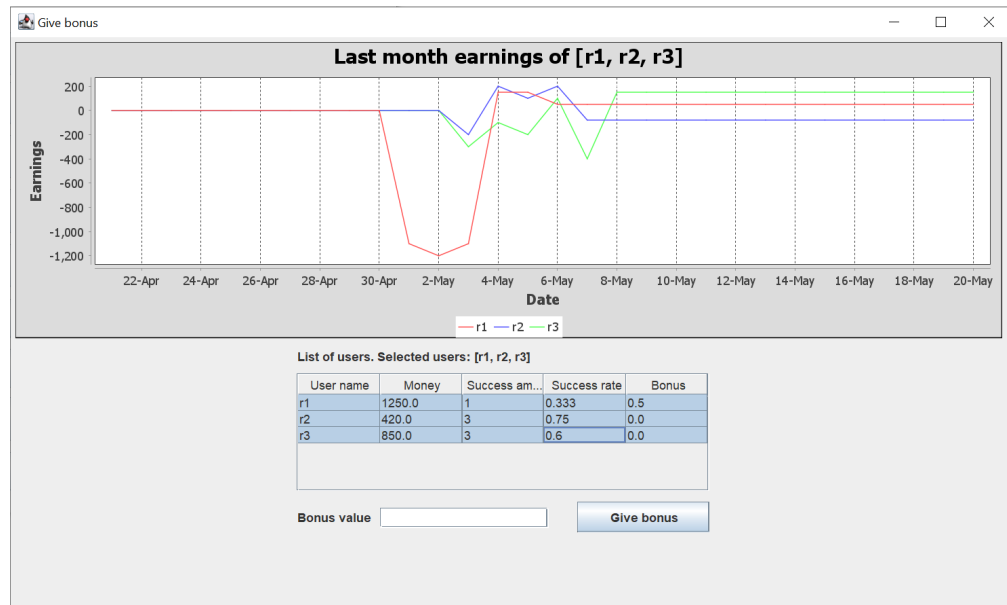
1. Username field is empty. Cannot follow another user.
2. Username is not registered. Cannot follow another user.
3. User already followed. Cannot follow this user.

-Give bonus:

Flow of events

Basic Flow

1. *System* shows users table ordered by winning quote amount.
2. *Admin* selects a **user** in table
3. *System* displays the **graph** of this **user**
4. *Admin* introduces **bonus** value
5. *Admin* gives **bonus** to selected **user**
6. *System* checks if **bonus** value is correct
7. *System* sets **bonus** value to **user**



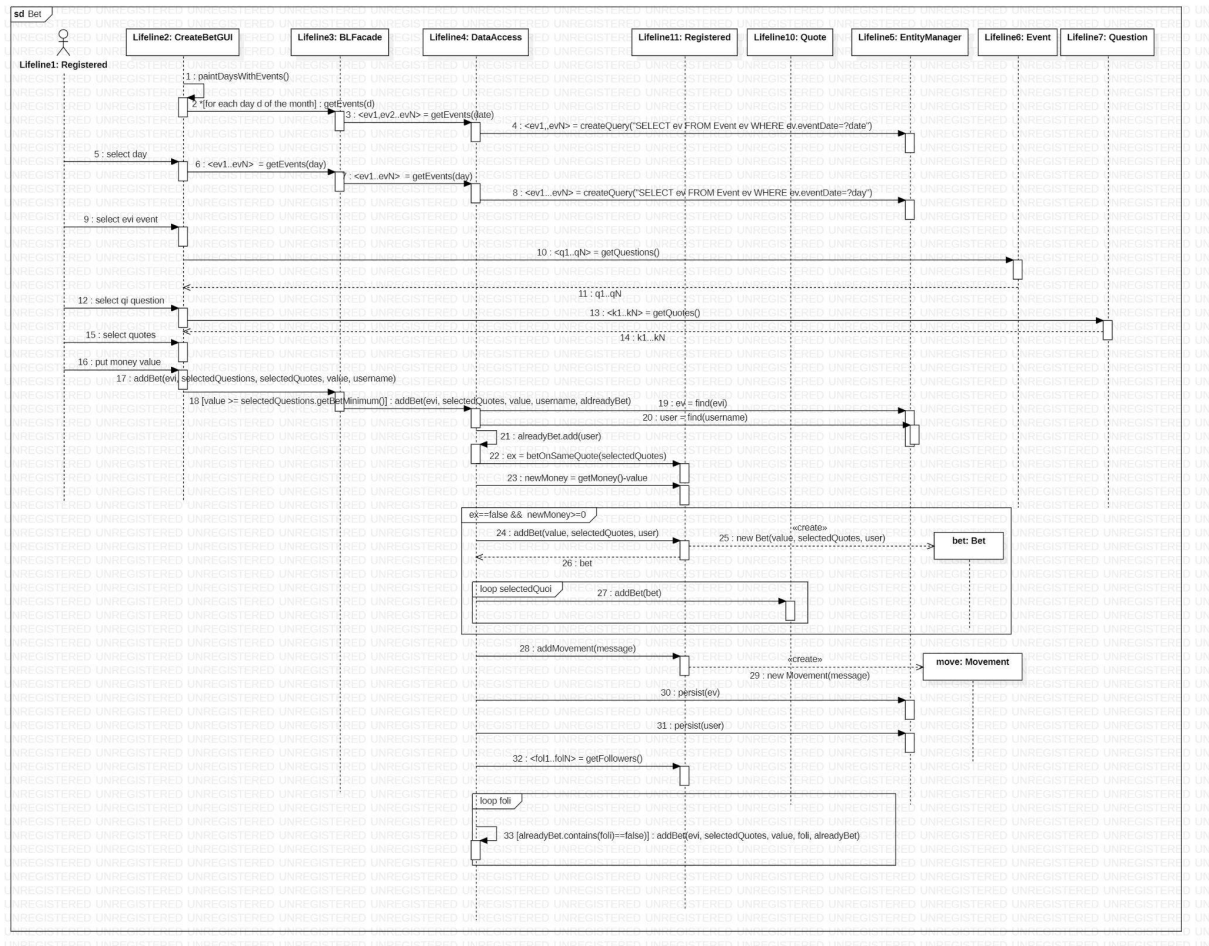
Alternative flow

1. There are no users. Graphs are not shown.
2. Bonus value is incorrect. Bonus is not given.

4. Diseinua

4.1 Sekuentzia diagramak

Bet:



Sekuentzia diagrama hau bigarren iteraziokoa zen, baina hirugarren iterazioan eguneratu behar izan dugu apustuak kuota anitzekin erlazionatzeko eta erabiltzailearen jarraitzaileak automatikoki apustuak egiteko.

Gertaera baten kuotak aukeratu ondoren, apustuaren balioa sartu behar da, eta metodo errekursibo bati deia egiten zaio, erabiltzailearen jarraitzaile guztiak apustu berdina egiteko. Erabiltzaile berdina apustu bera aldi bat baino gehiago ez egiteko, zerrenda bat jarri dugu parametro bezala, apustua egin duten erabiltzaileak gordetzen dituen. Gertaeren kuotetan eta erabiltzaileen kuotetan apustuak gehitzen ditugunez, gertaeran eta erabiltzaileetan persist egin behar da, datu-basea eguneratzeko.

Follow user:

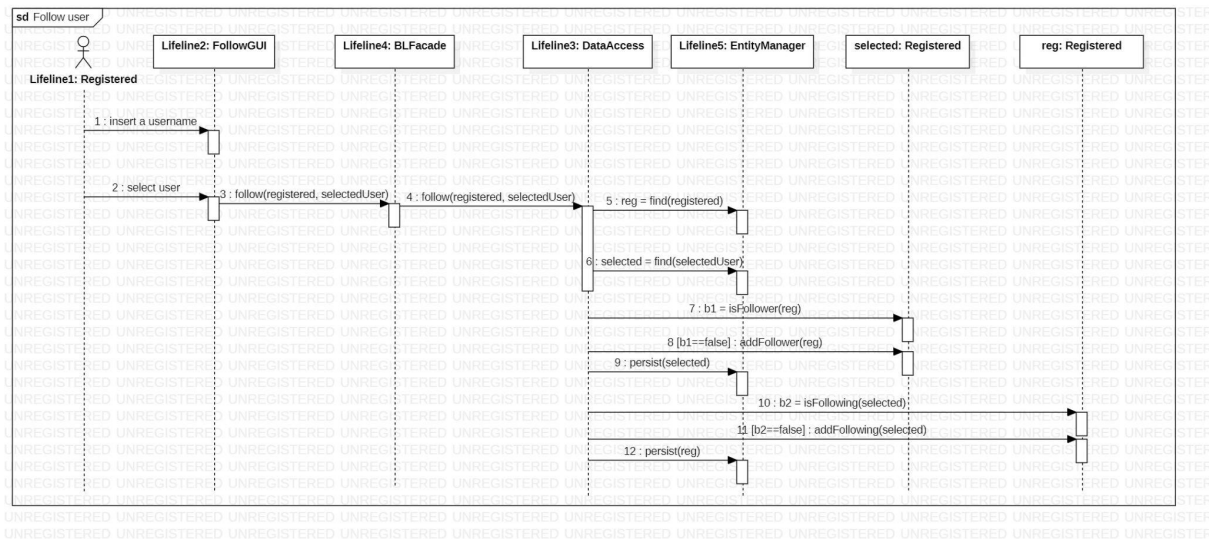
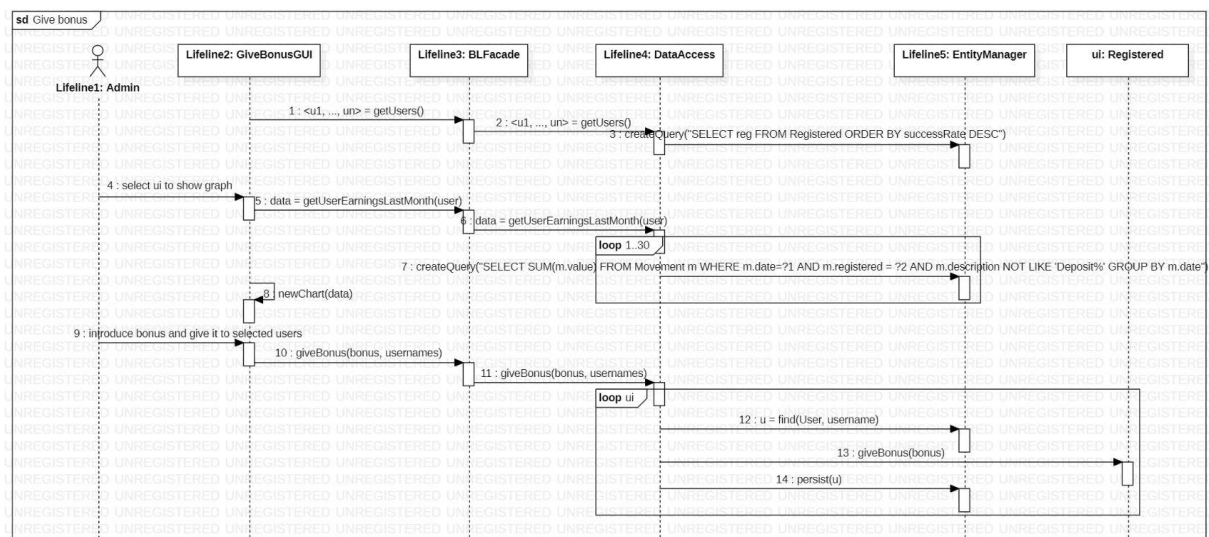


Diagrama hau sinpleagoa da, interfazeaz idatzi behar da jarraitu nahi duzun erabiltzaile erregistratua. Existitzen bada, datu-basetik hartzen dira bai bilatutakoa, bai zurea, eta bien jarraitzaile eta jarraitzen duten erabiltzaileen zerrendak eguneratzen dira.

Give bonus:



Lehenik eta behin, GUI-ak Erabiltzaile guztiak eta haien datu batzuk taula batean azaltzen ditu, horretarako datu basea irakurtzen du.

Ondoren, administratzaileak erabiltzaileak aukeratu eta datu basetik haien azken hileko irabaziak irakurriko ditu eta grafikoa azalduko zaio GUI-an, erabiltzaileen irabaziak konparatu ahal izateko.

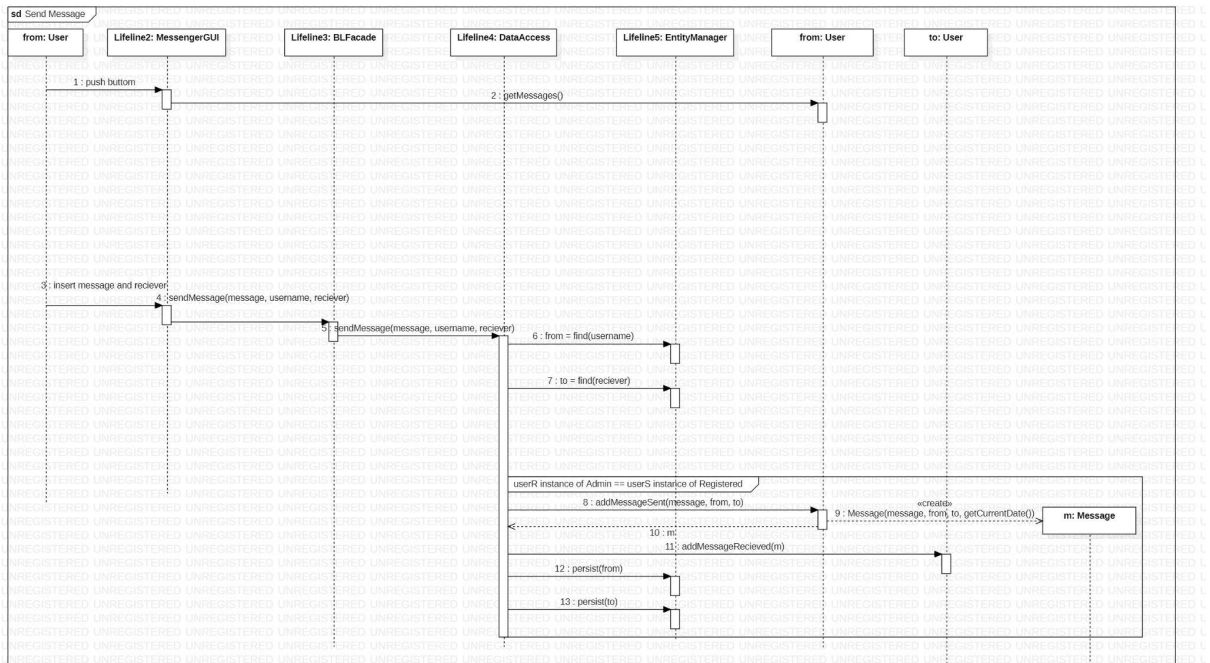
Azkenik, administratzaileak erabiltzaile bat edo gehiago aukeratuko ditu bonus bat emateko, eta erabiltzaileko persist egiten du datu basea eguneratuz.

[illegible]

Hau egin ondoren, datu-basetik aukeratutako gertaera jasotzen da, eta bere galdera eta kuota guztien atributu guztiak dituzten galdera eta kuota berriak sortzen dira, aurrekoen berdinak direnak, baina ID desberdinekin.

Gertaeraren kopia prest daqonez datu-basean sartzen da.

Send message:



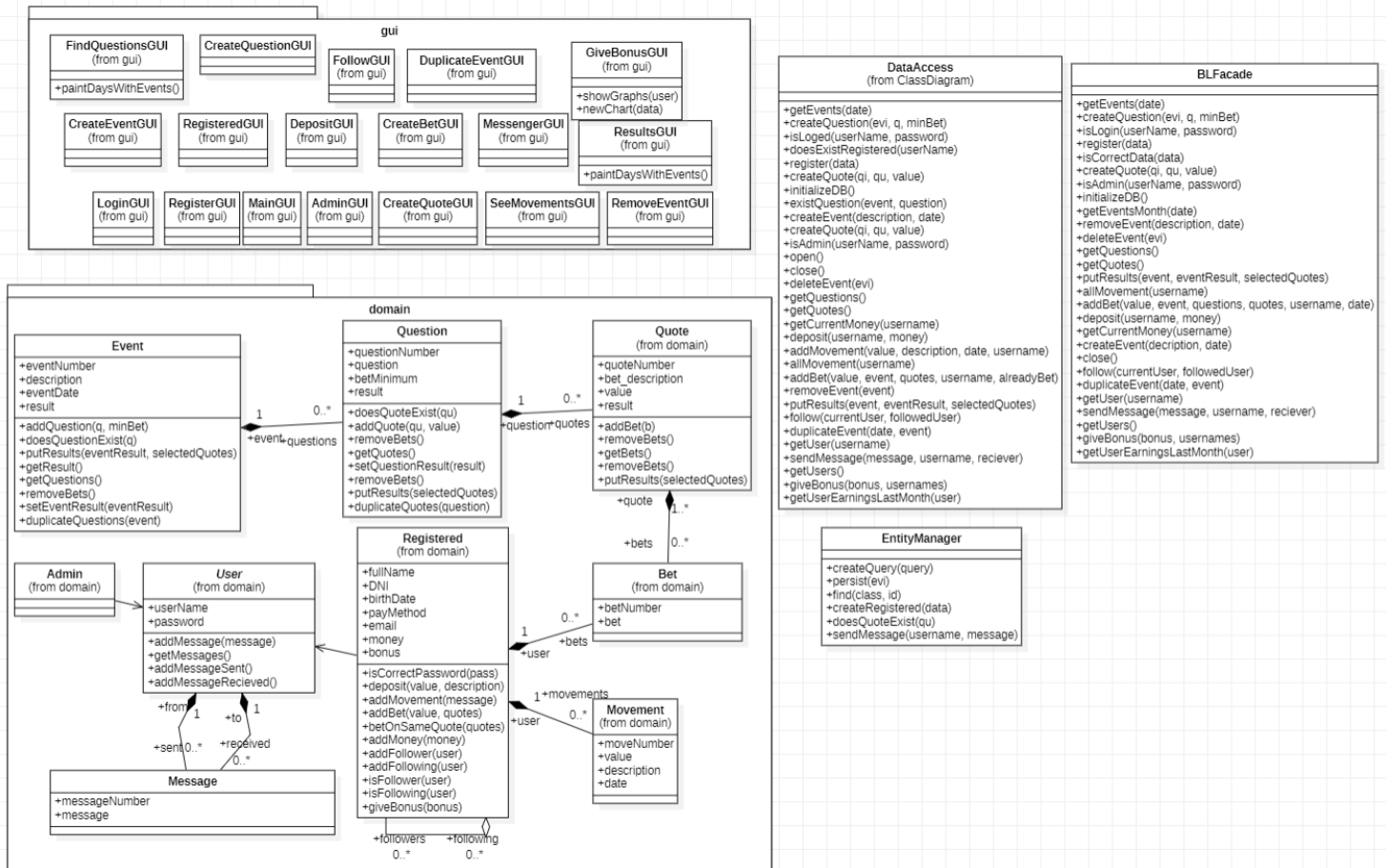
Lehenik eta behin, GUI-an bi zerrenda agertzen dira; jasotako mezuak eta bidalitako mezuak, datu-basetik jasotakoak.

Mezua bidaltzeko erabiltzailearen username idatzi behar da, erregistratuek bakarrik administraitzailearekin komunikatu daitezke eta alderantziz. Azkenik mezua idazten da.

Hartzailea existitzen bada, mezu berria jasotako mezuetan sartzen da. Mezua bidali duen erabiltzailearen bidalitako zerrenda ere eguneratzen da.

Bukatzeko, bi erabiltzaileen informazioa datu-basean eguneratzen da.

4.2 Klase diagrama



5. Izandako arazo nagusiak

1. Iterazioa

CreateEventGUI klasean, gertaera berriak sortzean aplikazioan ez ziren ikusten nahiz eta datu-basean sartu. Konpontzeko gertaera berriaren dataren ordua 00:00ra aldatu dugu UtilDate.trim() metodoarekin.

CreateQuoteGUI klasean gertaerarik gabeko data bat aukeratzean, exception bat altxatzen zen galderen ComboBox-agalderak kargatzen saiatzen zelako, hau gertatzen zen gertaeren ComboBox-ean aukeratutako gertaera null zelako. Baldintza bat jarri dugu galderak gertaera null ez denean bakarrik kargatzeko.

DataAccess klaseko createQuote() metodoak hasieran arazoak ematen zizkigun. Kuota sartzean, galdera gertaeraren listatik ezabatzen eta berriro gehitzen saiatzean, datu-baseak ez zigun uzten. Konpontzeko, galdera listatik zuzenean editatu dugu kuota bere listan sartuz.

Aurreko errorea konpondu ondoren, beste errore bat sortu genuela konturatu ginen. Kuota sortzeko lehioan, kuota bat sortzean eta berriro sortzen saiatzean, berriro sartzea uzten zuen aplikazioak nahiz eta berdinak izan. Hau konpontzeko, QuoteAlreadyExist exception altzatzeko baldintza aldatu genuen, comboBox-ean aukeratutako galderaren(berriro aukeratu arte eguneratzen ez dena) kuota bektorea konprobatu ordez, zuzenean gertaeraren galdera bektoreko dagokion posiziotik hartu genuen galdera (datu-basean eguneratuta dagoena).

Hau ondo funtzionatzen zuen, baina geroago metodo askoz sinpleago bat aurkitu genuen hau guztia egiteko. Gerteraren galdera bektoretik galderahartu ordez, parametrotik jasotako galderaren zenbakia hartu eta zuzenean find()-ekin bilatzen dugu. Horrela, galderari zuzenean kuota gehitzen diogu eta dena ondo eguneratzen da bektorerik ukitu gabe.

2. Iterazioa

Web servicean, BusinessLogicServer erabilitako funtzioak ezin zuten izen berdina izan main funtzioarekin. Izenak aldatuz konpondu dugu arazoa.

SeeMovementGUI-ean dirua erakusteko funtzioa ez zuen erakusten. Nahiz eta eraikitzailearen barruan egon, ez zuen funtzioa deitzen. Hau konpontzeko, add windowlistener akzioa erabili behar izan dugu.

DepositGUI-ean SeeMovement arazo berdina izan genuen, baina deposit funtzioarekin. DepositGUI-ean dirua txertatzen duzuean, ez zuen dirua aldatzen.

RemoveEventGUI-ean gertaerarekin erlazionatutako klase guztiak ez zuen ezabatzen. Klase horietan persist txertatu genuen. Ondorioz, ordezkatu genuen all-rekin. Gainera ahaztu egin zitzaigun cascade zehaztea.

RemoveEventGUI-ean gertaera ezabatzen zenean, apostatu zutenei dirua ez zuen bultatzen. Arazoa zen, gure funtzioa dataAccess ean zuen jasotzen quoten lista. Hasieran Set moduan jasotzen zuen. Ondorioz, vector moduan ordezkatu genuen.

3. Iterazioa

Send Message, mezu bat bidaltzean, datu-basean behar bezala gordetzen zuen. Hala ere, mezuak null gisa agertzen ziren. BusinessLogicServer erabiltzeagatik sortzen zen arazo hori, datu-basean null gisa gordetzen duena, baina null horren barruan, mezuaren informazio guztia agertzen zen. Arazo hori konpontzeko, User klasea, beste User-rekin jasotzeko atributua string moduan ordezkatu genuen.

Give bonus funtzionalitatearen inplementazioan, datu basetik erabiltzaileen datuak atzitzeko bi bektorez osatutako matrize bat itzultzea nahi genuen, baino matrize hutsa itzultzen zuen. Honen arrazoia web zerbitzua zen, beraz, datu base atzipenak alderatu behar izan ditugu datu base irakurketa egokia lortzeko.

Apostu anitzak funtzionalitatean, jarraitzaileak kontuan hartu behar genituen. Horrertarako erabilpen kasu honen inplementazioan aldaketak egin behar izan ditugu, errekursibitatea erabiliz. Hala ere, baliteke bi erabiltzaile (edo gehiago) haien artean jarraitu izana. Kasu honetan, begizta infinitu bat gertatuko zen errekursibitatearengatik. Horretarako, jadanik apostua sortu duten erabiltzaileen zerrenda bat inplementatu behar izan dugu, errepikapenak ekiditeko.

6. Ondorioak

Programazio-proiektu honen esperientzian, lankidetza handiko talde motibatu batean lan egiteko aukera izan dugu. Nahiz eta taldekide batek uko egin, aurrera egin dugu. Gure helburua apustuen web aplikazio bat garatzea zen. Proiektuan zehar, marroi izeneko zeregin desafiatsuak esleitu zizkiguten, gure jakintasuna bultzatuz, ondoren ikasitakoa proiektuan aplikatzeko.

PBL metodologiak hezkuntza-esperientzia bat eskintzen digu, non arazo praktikoak konponduz eta proiektu esanguratsuak eginez ezagutzak eta trebetasunak eskuratzen dituzten.

Klasearen dinamikak asko gustatu zaizkigu. Proiektuarekin aurrera eramateko askatasuna eman digu. Proiektuari aurrera egiteko behar genuen eduki guztia eskuratu genuen. Ariketek ikuspegi zehatzagoa ematen ziguten diseinuak planteatzerakoan. Laborategiak, ariketak bezala, onuragarriak izan dira.

StarUML tresna indartsua eta sistemak modelatzeko asko erabiltzen dela izan arren, erabiltzaile-interfazea nahasia eta inutizio-gutxikoa izan daiteke. Softwarean erabilgarri daude funtzio eta aukera ugariak asko harrapa ditzakete esperientziarik gabeko erabiltzaileak (gu kasu), eta zaildu egin dezakete haien ikaskuntza eta erabilera eraginkorra.

Hurrengo urteko ikasleentzako emango genukeen gomendioa hau izango litzateke: Lankide onak aukeratu behar direnean, garrantzitsua da lankidetza eta baterako arrakasta sustatuko duten funtsezko ezaugarriak bilatzea. Komunikazioa bultzatzen dutenak, errespetuzkoak eta taldean lan egiteko prest dauden pertsonak.

Bestalde, atzera begiratzeko eta egindako akatsak aztertzeke gai izatea oso garrantzitsua da, iteraziotan banandutako proiektu honetan behin eta berriz arituko zarela egindako akatsen aurrean soluzioak bilatzen eta pixkanaka proiektua aurrera eramaten.