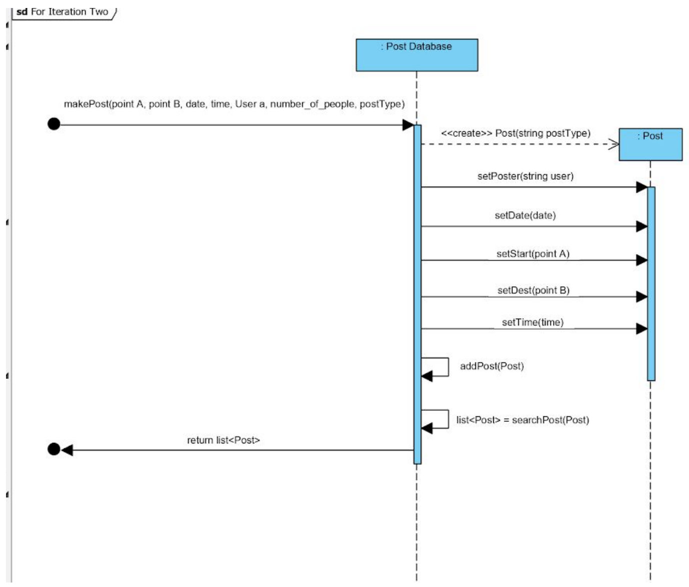
Iteration 2

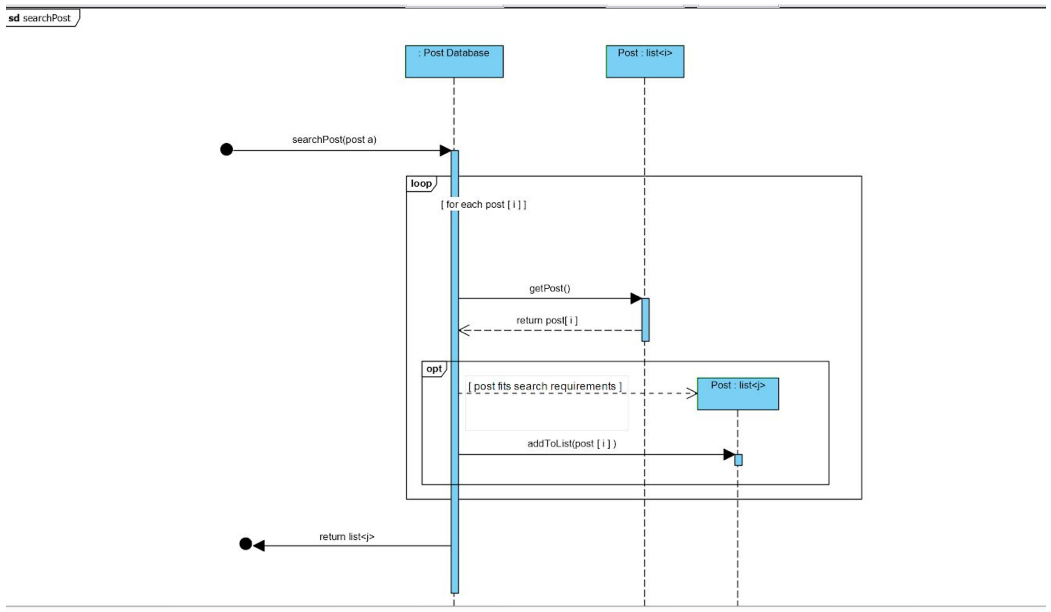
18 sequence diagrams:

1. makePost()
2. searchPost()
3. reportPost()
4. reportUser()
5. storeSurvey()
6. banUser()
7. deleteAccount()
8. readData()
9. requestPostAcceptance()
10. queryDatabase()
11. deletePost()
12. generateStatusReport()
13. createAccount()
14. updatePost()
15. updateUser()
16. sendFriendReq()
17. addFriend()
18. acceptRequest()

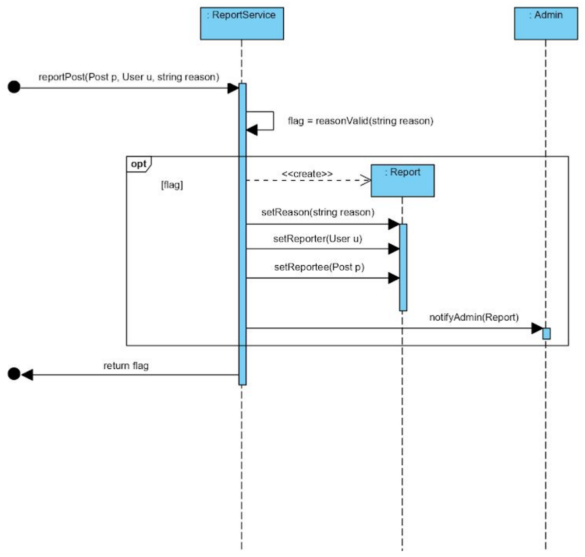
1. MakePost()


sd For Iteration Two
: Post Database
: Post
makePost(point A, point B, date, time, User a, number_of_people, postType)
<<create>> Post(string postType)
setPoster(string user)
setDate(date)
setStart(point A)
setDest(point B)
setTime(time)
addPost(Post)
list<Post> = searchPost(Post)
return list<Post>

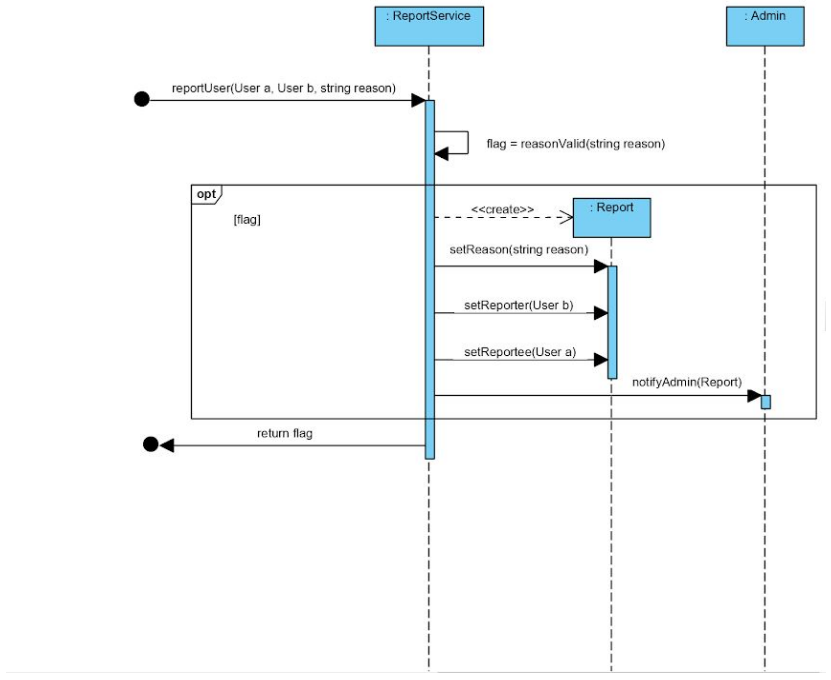
2.SearchPost()



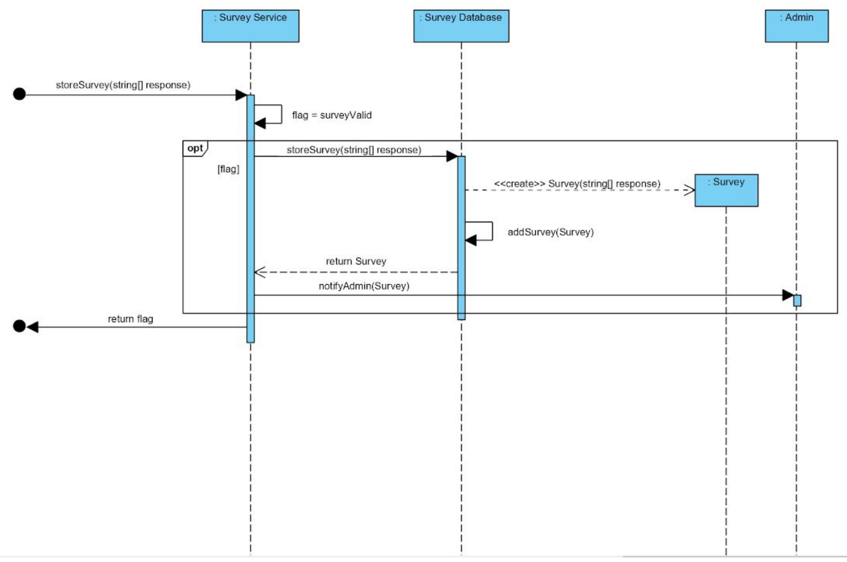
3. reportPost()



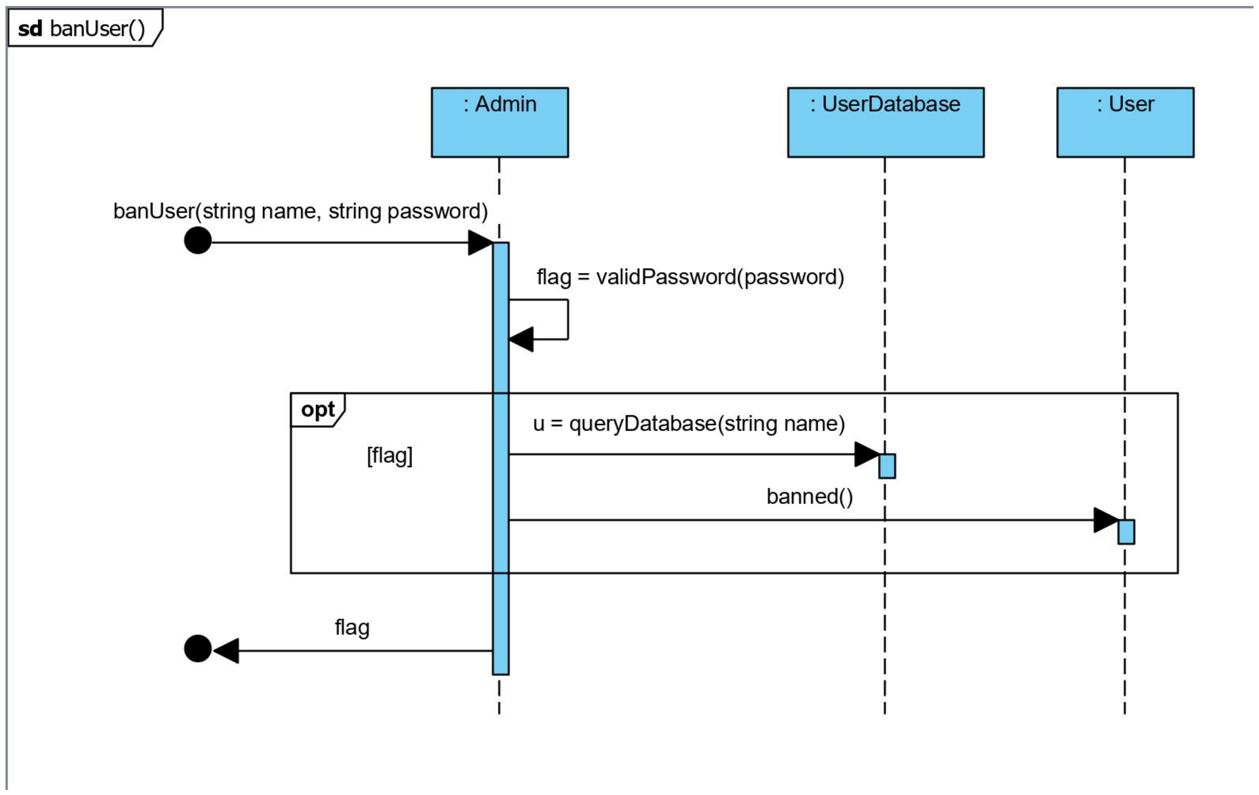
4. reportUser()



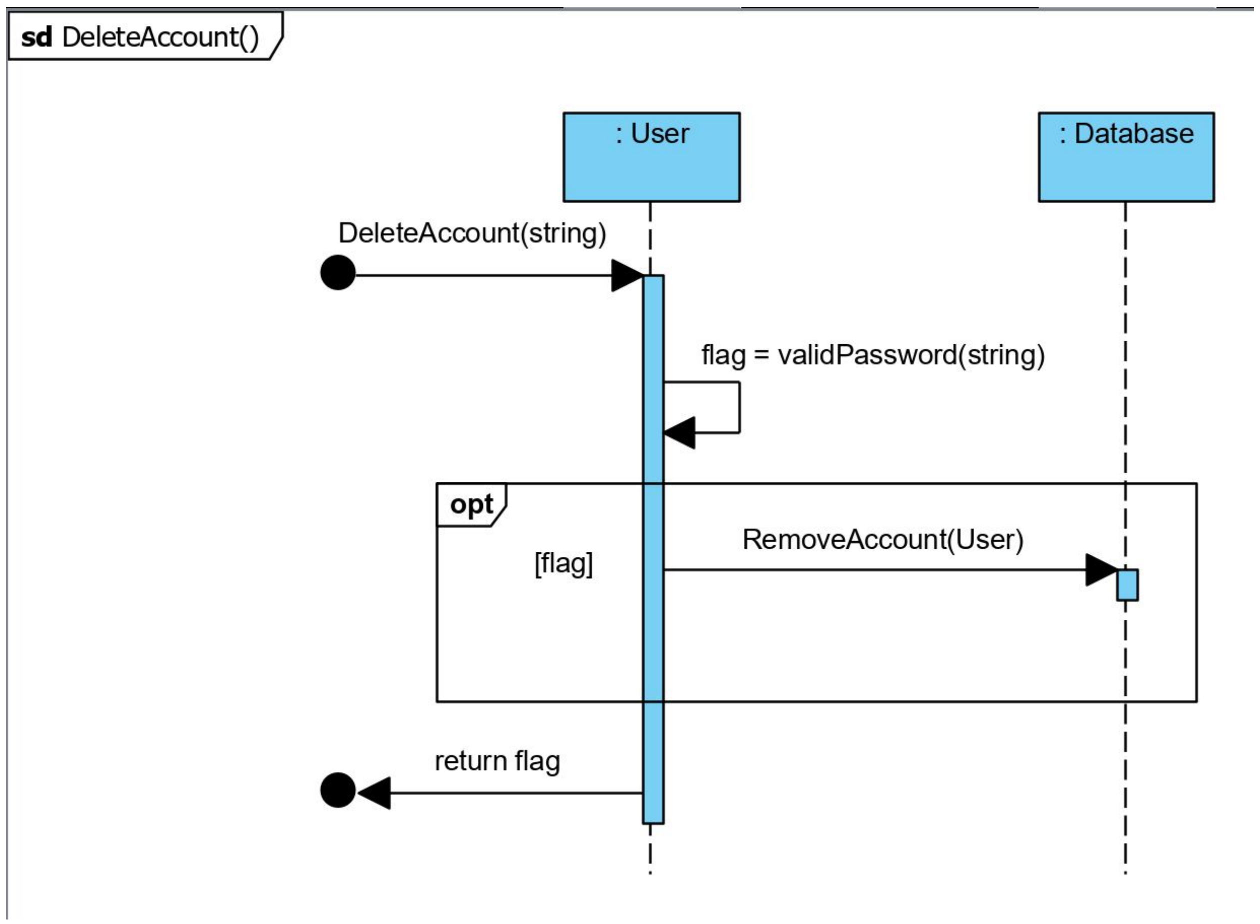
5. storeSurvey()



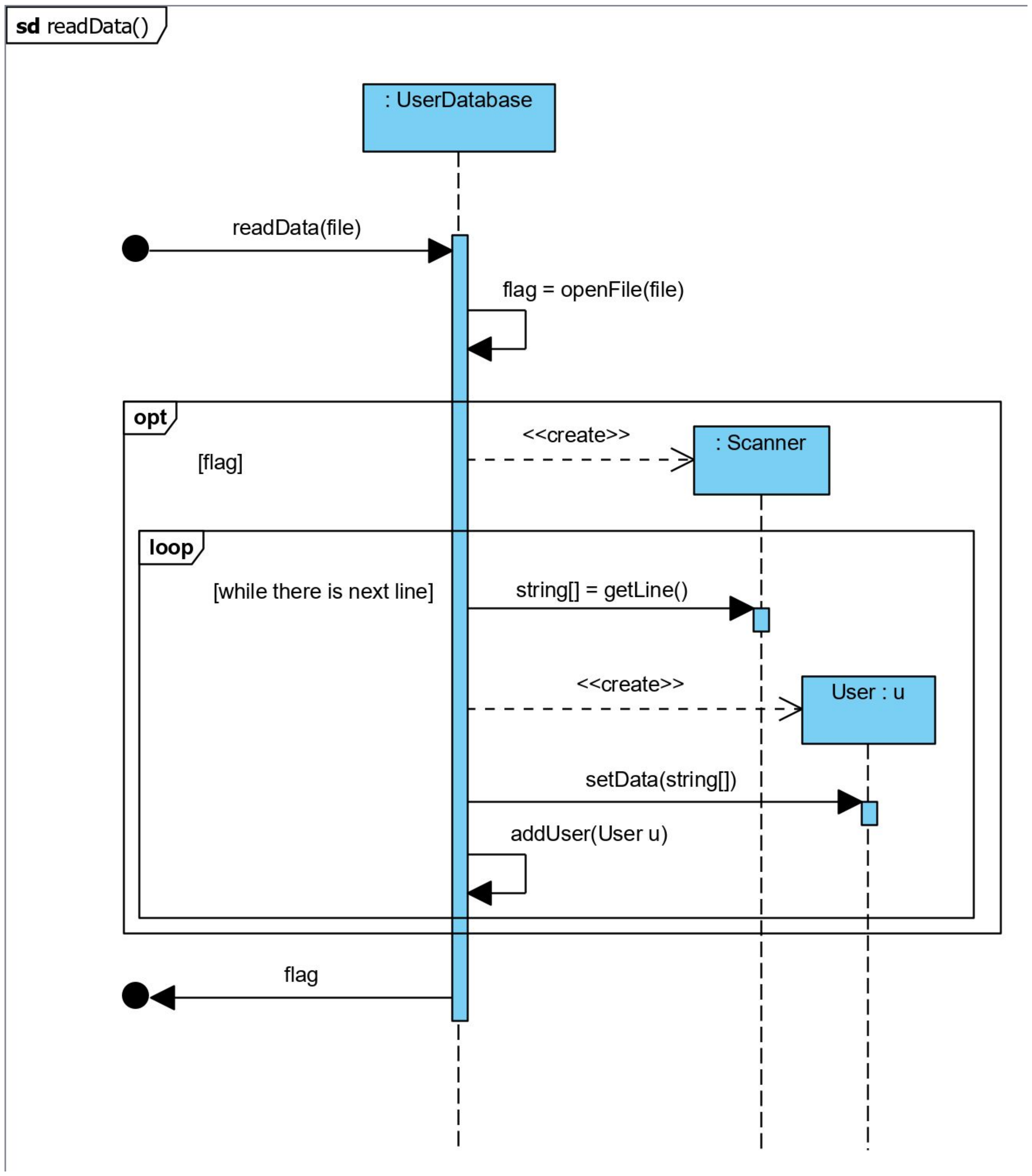
6. banUser()


sd banUser()
: Admin
: UserDatabase
: User
banUser(string name, string password)
flag = validPassword(password)
opt
u = queryDatabase(string name)
[flag]
banned()
flag

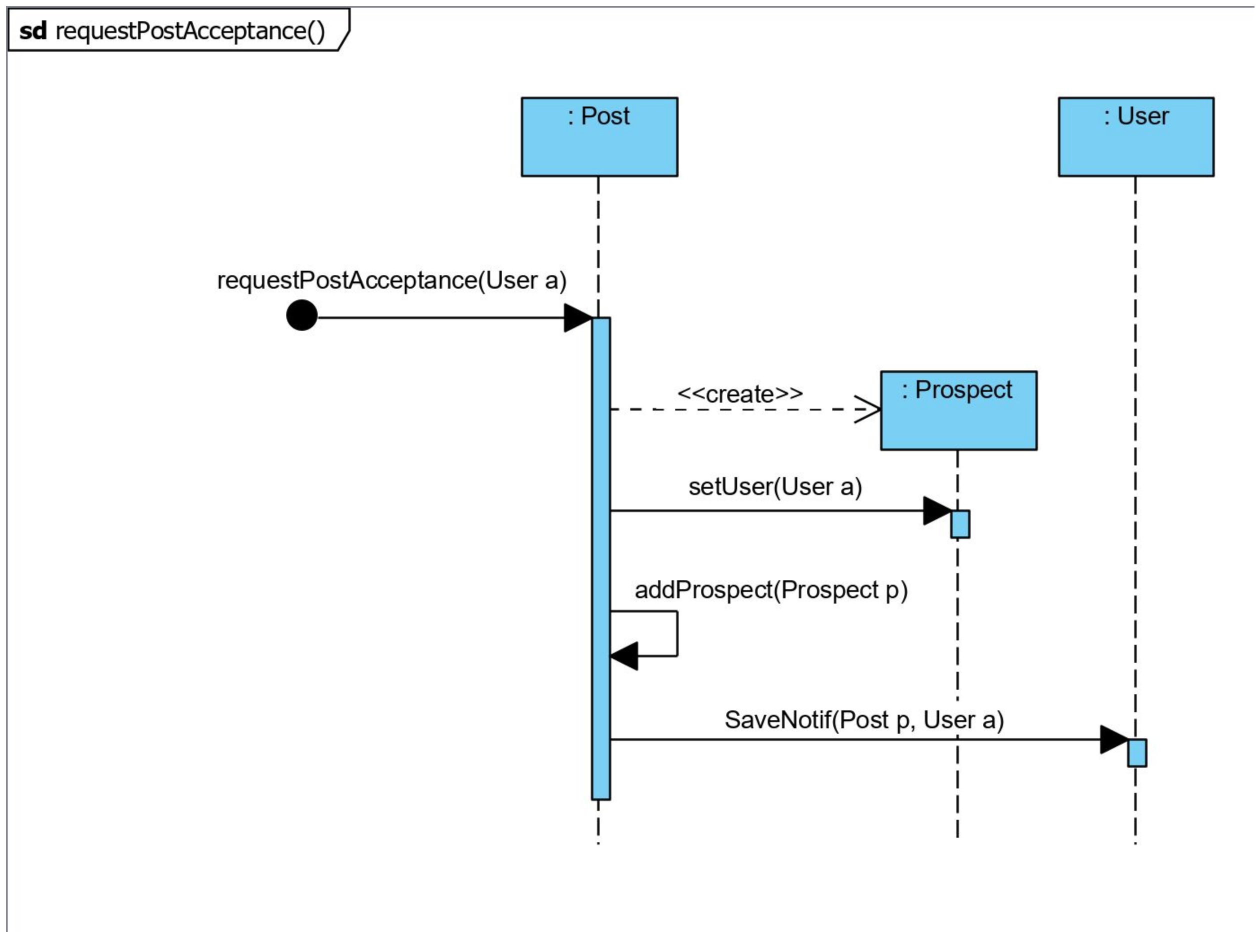
7. DeleteAccount()



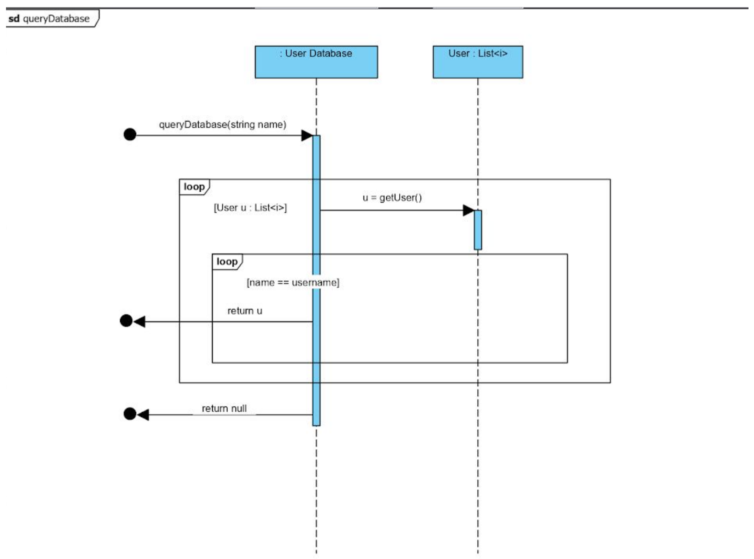
8. readData()


sd readData()
: UserDatabase
readData(file)
flag = openFile(file)
opt
[flag]
<<create>>
: Scanner
loop
[while there is next line]
string[] = getLine()
<<create>>
User : u
setData(string[])
addUser(User u)
flag

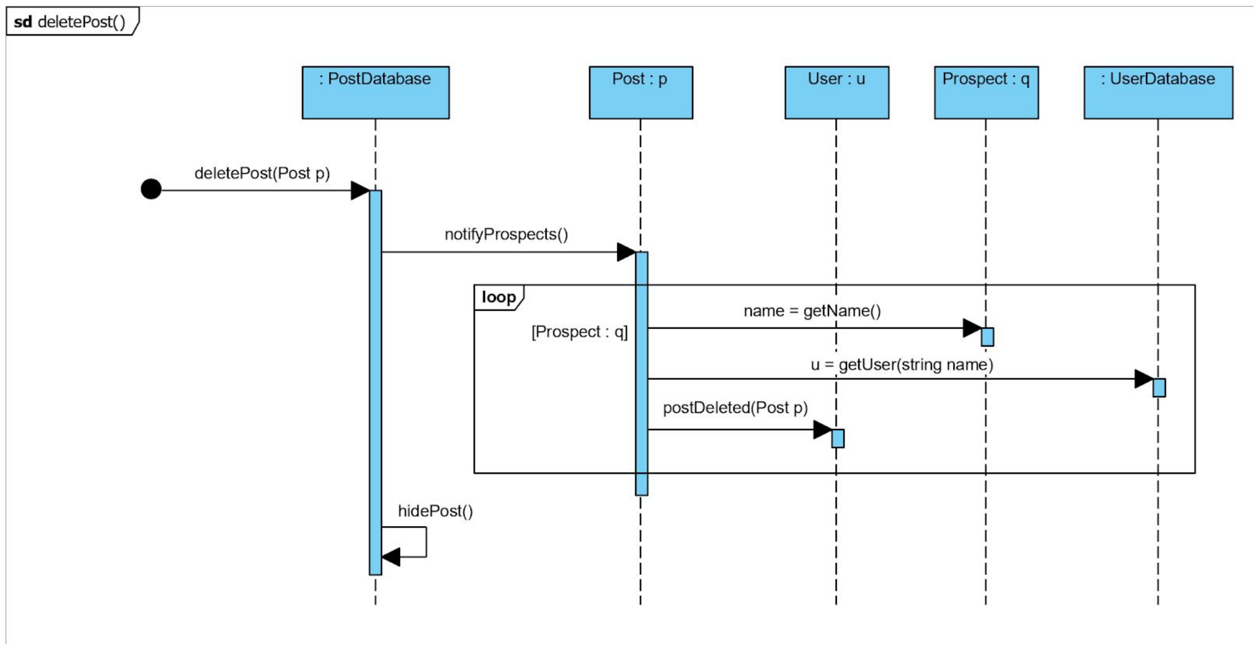
9. requestPostAccpetance()


sd requestPostAcceptance()
: Post
: User
requestPostAcceptance(User a)
<<create>>
: Prospect
setUser(User a)
addProspect(Prospect p)
SaveNotif(Post p, User a)

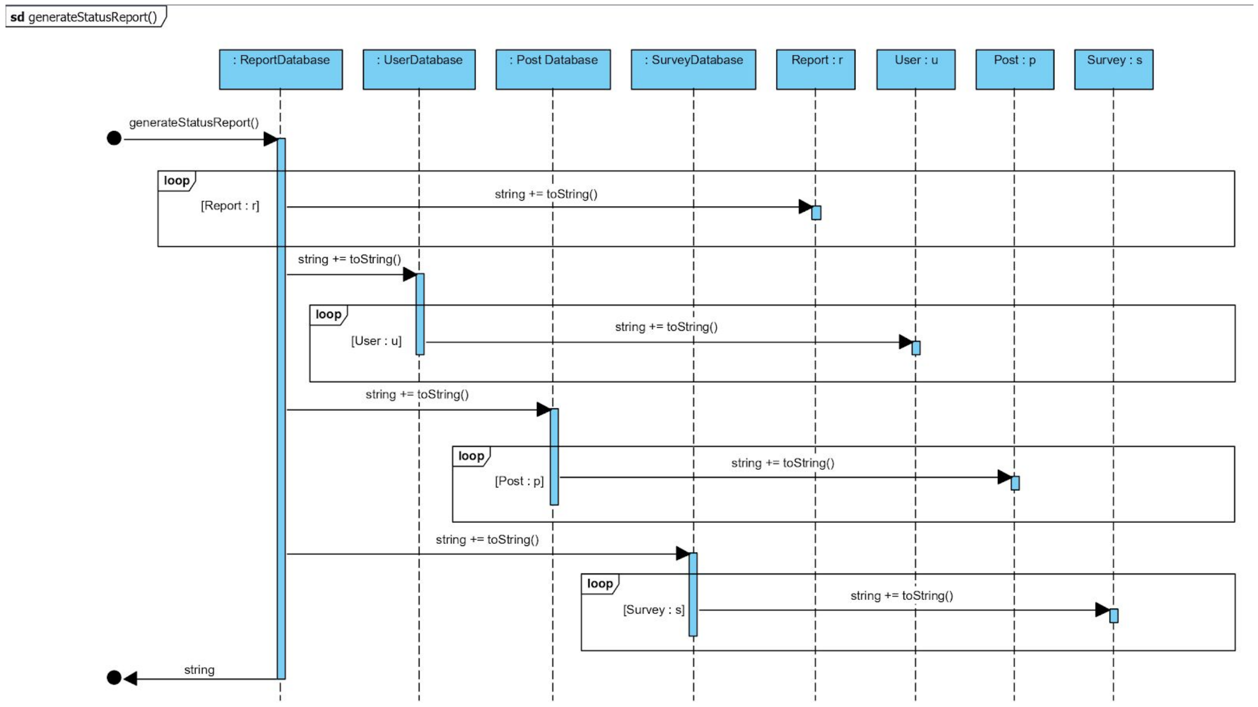
10.queryDatabase()



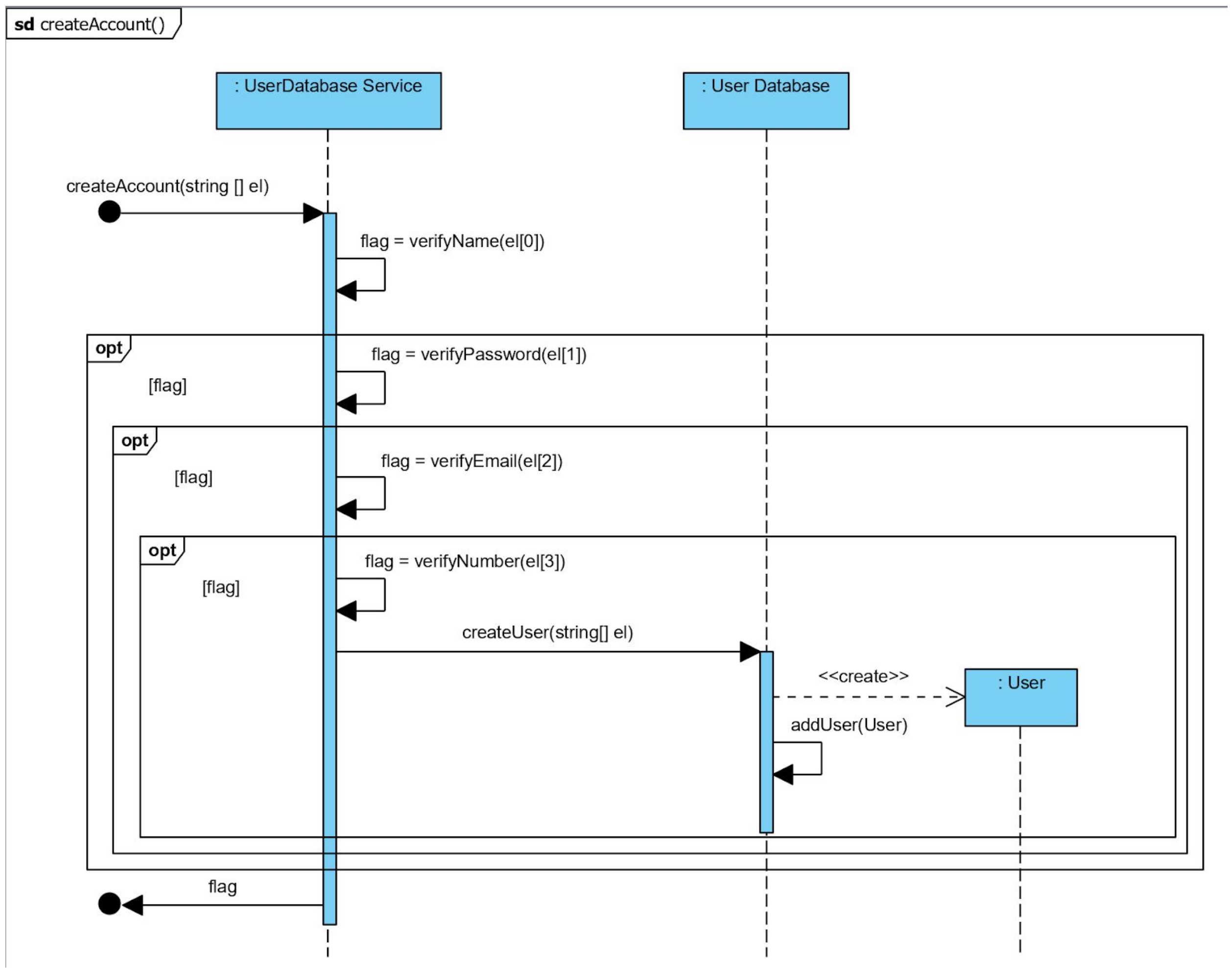
11. deletePost()



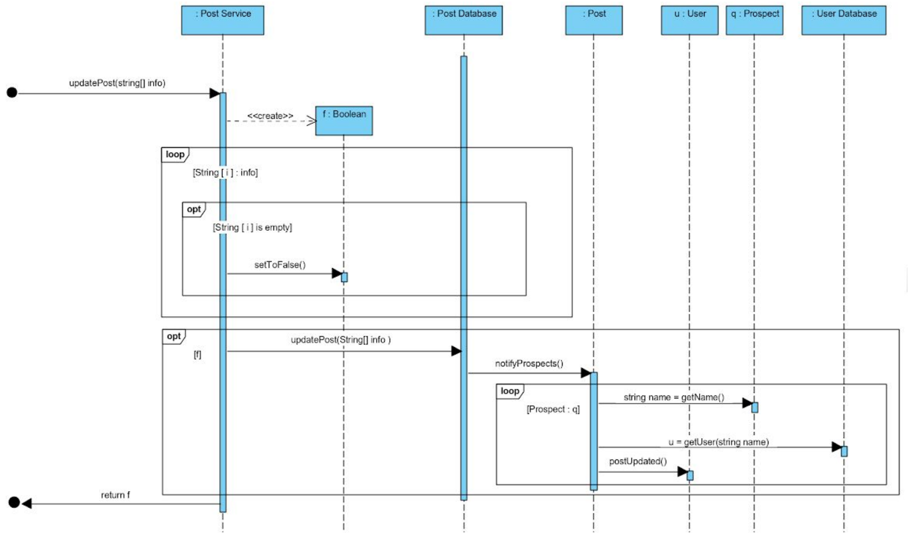
12. generateStatusReport()



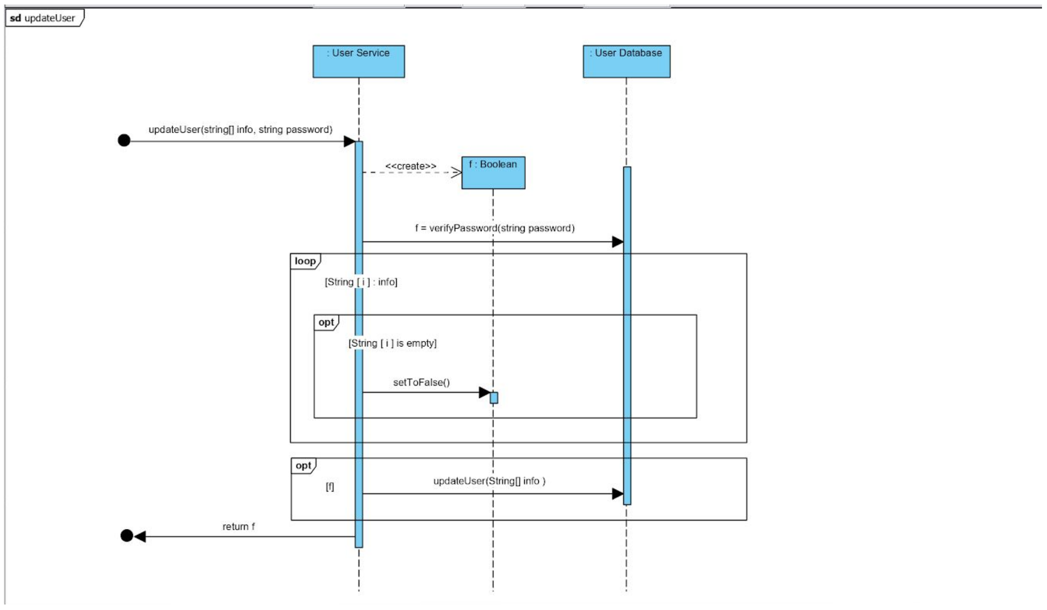
13. createAccount()



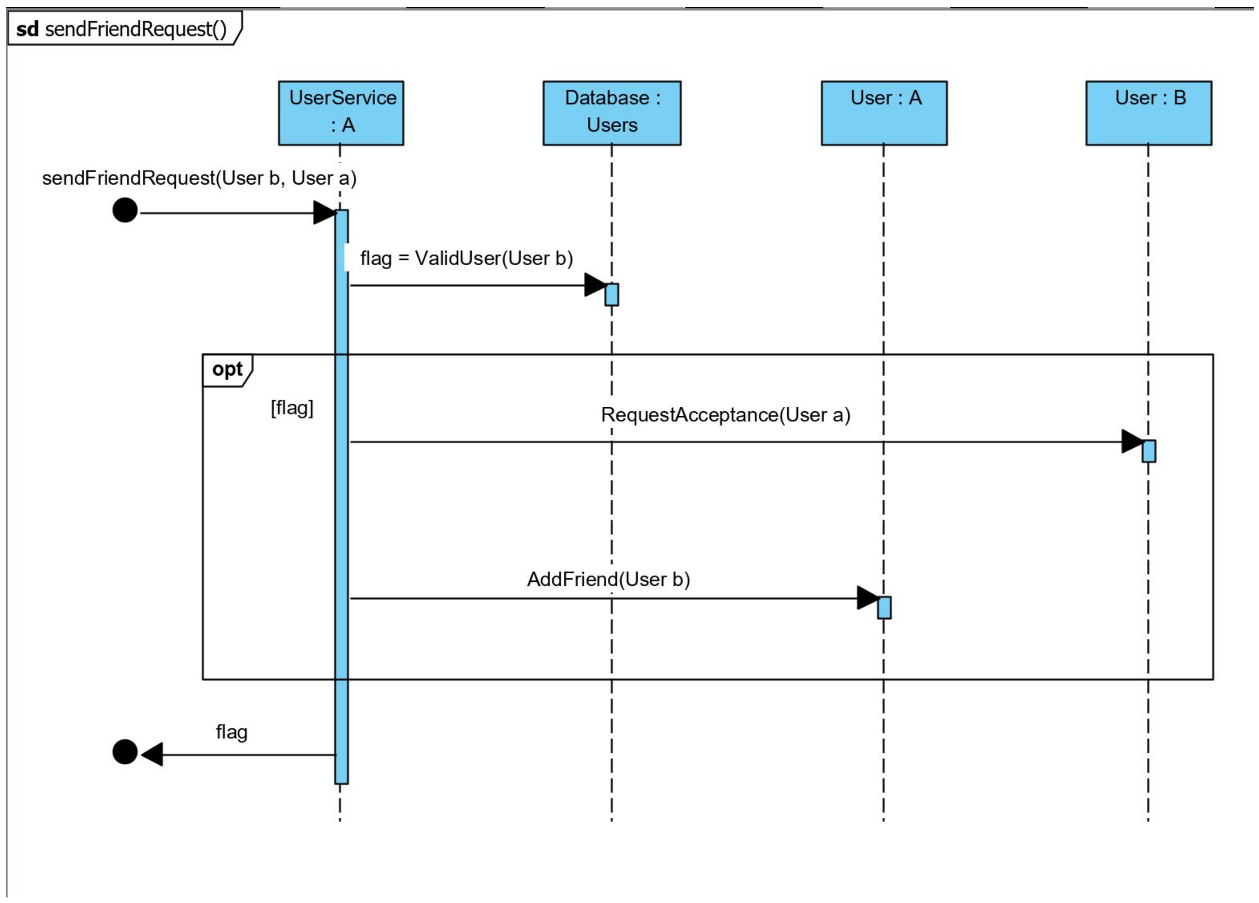
14. updatePost()



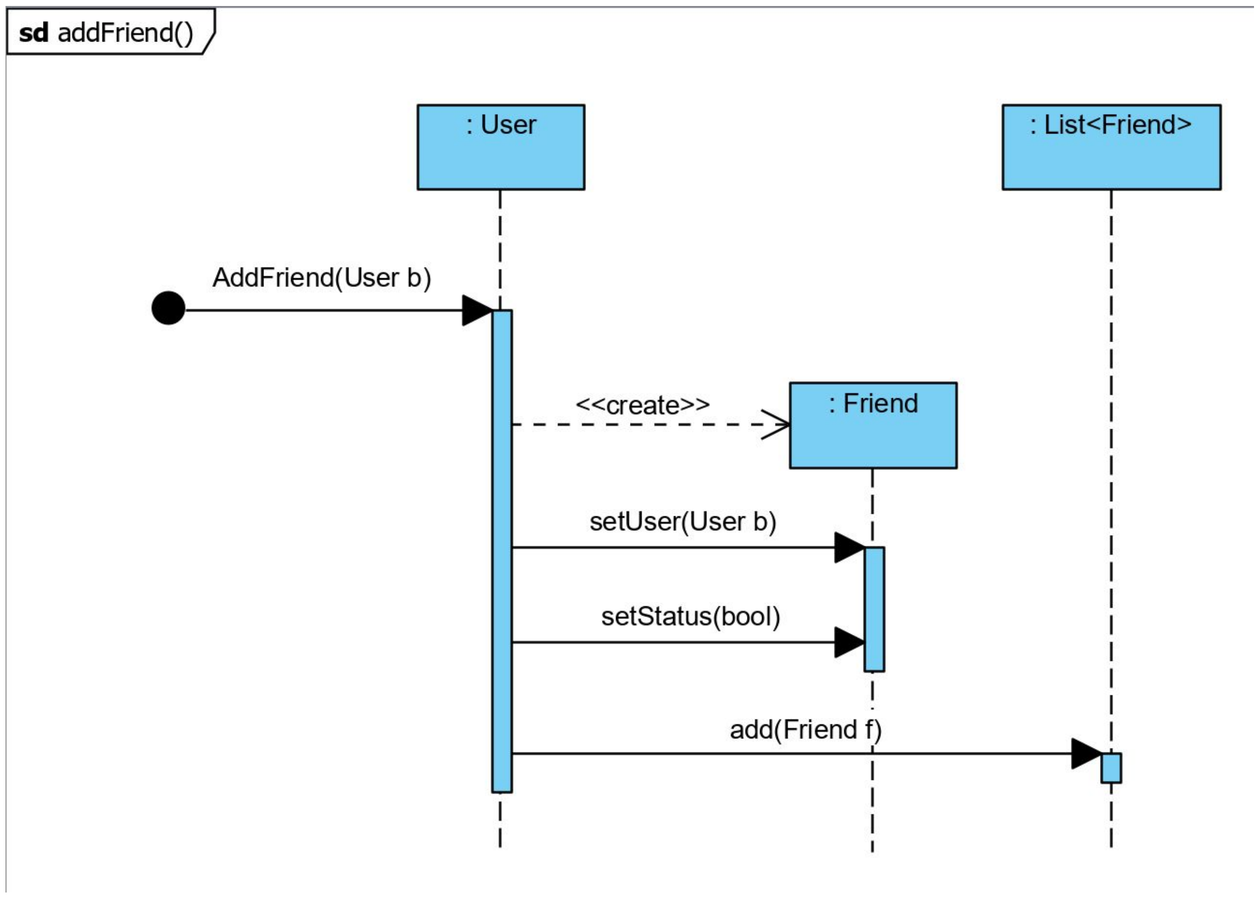
15. updateUser()



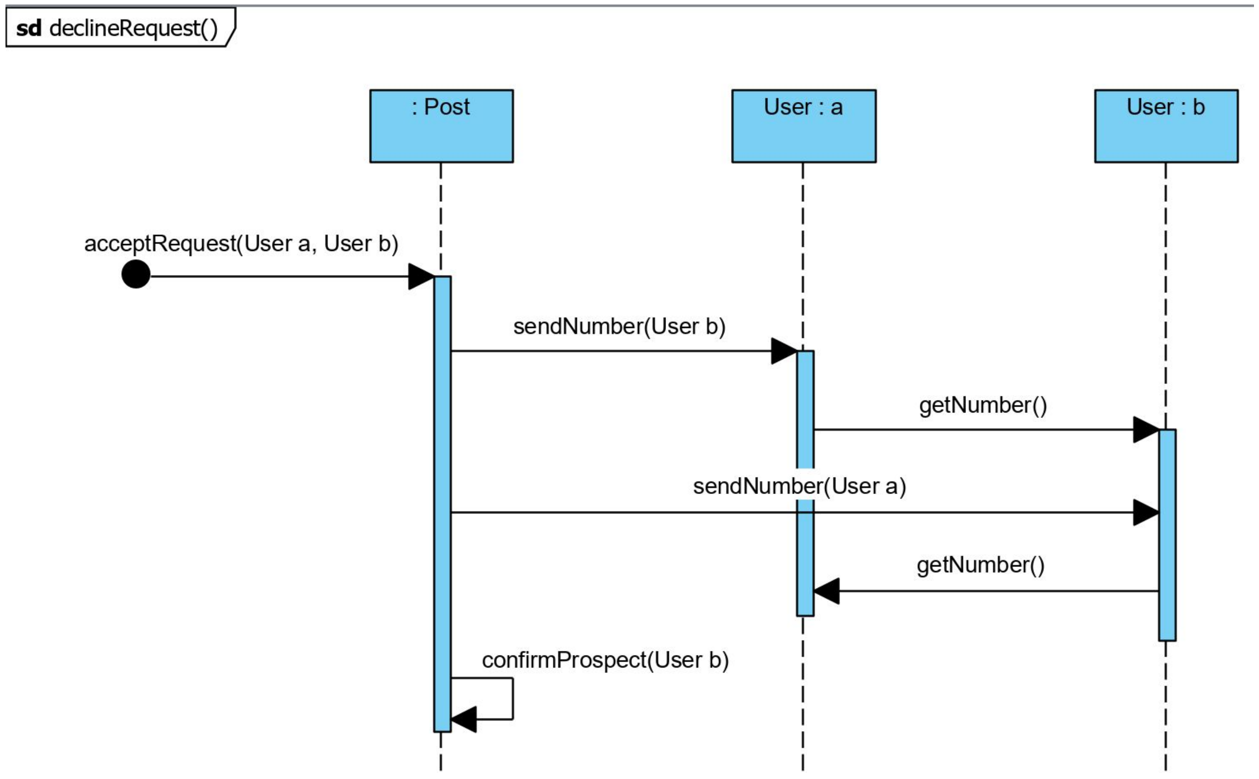
16. sendFriendRequest()



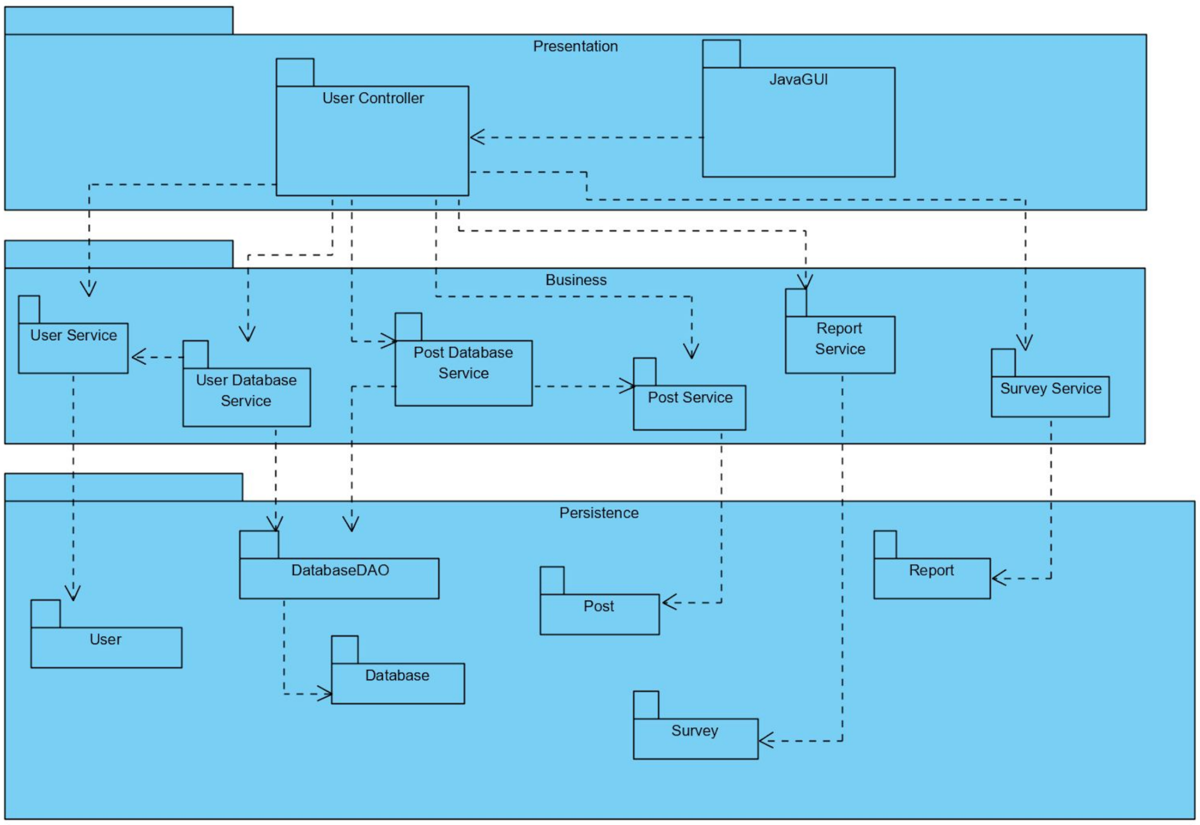
17. addFriend()



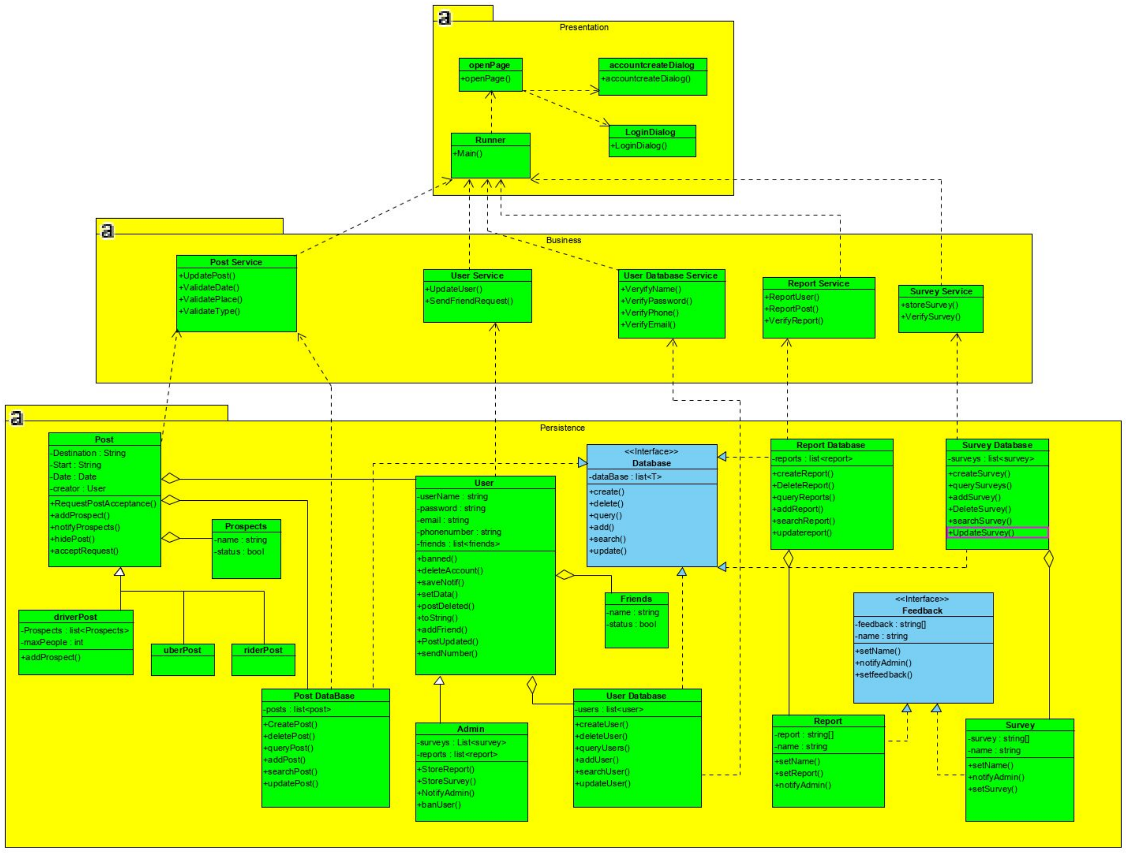
18. acceptRequest()



Package Diagram



Design Diagram



GRASP:

Class: Database

readData

Operation Rationale:

The readData has the creator pattern since it creates the database information.

Class: Post Database

makePost

Operation Rationale:

The makePost has the GRASP patterns of Creator since it creates new Post instances. The operation also has the information expert pattern since the Post Database records stores and knows about all the created posts.

searchPost

Operation Rationale:

The searchPost has the Information expert pattern since it knows about the posts that fit the requirement that the search needs.

Class: Post Service

updatePost()

Operation Rationale:

The operation has low coupling since the service is a buffer between the multiple classes involved so that the classes wouldn't have to know about each other.

Class: Report Service

reportPost

Operation Rationale:

The operation also has the creator pattern since the it creates a report instance and sends it to the admin.

Class: User Database Service

createAccount()

Operation Rationale:

The operation has the controller pattern since it receives the information from the UI inputted by the user. The operation also has the creator pattern where it creates a new User instance.

updateUser()

Operation Rationale:

The operation has low coupling since the user database is buffered by the service. The operation also has the controller pattern since the operation decides if the input is valid or not and directs the operations done after that depending on the validity of the input.

queryDatabase

Operation Rationale:

The queryDatabase has the Information expert patterns where the operation knows about the user that the operation is searching for.

Class: Survey Database

storeSurvey

Operation Rationale:

The operation has the controller pattern where the input is checked for its validity and the operation is directed afterward. The operation also has the pattern of the creator for creating an instance of the Survey.

Class: Report Service

reportUser()

Operation Rationale:

The operation has the controller pattern for getting input directly from the UI. The operation also has the high cohesion pattern since it’s only concerned with making a report for the admin.

Class: User

deleteAccount()

Operation Rationale:

The operation has the controller pattern because it gets confirmation from the User, then deletes the account if the inputs are valid.

sendFriendReq()

Operation Rationale:

The operation supports high cohesion since it only deals with users and nothing else.

addFriend()

Operation Rationale:

The operation has the creator pattern since it creates a friend instance.

requestPostAcceptance()

Operation Rationale:

This operation has the creator pattern since it creates a prospect instance to be added to the post prospects list

deletePost()

Operation Rationale:

This operation supports the Controller pattern because it gets information from the UI, then deletes the post if the inputs are valid.

acceptRequest()

Operation Rationale:

This operation supports high cohesion since it only deals with User to User interaction

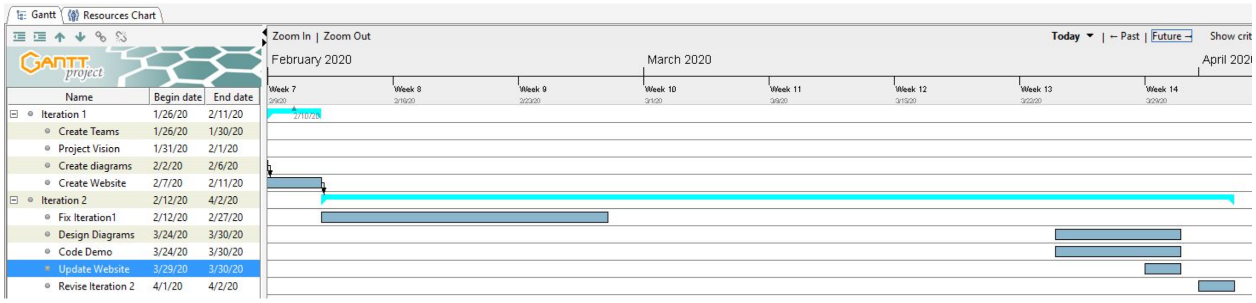
Class: Admin

banUser()

Operation Rationale:

The operation has the high cohesion pattern, because it only focuses on User status.

Gantt Diagram



Name	Begin date	End date
Iteration 1	1/26/20	2/11/20
Create Teams	1/26/20	1/30/20
Project Vision	1/31/20	2/1/20
Create diagrams	2/2/20	2/6/20
Create Website	2/7/20	2/11/20
Iteration 2	2/12/20	4/2/20
Fix Iteration1	2/12/20	2/27/20
Design Diagrams	3/24/20	3/30/20
Code Demo	3/24/20	3/30/20
Update Website	3/29/20	3/30/20
Revise Iteration 2	4/1/20	4/2/20

Test coverage:

Plan to cover:

- Posts
  - Test the makePost method, which creates a rider post if type is rider/ and driver if type rider.
  - Test the deletePost method, ensuring that a deleted post is marked as not visible
- Creating and removing accounts
  - Test the validateAccountInfoEntered method, which takes in the info provided from a user, and verifies the info entered is proper, such as making sure the input password is more than 7 characters
  - Test the deleteAccount method, which removes the user from the database

- Ensuring users can join rides successfully
  - Test the requestPostAcceptance method, which allows riders to enter a queue in a driver post which will allow them to take up a seat in the drivers post.
- Ensuring the logger works correctly
  - Testing that all major events such as account creation/deletion, successful logins are all properly logged at their appropriate levels, and exported to a log file.
- Ensuring the windows work as intended
  - Testing inside the Runner class to verify that all buttons pressed and window creation and destruction occurs as intended.
- Ensure database saves correctly
  - Testing the write method in both the user and post database. Verify is updating changes to external files, and when application is restarted verifying changes appear

- Ensure errors are handled correctly
  - Test all methods with unexpected input. Verify all methods return, exit, or throw an exception as intended
- Ensure edge cases are handled properly
  - Test all methods with input on extreme ends. Verify these methods return properly
- Creating database report
  - Test the generateStatusReport method. Verify it processes every member in their respective databases. Verify the output matches the contents of the database
- Retrieving surveys
  - Test the getSurveys method. Verify that every survey entered matches the output of the retrieved surveys.

We plan on testing all cases as thoroughly as possible.

Point distributions:

Joseph Yu	20/100
Joseph Perez	20/100
Joshua Huertas	20/100
Andrew Ammentorp	20/100
Leighton Glim	20/100

(adds to 100/100)

Updated Timecards:

Joseph Yu	53 hrs
Joseph Perez	54 hrs
Joshua Huertas	54 hrs
Andrew Ammentorp	46 hrs
Leighton Glim	47 hrs