# Low Level Design Document

Student Multi-Tool

---

12.010.2021

**Team Marvel**
Albert Toscano
Audrey Brio
Bradley Nickle
Joseph Cutri
Michael Kriesel (Team Leader)

**Technologies Used**

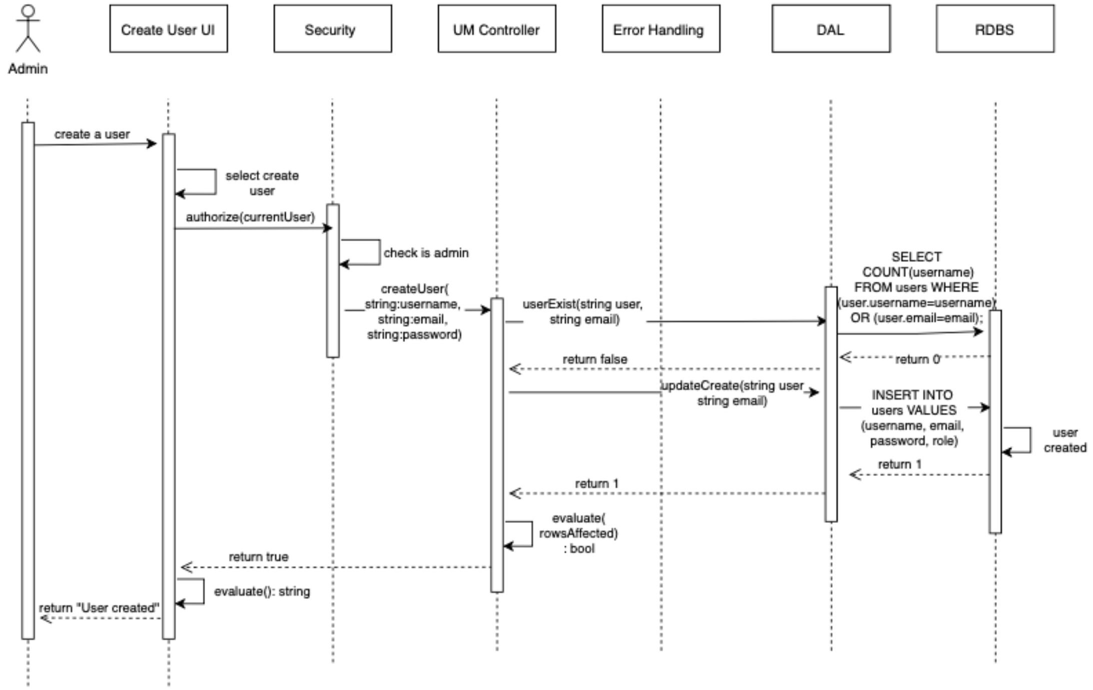
- C#
- .Net
- SQL Server 2019 Developer/Express Edition
- SQL Server Management
- Video Studio Code and Visual Studio Community

## UM

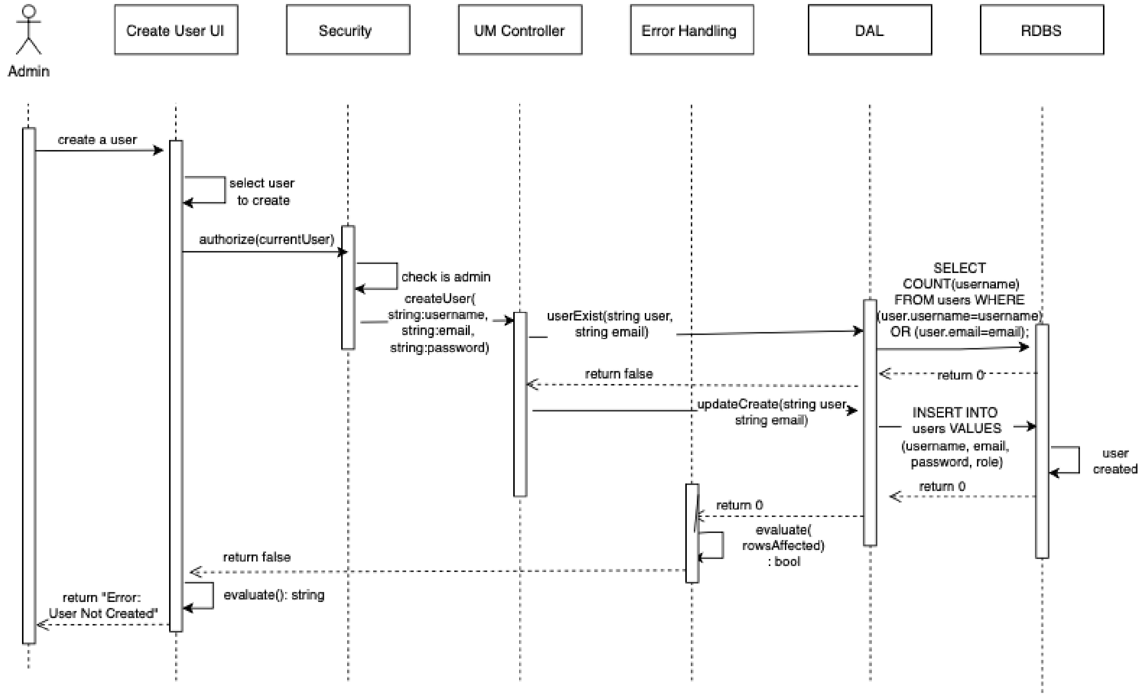
**Sequence Diagrams:**

- To be able to perform any of the user management functions, the user must be an admin (role must be admin) of the system
  - If user’s role is student, then they will not be authenticated and can therefore not perform any of the actions under user management
- Users must be logged into the system in order to perform any actions
- The sequence diagrams are under the assumption that the user is already logged into the system
- If a user's status is disabled, then they will not be able to log into the system nor use any of the other components or features.

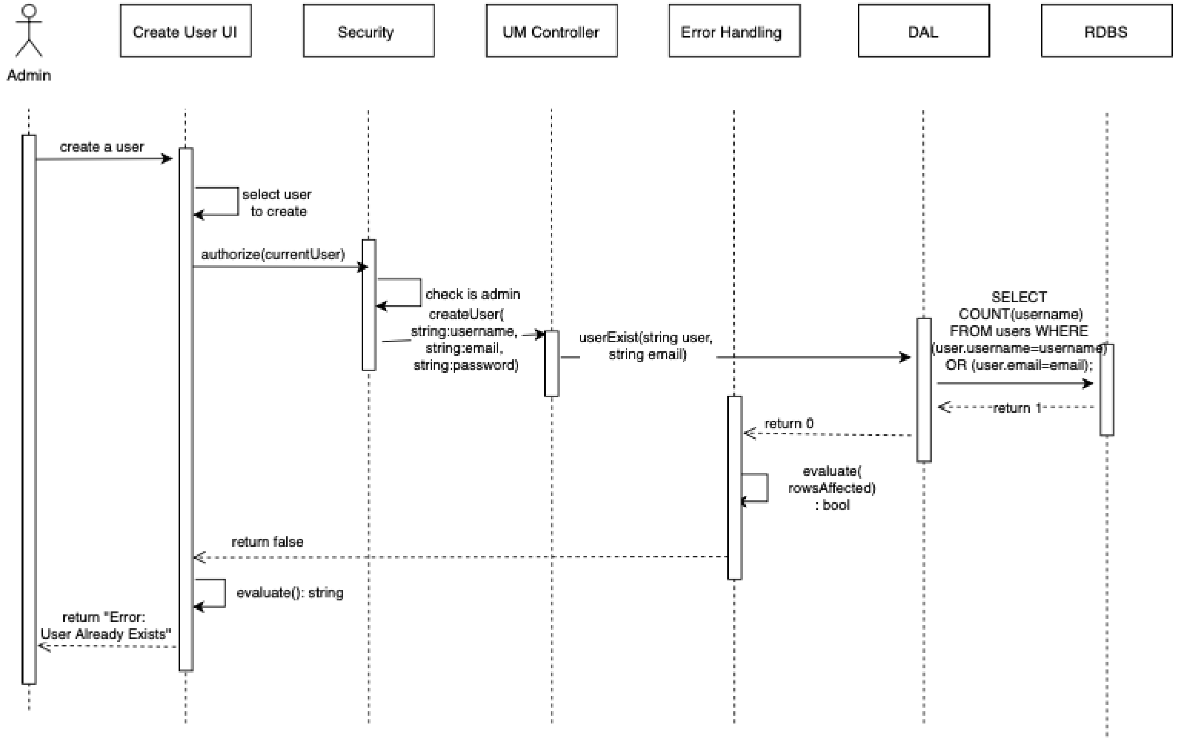
Create User (Success)


Admin
Create User UI
Security
UM Controller
Error Handling
DAL
RDBS
create a user
select create user
authorize(currentUser)
check is admin
createUser( string:username, string:email, string:password)
userExist(string user, string email)
SELECT COUNT(username) FROM users WHERE (user.username=username) OR (user.email=email);
return 0
return false
updateCreate(string user string email)
INSERT INTO users VALUES (username, email, password, role)
user created
return 1
return 1
evaluate( rowsAffected) : bool
return true
evaluate(): string
return "User created"

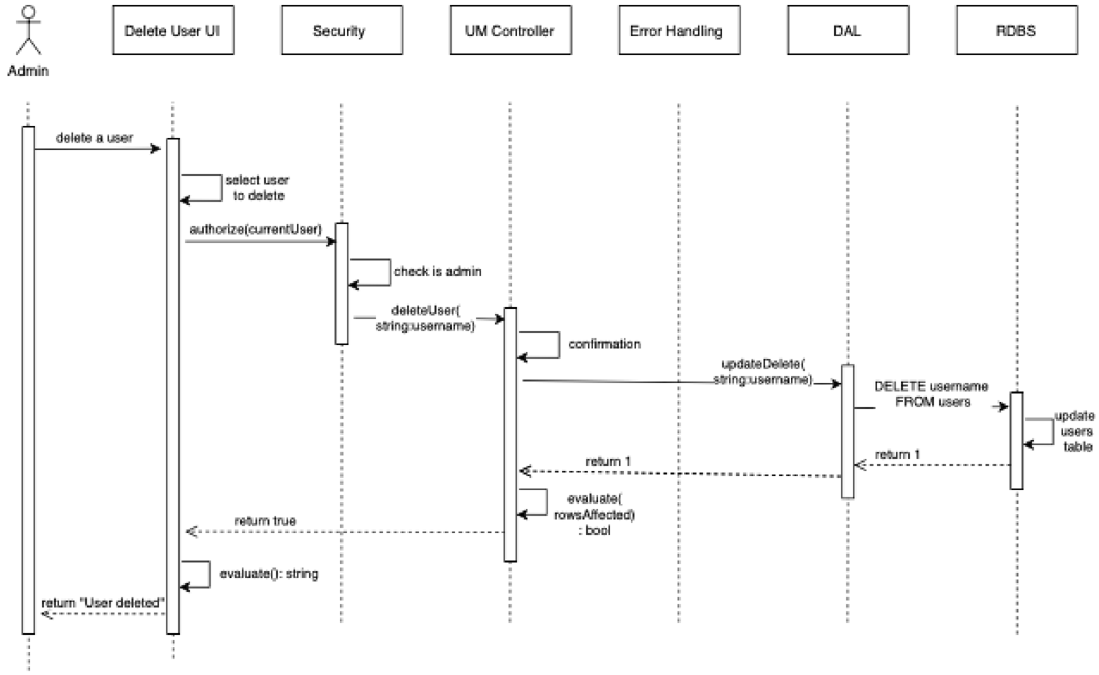
## Create User (Fail - user not created)



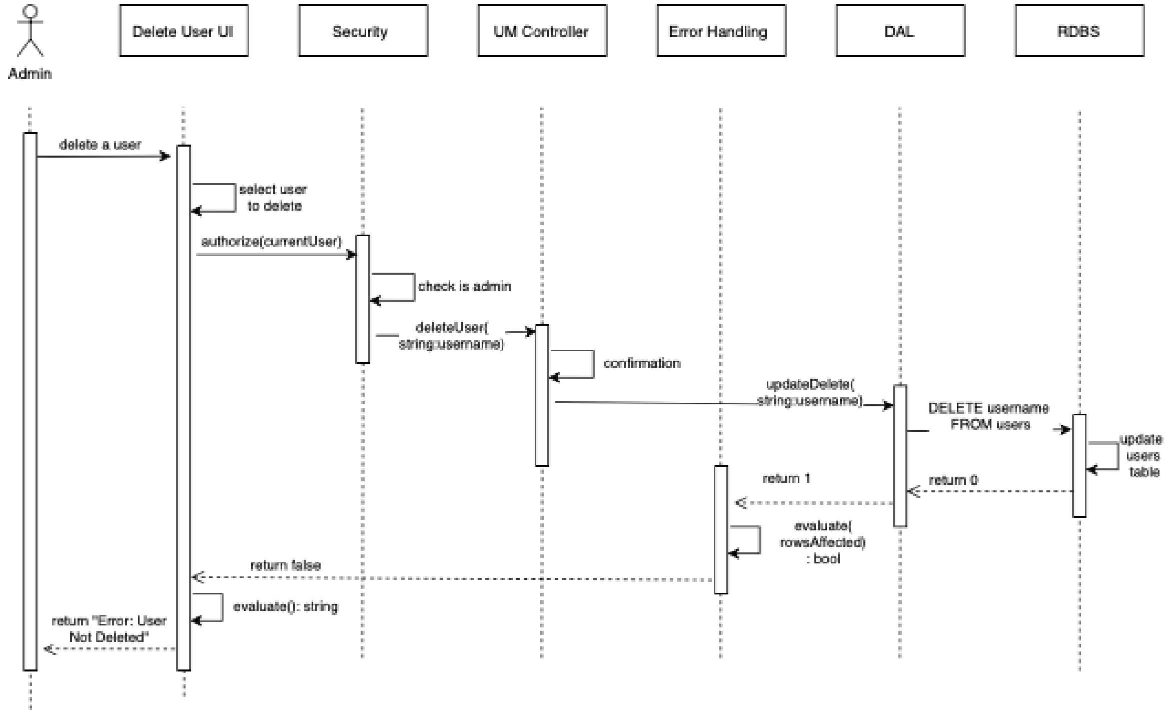
## Create User (Fail - user already exists)



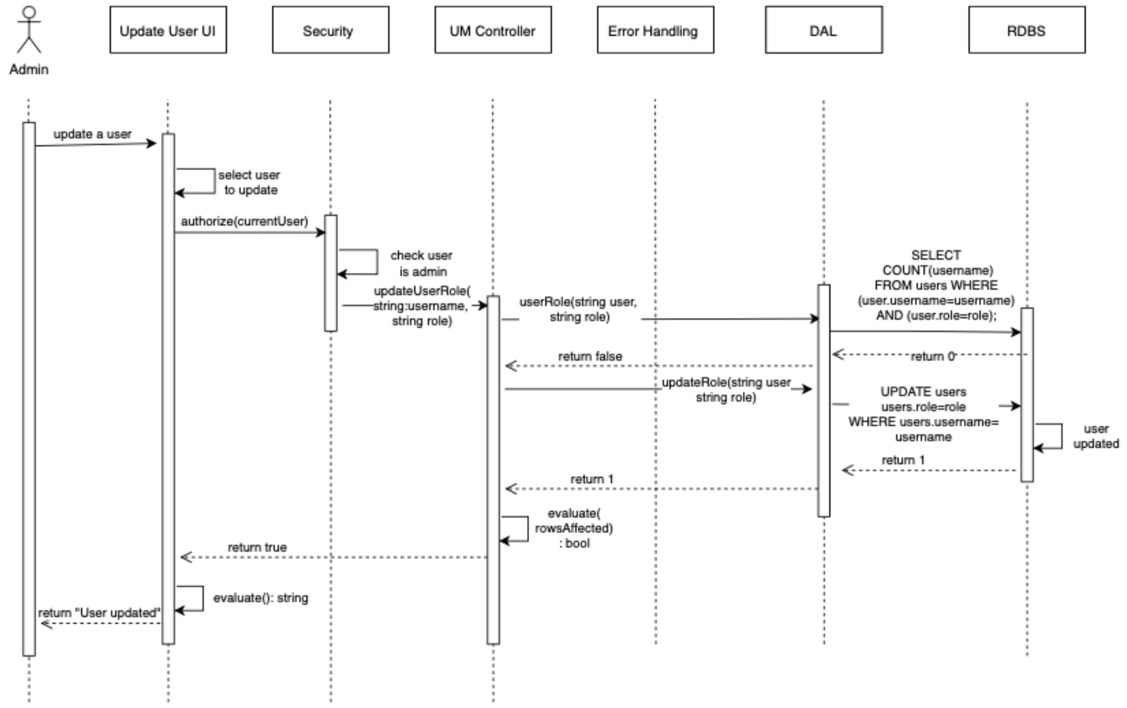
## Delete User (Success)



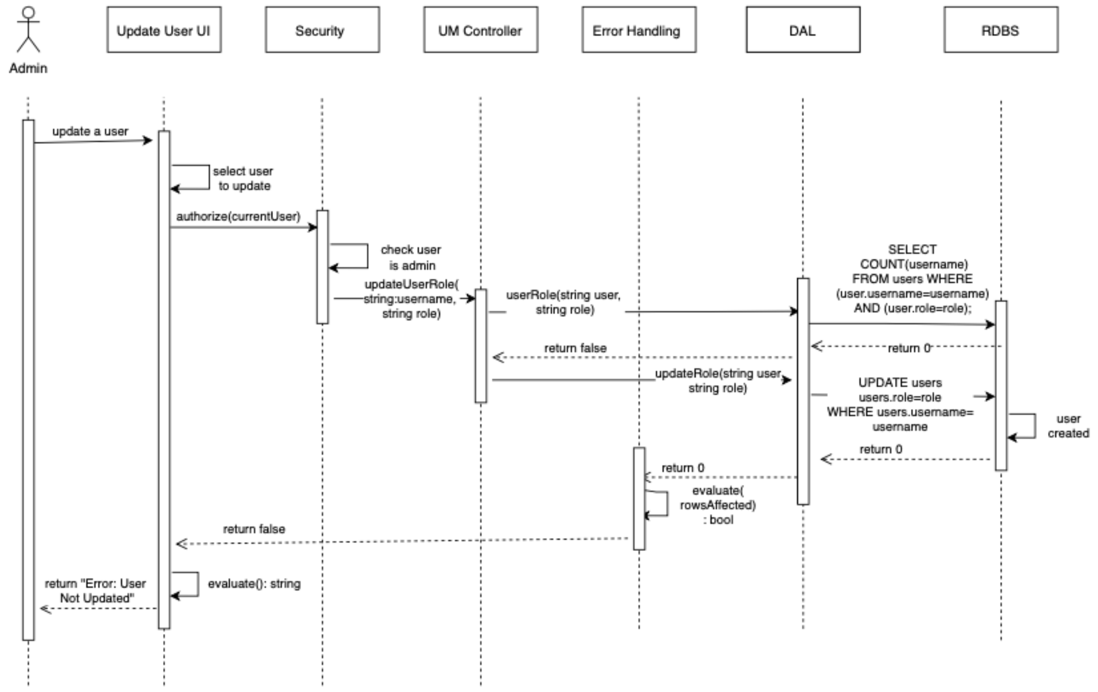
## Delete User (Fail - user not deleted)



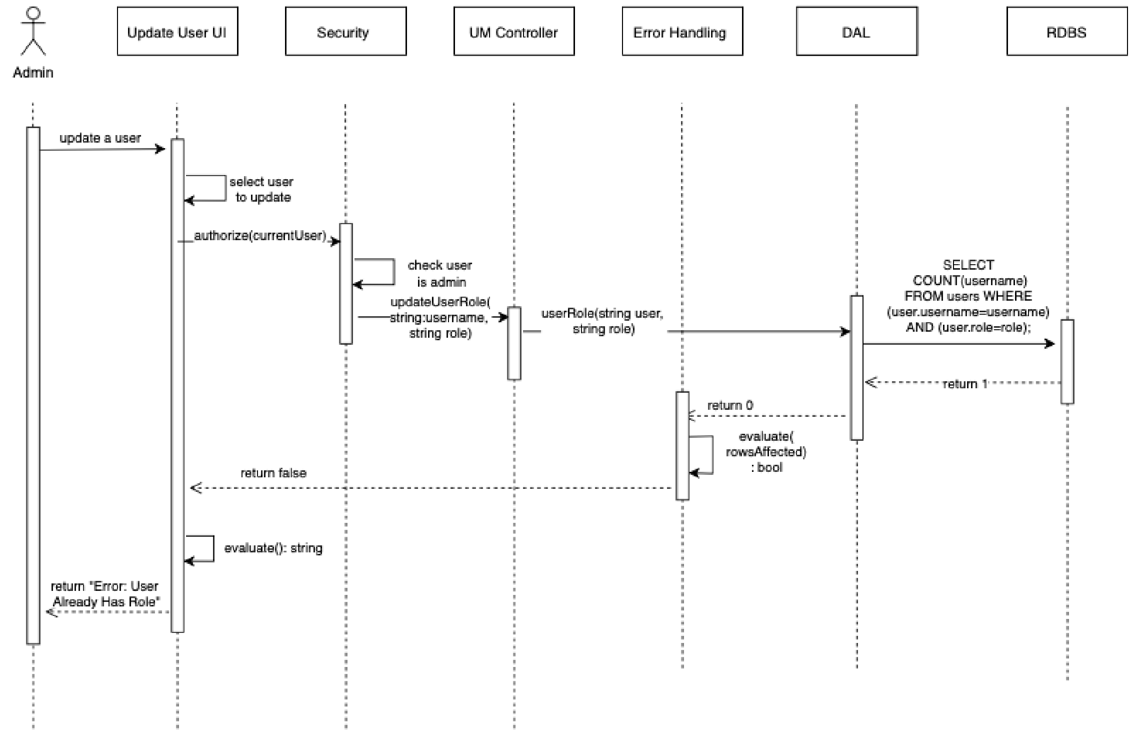
## Update User Role (Success)


Admin
Update User UI
Security
UM Controller
Error Handling
DAL
RDBS
update a user
select user to update
authorize(currentUser)
check user is admin
updateUserRole( string:username, string role)
userRole(string user, string role)
SELECT COUNT(username) FROM users WHERE (user.username=username) AND (user.role=role);
return 0
return false
updateRole(string user string role)
UPDATE users users.role=role WHERE users.username= username
user updated
return 1
return 1
evaluate( rowsAffected) : bool
return true
evaluate(): string
return "User updated"

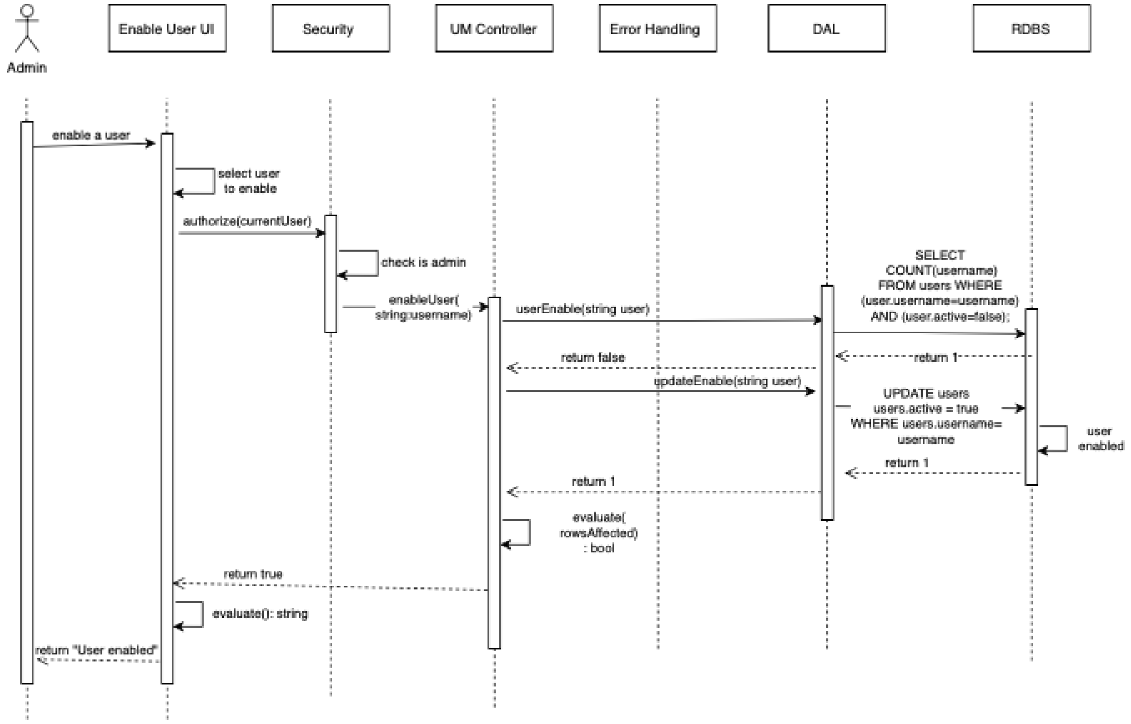
## Update User Role (Fail - role not updated)


Admin
Update User UI
Security
UM Controller
Error Handling
DAL
RDBS
update a user
select user to update
authorize(currentUser)
check user is admin
updateUserRole( string:username, string role)
userRole(string user, string role)
SELECT COUNT(username) FROM users WHERE (user.username=username) AND (user.role=role);
return 0
return false
updateRole(string user string role)
UPDATE users users.role=role WHERE users.username= username
user created
return 0
return 0
evaluate( rowsAffected) : bool
return false
evaluate(): string
return "Error: User Not Updated"

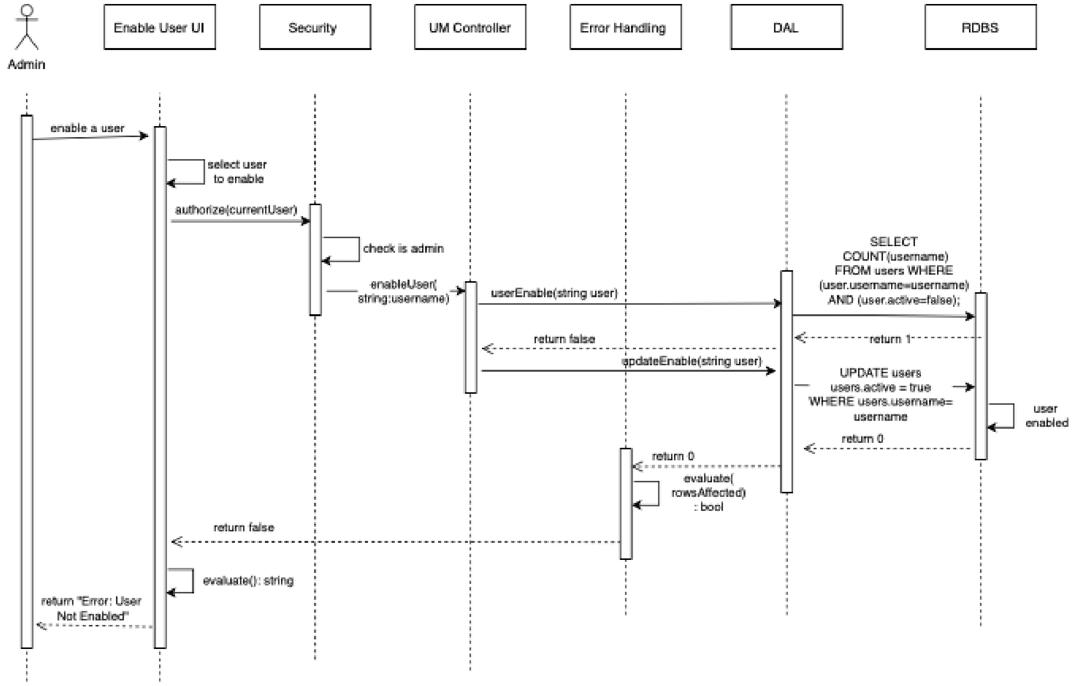
## Update User Role (Fail - not new role)



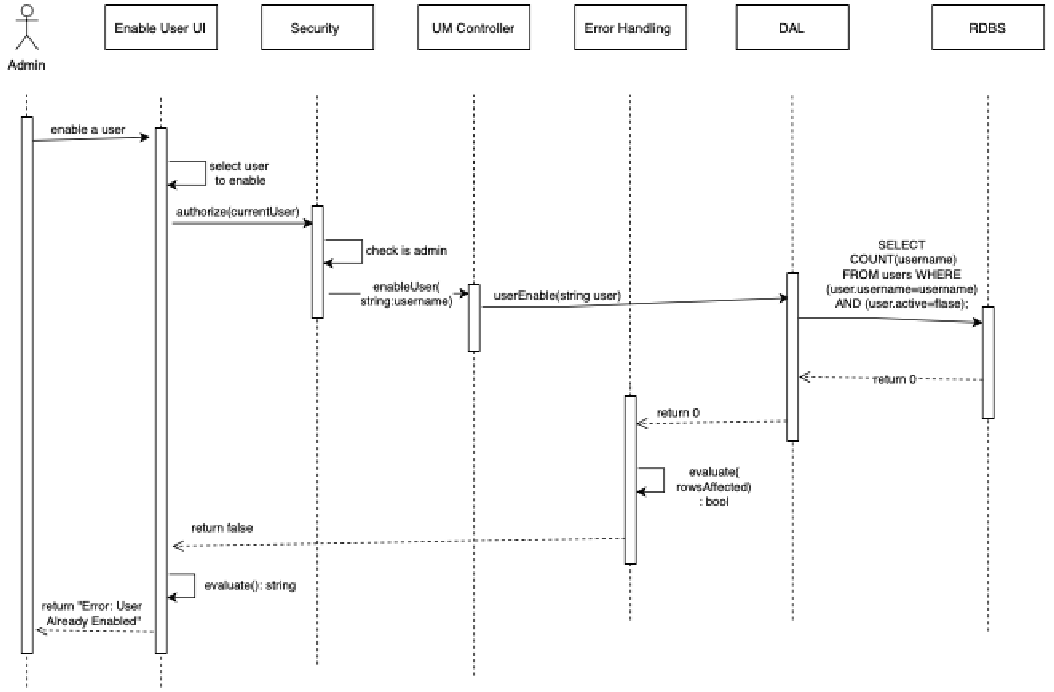
## Enable User (Success)



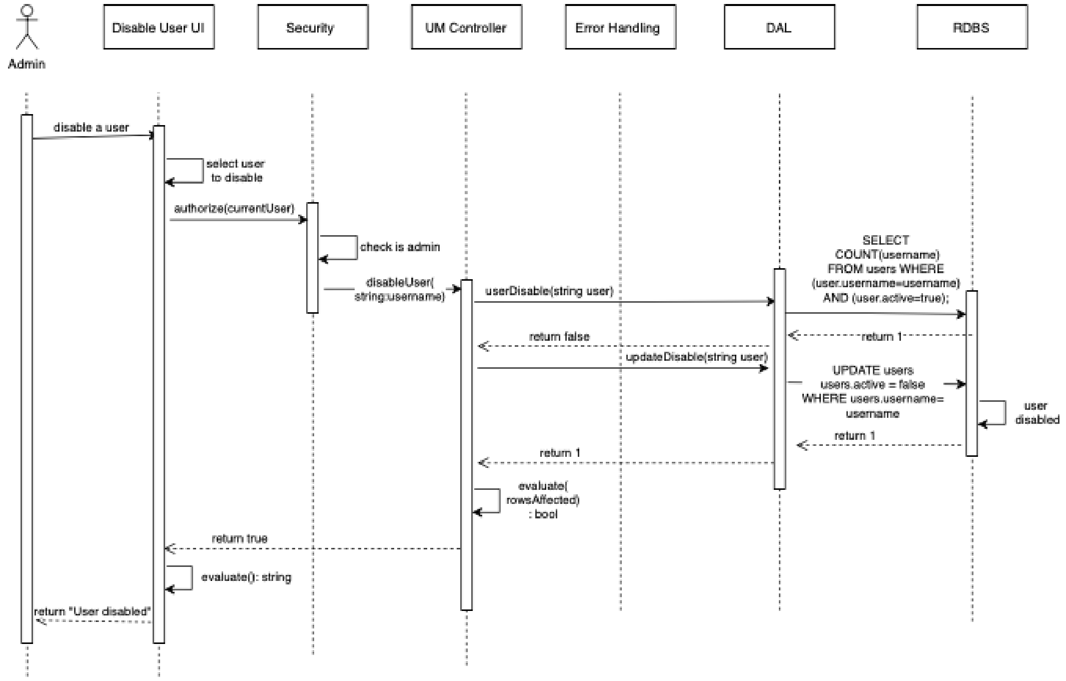
## Enable User (Fail - user not enabled)



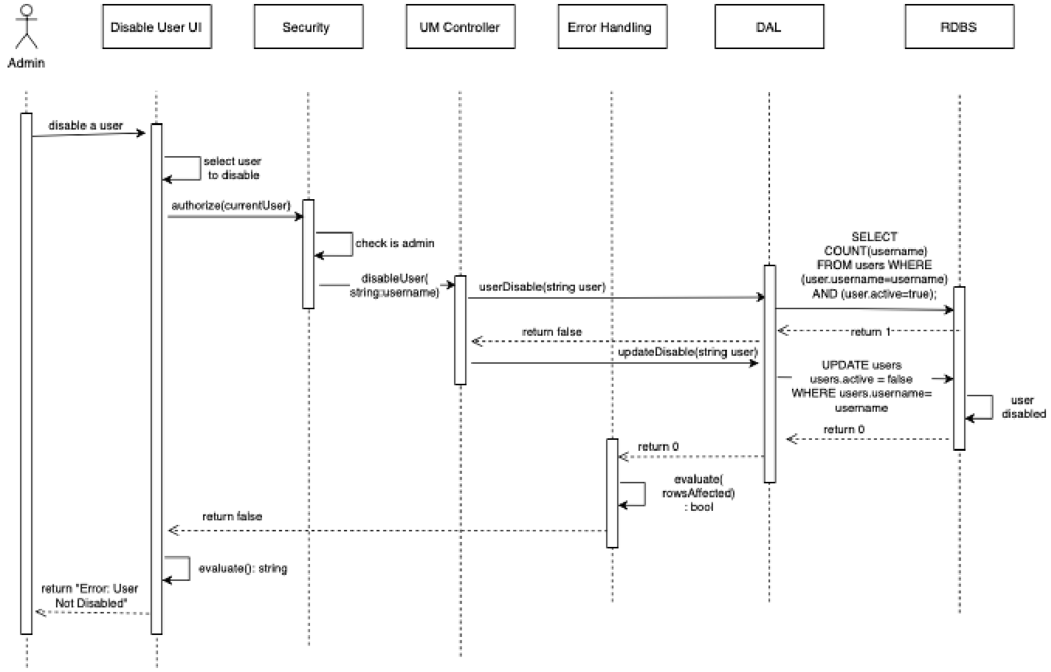
## Enable User (Fail - user already enabled)



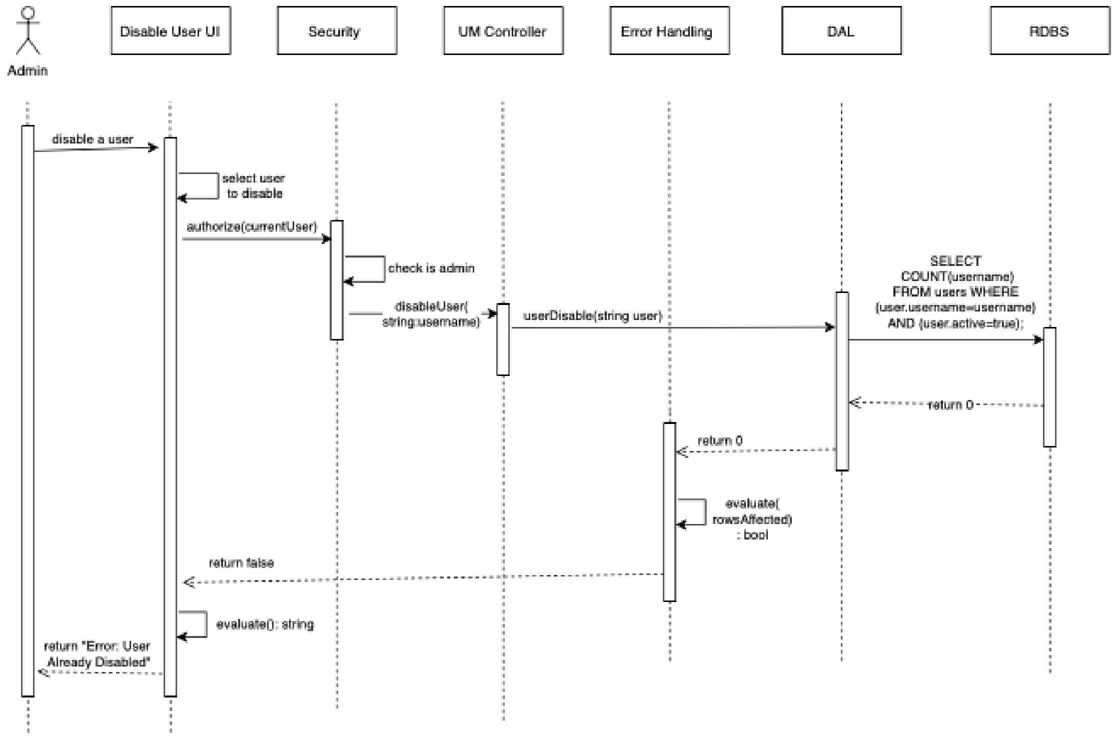
## Disable User (Success)



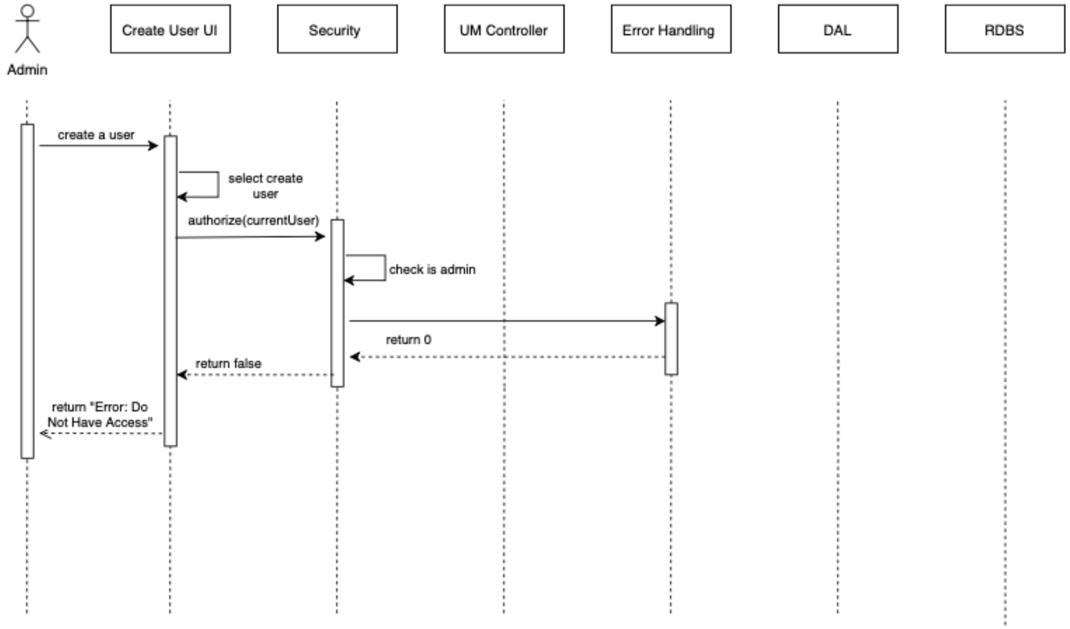
## Disable User (Fail - user not diabled)


Admin
Disable User UI
Security
UM Controller
Error Handling
DAL
RDBS
disable a user
select user to disable
authorize(currentUser)
check is admin
disableUser( string:username)
userDisable(string user)
SELECT COUNT(username) FROM users WHERE (user.username=username) AND (user.active=true);
return 1
return false
updateDisable(string user)
UPDATE users users.active = false WHERE users.username= username
user disabled
return 0
return 0
evaluate( rowsAffected) : bool
return false
evaluate(): string
return "Error: User Not Disabled"

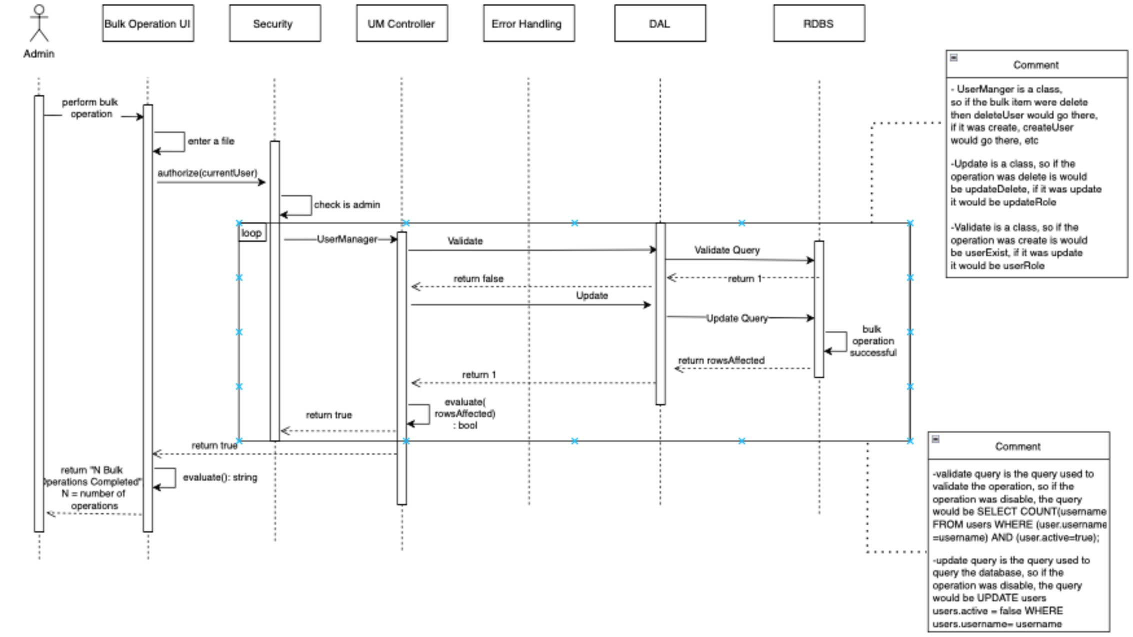
## Disable User (Fail - user already disabled)



## User not authorized (User not an admin)



## Bulk Operations (Success)



- UserManger is a class, so if the bulk item were delete then deleteUser would go there, if it was create, createUser would go there, etc

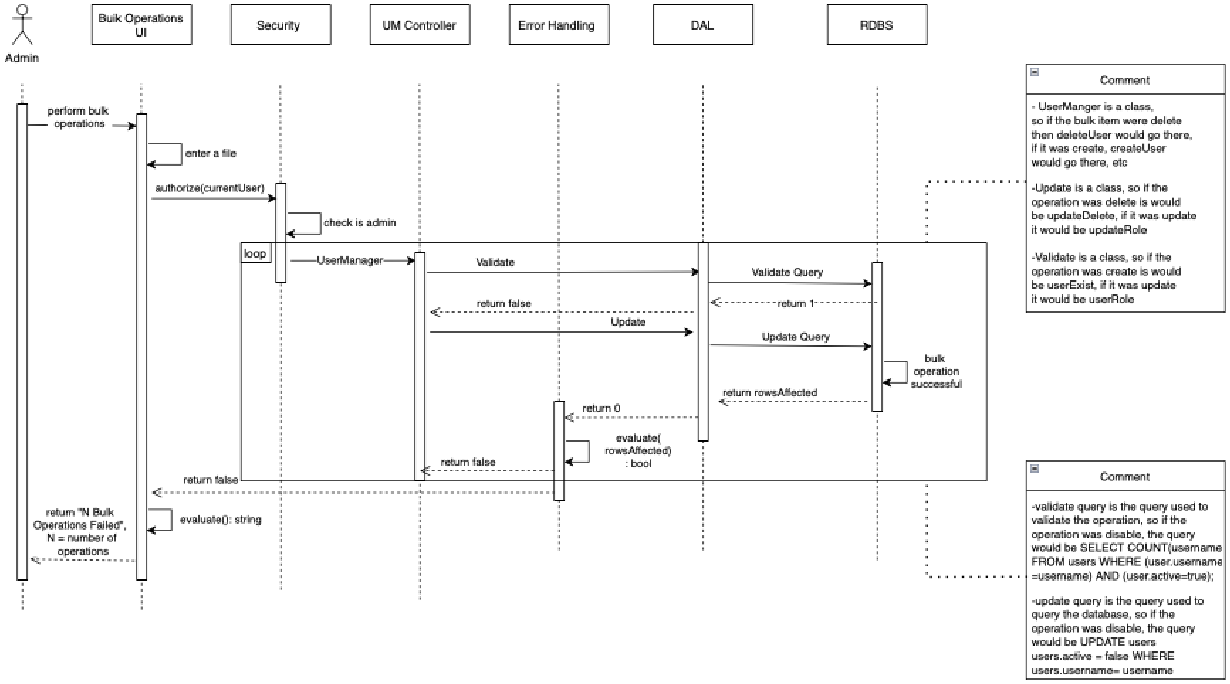
-Update is a class, so if the operation was delete is would be updateDelete, if it was update it would be updateRole

-Validate is a class, so if the operation was create is would be userExist, if it was update it would be userRole

-validate query is the query used to validate the operation, so if the operation was disable, the query would be SELECT COUNT(username FROM users WHERE (user.username =username) AND (user.active=true);

-update query is the query used to query the database, so if the operation was disable, the query would be UPDATE users users.active = false WHERE users.username= username

## Bulk Operations (Fail - not all completed successfully)



- UserManger is a class, so if the bulk item were delete then deleteUser would go there, if it was create, createUser would go there, etc

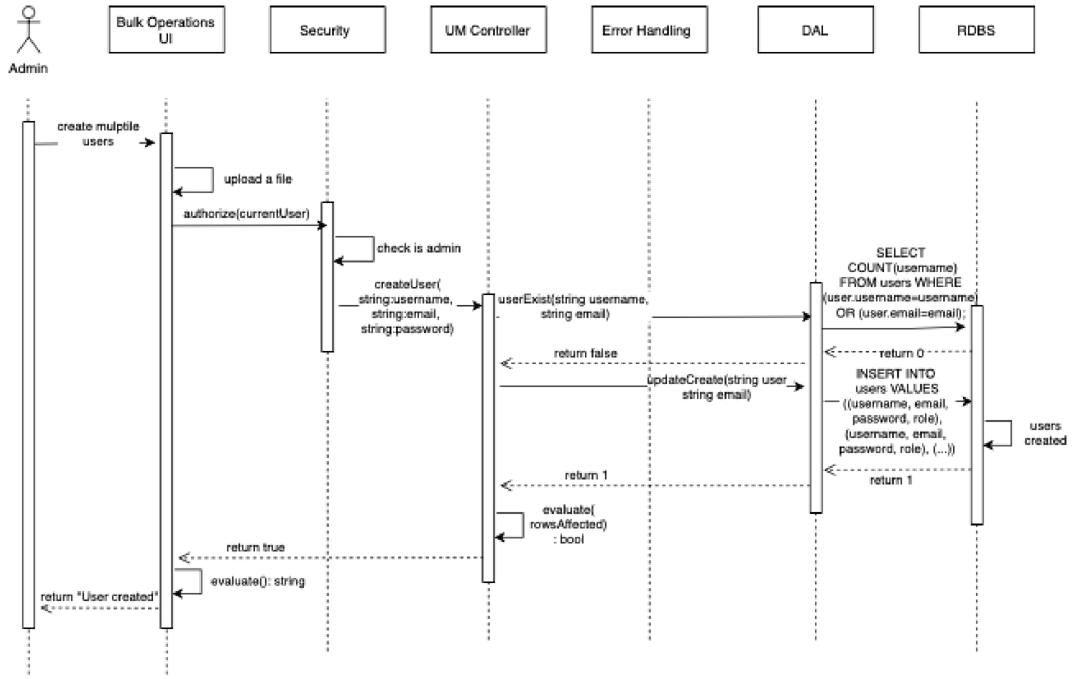
-Update is a class, so if the operation was delete is would be updateDelete, if it was update it would be updateRole

-Validate is a class, so if the operation was create is would be userExist, if it was update it would be userRole

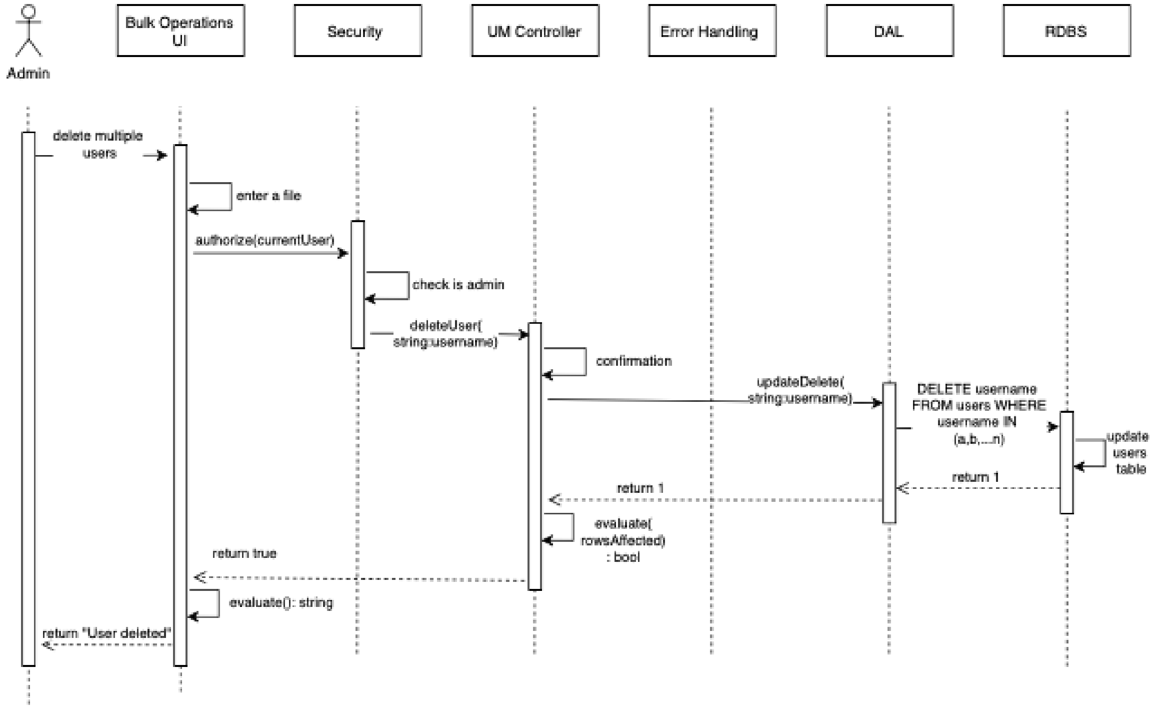
-validate query is the query used to validate the operation, so if the operation was disable, the query would be SELECT COUNT(username FROM users WHERE (user.username =username) AND (user.active=true);

-update query is the query used to query the database, so if the operation was disable, the query would be UPDATE users users.active = false WHERE users.username= username

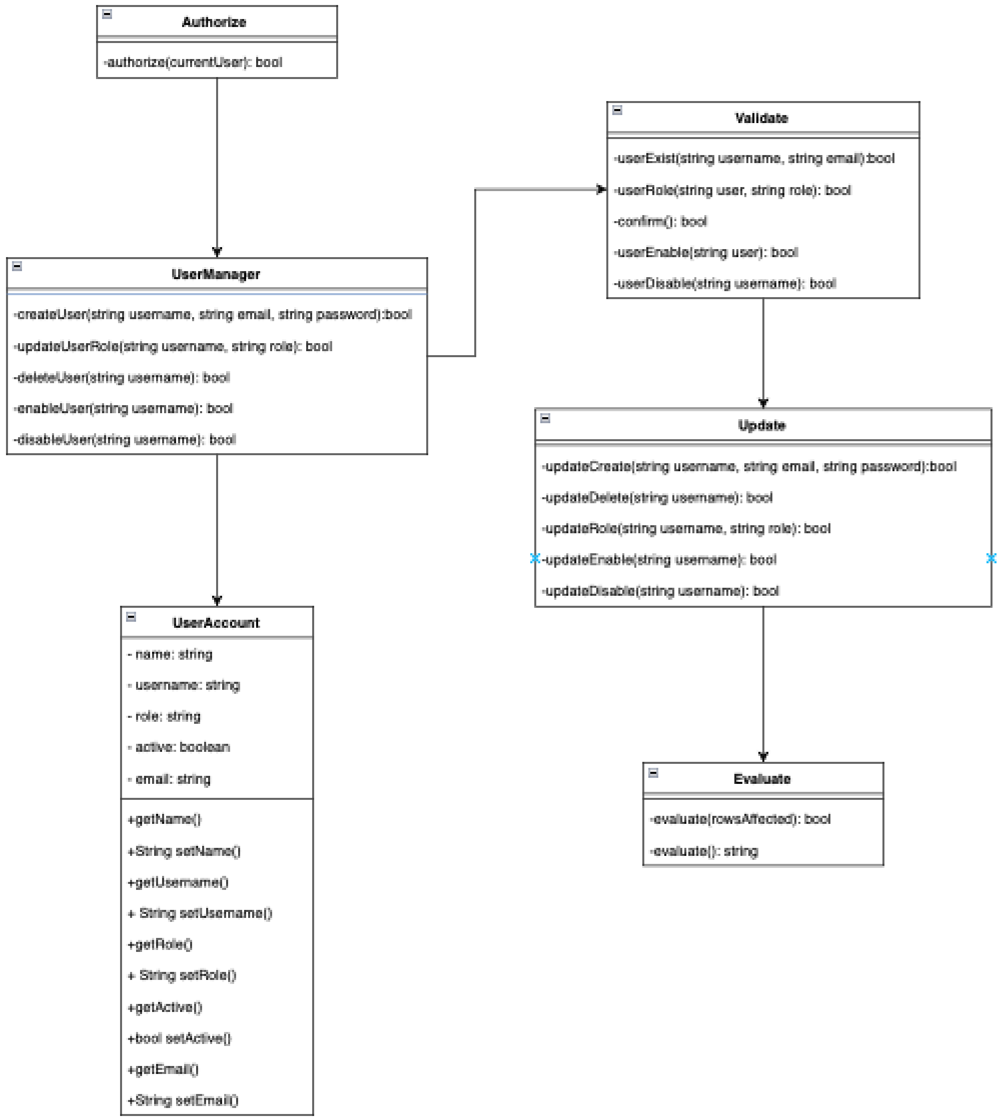
## Bulk Operations - Create


Admin
Bulk Operations UI
Security
UM Controller
Error Handling
DAL
RDBS
create mulptile users
upload a file
authorize(currentUser)
check is admin
createUser( string:username, string:email, string:password)
userExist(string username, string email)
SELECT COUNT(username) FROM users WHERE (user.username=username) OR (user.email=email);
return 0
return false
updateCreate(string user string email)
INSERT INTO users VALUES ((username, email, password, role), (username, email, password, role), (...))
users created
return 1
return 1
evaluate( rowsAffected) : bool
return true
evaluate(): string
return "User created"

## Bulk Operations - Delete

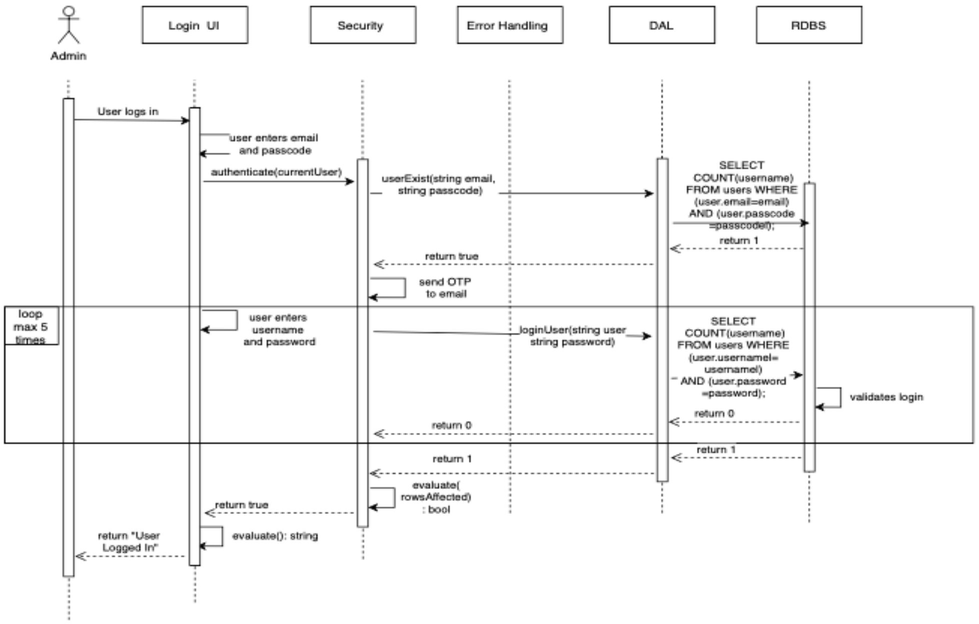


## Class DIagram


Authorize
-authorize(currentUser): bool
UserManager
-createUser(string username, string email, string password):bool
-updateUserRole(string username, string role): bool
-deleteUser(string username): bool
-enableUser(string username): bool
-disableUser(string username): bool
Validate
-userExist(string username, string email):bool
-userRole(string user, string role): bool
-confirm(): bool
-userEnable(string user): bool
-userDisable(string username): bool
Update
-updateCreate(string username, string email, string password):bool
-updateDelete(string username): bool
-updateRole(string username, string role): bool
-updateEnable(string username): bool
-updateDisable(string username): bool
UserAccount
- name: string
- username: string
- role: string
- active: boolean
- email: string
+getName()
+String setName()
+getUsername()
+ String setUsername()
+getRole()
+ String setRole()
+getActive()
+bool setActive()
+getEmail()
+String setEmail()
Evaluate
-evaluate(rowsAffected): bool
-evaluate(): string

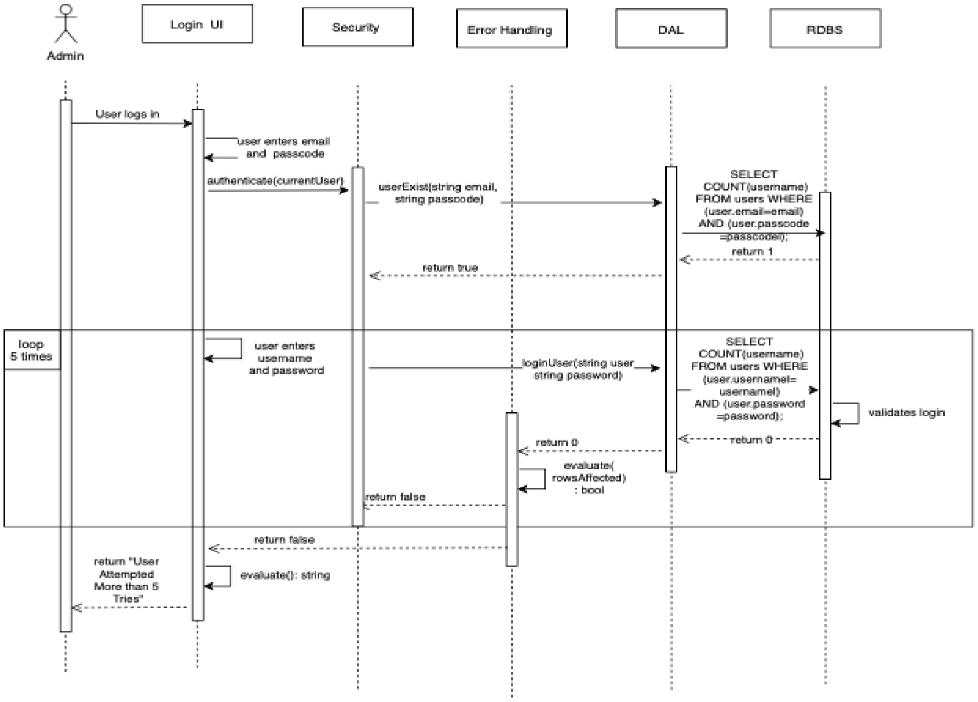
## Authentication Sequence Diagram

### Success - User Logged In



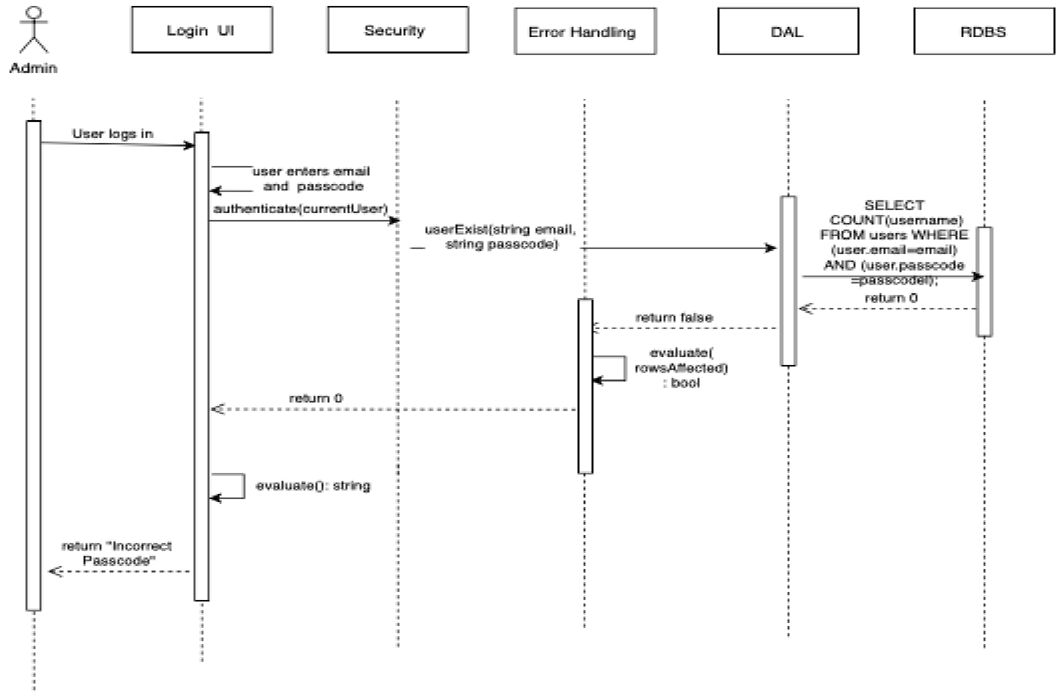
Admin
Login UI
Security
Error Handling
DAL
RDBS
User logs in
user enters email and passcode
authenticate(currentUser)
userExist(string email, string passcode)
SELECT COUNT(username) FROM users WHERE (user.email=email) AND (user.passcode =passcode);
return 1
return true
send OTP to email
loop max 5 times
user enters username and password
loginUser(string user string password)
SELECT COUNT(username) FROM users WHERE (user.usernamel= usernamel) AND (user.password =password);
validates login
return 0
return 0
return 1
return 1
evaluate( rowsAffected) : bool
return true
evaluate(): string
return "User Logged In"

### Failure - User Not Logged In

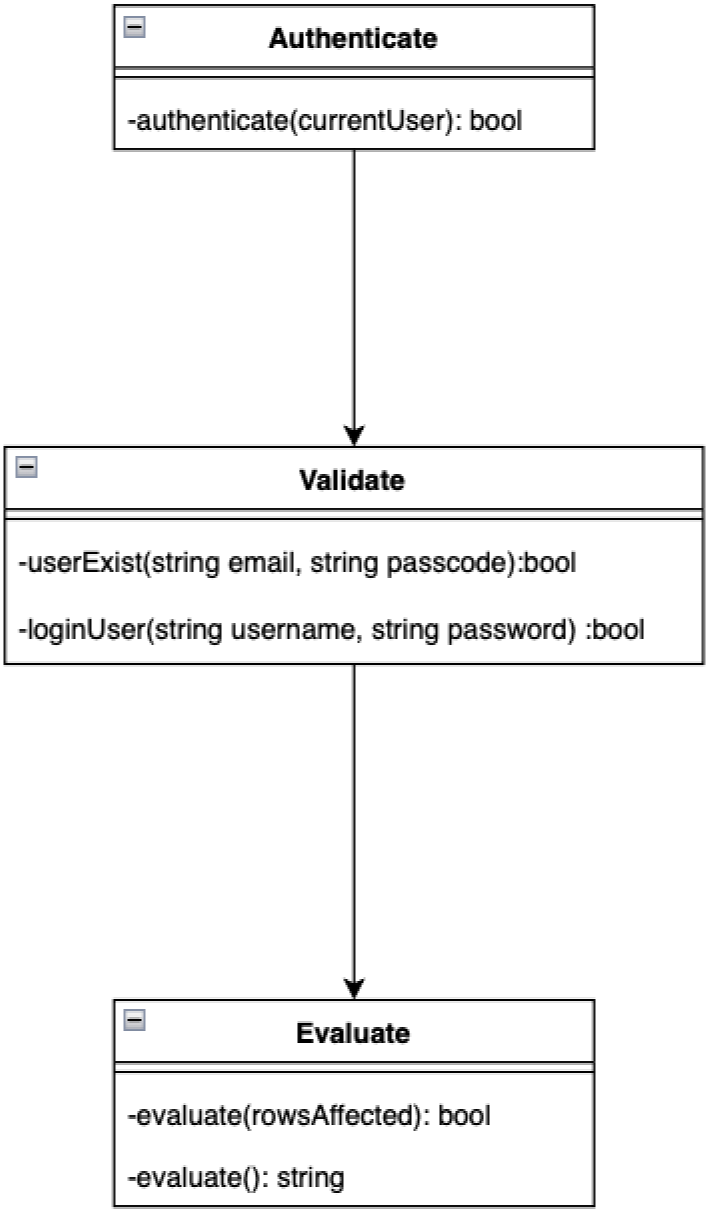


Admin
Login UI
Security
Error Handling
DAL
RDBS
User logs in
user enters email and passcode
authenticate(currentUser)
userExist(string email, string passcode)
SELECT COUNT(username) FROM users WHERE (user.email=email) AND (user.passcode =passcode);
return 1
return true
loop 5 times
user enters username and password
loginUser(string user string password)
SELECT COUNT(username) FROM users WHERE (user.usernamel= usernamel) AND (user.password =password);
validates login
return 0
return 0
evaluate( rowsAffected) : bool
return false
return false
evaluate(): string
return "User Attempted More than 5 Tries"

## Failure - Passcode Incorrect

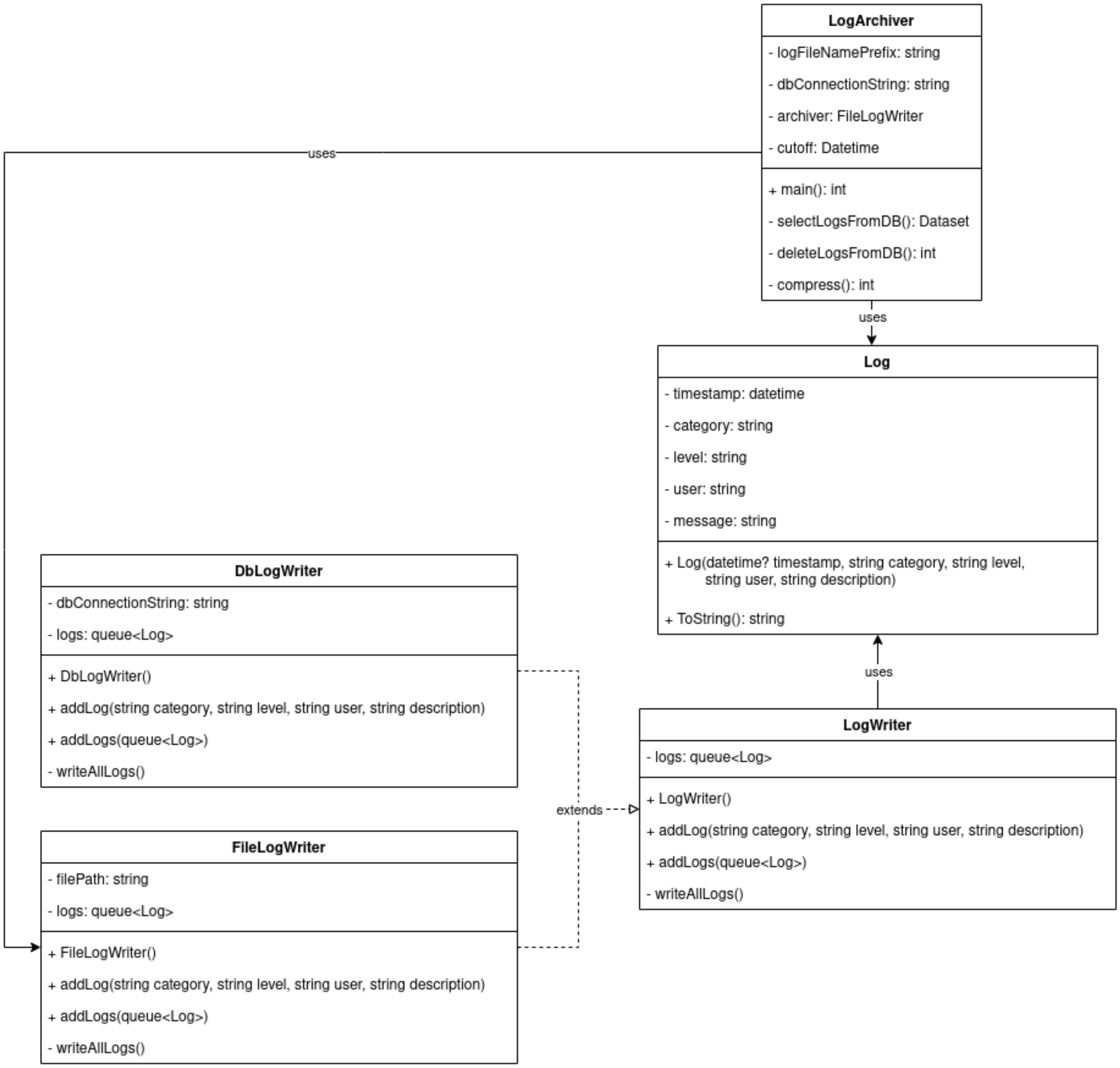

Admin
Login UI
Security
Error Handling
DAL
RDBS
User logs in
user enters email and passcode
authenticate(currentUser)
userExist(string email, string passcode)
SELECT COUNT(username) FROM users WHERE (user.email=email) AND (user.passcode =passcode]);
return 0
return false
evaluate( rowsAffected) : bool
return 0
evaluate(): string
return "Incorrect Passcode"

## Authentication Class Diagram


Authenticate
-authenticate(currentUser): bool
Validate
-userExist(string email, string passcode):bool
-loginUser(string username, string password) :bool
Evaluate
-evaluate(rowsAffected): bool
-evaluate(): string

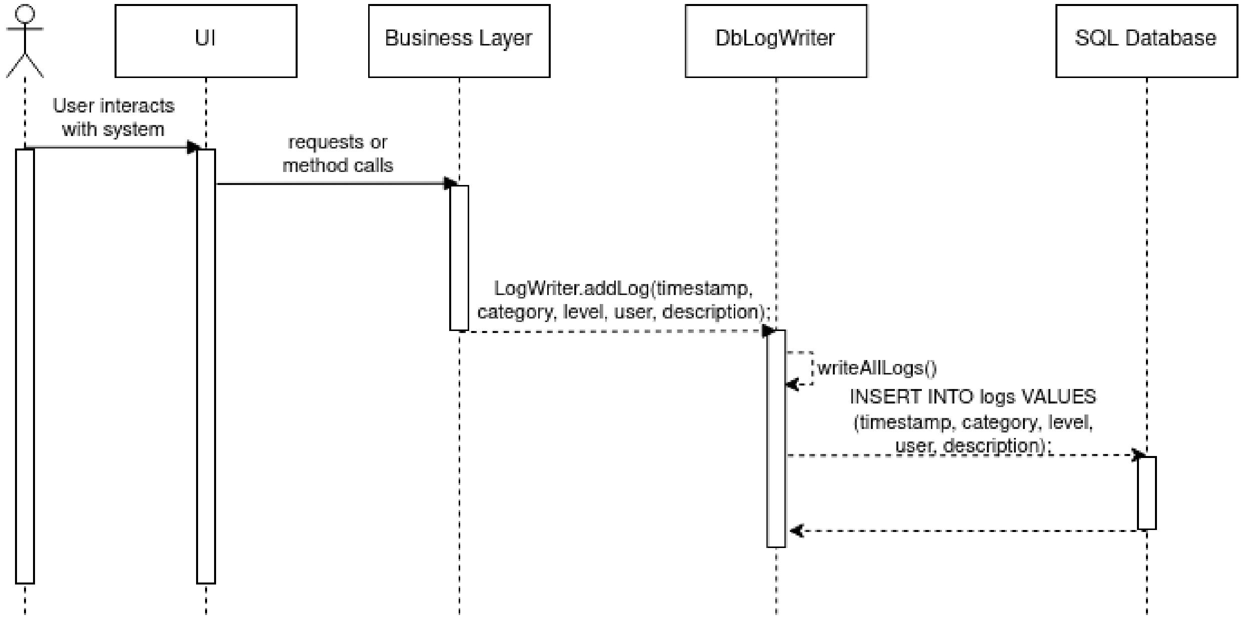
# Logging and Archiving

## Logging and Archiving Class DIagram

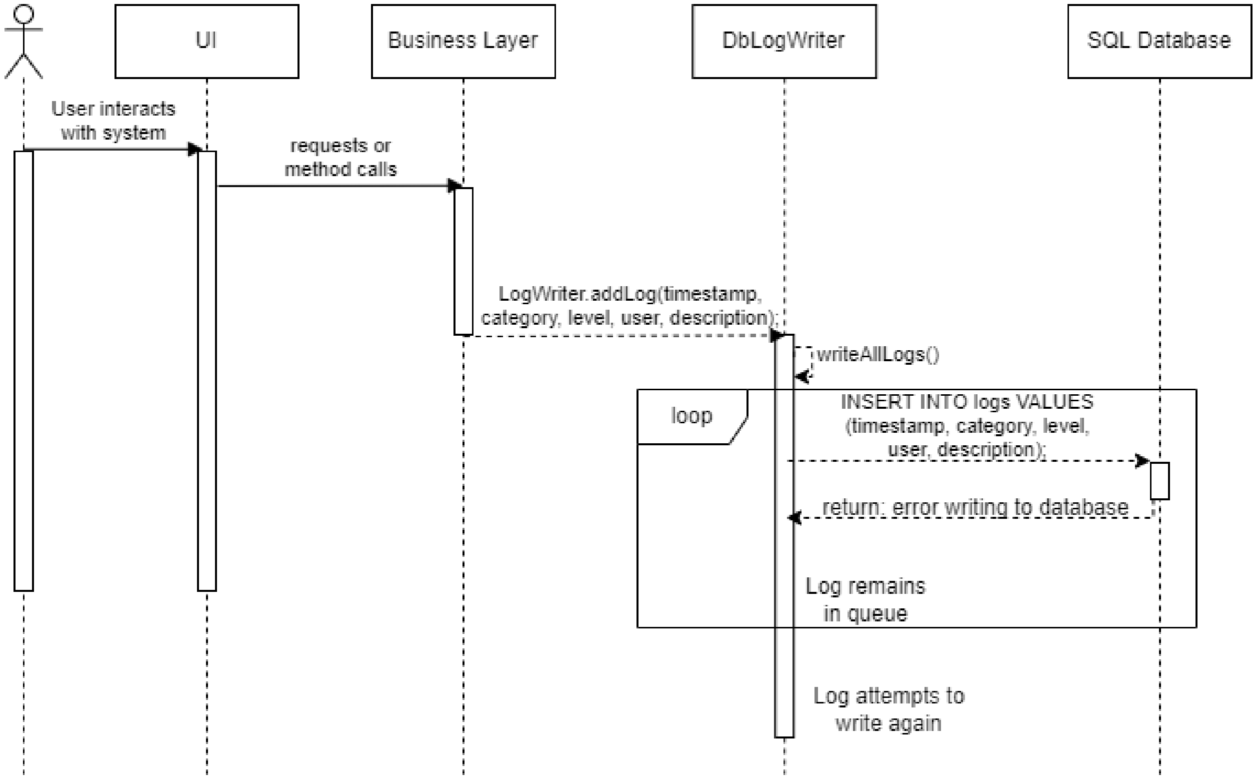

LogArchiver
- logFileNamePrefix: string
- dbConnectionString: string
- archiver: FileLogWriter
- cutoff: Datetime
+ main(): int
- selectLogsFromDB(): Dataset
- deleteLogsFromDB(): int
- compress(): int
uses
uses
Log
- timestamp: datetime
- category: string
- level: string
- user: string
- message: string
+ Log(datetime? timestamp, string category, string level, string user, string description)
+ ToString(): string
DbLogWriter
- dbConnectionString: string
- logs: queue<Log>
+ DbLogWriter()
+ addLog(string category, string level, string user, string description)
+ addLogs(queue<Log>)
- writeAllLogs()
uses
LogWriter
- logs: queue<Log>
+ LogWriter()
+ addLog(string category, string level, string user, string description)
+ addLogs(queue<Log>)
- writeAllLogs()
extends
FileLogWriter
- filePath: string
- logs: queue<Log>
+ FileLogWriter()
+ addLog(string category, string level, string user, string description)
+ addLogs(queue<Log>)
- writeAllLogs()

**Logging Sequence Diagram**

- Since logging is a cross-cutting concern, we do not know which objects or methods will need to generate logs. Therefore we have tried to design classes that can write logs from anywhere in our code.
- Datetime.UtcNow is passed in as the Log's timestamp argument, setting the time as immediately as possible.
- LogWriter.addLog() is used asynchronously in a "fire and forget" manner. This diagram shows the method being called at the end of a method, but it can be called at any time.

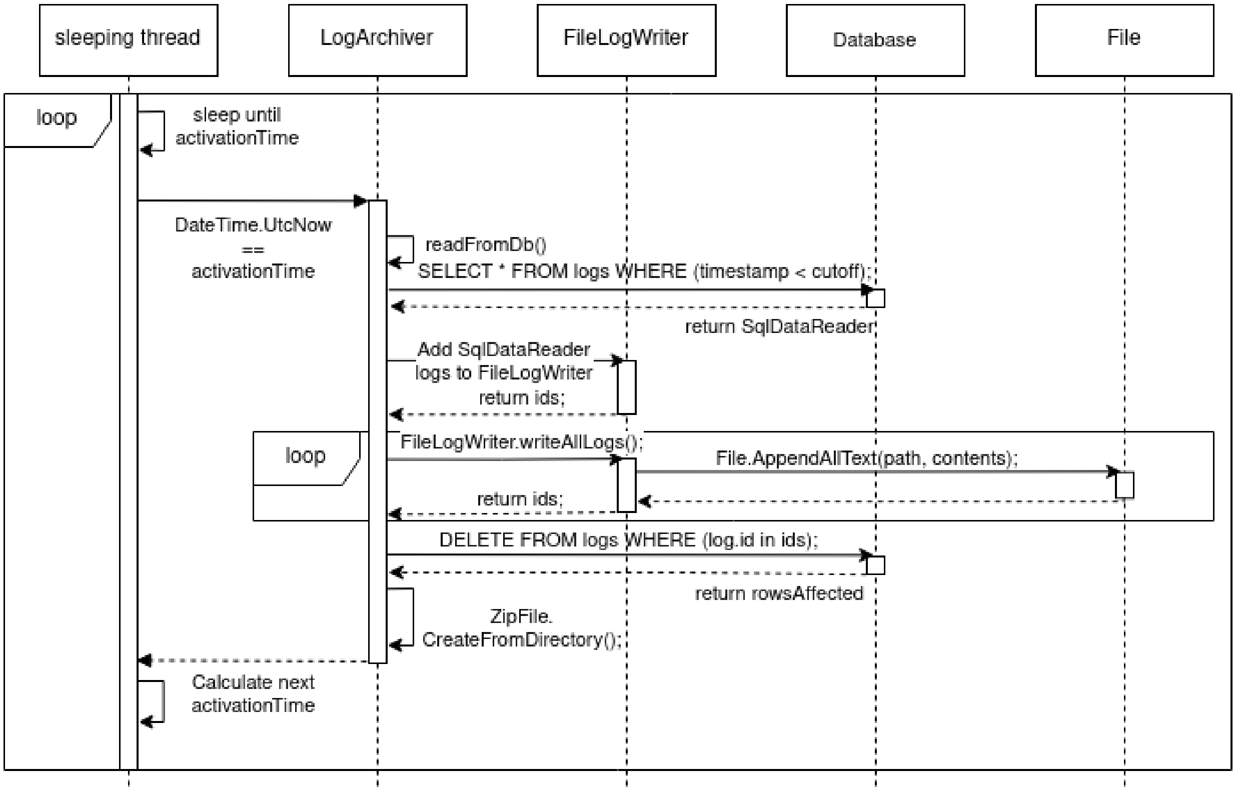

UI
Business Layer
DbLogWriter
SQL Database
User interacts with system
requests or method calls
LogWriter.addLog(timestamp, category, level, user, description);
writeAllLogs()
INSERT INTO logs VALUES (timestamp, category, level, user, description);

Failure - Could not connect to SQL Database

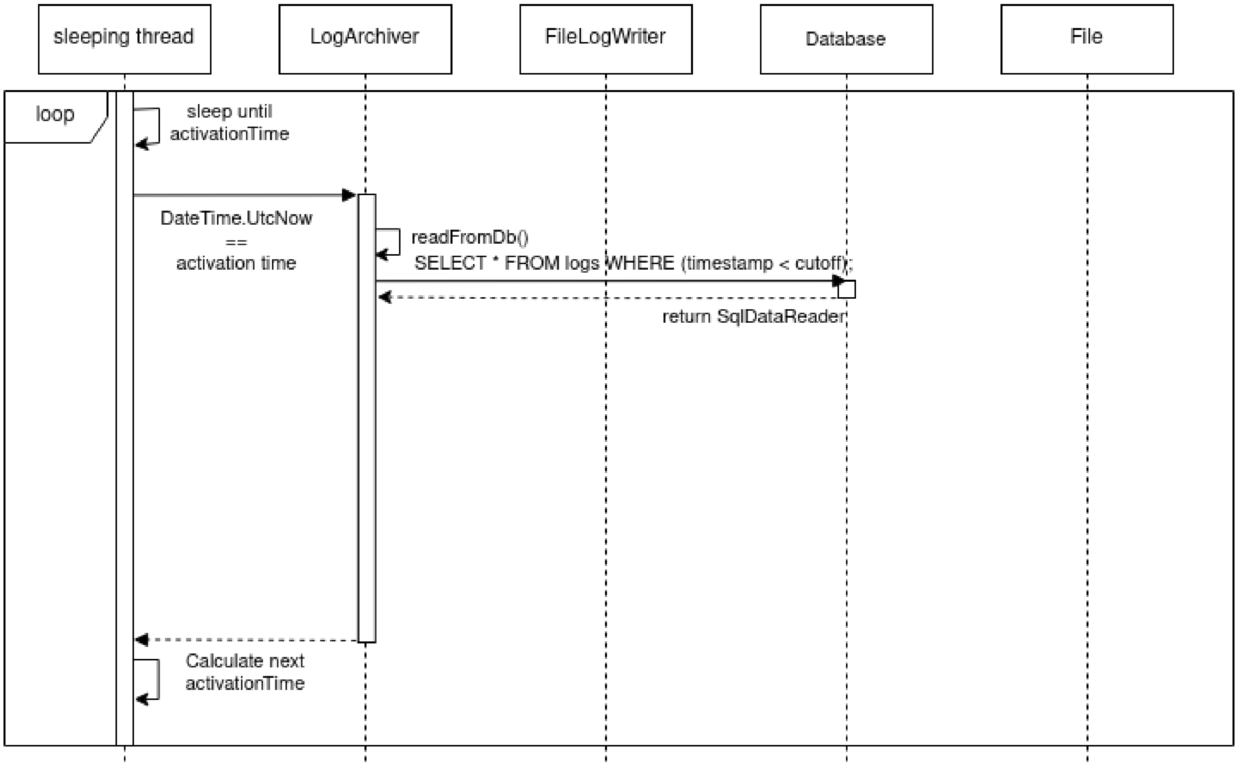

UI
Business Layer
DbLogWriter
SQL Database
User interacts with system
requests or method calls
LogWriter.addLog(timestamp, category, level, user, description);
writeAllLogs()
loop
INSERT INTO logs VALUES (timestamp, category, level, user, description);
return: error writing to database
Log remains in queue
Log attempts to write again

**Archiving Sequence Diagram (Success)**

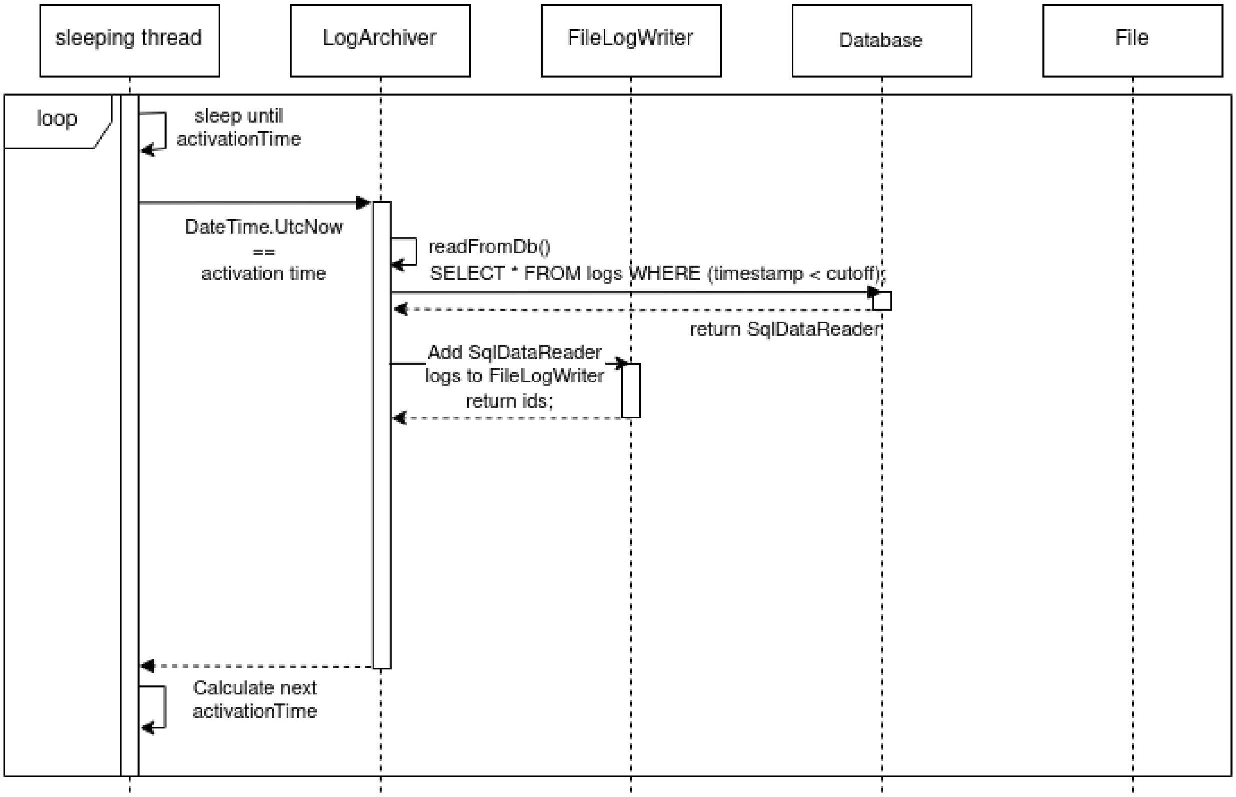
- We will schedule this program to be called on the first of each month at midnight ("activation time") using a thread that sleeps until the next activation time. After attempting to archive the files, the thread will calculate the next activation time.
- This program will get all logs from the database from before the "cutoff" (activation time - 30 days), store them in a file in our block storage, and compress the file.
- FileLogWriter functions will return the set of IDs (ints) that were successfully read from the database or successfully written to the file.
  - The size of the set can be checked to determine whether or not an error occurred. If fewer logs were written to the file than were read, then a failure occurred while writing to the file.
  - Upon deletion, only logs successfully written to the file will be deleted.


sleeping thread
LogArchiver
FileLogWriter
Database
File
loop
sleep until activationTime
DateTime.UtcNow == activationTime
readFromDb()
SELECT * FROM logs WHERE (timestamp < cutoff);
return SqlDataReader;
Add SqlDataReader logs to FileLogWriter
return ids;
loop
FileLogWriter.writeAllLogs();
File.AppendAllText(path, contents);
return ids;
DELETE FROM logs WHERE (log.id in ids);
return rowsAffected
ZipFile. CreateFromDirectory();
Calculate next activationTime

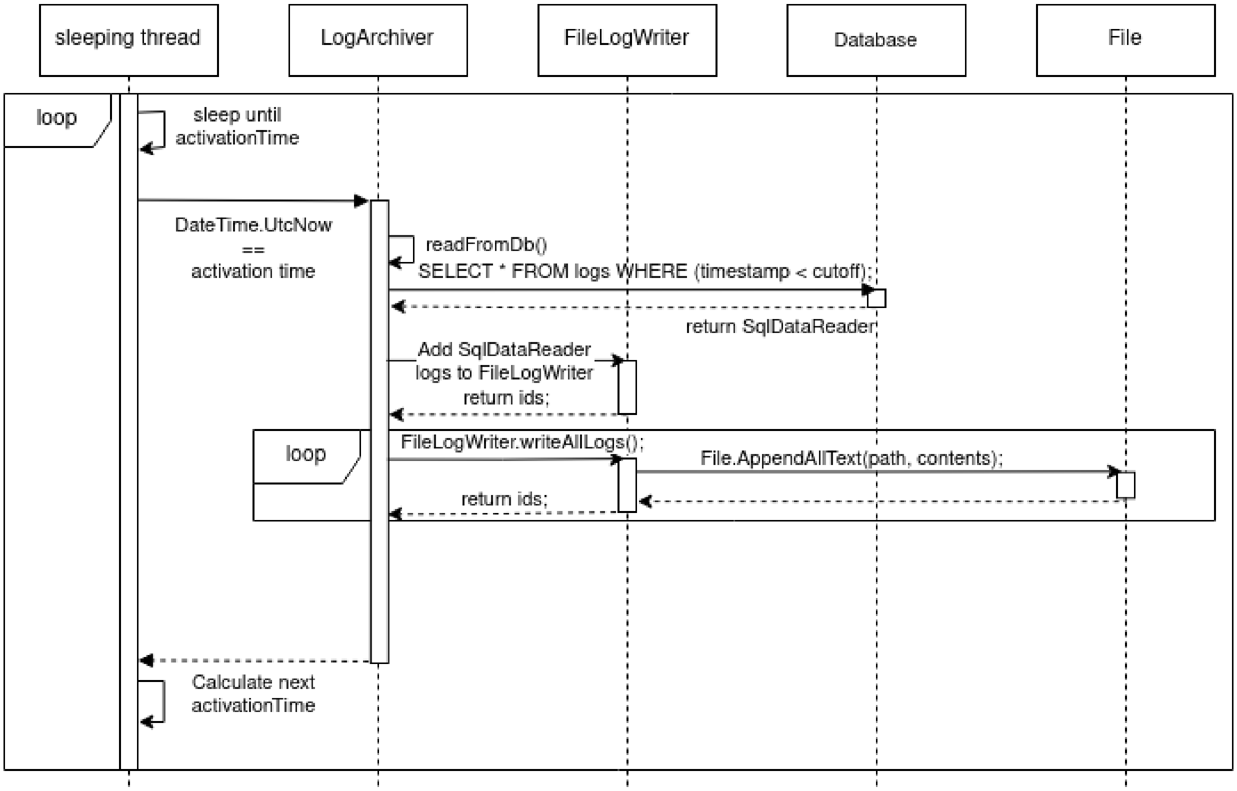
## Archiving Sequence Diagram (Failure during SELECT query)


sleeping thread
LogArchiver
FileLogWriter
Database
File
loop
sleep until activationTime
DateTime.UtcNow == activation time
readFromDb()
SELECT * FROM logs WHERE (timestamp < cutoff);
return SqlDataReader
Calculate next activationTime

## Archiving Sequence Diagram (Failure when reading SELECT query results)



## Archiving Sequence Diagram (Failure when writing to file)


sleeping thread
LogArchiver
FileLogWriter
Database
File
loop
sleep until activationTime
DateTime.UtcNow == activation time
readFromDb()
SELECT * FROM logs WHERE (timestamp < cutoff);
return SqlDataReader
Add SqlDataReader logs to FileLogWriter
return ids;
loop
FileLogWriter.writeAllLogs();
File.AppendAllText(path, contents);
return ids;
Calculate next activationTime