TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS

OOAD
Lab Case Study
Final Report

Submitted By:
Anish Sapkota (076BCT008)
Kushal Subedi (076BCT031)
Nabin Khanal (076BCT036)

Submitted To:
Department of Electronics and
Computer Engineering

Submitted on:
28th February, 2023

Functional Requirements:

1. **User Management:**

   Description: Users will be able to register to the system, Confirm their email and then finally login to the system. If any user forgets their password, there is also a functionality to reset the password.

   a. **Signup**:
      - Input: First Name, Last Name, Email, Phone, Password.
      - Output: User gets the error/success message.

   b. **Login:**
      - Input: Email and Password
      - Output: If email/password matches, user is logged in, else the error message is shown.

   c. **Password Reset:**
      - Input: Valid Email
      - Output: If the email does not exist in the system, an error message is shown. If email exists in the system, a password reset email will be sent.

   d. **Set new Password:**
      - Input: Valid password reset link along with the new password to be set.
      - Output: Password reset message.

2. **Friends Management:**

   Description: Users can search each other using email or username and can send them the connection requests. The user who gets the connection request will be able to either accept or reject it.

   a. **Search User:**
      - Input: Users will be able to search other users in the system by their username/email.

  - Output: Profile of matched user.

b. **Send Friend Request:**
  - Input: Friend request along with the username.
  - Output: Success/error message.

c. **Accept Friend Request:**
  - Input: User who receives the friend request will be able to accept/decline it.
  - Output: Accepted/Declined message.

3. Transaction Management:

Description: User can send the payment request to any of their friends. The user who gets the payment request will be able to either accept or reject it. If the user accepts the payment request, the requested amount is debited in the receiver's account and an equal amount is credited in the sender’s account.

a. **Create Transaction Request:**
  - Input: One user will be able to create a transaction request of desired amount to one of their friends along with the remarks.
  - Output: Transaction request is created.

b. **Accept Payment Request:**
  - Input: Input accepting the transaction request.
  - Output: Amount debited from the user’s account.

c. **Reject Payment Request:**
  - Input: Input rejecting the transaction request.
  - Output: Success/Error message.

Non-Functional Requirements:

1. **Security:**

a. Any user must not be able to access/compromise other user’s accounts.
b. System must have proper validation for email/password and against known sql injections.

2. **Scalability:**
    a. The system should be easily scalable in order to reach the maximum user base.

3. **Performance:**
    a. It must be fast and lightweight to work in poor network connections and low end devices.
    b. The load time for each page in the web-app/mobile-app should be less than 3 seconds

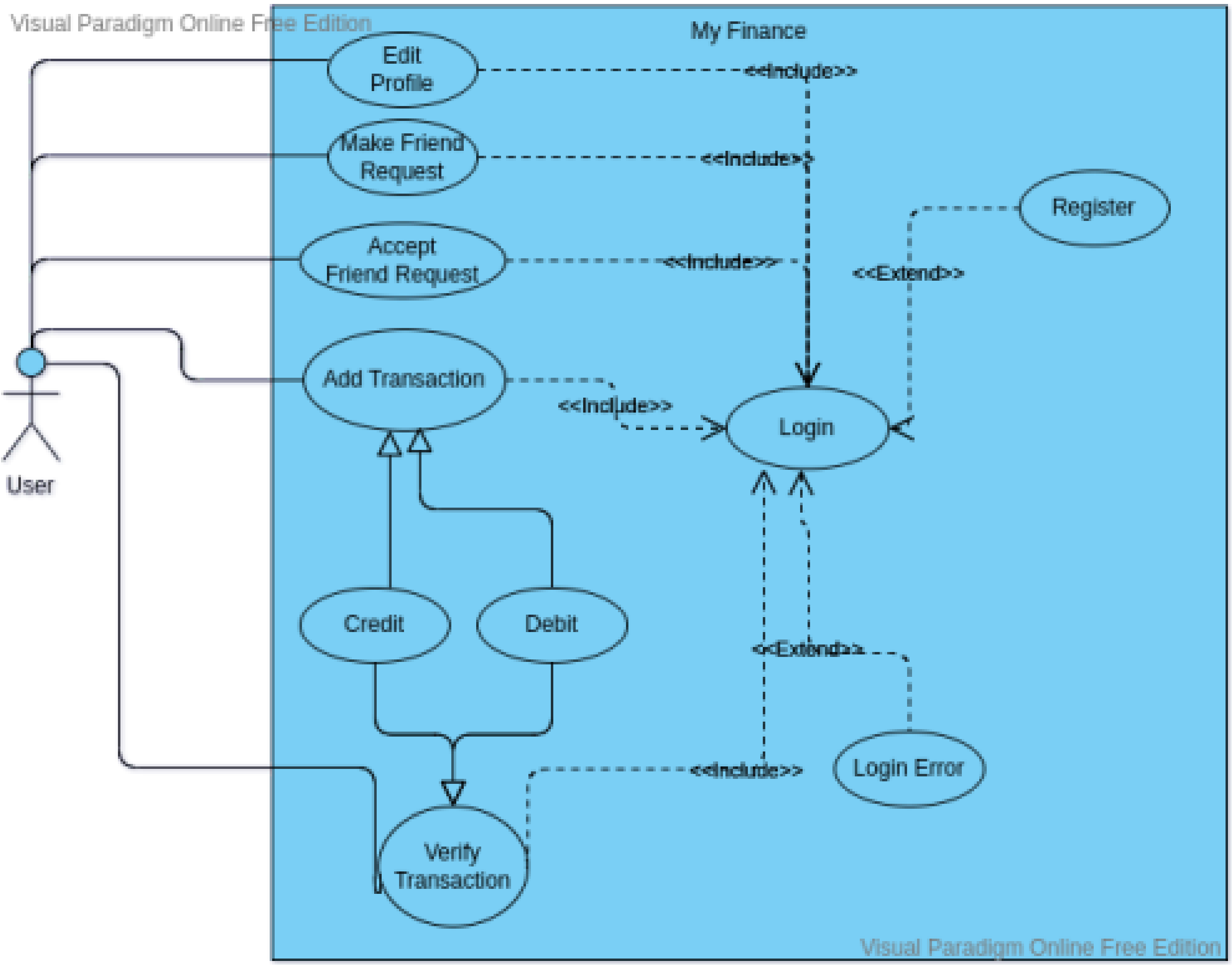
4. **Usability:**
    a. There must be easy navigation and easy-to-use features to be used by users.

5. **Maintainability:**
    a. The system should be easily maintainable and updatable.
    b. It should have clear and well-documented code and have a clear and concise design structure.

6. **Reliability:**
    a. The system should not give unexpected errors/glitches while in operation.

7. **Availability:**
    a. The system should be available 24X7. There must not be more than 5 hours of downtime each month.

Use Case Diagram:



Use case description:

1. Register

Actors	User

Description	Users can register to the system using their email and password.
Priority	Must have
Precondition	None
Post Condition	Users can login to the system using the registered email and password.

2. Login

Actors	User
Description	Users must be logged in to the system to use the app. User can login with their email and password.
Priority	Must have
Precondition	Registered
Post Condition	Users can use all the features of the app.

3. Make a Friend Request

Actors	User
Description	Users need to be friends with another user to start making track of their transactions between them. So, a user can send friend requests to another user to be friends.
Priority	Must have
Precondition	Logged In, Two user should be friend of each other
Post Condition	After a friend request is accepted, the two can start having their transaction going.

.

4. Accept Friend Request

Actors	User

Description	Users need to be friends with another user to start making track of their transactions between them. So, a user can accept friend requests from another user to be friends.
Priority	Must have
Precondition	Logged In
Post Condition	After a friend request is accepted, the two can start having their transaction going.

5. Add transaction

Actors	User
Description	Users can add either money borrowed from or lend to any other user who is their friends.
Priority	Must have
Precondition	Logged In, Two user should be friend of each other
Post Condition	After a transaction is added, notification or email is sent to the friend user for the verification purpose.

6. Verify Transaction

Actors	User
Description	Once the transaction is added by one user, the other friend user must verify it for the validity of the transaction. When a user adds a transaction, another friend user gets notified and needs to verify the transaction using the verify button in the UI.
Priority	Must have
Precondition	Logged In,Friend with another user
Post Condition	After a transaction is verified, the net total either to pay or received is updated for the validated transactions.

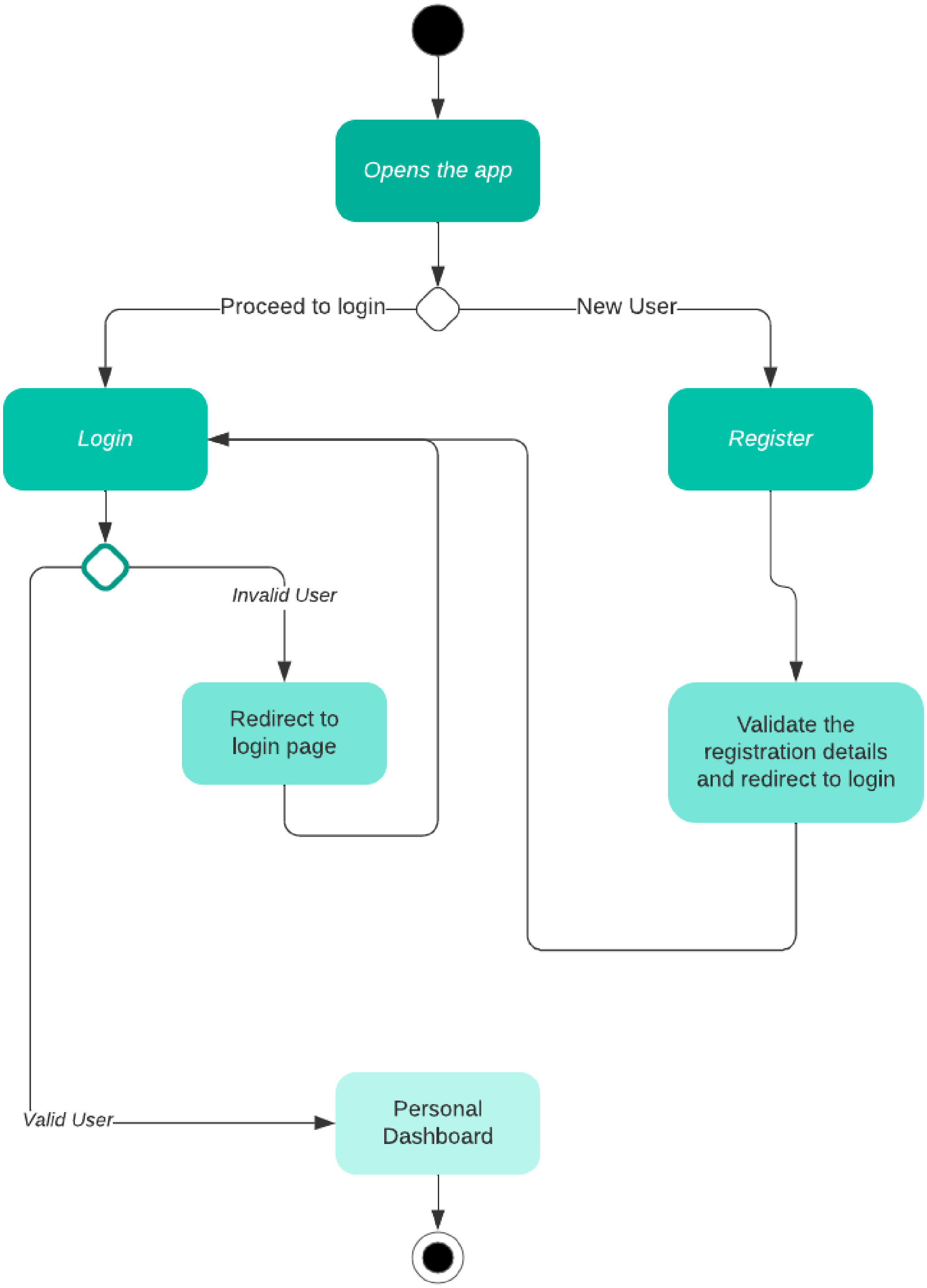
7. Edit profile

Actors	User

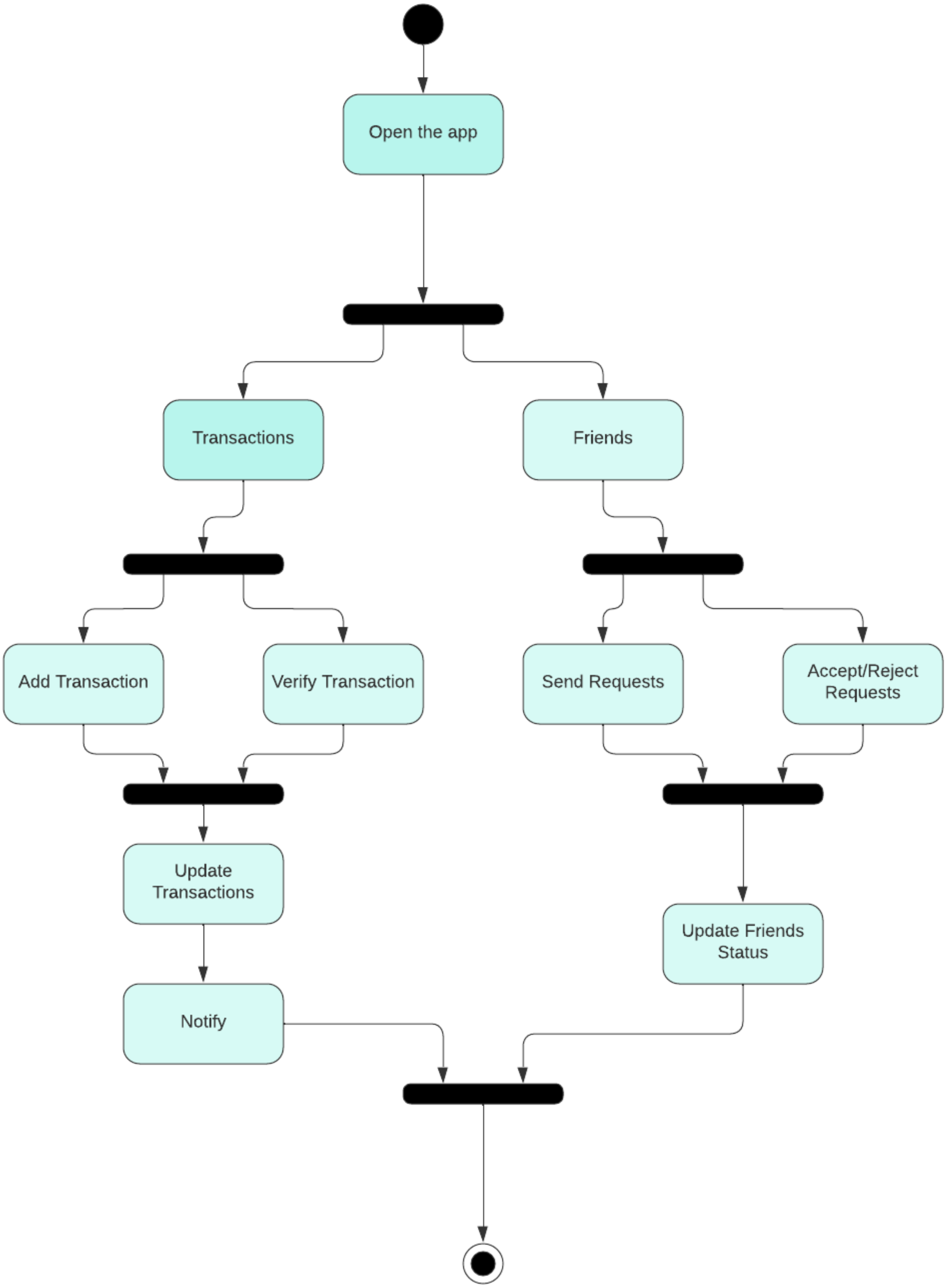
Description Priority	Users can edit their profile details like phone number, address, profile picture and others. Must have
Precondition	Logged in
Post Condition	None

Activity Diagrams

1. Login/Sign Up

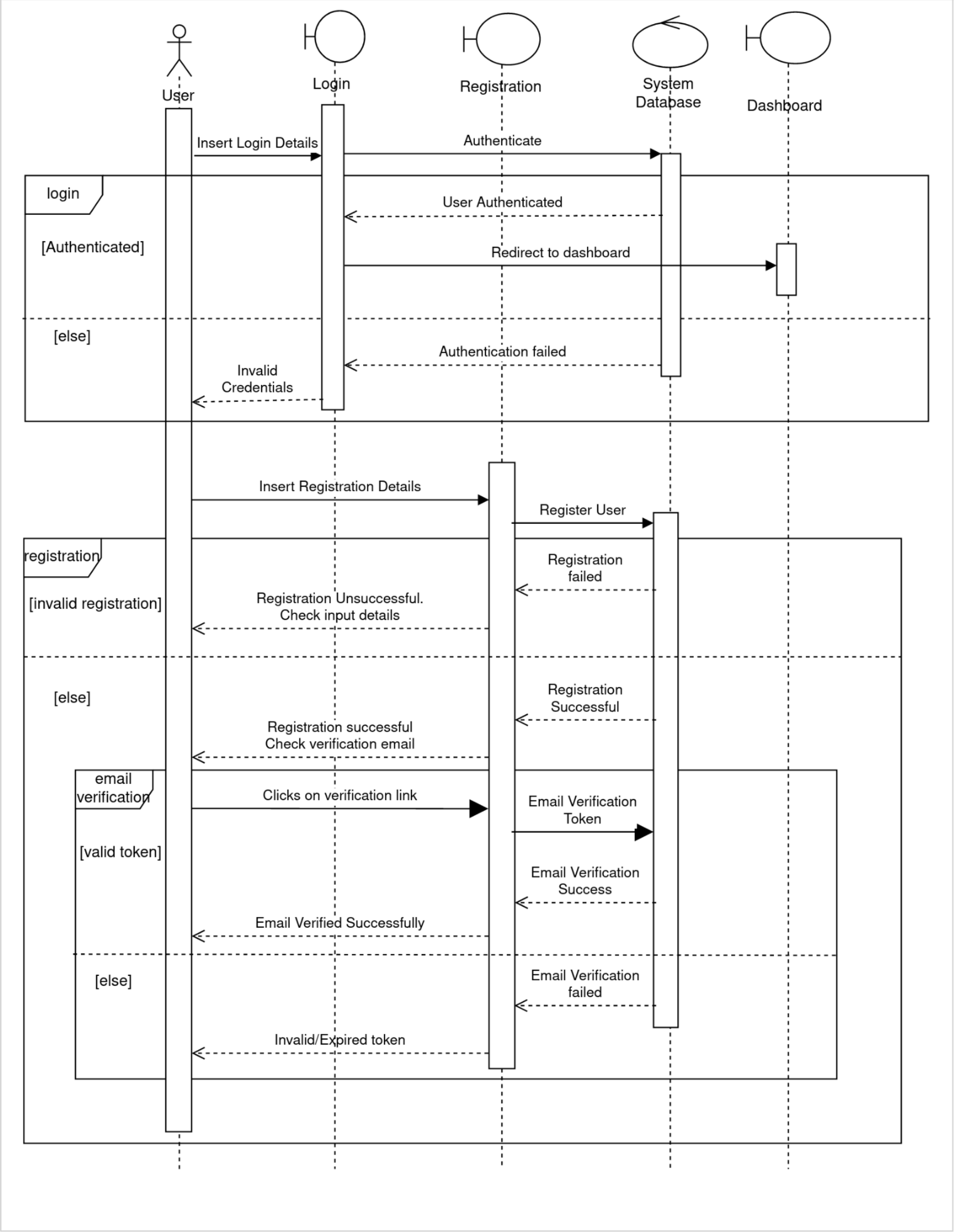


2. Transaction and Friends

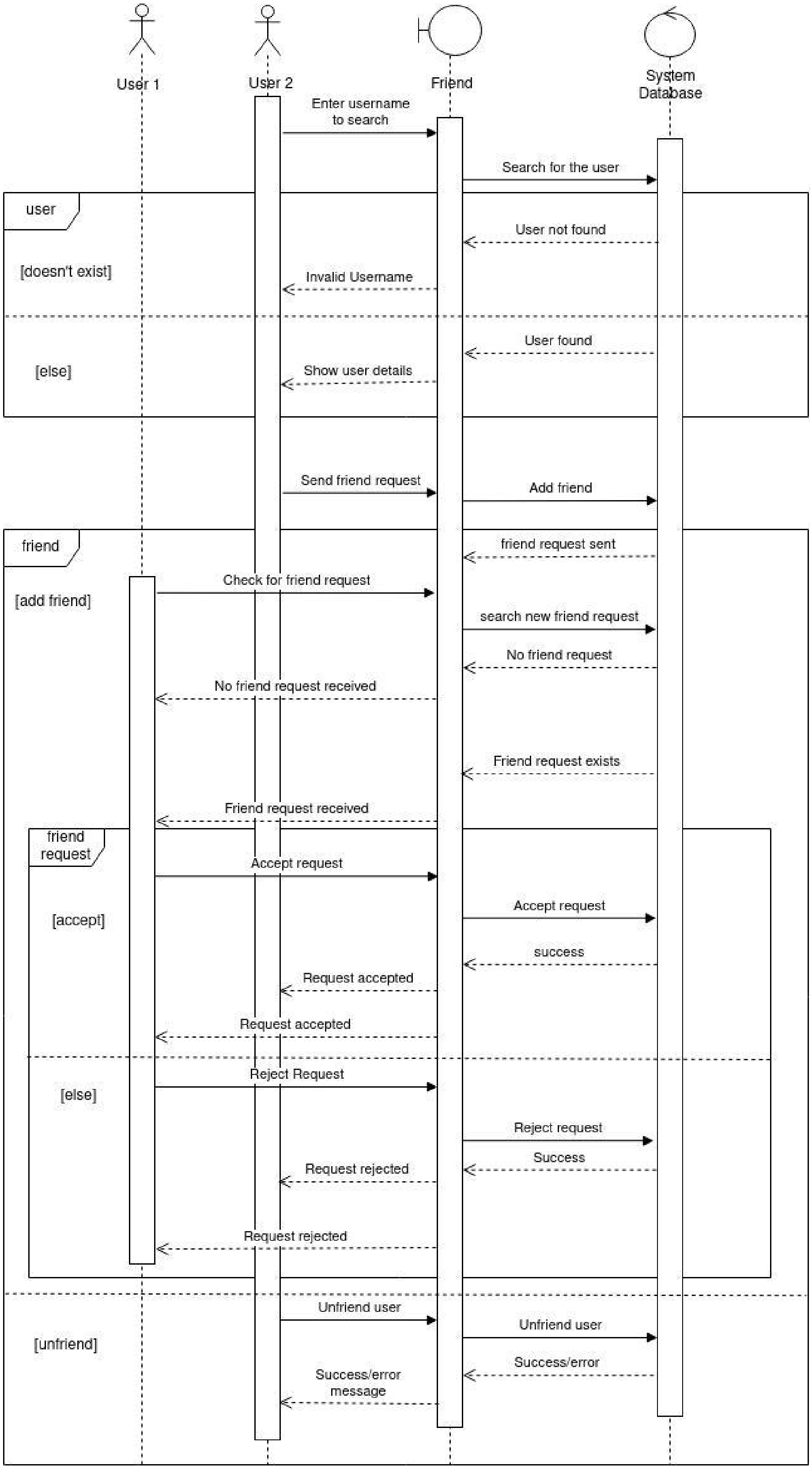


Sequence Diagrams:

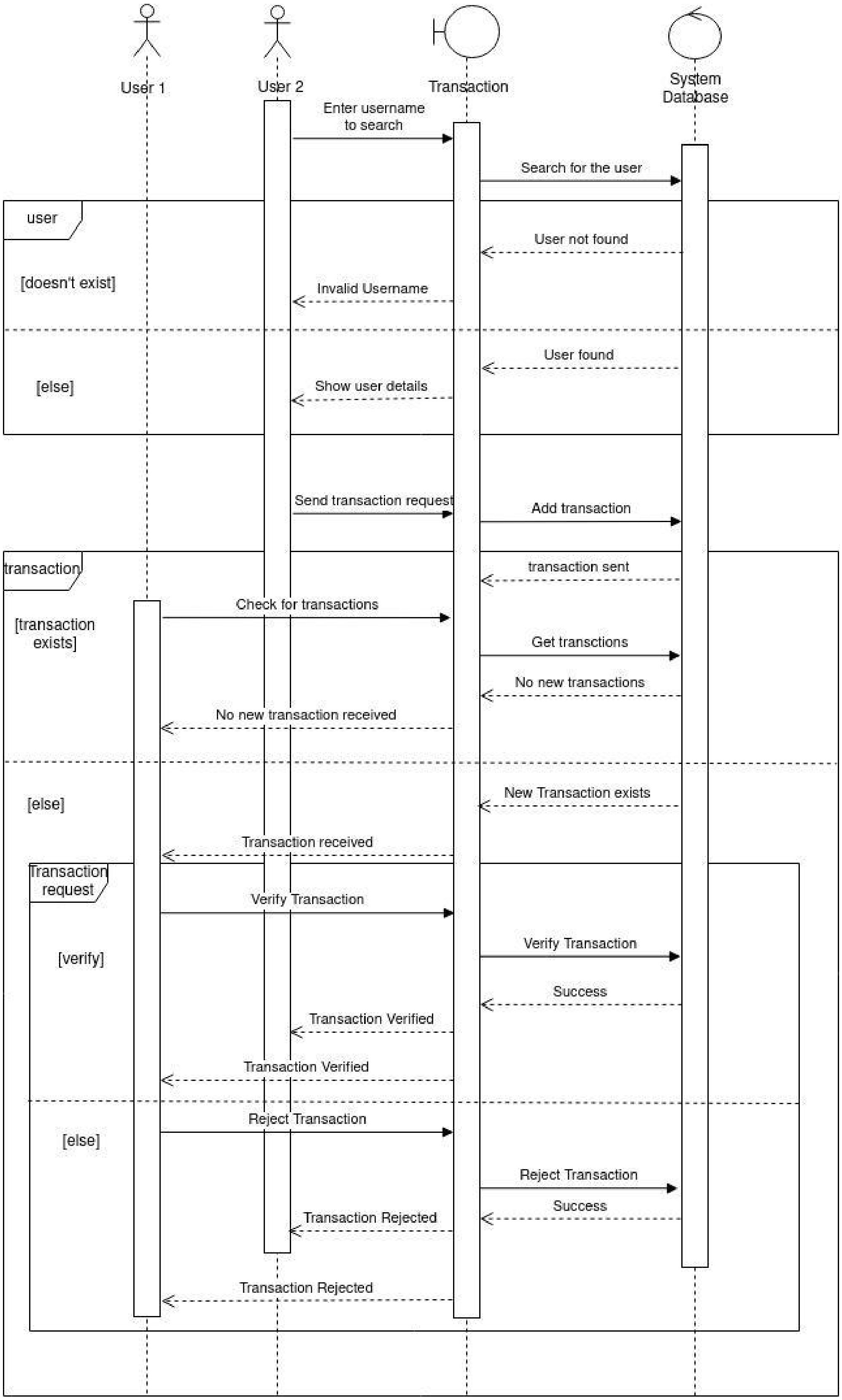
Sequence Diagram for User Authentication:



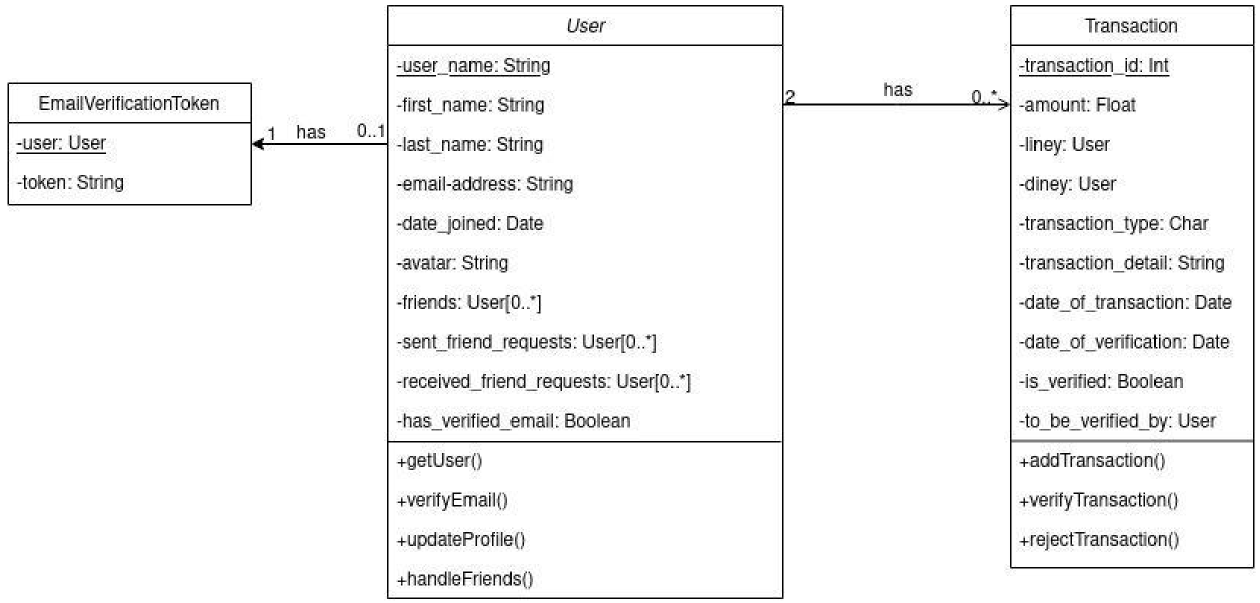
Sequence Diagram for Friends:



Sequence Diagram for Transactions:



Class Diagram:



Class Diagram Description:

- User: User model consists of following fields and methods:
    a. user_name: It is the primary key for the class User.
    b. first_name: Denotes the first name of the user.
    c. last_name: Denotes the last name of the user.
    d. email_address: Denotes the email address of the user.
    e. date_joined: Stores the information at which the user is registered in the system.
    f. avatar: Denotes the url of the user’s avatar.
    g. friends: It is an array of user_name from User model and denotes the users to which the user is friends with.
    h. sent_friend_requests: It is an array of user_name from User model and denotes the users to which the user has sent the friend request.
    i. received_friend_requests: It is an array of user_name from User model and denotes the users from which the user has received the friend request.

j. has_verified_email: This boolean field indicates whether the user's email is verified or not.
k. getUser(): This method returns all the details about the user.
l. verifyEmail(): It tries to verify the email of the user using the data from EmailVerificationToken model.
m. handleFriends(): It implements features like sending/deleting friend requests.

- Transaction: This model consists of following attributes and models:
  a. transaction_id: It is the primary key of this model.
  b. amount: This field represents the amount of transaction.
  c. liney: This holds the foreign key to the User model and represents the receiver of the transaction.
  d. diney: This holds the foreign key to the User model and represents the lender of the transaction.
  e. transaction_type: "D" represents debit and "C" represents credit.
  f. transaction_detail: It represents the remarks of the transaction.
  g. date_of_transaction: It represents the date in which a transaction occurred.
  h. date_of_verification: It represents the date in which a transaction was verified.
  i. is_verified: It indicates whether the transaction is verified or not.
  j. to_be_verified_by: It indicates who is responsible for the transaction verification.
  k. addTransaction(): This method adds a transaction entry.
  l. verifyTransaction(): This method confirms the transaction verification.
  m. rejectTransaction(): This method confirms the transaction rejection.
- EmailVerificationToken: This model has following attributes:
  a. user: This holds the foreign key to the User model to which this verification token belongs to.

b. token: This holds the actual verification token.