SE-I COURSE PROJECT (PHASES 1 & 2 COVER SHEET)

FOR TEAMS OF FIVE MEMBERS ONLY

Discussions Scheduled for Week 11 or 12 *(Specific dates TBA later).*

- Print 1 copy of this cover sheet and attach it to a printed copy of the documentation *(SRS, … etc.)*. You must submit softcopies of all your documents *(as PDFs)*; details will be announced later.
- Please write all your names in Arabic.
- Please make sure that your students' IDs are correct.
- Handwritten Signatures for the attendance of all team members should be filled in before the discussion.
- Please attend the discussion on time *(announced separately)*, late teams will lose 3 grades.

Project Name: _____

Team Information *(typed not handwritten, except for the attendance signature)*:

	ID [Ordered by ID]	Full Name [In Arabic]	Attendance [Handwritten Signature]	Final Grade
1	201900273	حسين مصطفي عبدالعال ابوزيد		
2	201900403	عبدالرحمن احمد فريج حامد		
3	201900298	رضوان احمد محمد عاطف		
4	201900021	احمد جمال منصور رمضان		
5	201900420	عبدالرحمن عبدالحليم سعد عبدالحليم		

Grading Criteria:

Items		Grade	Notes
Functional Requirements	**1.5**		
Non-Functional Requirements	**1.5**		
Use-Case Diagram(s) *including general use-cases for the system, and the detailed use-cases description*	**2**		
Activity Diagram(s)	**2**		

Database Specification *(ERD, Tables)*	**2**		
System Architecture – *including applied Architectural Pattern(s)*	**1**		
Sequence Diagram(s) *including* ***System Sequence Diagrams (SSDs)***	**2**		
Collaboration/Communication Diagram(s)	**2**		
Class Diagram *(3 versions)* **1) An initial version based on the requirements and Use-Case/Activity diagrams.** **2) An intermediate version based on the interaction diagrams.** **3) A final version, after applying the design pattern(s) and any other modifications.**	**6**		
Package Diagram(s)	**1**		
2 Mandatory Design Pattern Applied *(Including a typed description)*	**4**		

Teaching-Assistant's Signature: _____

_____
25

On Road Vehicle Breakdown Assistance

Project requirements:

First: Functional requirements:

- Admin requirements:

1-Admin can log in to view the system.

2-Admin can view registered mechanics with their details

3-Admin has the access to allow or block mechanics

4-Admin can view all registered users' details

5-Admin can view all the feedback given by a user or a mechanics

- User

1- Users can register with all their details

2- Registered users can log in with all their credential

3- Users can search for local mechanics on the basis of their location

4- Users can send a request to a certain mechanic

5- Users can give their feedback accordingly.

-Mechanic

1- Mechanics can register with all their details

2- Registered mechanics with premission from the admin to login will be able to login

3- Mechanics can view the requests sent my the users

4- Mechanics can provide their own feedback

Second: NonFunctional requirements:

1- The system must be secure
   - the data collected from registertion and requests must be secured from any breach

2- The system must be easy to access and view
   - Registered users and mechanics must face no difficulties while accessing the system

3-The system must be user-friendly
   - Users must face no difficulties while using any part of the system

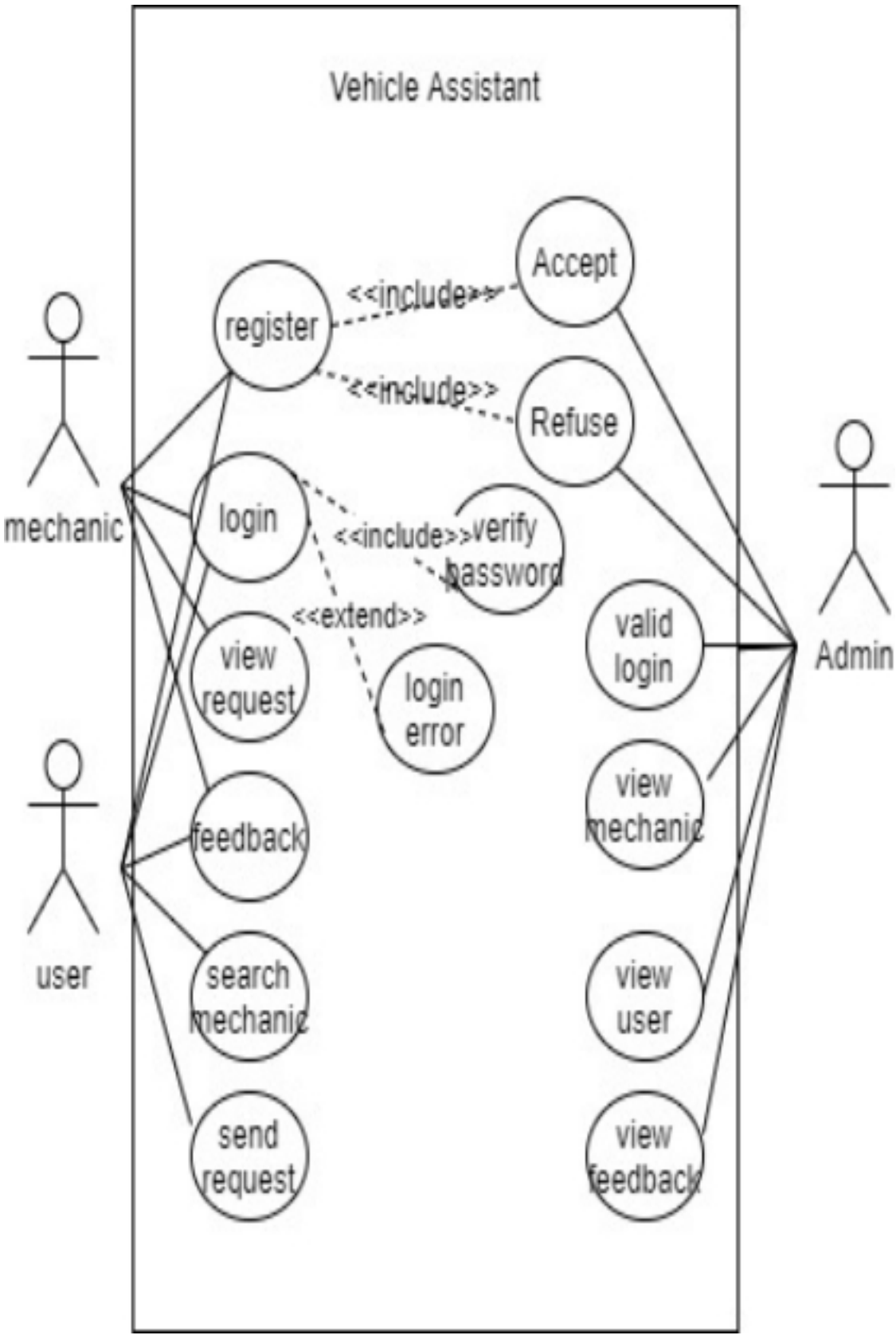
4- The system must be reliable
   - the system must face no drawbacks that affect the service in production

5- The system must be flexible
   - The system must be flexible with any modifications due to customers' needs

6- The system must reduce the manual work
   - customer should rely on the system for the previous Manual work(searching for mechanics physicly)

7- The system must collect users and mechanics feedback for improvements

- Use-Case Diagram



Use case description for each method in the use case diagram:

Admin login use-case:

Identifier and Name: Admin's username
Initiator: Admin
Goal: login
Precondition: user must exist in database
Postcondition: user logged in successfully
Main success scenario:
1. user choose to make login
2. user Writes his email and password
3. user clicks on login

Extensions:
3.a. his email or password isn't correct
3.a.1. System alert him to write the right email and right password
3.b. the user is an admin
3.b.1. the system opens admin dashboard

View user or mechanic:

Identifier and Name: user or mechanic
Initiator: Admin
Goal: display details about user or mechanic info
Precondition: Admin should be logged in and the user should be on the system
Postcondition: Admin knows the details about user or mechanic
Main success scenario:
1. Admin logged in.
2. Enter the ID of the user (user or mechanic).
3. Click on get details about user or mechanic by ID, the database check the id and If it exists or not
4. The database return the details about user or mechanic.

View user Feedback:

Identifier and Name: User give Feedback
Initiator: Admin
Goal: Get a summarized Feedback about the mechanic
Precondition: Admin must be log in to the system.

Postcondition: Admin knows a summarized Feedback about the mechanic

Main success scenario:

1. Admin logged in.
2. Admin selects the “Feedback” option.
3-System displays a summarized Feedback about that mechanic.

View mechanic Feedback:

Identifier and Name: mechanic can provide their feedback

Initiator: Admin

Goal: Get a summarized Feedback from the mechanic

Precondition: Admin must be log in to the system.

Postcondition: Admin knows a summarized Feedback from the mechanic

Main success scenario:

1. Admin logged in.
2. Admin selects the “Feedback” option.
3-System displays a summarized Feedback from that mechanic.

User register use-case:

Identifier and Name: User’s info

Initiator: user

Goal: Registered successfully and have an Email in system

Precondition: user must exist in database

Postcondition: user registered and his information is added to database

Main success scenario:

1. user choose to make register
2. user Writes his all information
3. user clicks on submit

Extensions:

3.a. invalid register

3.a.1. System alert him to write all infomation

3.b. User information already Existing in database

3.b.1. System alert him to go to login

User login use-case:

Identifier and Name: User’s username

Initiator: User

Goal: login

Precondition: user must exist in database

Postcondition: user logged in successfully
Main success scenario:
1. user choose to make login
2. user Writes his email and password
3. user clicks on login
Extensions:
3.a. his email or password isn't correct
3.a.1. System alert him to write the right email and right password
3.b. the user is an customer
3.b.1. the system opens customer dashboard

Search mechanics:

Identifier and Name: User search
Initiator: User
Goal: find mechanic
Precondition: The user is looking for a mechanic
Postcondition: The user finded a mechanic
Main success scenario:
1. User logged in.
2. User selects the "search" option.
3. Customer write his location to find mechanic in same location
Extensions:
1.a. his email or password isn't correct
1.a.1. System alert him to write the right email and right password
3.b. the system fails to search
3.b.2. doesn't find a mechanic in this location

Send request:

Identifier and Name: User's request
Initiator: User
Goal: Send request
Precondition: The user finded a mechanic in database
Postcondition: The user send request to mechanic
Main success scenario:
1. User logged in.
2. User selects the "request" option.
3. Customer send his request to Database
Extensions:
1.a. his email or password isn't correct
1.a.1. System alert him to write the right email and right password

3.b. the system fails to send request
3.b.1. user choose to cancel request
3.b.2. mechanic refuse this request

Mechanic register use-case:

Identifier and Name: mechanic's info
Initiator: mechanic
Goal: Registered successfully and have an Email in system
Precondition: mechanic must exist in database
Postcondition: mechanic registered and his information is added to database
Main success scenario:
1. mechanic choose to make register
2. mechanic Writes his all information
3. user clicks on submit
4- admin allow or block a mechanics

Extensions:
3.a. invalid register
3.a.1. System alert him to write all infomation
3.b. User information already Existing in database
3.b.1. System alert him to go to login
4.a. admin allow a mechanic
4.a.1 mechanic now in system
4.b. admin block a mechanic
4.b.1 mechanic out the system

Mechanic Login use-case:

Identifier and Name: Mechanic's username
Initiator: mechanic
Goal: login
Precondition: mechanic must exist in database
Postcondition: mechanic logged in successfully
Main success scenario:
1. mechanic choose to make login
2. mechanic Writes his email and password
3. mechanic clicks on login
4- admin allow or block a mechanics

Extensions:
3.a. his email or password isn't correct
3.a.1. System alert him to write the right email and right password
3.b. the user is an mechanic

3.b.1. the system opens mechanic dashboard
4.a. admin allow a mechanic
4.a.1 mechanic now in system
4.b. admin block a mechanic
4.b.1 mechanic out the system

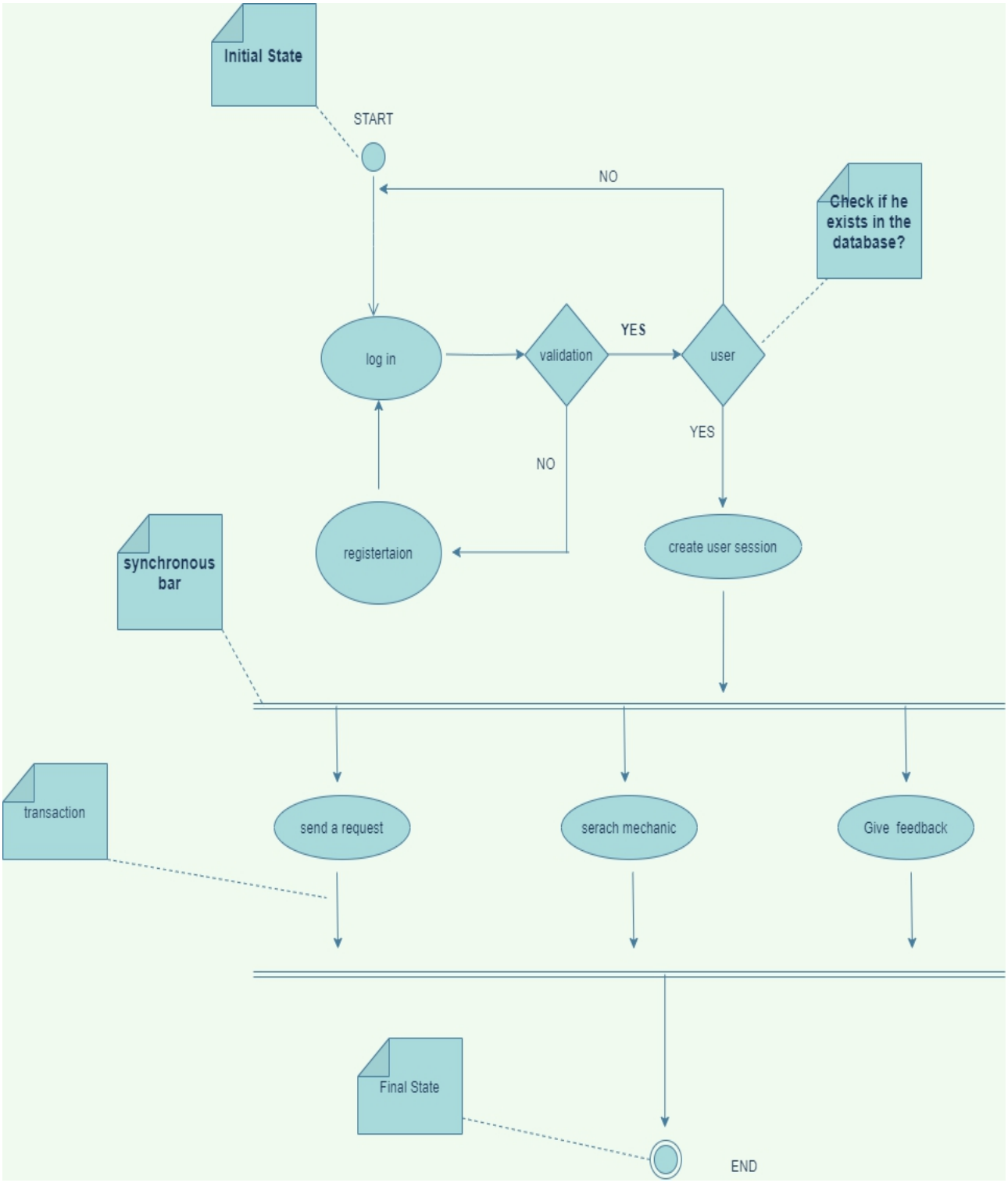
View User request:

Identifier and Name: mechanic view user's request
Initiator: mechanic
Goal: Show request
Precondition: mechanic must be log in to the system.
Postcondition: mechanic view request from the user
Main success scenario:
1. mechanic logged in.
2. mechanic selects the "view request" option.
3- mechanic reply to the request

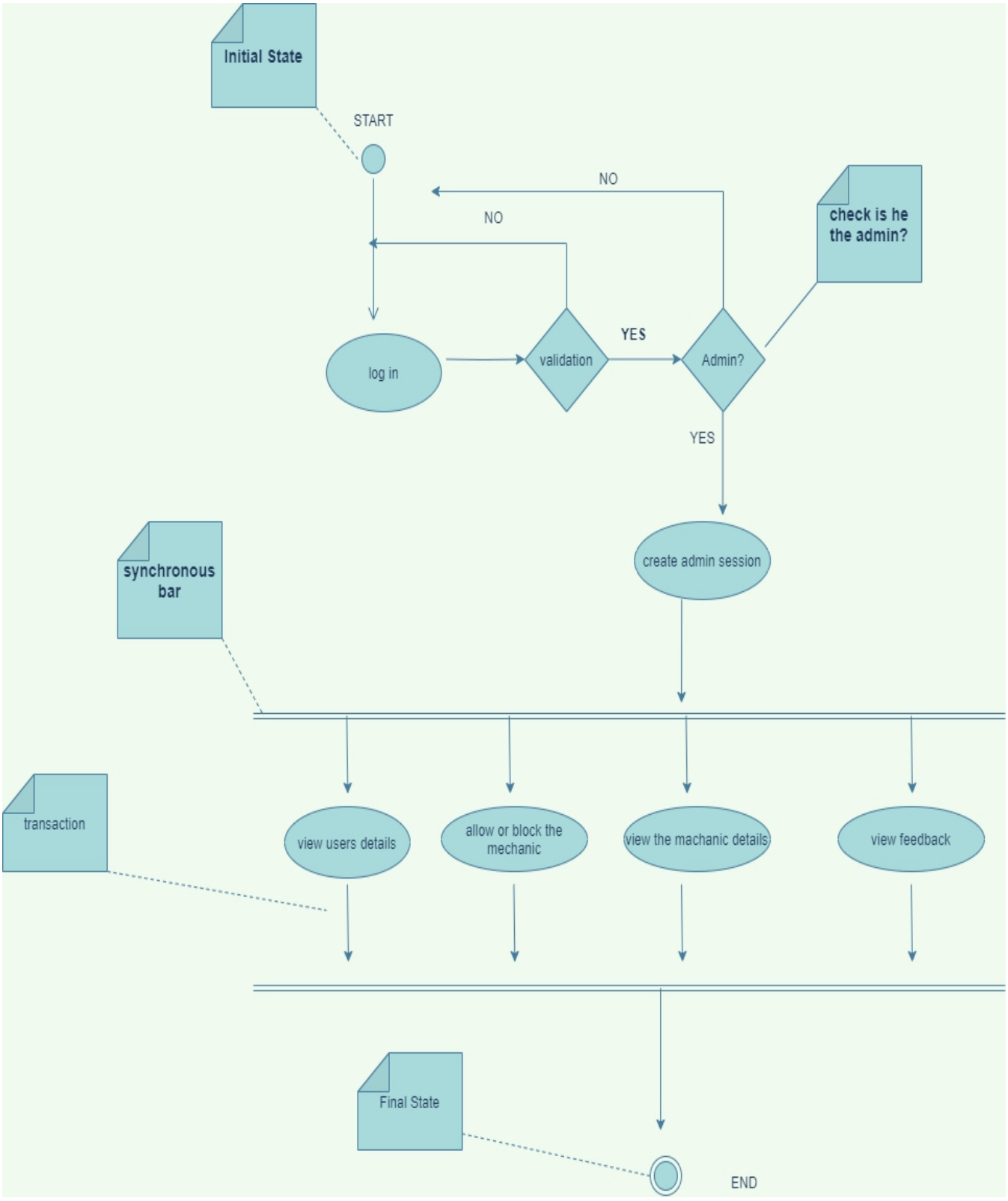
Extensions:
1.a. his email or password isn't correct
1.a.1. System alert him to write the right email and right password
3.a. if mechanic accept the request
3.a.1.assign request in database
3.b.mechanic refuse the request

Forth: Activity Diagram

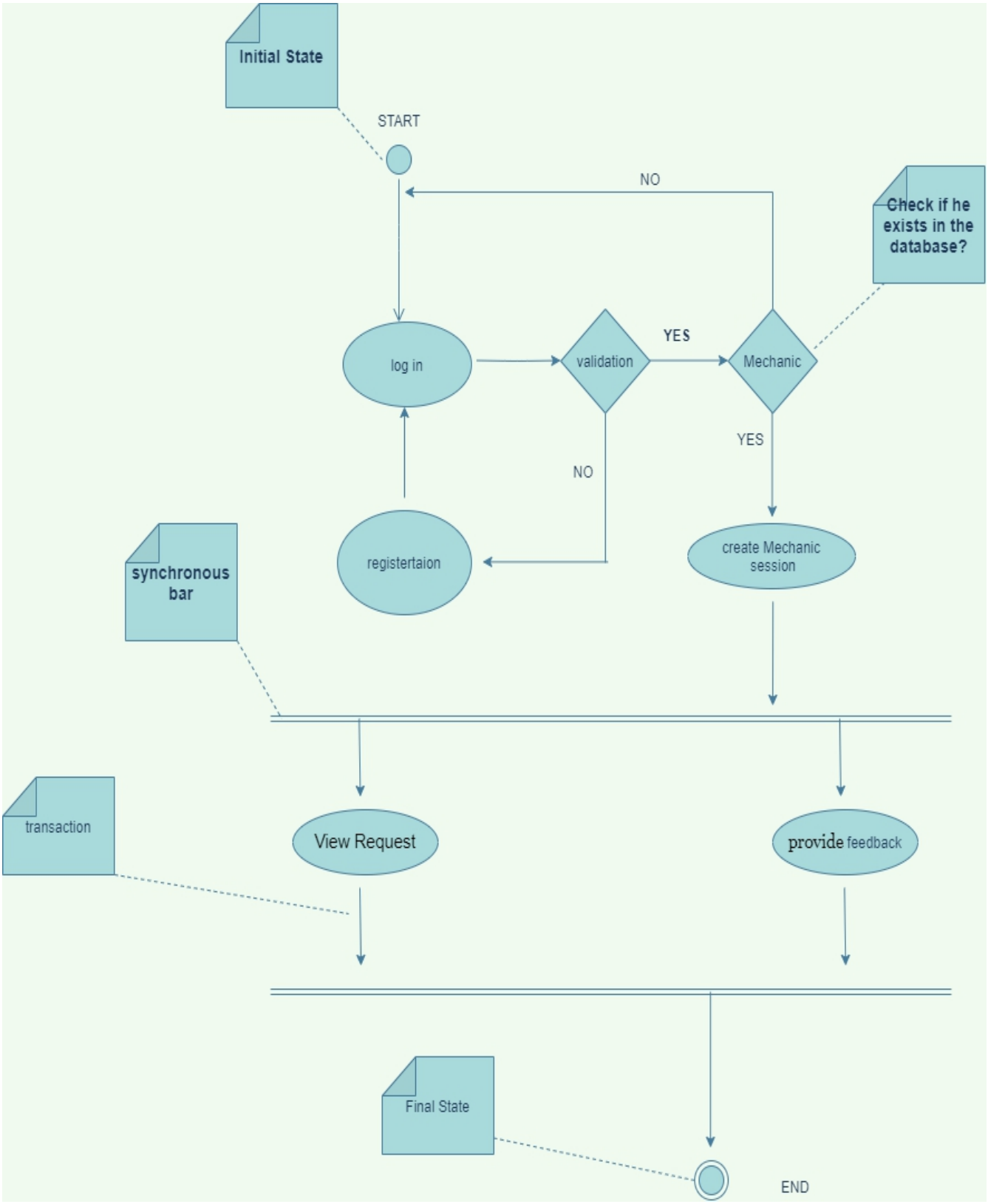
User:



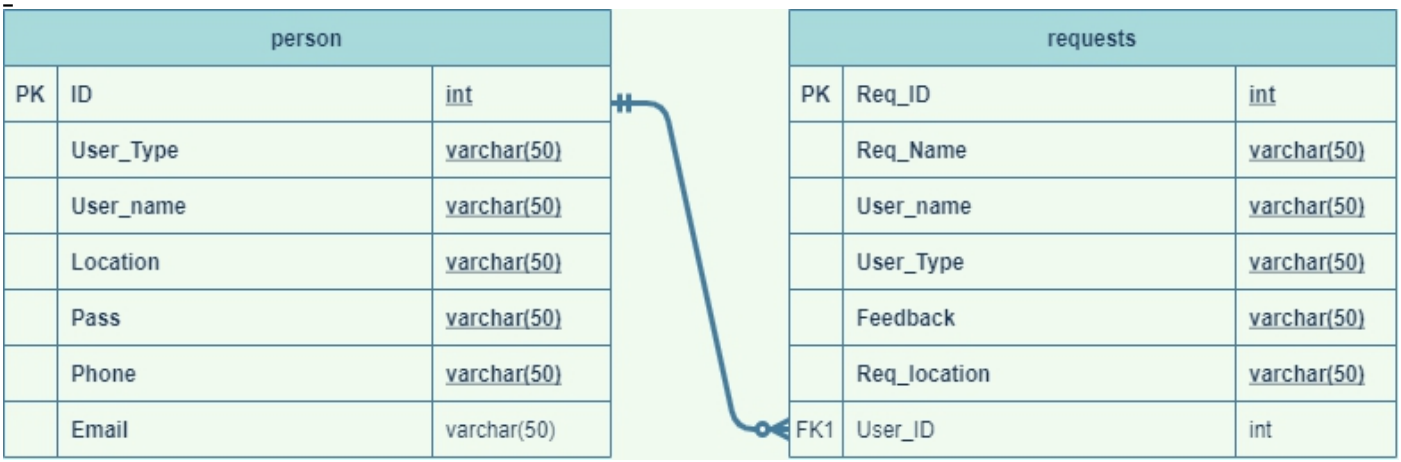
Admin:


Initial State
START
NO
NO
check is he the admin?
log in
validation
YES
Admin?
YES
create admin session
synchronous bar
transaction
view users details
allow or block the mechanic
view the machanic details
view feedback
Final State
END

Mechanic:


Initial State
START
NO
Check if he exists in the database?
log in
validation
YES
Mechanic
YES
NO
registertaion
create Mechanic session
synchronous bar
transaction
View Request
provide feedback
Final State
END

Database Specification :



person		
PK	ID	int
	User_Type	varchar(50)
	User_name	varchar(50)
	Location	varchar(50)
	Pass	varchar(50)
	Phone	varchar(50)
	Email	varchar(50)

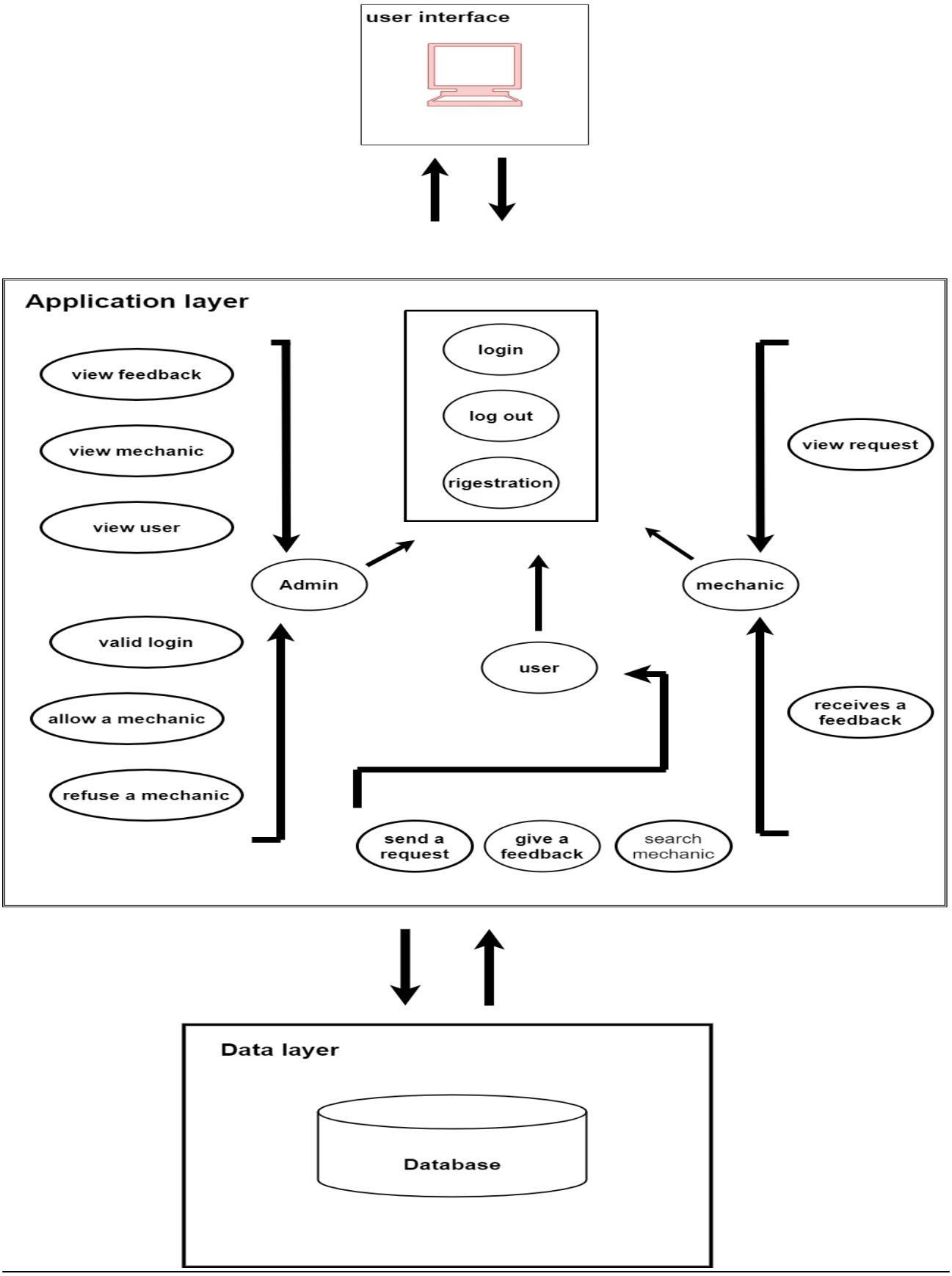
requests		
PK	Req_ID	int
	Req_Name	varchar(50)
	User_name	varchar(50)
	User_Type	varchar(50)
	Feedback	varchar(50)
	Req_location	varchar(50)
FK1	User_ID	int

Person	
int	ID
varchar(50)	User_Type
varchar(50)	User_name
varchar(50)	Location
varchar(50)	Pass
varchar(50)	Phone
varchar(50)	Email

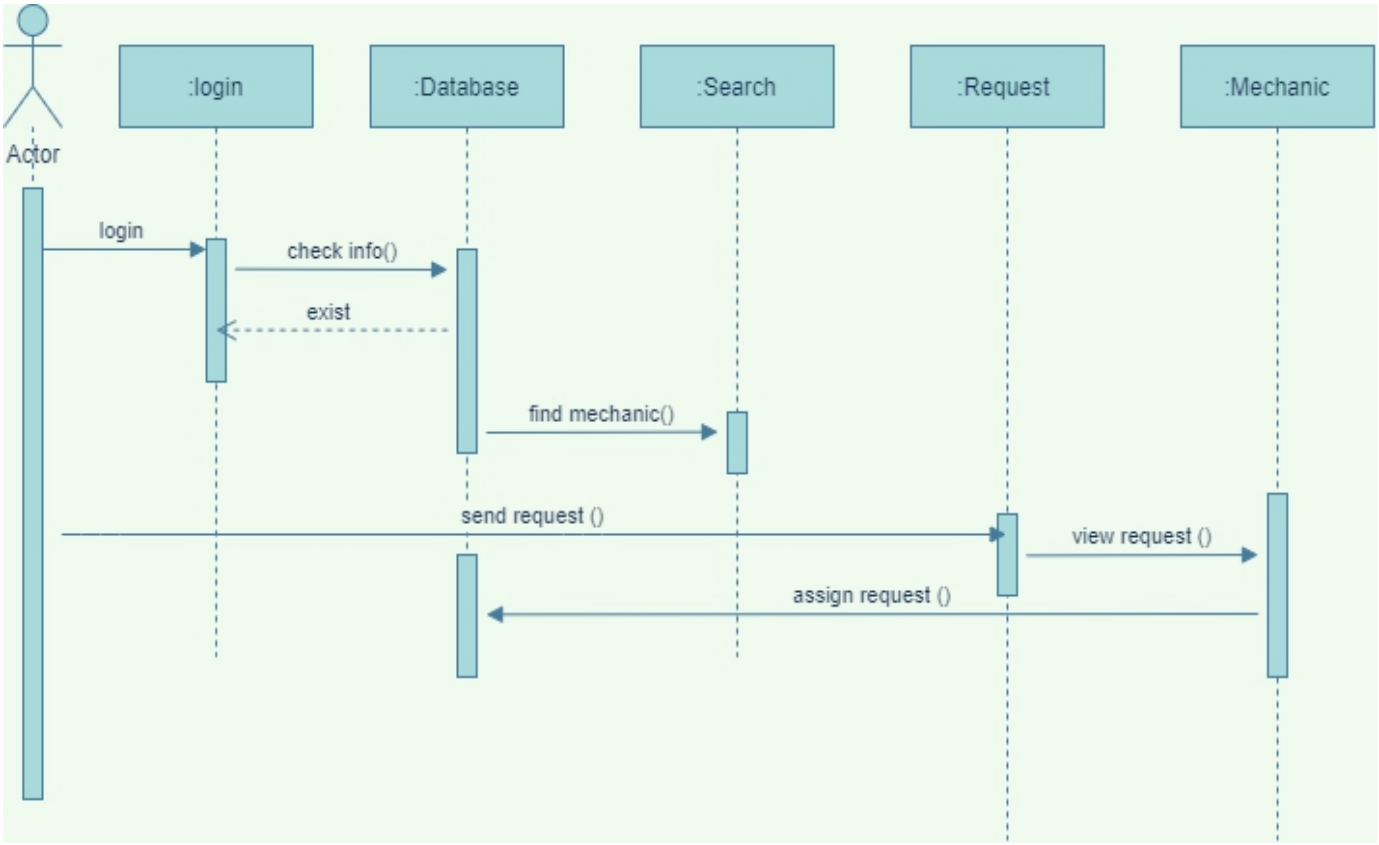
requests	
int	Req_ID
varchar(50)	Req_Name
varchar(50)	User_name
varchar(50)	User_Type
varchar(50)	Feedback
varchar(50)	Req_location
int	User_ID

System Architecture :

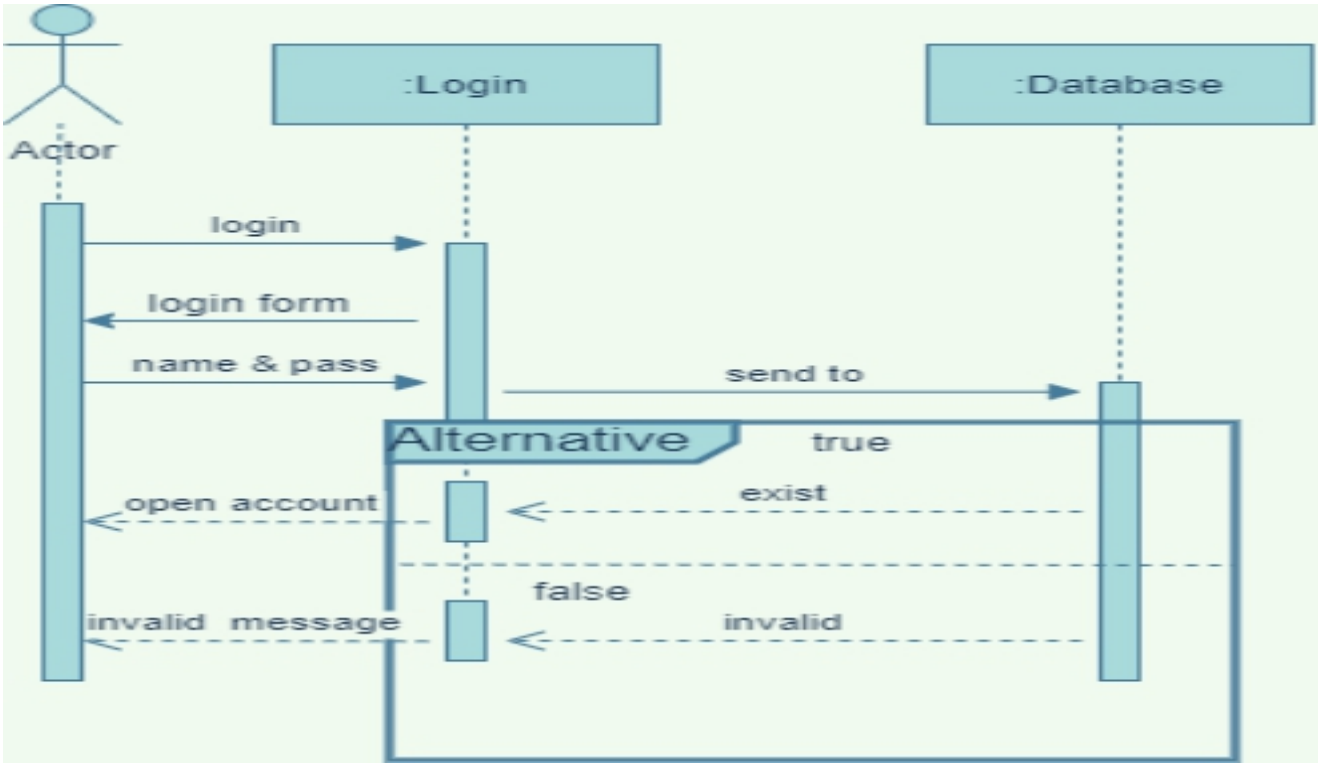
system architecture



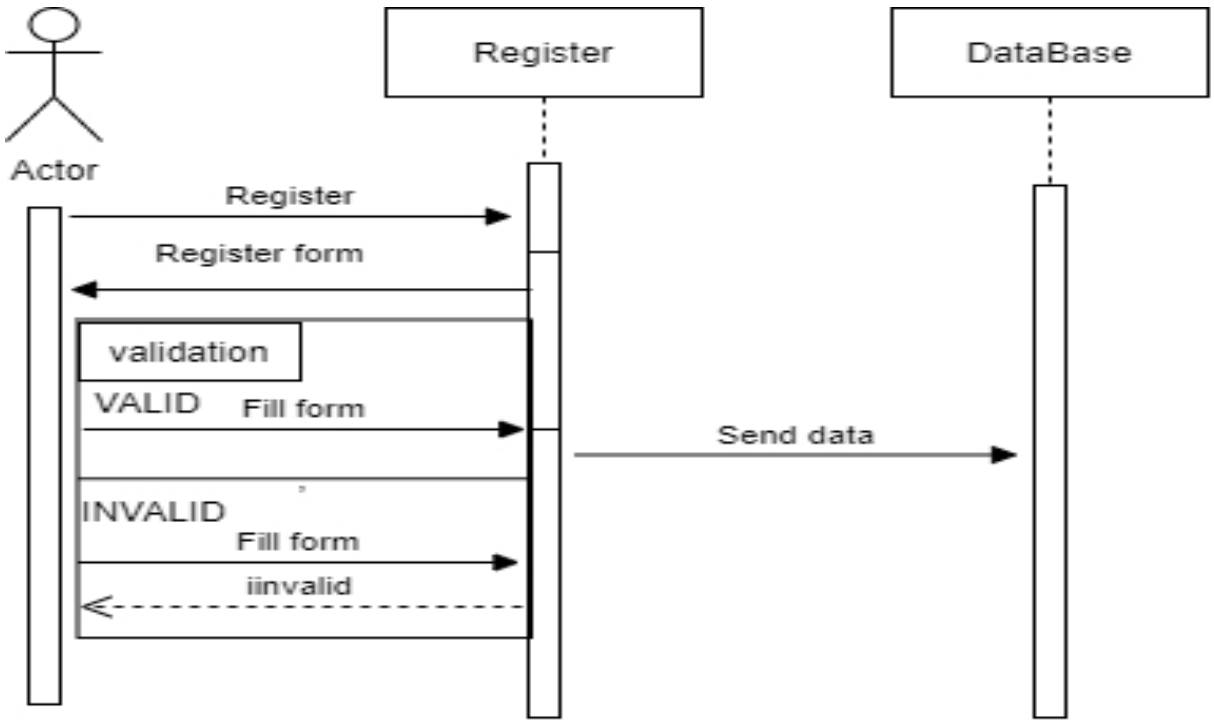
Sequence Diagram :


Actor
:login
:Database
:Search
:Request
:Mechanic
login
check info()
exist
find mechanic()
send request ()
view request ()
assign request ()

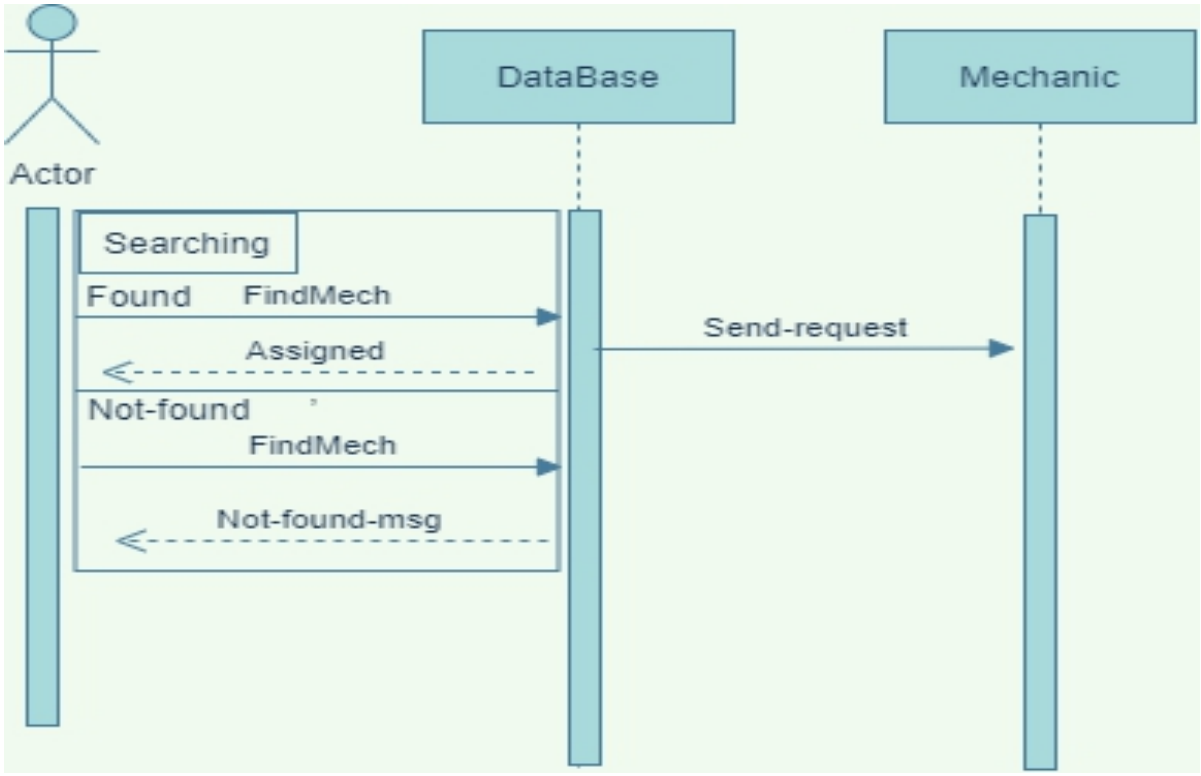
- Login


Actor
:Login
:Database
login
login form
name & pass
send to
Alternative
true
open account
exist
false
invalid message
invalid

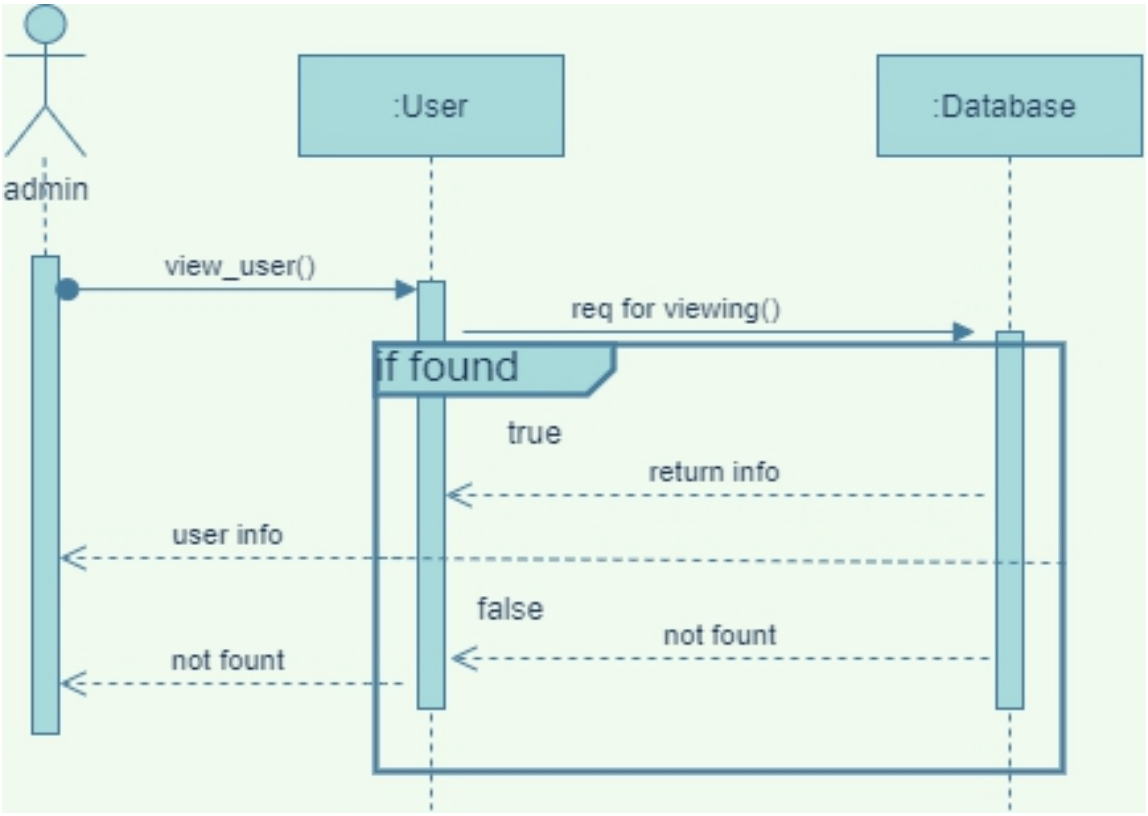
- Register



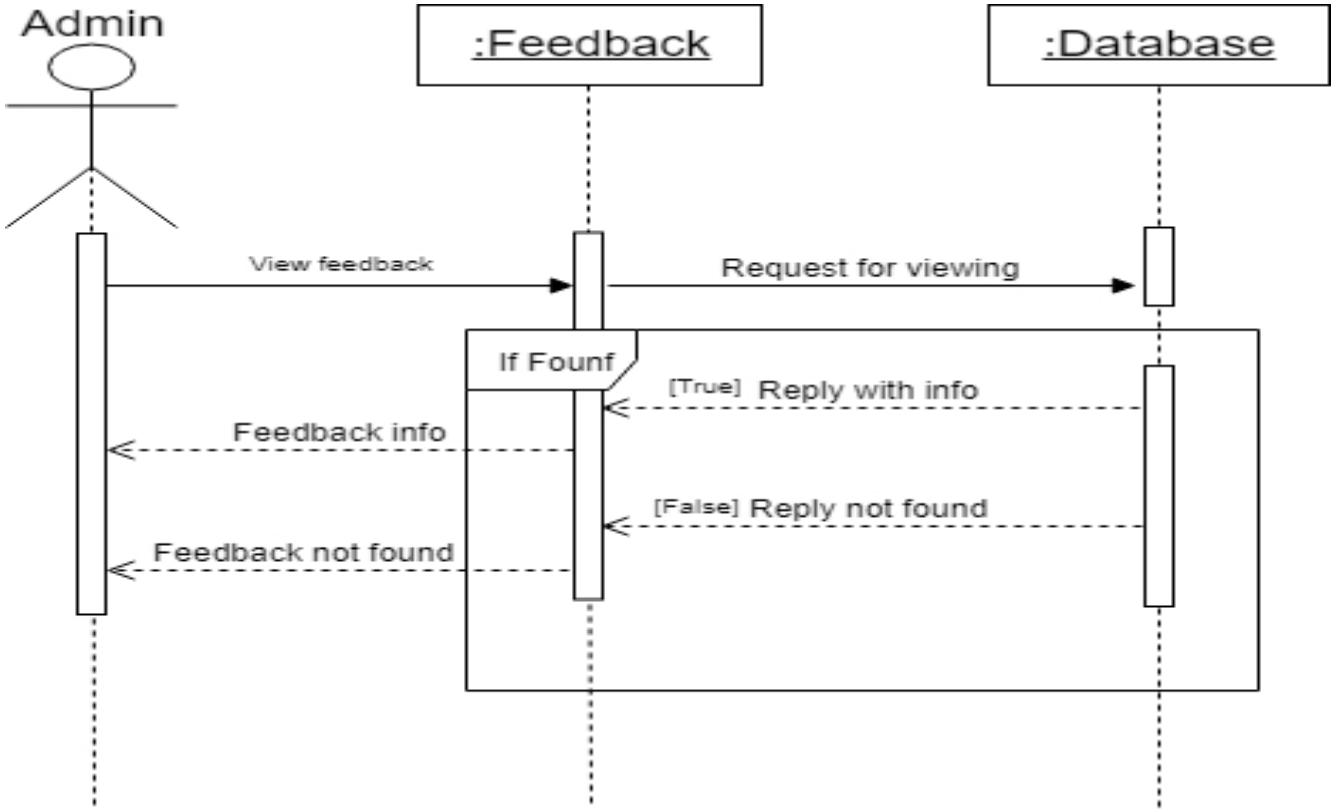
- Search



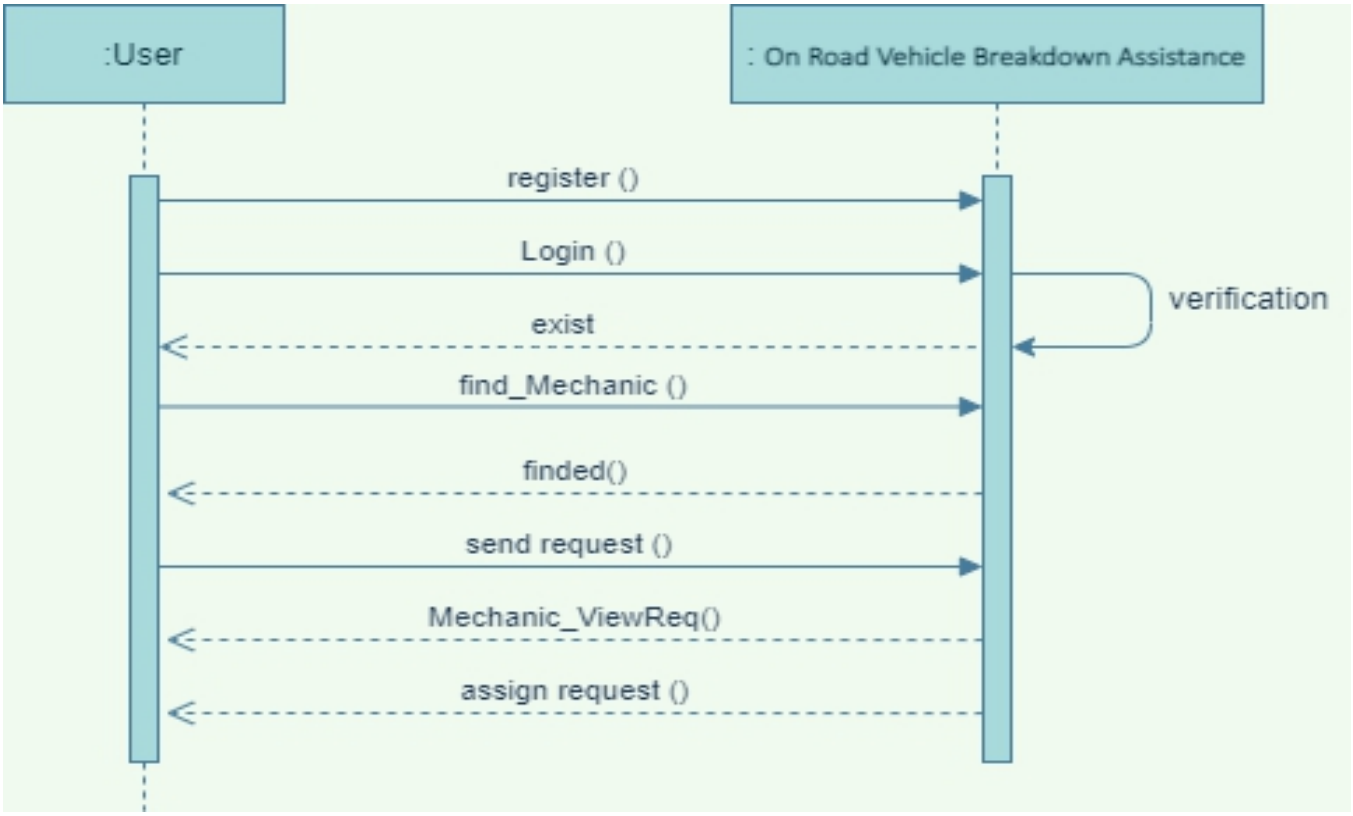
-View



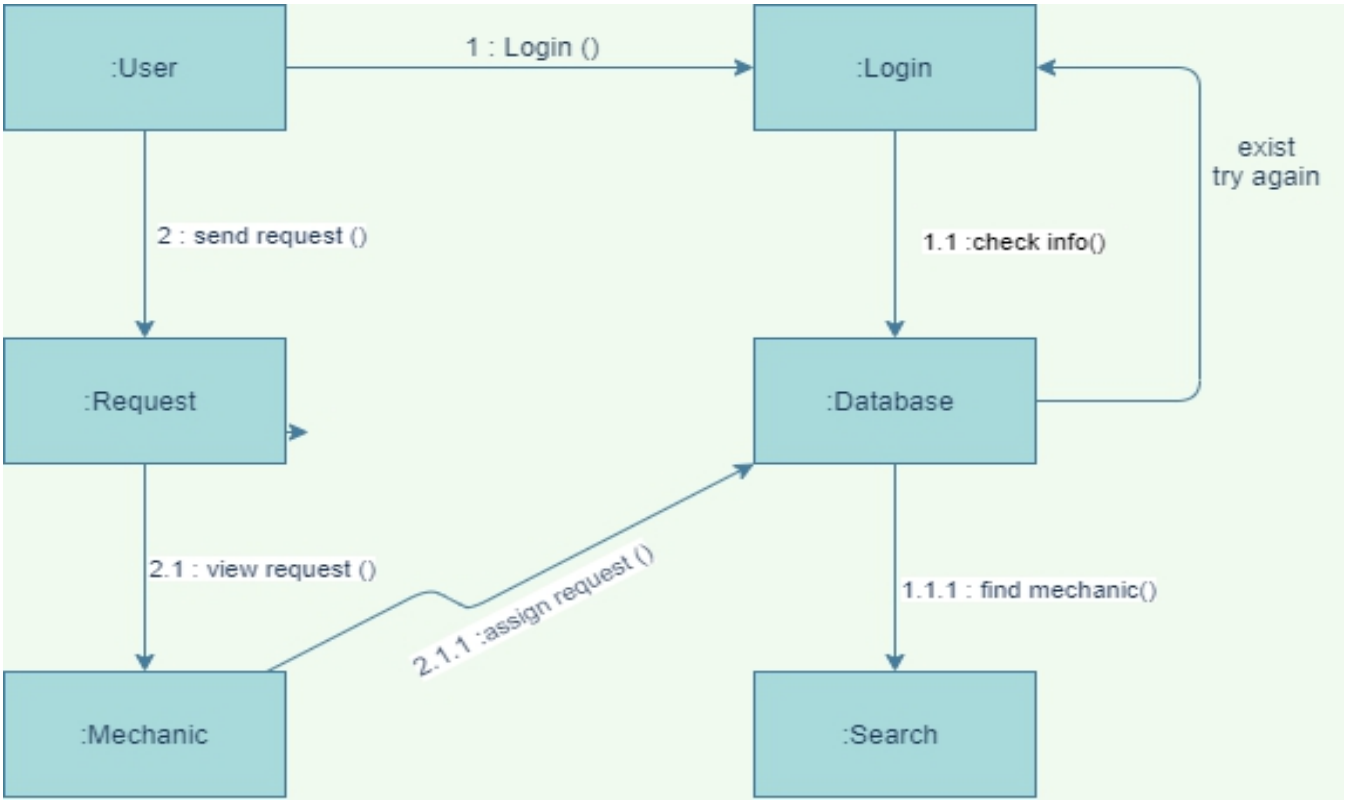
-Feedback



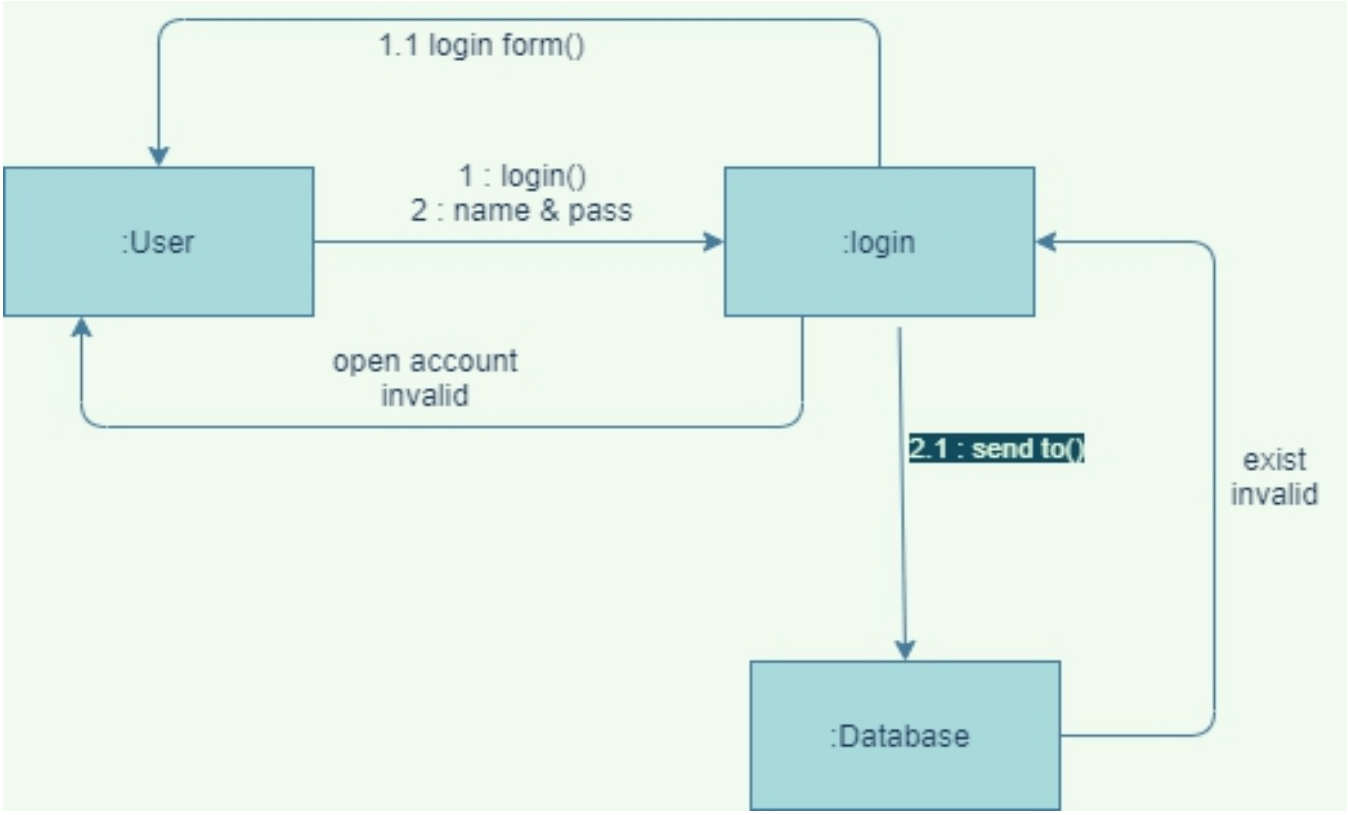
System Sequence Diagrams



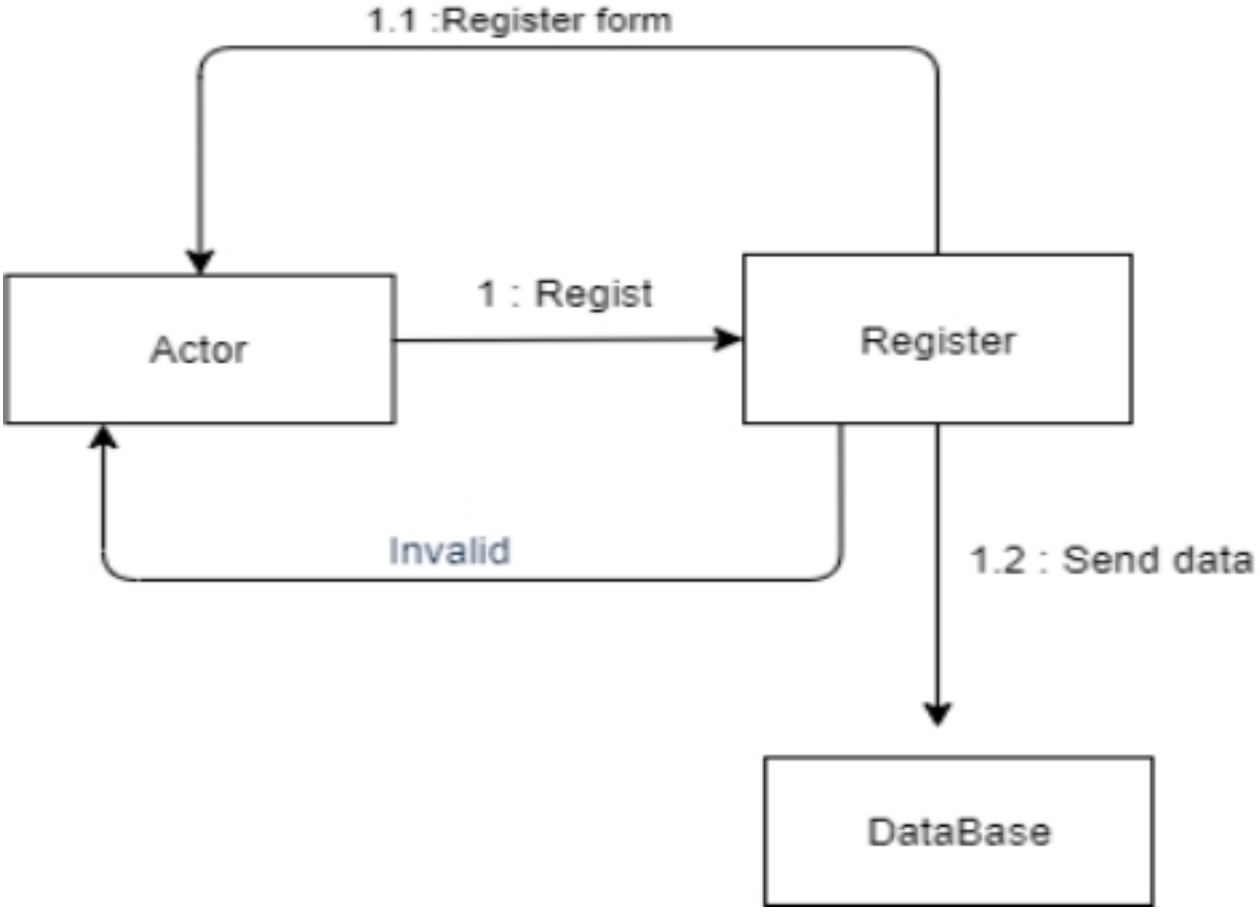
Collaboration/Communication Diagram



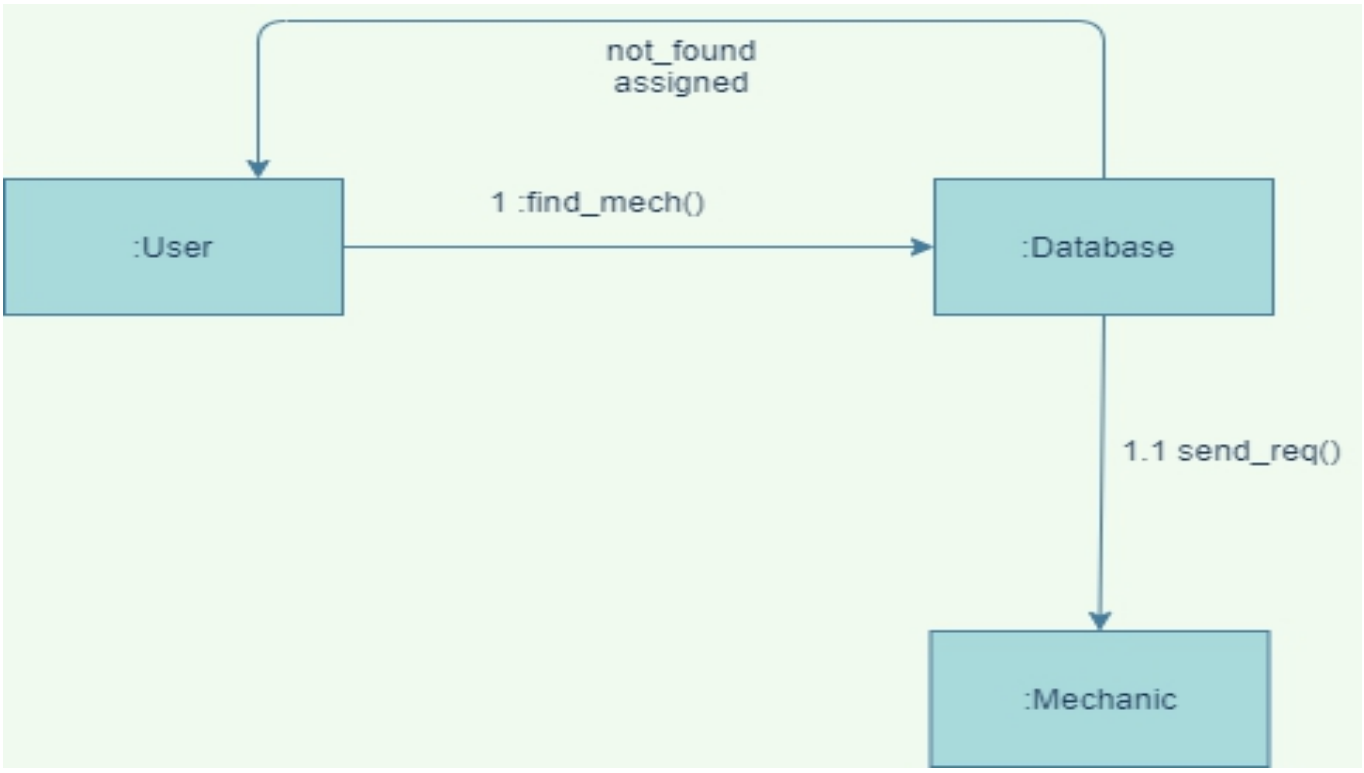
- Login



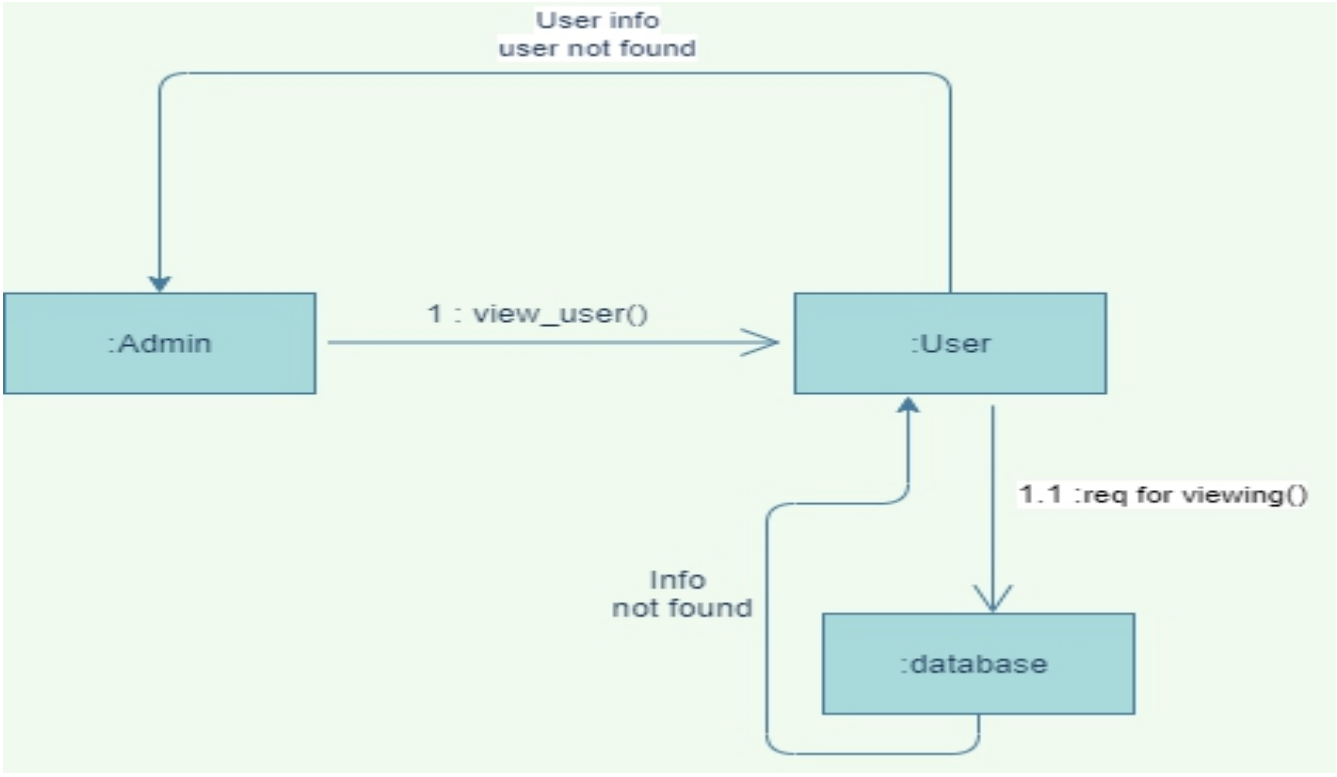
- Register



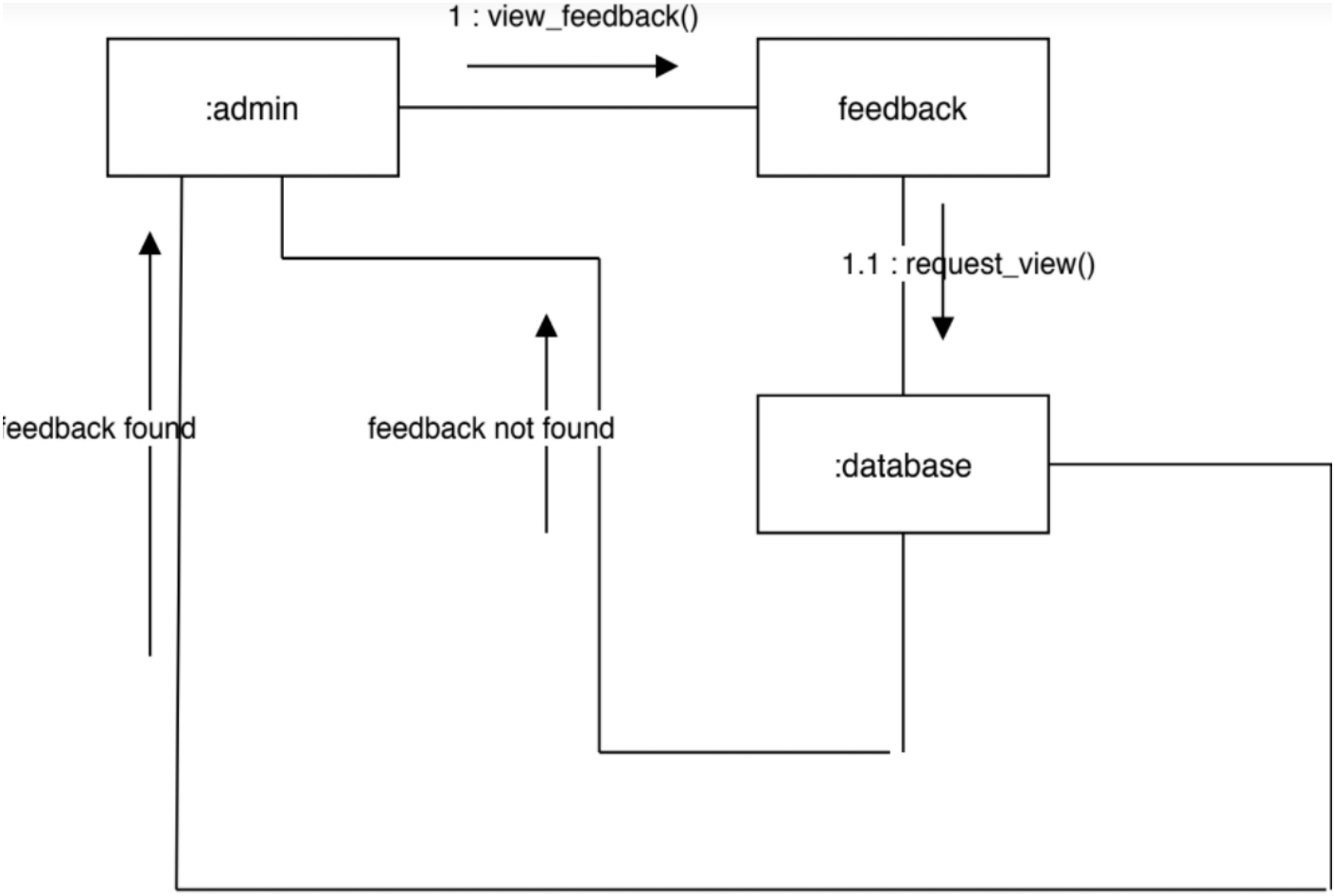
- Search


not_found
assigned
1 :find_mech()
:User
:Database
1.1 send_req()
:Mechanic

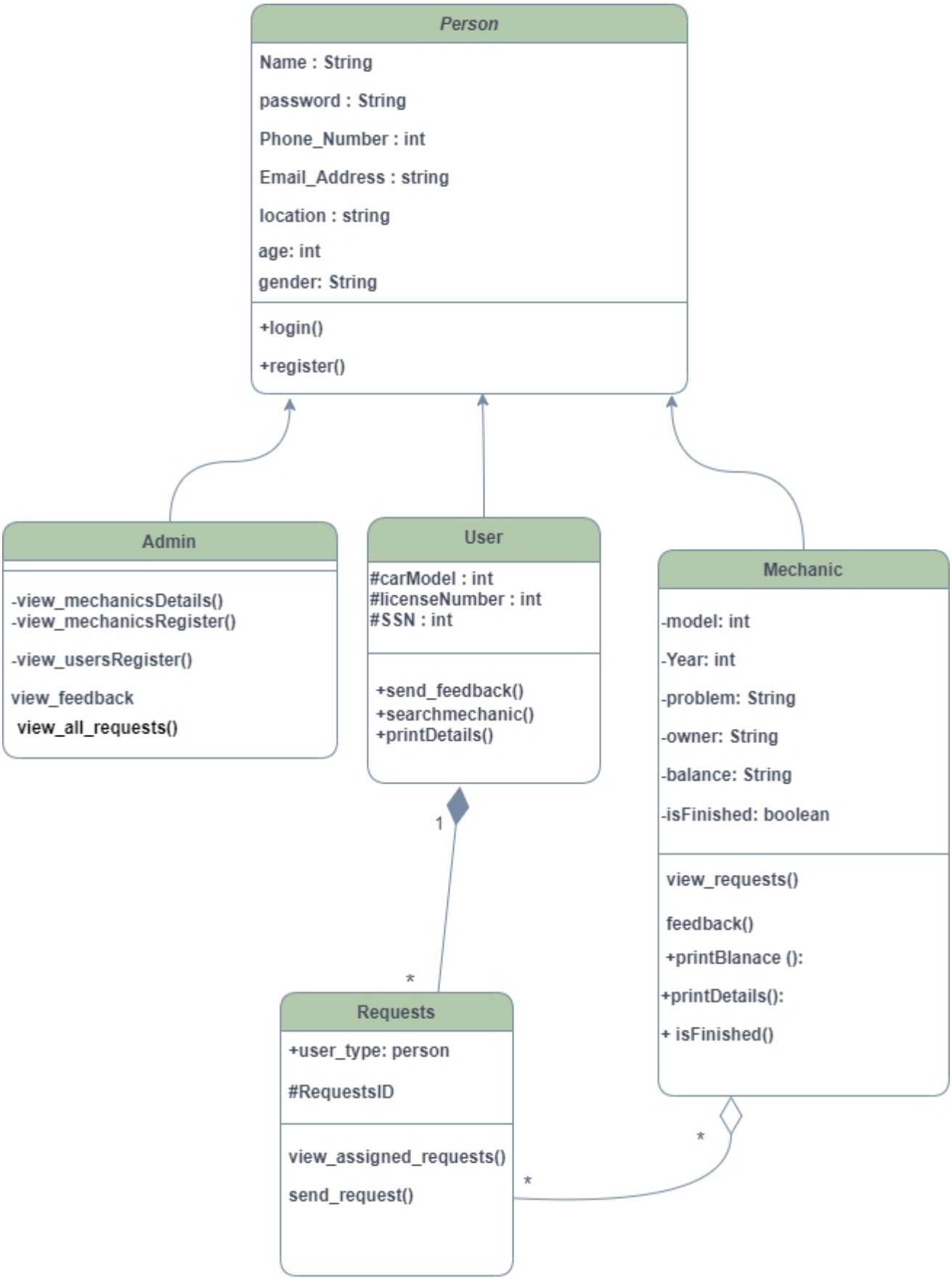
-View


User info
user not found
:Admin
1 : view_user()
:User
1.1 :req for viewing()
Info
not found
:database

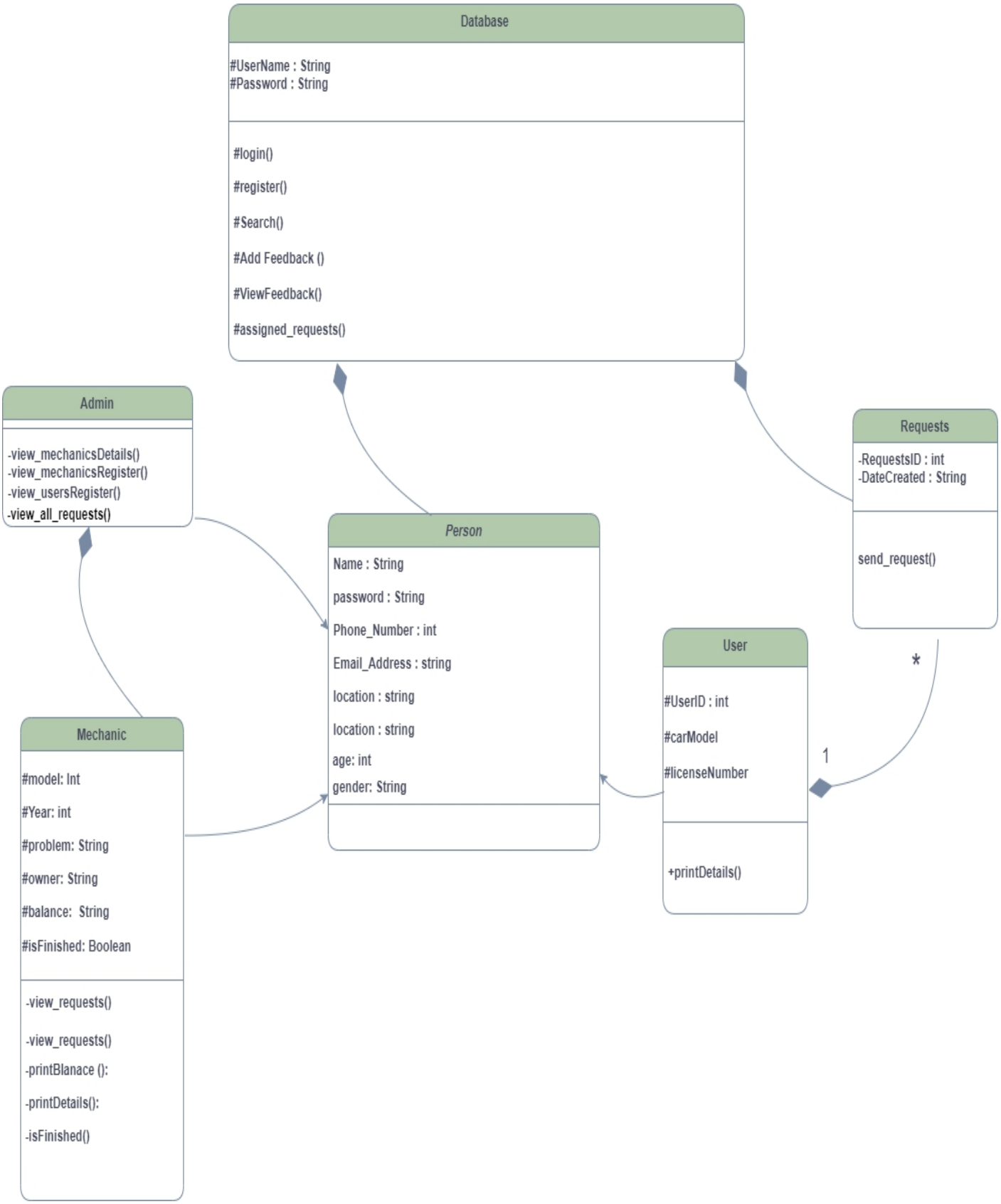
-Feedback


1 : view_feedback()
:admin
feedback
1.1 : request_view()
feedback found
feedback not found
:database

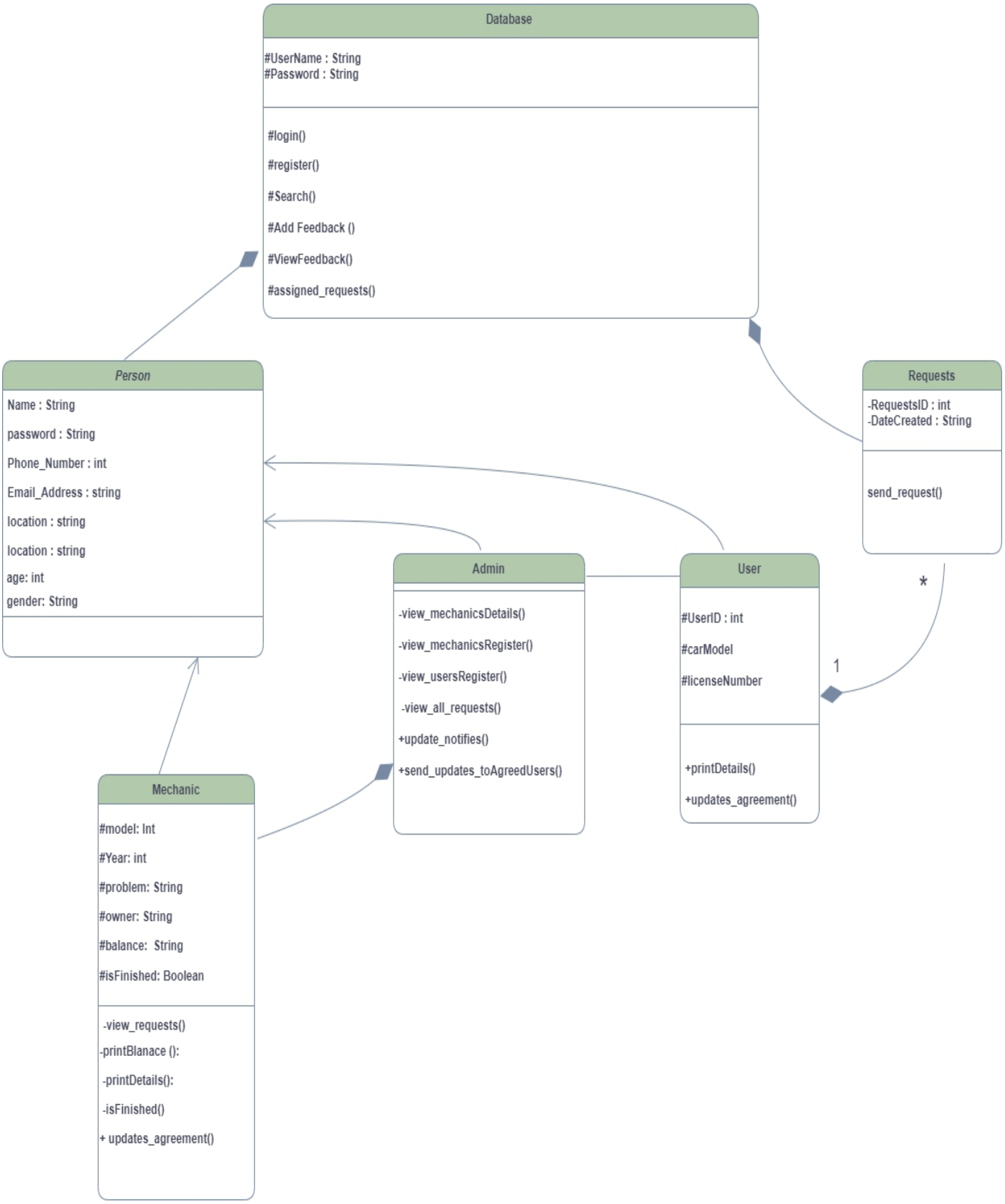
-Class Diagram *Version 1*



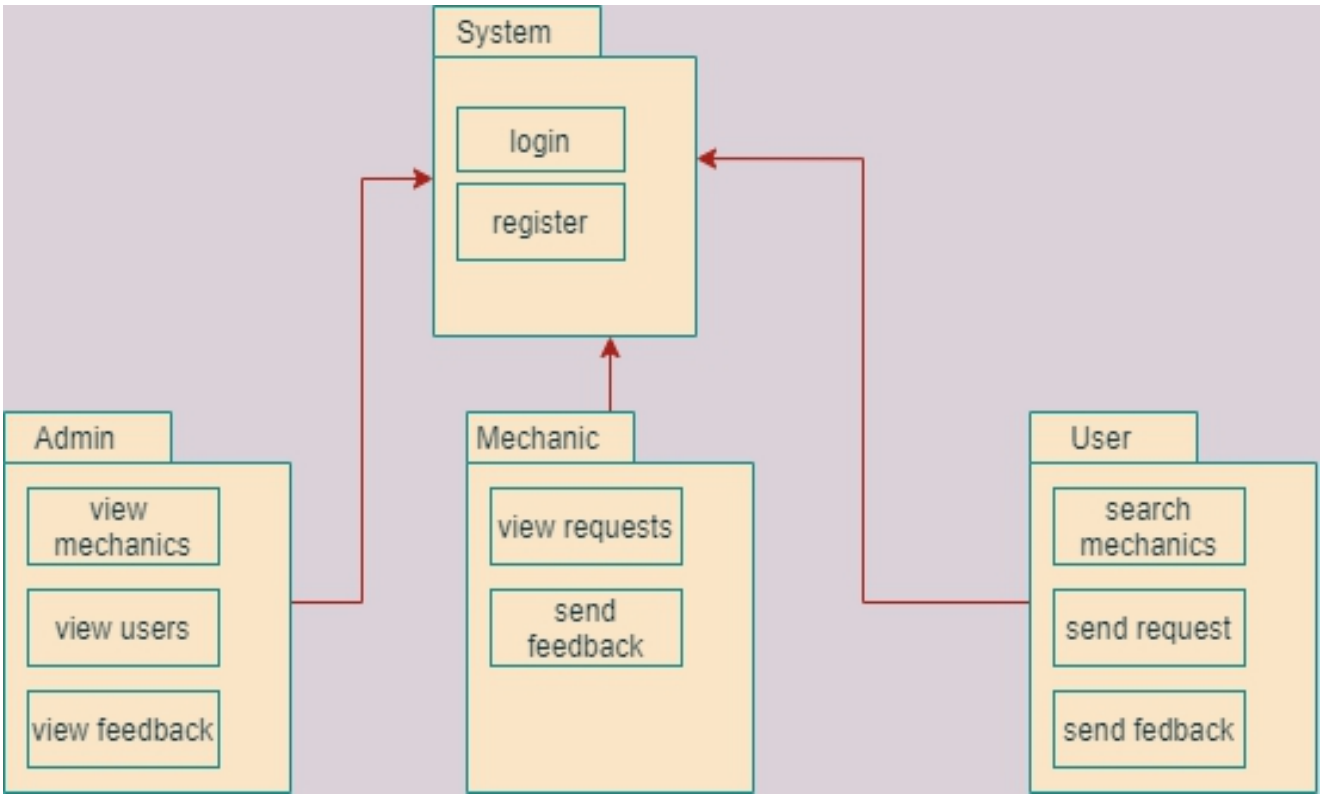
-Class Diagram *Version 2*



-Class Diagram *Version 3*



- Package Diagram :



Mandatory Design Pattern Applied

1-**Structural: Abstraction-Occurrence**

-Context:
All users will make a request on the application.

Problem:
What is the best way to represent such sets of occurrences?

Forces:
You want to represent the members of each set of occurrences
without duplicating the common information.

Solution:
1. Create an ‘‘abstraction’’ class that contains the common information of users.

2. Then create an ‘‘occurrences’’ class representing the occurrences of this abstraction.

3.Connect these classes with a one-to-many association.

2-Behavioural: Observer/Publish-Subscribe

Context:

When partitioning a system into individual classes you want the coupling between them to be loose so you have the flexibility to vary them independently.

Problem:

A mechanism is needed to ensure that when the state of an object changes, related objects are updated to keep them in step.

Forces:

The different parts of a system have to kept in step with one another without being too tightly coupled.

Solution:

1. One object has the role of the system/admin and one or more other objects the role of user/mechanic. The user/mechanic register themselves with the system, & if the state of the system changes, the user/mechanic notified & can then update themselves.

2. There are two variants:

- The ‘‘Push Model’’ where the system/admin send the user/mechanic detailed information about the change that has occurred.

- The ‘‘Pull Model’’ where the system/admin notifies the user/mechanic that there have been changes, and it’s the responsibility of the system/admin to find out the details they need to update themselves.