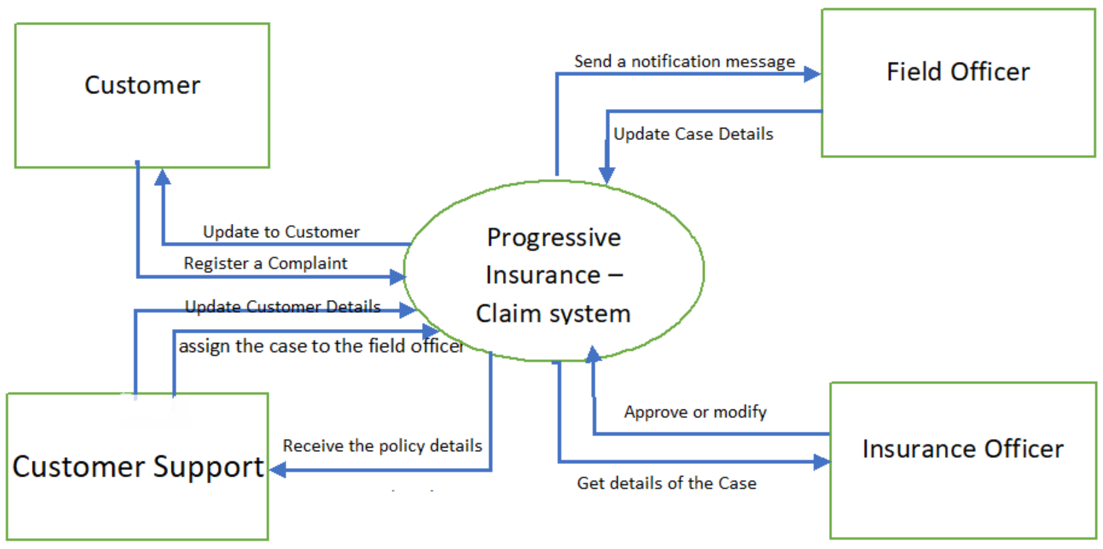
Progressive Insurance – Claim system

System Context Diagram:



Key Stakeholders and stakeholder goals:

Customer:

Call the call centre regarding an accident.

Customer Support:

Register a ‘Case’ in the system, capturing details such as location of accident, nature of accident, etc. Identify a field officer whose current location is nearest to that of the accident and assigns him the case.

Field Officer:

Reach the accident spot and examines the damage to the car. He calculates the compensation to be paid, using the app on his mobile phone. The app calculates the amount based on the coverage mentioned in the insurance policy which is retrieved from the central system.

Insurance Officer:

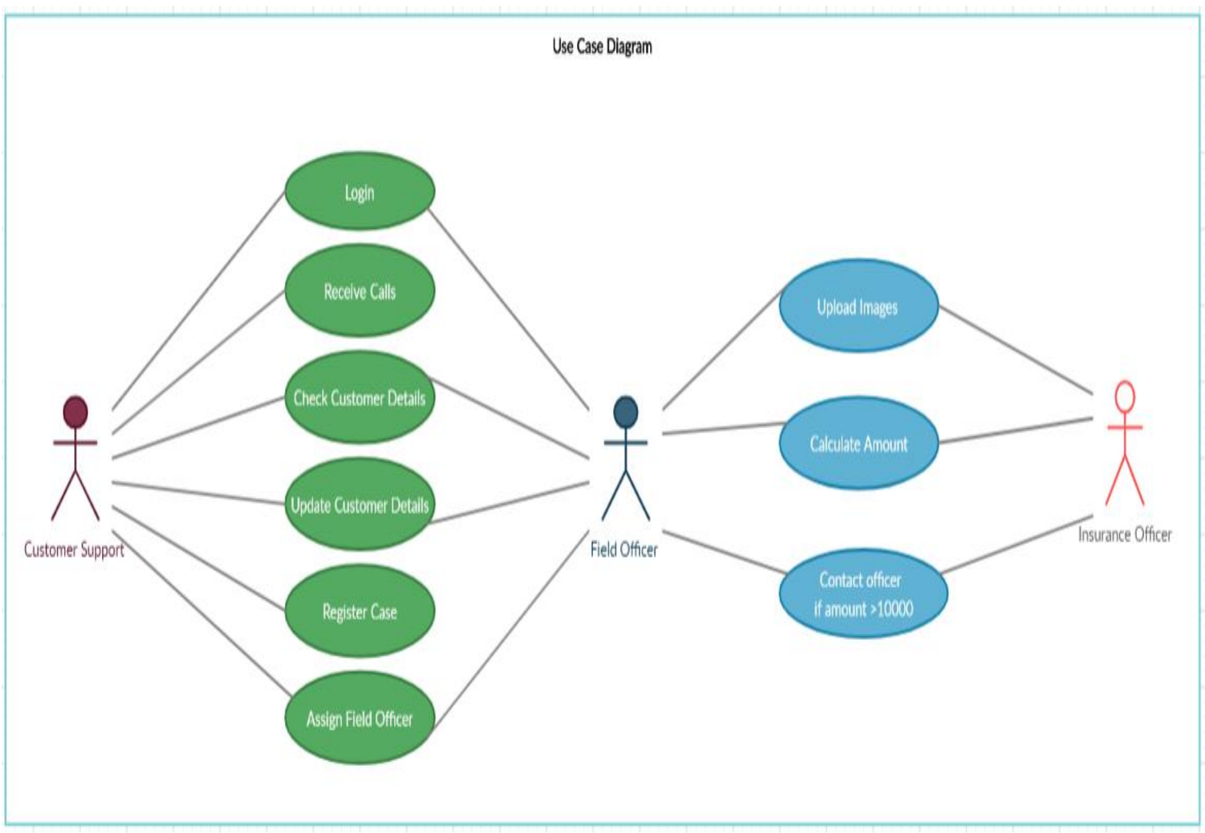
The insurance officer either approves the amount or modifies the amount. He transfers the amount to the customer’s bank account in a similar manner.

Quality Attribute Requirements:

Quality attribute	Attribute refinement	Scenario / ASR	Business Value	Impact on architecture
Usability	Field officer status indication	Call centre personnel should know the location of Field Officers.	High	High
	Customer's status indication	Field officer should know the location of the customer.	High	High
	Accurate compensation calculator	The field officer will fill up a form in his mobile app with the details of the damage and the system will calculate the compensation based on an algorithm.	High	Low
Interoperability	Field officer to the Central System	Mobile app should be able to interact with central insurance system to make payments.	High	High
	Call centre to Customer and Central System	The Call centre person should be able to view customer details from the DB.	High	High
	Central System to Payment Gateway	The central system will hit the API provided by the payment merchant with the required info for the payment to be made.	High	High
Security	Authorization	Mobile App should allow only authorized field officers to use it.	Low	Medium
	Payment Gateway	Central system should be able to make secure payment once approved.	Medium	Medium
	Confidentiality	Communication between mobile and server should be secure.	High	Medium
Availability	Detect	Central Server should be able to detect downtime.	High	Medium
	Recover	Central Server should be available to recover from failures in minimal interval.	High	Medium
	Source Detection	The system should be able to log the source of the error for post-mortem of the failure.	Medium	Low

Performance	Concurrency	Asynchronous updates to be carried out.	Medium	High
	Accuracy	Location of Field Officers nearby should be accurate.	High	High
	Well defined control flow	Synchronous allocation for individual customer processes.	Medium	Low
Modifiability	Updates to Field Officer	Call centre personal should be able to update details of customer in the central system.	High	Medium
	Vehicle Status	Field Officer should be able to upload details of the condition of the damaged vehicle.	Medium	Medium
	Updates to Insurance Officer.	The Accurate compensation calculator will update the Insurance.	Medium	Low
Testability	Individual Units	Customer DB, Payment, Field Officer DB.	Medium	Low
	Integrated Units	Manual testing is complex since it would require on field testing	High	Low
	Continuous Testing	This test is required to make sure that system is in a healthy state.	Low	Low

Use Cases:



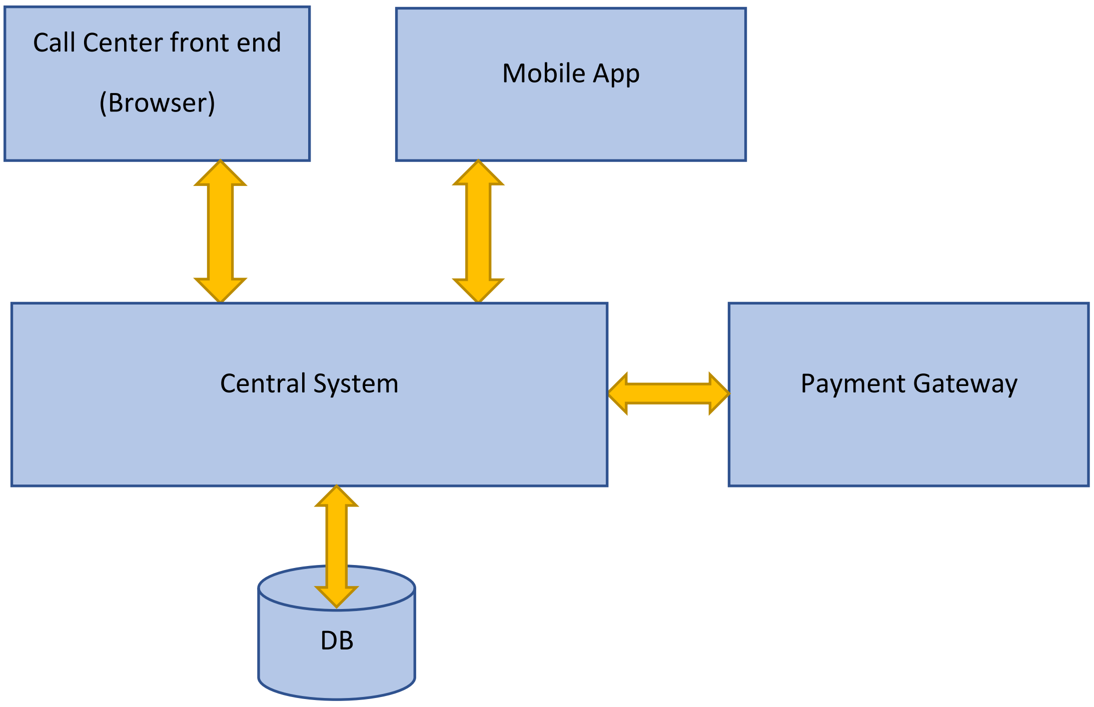
a. Customer Support:

- Receive call from customer.
- Check details of the customer in the Database and update the details about the accident from the customer.
- Then register a case in the System.
- Then search for the nearby field officer available by tracing the location of the customer assign the case to the field officer.

b. Field Officer:

- Receive notification message from the app.
- Examines the damage to the vehicle.
- Calculates the compensation to be paid, using the app on his mobile phone.
- If the compensation amount is less than Rs. 10,000, then the field officer transfers the amount to the customer.
- If the compensation amount is greater than Rs. 10,000, the field officer sends the case to his insurance officer for approval.

The architecture used is the 4-tier architecture with the following layers:



1. **Client (Frontend):**

   This is the interface through which the various stakeholders will interact with the system. It may be used as follows:

   - **Customer/Call Centre Person:**
     In case of an accident, the customer will contact the customer care centre via a phone call. The call centre person in turn will use this frontend to register the accident to generate a case along with the case number. The following details will be captured: location of accident, nature of accident, etc.

   - **Field Officer:**
     This interface will be available as a mobile app for the field officer. This app will be used file detailed report of the accident to calculate the compensation based on various factors. In case of compensation is less than Rs. 10,000.00 the app will send out a request to initiate transfer of compensation to the customer.

     Also, the location of field officer will be constantly sent to the central system via the same app.

- **Insurance Officer:**
  The insurance officer will use the front end to view detailed report of the incident in case the compensation claim is greater than Rs. 10,000. He will then approve or reject the claim at his discretion.

2. **Server (Central System)**

   The central system has the following tasks:

   - Receive and maintain the location of field officers into the DB layer.
   - Respond with a case number when a new case is requested by storing the data related to the case in the DB
   - Find the nearest field officer to a live case, assign him to the case and send notification to the officer's mobile app with the relevant details.
   - In case of compensation amount below Rs. 10,000.00, process the compensation of the customer using the customer ID. The bank account details will be fetched from the DB and send to the payment gateway for further processing.
   - In case of compensation amount above Rs. 10,000.00, forward the case to insurance officer and remit as per the amount that is approved by the officer.

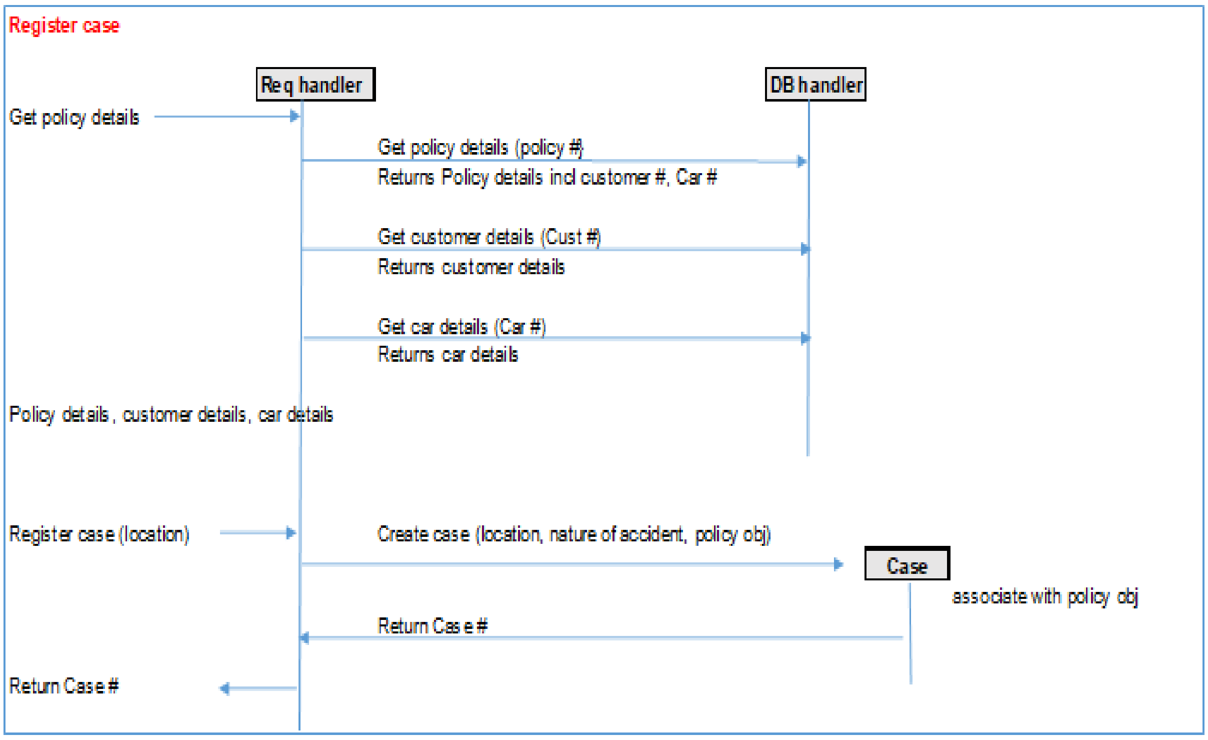
3. **Database layer**

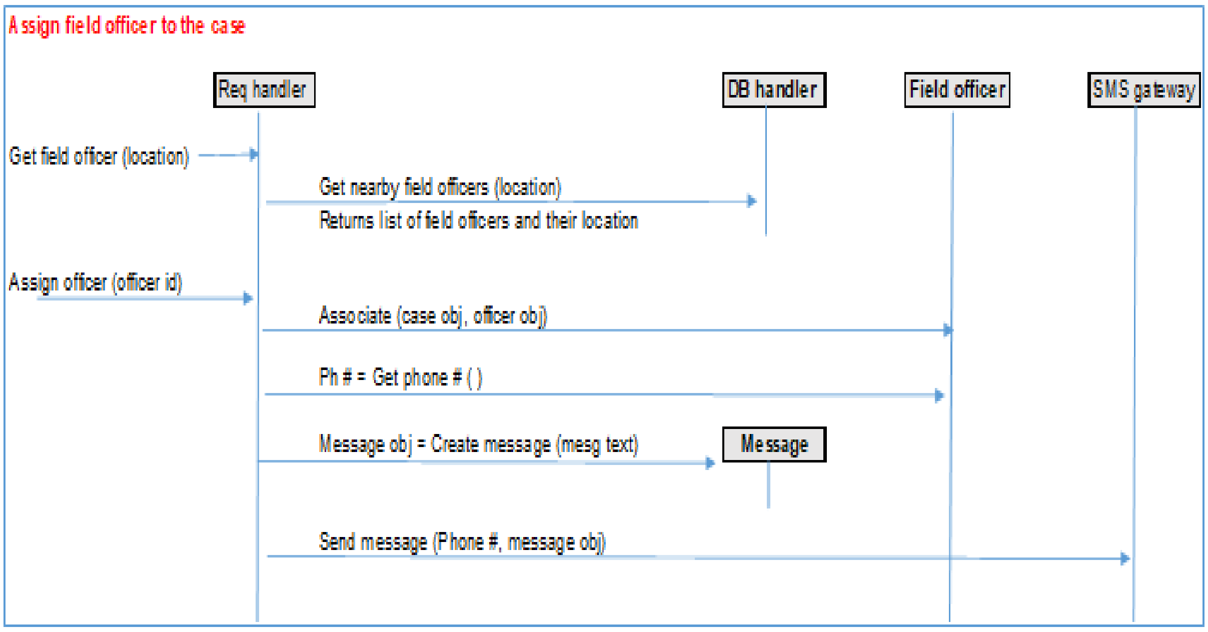
   The database will store the following data:
   - Customer
   - Cases
   - Bank details
   - Field Officer location

4. **Payment Gateway**

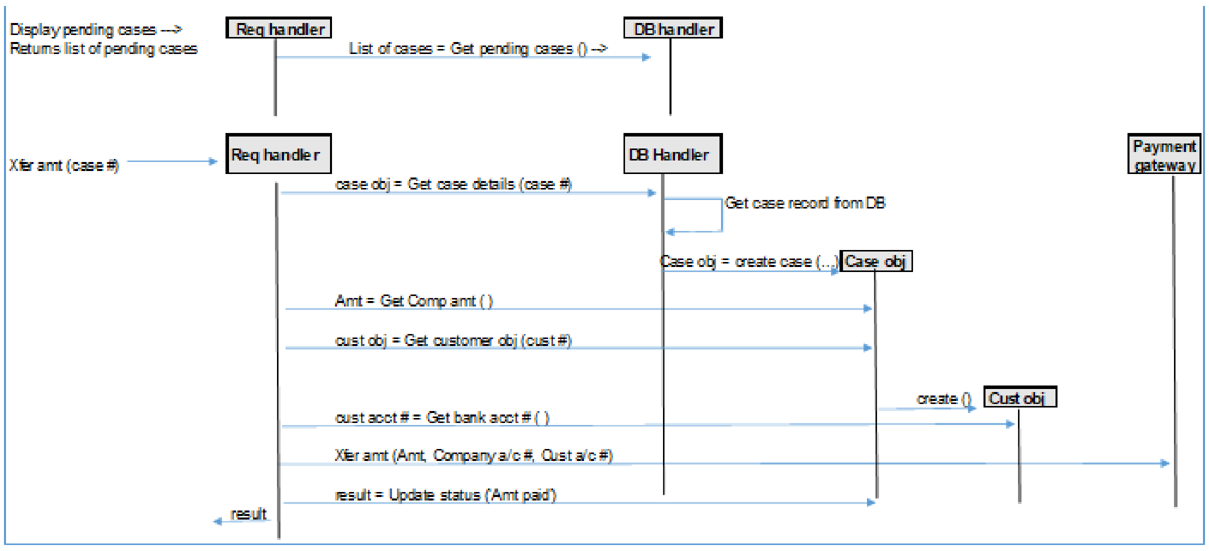
   This layer will be exposed by a merchant such as Razor-pay which will provide an API specific to our need that the central server may hit in case of a payment.

Sequence Diagram:


Register case
Req handler
DB handler
Get policy details
Get policy details (policy #)
Returns Policy details incl customer #, Car #
Get customer details (Cust #)
Returns customer details
Get car details (Car #)
Returns car details
Policy details, customer details, car details
Register case (location)
Create case (location, nature of accident, policy obj)
Case
associate with policy obj
Return Case #
Return Case #


Assign field officer to the case
Req handler
DB handler
Field officer
SMS gateway
Get field officer (location)
Get nearby field officers (location)
Returns list of field officers and their location
Assign officer (officer id)
Associate (case obj, officer obj)
Ph # = Get phone # ( )
Message obj = Create message (mesg text)
Message
Send message (Phone #, message obj)

Transfer Amount:


Display pending cases --> Returns list of pending cases
Req handler
List of cases = Get pending cases () -->
DB handler
Xfer amt (case #)
Req handler
DB Handler
Payment gateway
case obj = Get case details (case #)
Get case record from DB
Case obj = create case (...)
Case obj
Amt = Get Comp amt ( )
cust obj = Get customer obj (cust #)
create ()
Cust obj
cust acct # = Get bank acct # ( )
Xfer amt (Amt, Company a/c #, Cust a/c #)
result = Update status ('Amt paid')
result

The deployment view with choice of technologies:

1. **Client (Frontend):**

   Since the application should be accessible both on desktop as well as mobile, Flask would be an appropriate framework as it would reduce development effort needed across different OS and Platform.

   For deployment, standard publication of the app and website on app stores and web hosting platforms should do.

2. **Server (Central System)**

   Frameworks such as NodeJS or Spring boot may be used.

   Once developed, the servers may be deployed on cloud platforms such as AWS Beanstalk or GCP.

3. **Database layer**

   Since our data is highly structured, a standard Relational Database Management System is enough.

   Possible DBs that can be used: MySQL, MSSQL, Oracle

   If the data volume is too much, we may implement Big Data solutions such as Hive or Teradata to implement it.

4. **Payment Gateway**
   The deployment is abstract for the Insurance application as the payment merchant will just provide us with an API.

Tactics:

Interoperability:

In order to achieve the intended interoperability between all the layers of the system, proper configuration needed to provide and maintain interoperations with the minimum required security.

Configurations such as secure host id, port number and authentication token will ensure that operations between individual layers can be executed.

Availability:

<u>Fault detection</u>

A fault in the system may be detected by either sending a ping to payment gateway to check if its functional. Also, the central server's health may be checked by analysing resource utilization.

<u>Fault recovery</u>

Proper transaction management may be implemented to make sure that any failed transaction be rolled back and so that they do not leave the system with integrity issues. Also, making the central server stateless helps in case of failure as a simple server restart should be able to fix the issue.

<u>Fault prevention</u>

To prevent faults, proper validation may be implemented to make sure that failures don’t happen due to corrupt data.

Security:

The Mobile app has an interface to check the authenticity of the user to check only the intended user has access to the system.

Password and User-Id are set for every Filed Officer, Insurance Officer, Call Centre Personal who access the Central System.

A One-time Password is also sent to the registered email and mobile number for the user to authenticate them.

Performance:

The processes for each customer occur synchronously i.e. from Call to the Customer Support Personal – Call Centre Personal Assigning Field Officer – Filed Officer approving the claim or forwarding it to the Insurance Officer.

Usability:

To make the application more efficient for the user, adding more simpler features to the frontend so that the user can easily use the application for registering the case.

Example: Undo/Redo Buttons, Cancel Buttons etc.

Adding simpler feature also helps the system admin to handle the processes and efficient modelling of the task.

Modifiability:

To make the system modifiable, we may reduce the size of modules. Reducing the size of individual modules will make it easier to modify as it will reduce dependency on other modules.

Also, reducing the size will help to detect faults in case of failures as now the faults can be scoped to a certain module.

Testability:

We may implement the control and observe tactic to test the system. The developer may create some test cases for the module that has been developed and execute the test to observe the response.

Since we are using a layered architecture, a stub may be needed to test interaction with the other layers.