**[MFRP]**

**Blood Donation Management Application**

**Team Members:**

KRITIK ANIL WANKHEDE (2329177)

RAJU DESAI (2329179)

SRI CHARAN (2329175)

LOKESWARA REDDY (2329176)

MEDHINI (2329181)

|  |  |  |
| --- | --- | --- |
| S.NO | TOPIC | PAGE.NO |
| 1 | Introduction | 3 |
| 2 | Main Objectives | 3 |
| 3 | Requirements | 3 |
| 4 | Object | 4 |
| 5 | Profile | 4 |
| 6 | Overall flow of Project | 4 |
| 7 | Validation rule | 6 |
| 8 | Flows | 6 |
| 9 | Experience cloud | 7 |
| 10 | Triggers, Test Classes and LWC | 9 |
| 11 | Reports and dashboards | 30 |

**INDEX**

**INTRODUCTION**

A blood bank management application [streamlines the operations of a blood bank, improving the efficiency of blood collection, processing, and distribution](https://devpost.com/software/blood-bank-management-system-5e1toh). [It helps in monitoring expiration dates(blood should expire after 30 days of donation date) and ensuring the availability of appropriate blood products for transfusions](https://devpost.com/software/blood-bank-management-system-5e1toh).

[Donor Management in blood bank maintains a database of donors, Provide way to show their interest of donating through registering, and tracks donor information, which is crucial for maintaining a steady supply of blood.](https://devpost.com/software/blood-bank-management-system-5e1toh).

Blood receiver’s data management are done properly. The blood group, and other basic details are maintained properly. Reports and dashboards are used for [data analytics and reporting functions, allowing for tracking of key performance indicators and optimizing blood usage](https://devpost.com/software/blood-bank-management-system-5e1toh).

[Overall, such an application is vital to ensure that safe and timely blood transfusions are available to patients in need, thereby saving lives1](https://devpost.com/software/blood-bank-management-system-5e1toh). [It also addresses the challenge of inappropriate use of blood transfusions, which can be costly and associated with adverse events2](https://ashpublications.org/hematology/article/2019/1/583/422550/Patient-blood-management-as-the-standard-of-care). [By optimizing patient blood management, it becomes a standard of care that conserves the patient’s own blood and uses transfusions judiciously](https://ashpublications.org/hematology/article/2019/1/583/422550/Patient-blood-management-as-the-standard-of-care)

**Main objectives:**

Modules

* Blood Bank Directory
* Blood Donation Campaigns
* Donor Login
* Requestor Login
* Blood Availability Search

**Other Key functionalities and requirements for our projects:**

1. Ensures hospitals have good supply or inventories of blood bags.
2. List the availability of blood bags at any given time.
3. Ability to manage the information of its blood donor.
4. Alerts for blood requirement from registered donors.
5. Auto-check if the person donated blood in the last 3 months.
6. Allows good documentation about the donor and their blood donation activities.
7. Support fast searching to find match blood bags for the right person
8. Effectively manage blood camps
9. Collecting and storing data easier
10. Upgrade the system as per new technology
11. Digitize the system for easy usage
12. Keep a check on inventory of blood
13. Monitor the Achievement

**Objects**

* Blood campaign
* Donor
* Blood Bag
* Blood Bank
* Employee
* Blood Request

**Profiles**

* Donor
* Request Staff
* Blood Bank Manager
* Blood Bank Staff

**Overall Flow of project:**

The project has major two perspectives. One is from donor view and other one is from blood request. Firstly we can see the donor

Whenever the donor login in our application, the will be able to see the campaign details and the form which was created using screen flow(admin) /LWC (development).once that form is filled, their registration is done and the data is stored in the donor object.

A screenshot of a computer

Description automatically generated

*Fig-01 Donor App Home Page*

Then on the date of donation, the donor comes to the venue of ongoing campaign, ther when the blood is donated, the blood bank staff will create a blood bag. Once a blood bag is created, the unit quantity of blood bag donated will be automatically added in the blood bank object. This was done by using (record trigger flow(admin) /Trigger(development).  
  
In the blood bank object, many blood banks are there and each blood group and their respective quantity are stored and maintained up to date by automation.

Coming to second Perspective of our project, once the patient requires blood, the request will be made from a Blood request staff, by approval process, The request will be sent to blood bank manager, and once the particular blood group availability is checked, the blood bank manager will either approve or reject the request. Along with this, email is also sent for notifying them.

 *Fig-02 Blood Requestor App Home Page*

**Validation Rules**

Donor’s Weight should be above 50 kgs

Aadhar card number should contain 12 digits only and cannot be null

Donor age should be between the range of 18 to 65

Donation Date cannot be in the past and should be between the campaign dates

Donor can donate only after 90 days of their previous donation.

Donor can donate maximum of only 3 units on a single transaction.

**Flows:**

For admin part, we created 2 flows for each blood group, so total it is 16 flows for automation. We have attached the 2 flows of A+ blood group.

A screenshot of a computer

Description automatically generated

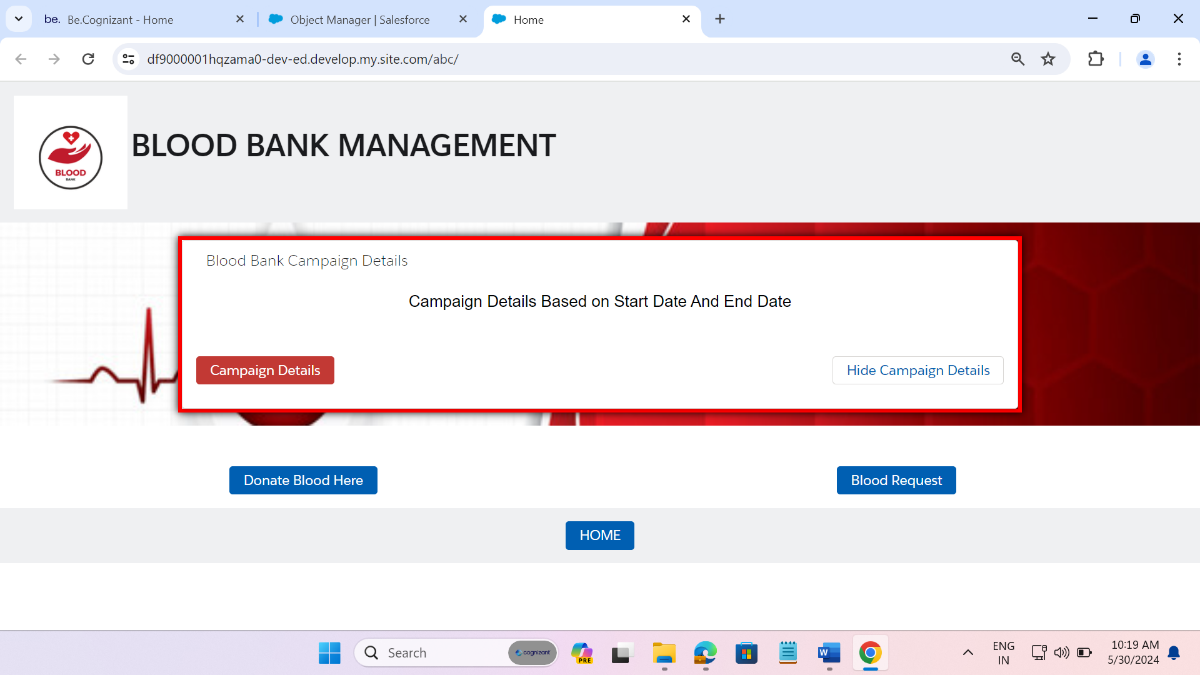
A screenshot of a computer

Description automatically generated

Whenever, Blood Donation is done, all the A+ blood donated is collected and the using loop summation is assigned. Then the A+ Approved Blood is collected and assigned. Then it is updated in blood bank object.

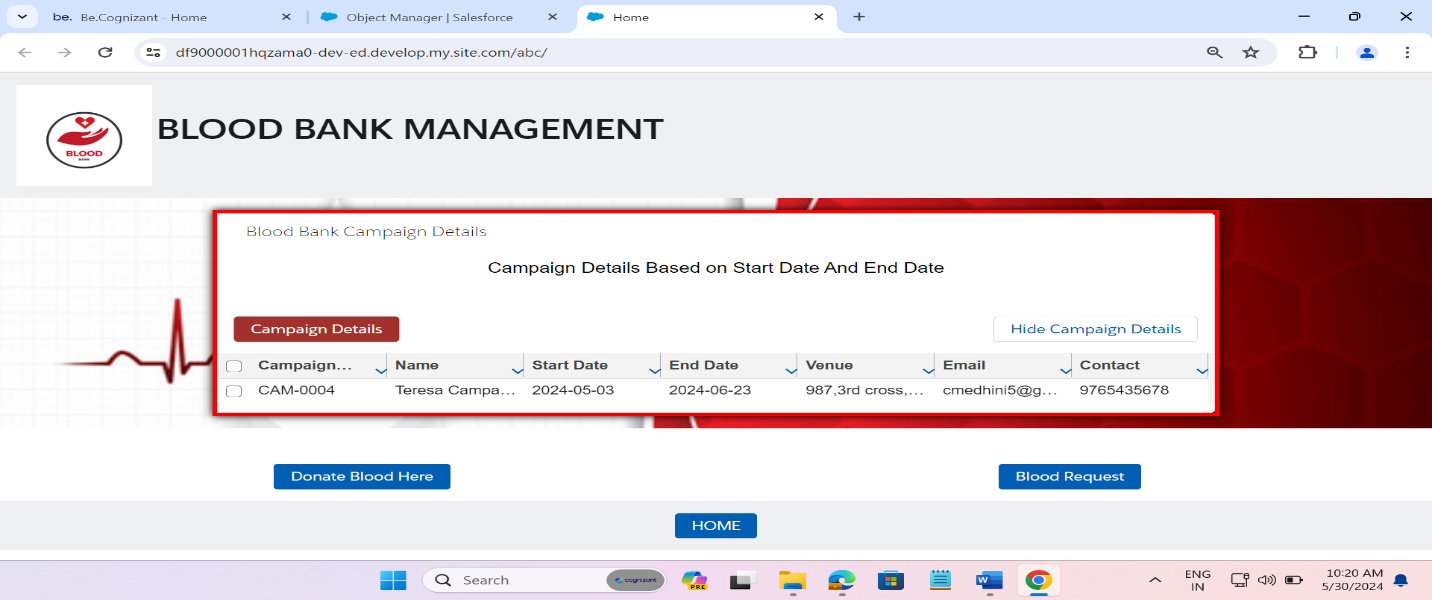
Whenever Blood request is made and approved, it is collected, and Summation is assigned. Then the total blood donated in that particular blood group is also calculated and it updates the record.

**Experience Cloud:**

****

*Fig-03 Experience Cloud*

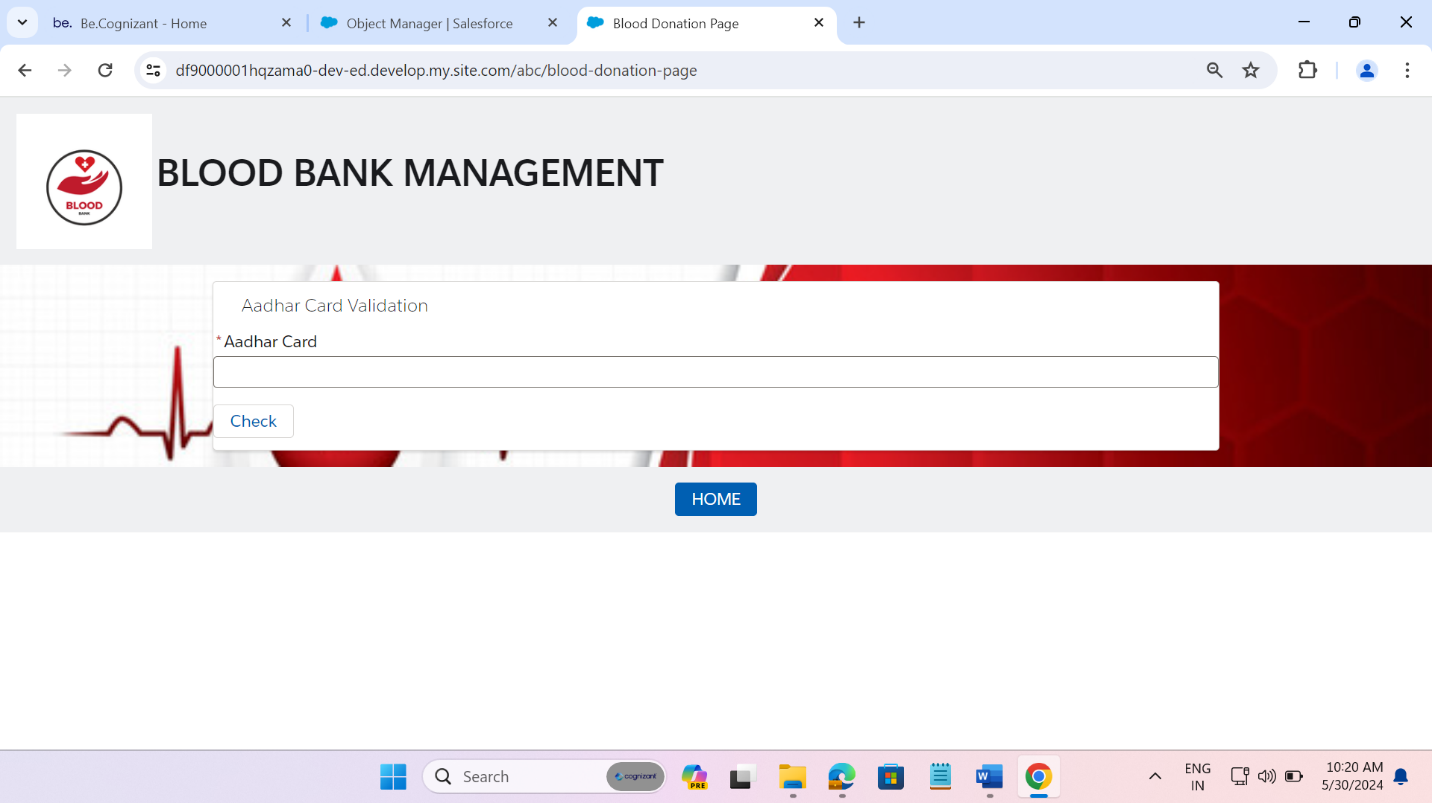
The above mentioned image is the experience cloud User interface. The component is blood campaign and if we click on campaign details button, the campaign will be displayed as below.



*Fig-04 Experience Cloud when “campaign details” button is clicked*

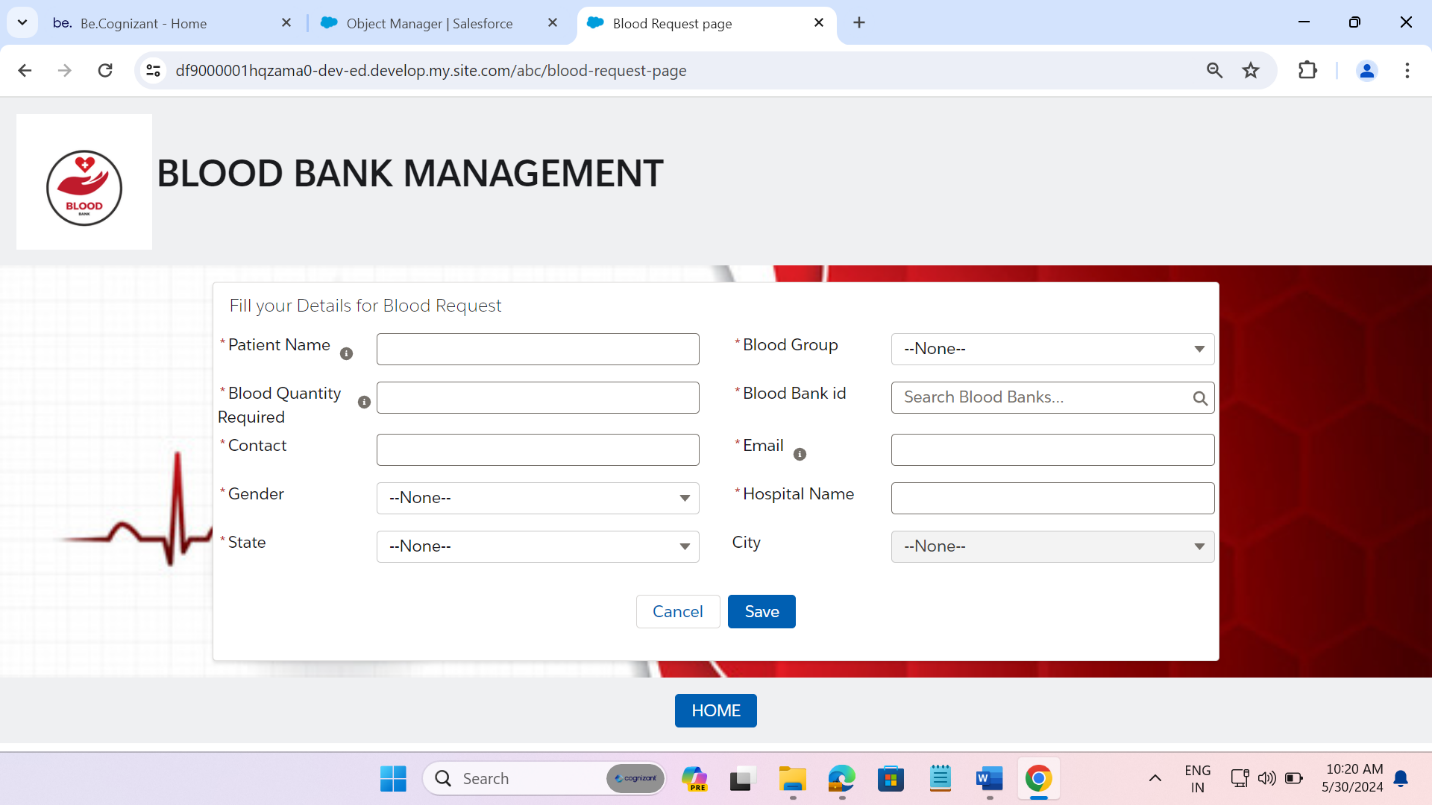
If we click on the “*Donate Blood Here”* button the Aadhar Card Validation Component will be displayed. And it checks for three condition.

1. If new donor enters their Aadhar card number, then it displays a message “You are a new user, you can fill your details” -Donor Component with all fields will be displayed.
2. If an old donor comes after 90 days of last blood donation, he will need to enter only 3 fields. Not the whole fields in the donor component and those details will be updated.
3. If an old donor comes within 90 days has completed, he will be displayed a message “you can donate only after 90 days from your last donation”.

****

*Fig-05 Aadhar card Component*

If we click on the “*Blood Request”* button, the blood request component will be displayed, and then details can be filled. The below image shows it

****

*Fig-06 Blood Request Component*

We have used trigger on two object namely blood bag and blood request for automation purposes. Those codes are attached for reference.

**Blood Bag Trigger:**

trigger TotalBlood on Blood\_bag\_\_c (after insert, after delete,before delete,before insert) {

if(Trigger.isDelete && Trigger.isBefore){

BloodBagTriggerHandler.handleBeforeDelete(Trigger.old);

}

if(Trigger.isInsert && Trigger.isAfter){

BloodBagTriggerHandler.handleAfterInsert(Trigger.new);

}

if (Trigger.isBefore && Trigger.isInsert) {

BloodBagTriggerHandler.handleBeforeInsert(Trigger.new);

}}

**Blood Bag Trigger Handler**

public class BloodBagTriggerHandler {

public static void handleBeforeInsert(List<Blood\_bag\_\_c> newBloodBags) {

// Collect the donor IDs from the new blood bags created

Set<Id> donorIds = new Set<Id>();

for (Blood\_bag\_\_c bloodBag : newBloodBags) {

if ( bloodBag.Donor\_id\_\_c!= null) {

donorIds.add(bloodBag.Donor\_id\_\_c);

}

}

if (!donorIds.isEmpty()) {

// Query the existing blood bags for the donors

Map<Id, Blood\_bag\_\_c> donorToBloodBagMap = new Map<Id, Blood\_bag\_\_c>();

for (Blood\_bag\_\_c existingBloodBag : [

SELECT Donor\_id\_\_c, Blood\_Group\_\_c

FROM Blood\_bag\_\_c

WHERE Donor\_id\_\_c IN :donorIds

ORDER BY CreatedDate DESC

]) {

// Only store the latest blood bag per donor

if (!donorToBloodBagMap.containsKey(existingBloodBag.Donor\_id\_\_c)) {

donorToBloodBagMap.put(existingBloodBag.Donor\_id\_\_c, existingBloodBag);

}

}

// Ensure new blood bags have the same blood group as the previous blood bag for the donor

for (Blood\_bag\_\_c bloodBag : newBloodBags) {

if (bloodBag.Donor\_id\_\_c != null && donorToBloodBagMap.containsKey(bloodBag.Donor\_id\_\_c)) {

Blood\_Bag\_\_c latestBloodBag = donorToBloodBagMap.get(bloodBag.Donor\_id\_\_c);

bloodBag.Blood\_Group\_\_c = latestBloodBag.Blood\_Group\_\_c;

}

}

}

}

public static void handleAfterInsert(List<Blood\_bag\_\_c> bagList){

Set<Id> donorId=new Set<Id>();

Map<Id,Map<String,Decimal>> bankToBloodGroup=new Map<Id,Map<String,Decimal>>();

for(Blood\_bag\_\_c bag:bagList){

Id bankId=bag.Blood\_Bank\_\_c;

Id donId=bag.Donor\_id\_\_c;

String bloodGroup=bag.Blood\_Group\_\_c;

Decimal quantity=bag.Blood\_Quantity\_\_c;

donorId.add(donId);

if(!bankToBloodGroup.containsKey(bankId)){

bankToBloodGroup.put(bankId,new Map<String,Decimal>());

}

Map<String,decimal> bloodGroupQuantity= bankToBloodGroup.get(bankId);

if(!bloodGroupQuantity.containsKey(bloodGroup)){

bloodGroupQuantity.put(bloodGroup,0);

}

bloodGroupQuantity.put(bloodGroup,bloodGroupQuantity.get(bloodGroup)+quantity);

System.debug(bankId);

System.debug(donId);

}

List<Donor\_\_c> donList=[Select Id,Status\_\_c from Donor\_\_c where Id IN :donorId];

for(Donor\_\_c don:donList){

don.Status\_\_c='Closed';

System.debug('Status:-'+don.Status\_\_c);

}

update donList;

List<Blood\_Bank\_\_c> bankList=[select Id,A\_Quantity\_\_c,A\_negative\_Quantity\_\_c,

B\_positive\_Quantity\_\_c,B\_Negative\_Quantity\_\_c,

AB\_Negative\_Quantity\_\_c,AB\_Positive\_Quantity\_\_c,

O\_Positive\_Quantity\_\_c,O\_Negative\_Quantity\_\_c

from Blood\_Bank\_\_c where Id=:bankToBloodGroup.keySet() ];

for(Blood\_Bank\_\_c bank:bankList){

Map<String,Decimal> bloodGroupQuantity=bankToBloodGroup.get(bank.Id);

System.debug(bank.Id);

// System.debug('Before Adding:-'+bank.A\_positive\_Quantity\_\_c);

for(String bloodGp:bloodGroupQuantity.keySet()){

Decimal bloodQuantity=bloodGroupQuantity.get(bloodGp);

switch on bloodGp{

when 'A+'{

bank.A\_Quantity\_\_c+=bloodQuantity;

}

when 'A-'{

bank.A\_negative\_Quantity\_\_c+=bloodQuantity;

}

when 'B+'{

bank.B\_positive\_Quantity\_\_c+=bloodQuantity;

}

when 'B-'{

bank.B\_Negative\_Quantity\_\_c+=bloodQuantity;

}

when 'AB+' {

bank.AB\_Positive\_Quantity\_\_c += bloodQuantity;

}

when 'AB-' {

bank.AB\_Negative\_Quantity\_\_c += bloodQuantity;

}

when 'O+' {

bank.O\_Positive\_Quantity\_\_c += bloodQuantity;

}

when 'O-' {

bank.O\_Negative\_Quantity\_\_c += bloodQuantity;

}

}

}

}

update bankList;

}

public static void handleBeforeDelete(List<Blood\_bag\_\_c> bagList){

for(Blood\_bag\_\_c bag : bagList){

if(bag.Blood\_Expiry\_Date\_\_c>=System.today()){

bag.addError('The blood is yet to expire. You Cannot delete it');

}

}

}

}

**Blood Request Trigger**

trigger BloodRequestTrigger on Blood\_Request\_\_c (after update)

{

BloodRequestTriggerHandler.handleAfterUpdate(Trigger.new, Trigger.oldMap);

}

**Blood Request Trigger Handler**

public class BloodRequestTriggerHandler {

public static void handleAfterUpdate(List<Blood\_Request\_\_c> reqList,Map<Id, Blood\_Request\_\_c> oldMap)

{

Map<Id,Map<String,Decimal>> bankToRequest=new Map<Id,Map<String,Decimal>>();

for(Blood\_Request\_\_c req: reqList) //loop blood request

{

if(req.Status\_\_c == 'Approved' && oldMap.get(req.Id).Status\_\_c=='Requested')

{

Id bankId=req.Blood\_Bank\_id\_\_c;

String bloodGroup=req.Blood\_Group\_\_c;

Decimal quantity = req.Blood\_Quantity\_Required\_\_c;

if(!bankToRequest.containsKey(bankId)) //check for new records id in map

{

bankToRequest.put(bankId,new Map<String,Decimal>()); //initialise new map

}

Map<String,Decimal> bloodGroupquan= bankToRequest.get(bankId);

if(!bloodGroupquan.containsKey(bloodGroup)){

bloodGroupquan.put(bloodGroup,0);

}

bloodGroupquan.put(bloodGroup,bloodGroupquan.get(bloodGroup)+quantity);

}

}

List<Blood\_Bank\_\_c> bankList = [SELECT Id, A\_Quantity\_\_c, A\_negative\_Quantity\_\_c,

B\_positive\_Quantity\_\_c, B\_Negative\_Quantity\_\_c,

AB\_Negative\_Quantity\_\_c, AB\_Positive\_Quantity\_\_c,

O\_Positive\_Quantity\_\_c, O\_Negative\_Quantity\_\_c

FROM Blood\_Bank\_\_c WHERE Id IN :bankToRequest.keySet()];

for(Blood\_Bank\_\_c bank:bankList){

Map<String,Decimal> bankQuantity =bankToRequest.get(bank.id);

for(String bloodGp:bankQuantity.keySet()){

Decimal quan=bankQuantity.get(bloodGp);

switch on bloodGp{

when 'A+' {

bank.A\_Quantity\_\_c -= quan;

}

when 'A-' {

bank.A\_negative\_Quantity\_\_c -= quan;

}

when 'B+' {

bank.B\_positive\_Quantity\_\_c -= quan;

}

when 'B-' {

bank.B\_Negative\_Quantity\_\_c -= quan;

}

when 'AB+' {

bank.AB\_Positive\_Quantity\_\_c -= quan;

}

when 'AB-' {

bank.AB\_Negative\_Quantity\_\_c -= quan;

}

when 'O+' {

bank.O\_Positive\_Quantity\_\_c -= quan;

}

when 'O-' {

bank.O\_Negative\_Quantity\_\_c -= quan;

}

}

}

}

if (!bankList.isEmpty()) {

update bankList;

}

}

}

**Test Classes**

@isTest

public class TestBloodRequestApproved {

public testmethod static void main(){

Test.startTest();

Blood\_Inventory\_\_c testBank = new Blood\_Inventory\_\_c(

Blood\_Inventory\_Name\_\_c = 'Test Blood Bank',

State\_\_c='Maharashtra',

Area\_\_c='Nagpur',

Email\_Id\_\_c='test@gmail.com',

Contact\_Info\_\_c='9878903456',

A\_positive\_Quantity\_\_c = 100,

A\_Negative\_Quantity\_\_c = 50,

B\_Quantity\_\_c = 80,

B\_negative\_Quantity\_\_c = 30,

AB\_positive\_Quantity\_\_c = 60,

AB\_negative\_Quantity\_\_c = 20,

O\_positive\_Quantity\_\_c = 120,

O\_negative\_Quantity\_\_c = 70

);

insert testBank;

// Create a test Blood Request record

Blood\_Request\_\_c testRequest = new Blood\_Request\_\_c(

Patient\_Name\_\_c = 'Test Request',

Email\_Id\_\_c='kritikTest@gmail.com',

Hospital\_Name\_\_c='Top Hospital',

Mobile\_Number\_\_c='8976589321',

State\_\_c='Maharashtra',

Area\_\_c='Nagpur',

Status\_\_c = 'Requested',

Blood\_Bank\_Id\_\_c = testBank.Id,

Blood\_Group\_\_c = 'A+',

Blood\_Quantity\_Required\_\_c = 3

);

insert testRequest;

testRequest.Status\_\_c='Approved';

update testRequest;

Test.stopTest();

// Verify the updated Blood Bank record

Blood\_Inventory\_\_c updatedBank = [SELECT Id, A\_positive\_Quantity\_\_c FROM Blood\_Inventory\_\_c WHERE Id = :testBank.Id];

System.assertEquals(97, updatedBank.A\_positive\_Quantity\_\_c, 'A+ quantity should be reduced by 97');

}

}

------------------------------------------------------------------------------------------------

@isTest

public class TestTotal {

public testmethod static void main(){

Test.startTest();

Blood\_Inventory\_\_c testBank = new Blood\_Inventory\_\_c(

Blood\_Inventory\_Name\_\_c = 'Test Blood Bank',

State\_\_c='Maharashtra',

Area\_\_c='Nagpur',

Email\_Id\_\_c='test@gmail.com',

Contact\_Info\_\_c='9878903456',

A\_positive\_Quantity\_\_c = 100,

A\_Negative\_Quantity\_\_c = 50,

B\_Quantity\_\_c = 80,

B\_negative\_Quantity\_\_c = 30,

AB\_positive\_Quantity\_\_c = 60,

AB\_negative\_Quantity\_\_c = 20,

O\_positive\_Quantity\_\_c = 120,

O\_negative\_Quantity\_\_c = 70

);

insert testBank;

Blood\_Bank\_Campaign\_\_c testCamp=new Blood\_Bank\_Campaign\_\_c(

Campaign\_Name\_\_c='Test to be done',

Contact\_Number\_\_c='6789056780',

Email\_Id\_\_c='testsub@gmail.com',

Start\_Date\_\_c=Date.parse('29/05/2024'),

End\_Date\_\_c=Date.parse('31/05/2024'),

Venue\_\_c='XYZ complex',

Blood\_Bank\_Id\_\_c=testBank.Id

);

insert testCamp;

Donor\_\_c testDon=new Donor\_\_c(

Donor\_Name\_\_c = 'Test Subject',

Aadhar\_Card\_Number\_\_c='345678998710',

Age\_\_c=20,

Contact\_Number\_\_c='9876432210',

Donation\_Date\_\_c=Date.parse('29/05/2024'),

Email\_Id\_\_c='testSubject@gmail.com',

Gender\_\_c='Male',

Status\_\_c='Open',

Weight\_\_c=65,

Blood\_Bank\_Campaign\_Id\_\_c=testCamp.Id

);

insert testDon;

Blood\_Bag\_\_c testBag=new Blood\_Bag\_\_c(

Donar\_Id\_\_c=testDon.Id,

Blood\_Group\_\_c='A+',

Blood\_Quantity\_\_c=3,

Blood\_Inventory\_\_c=testBank.id

);

insert testBag;

Test.StopTest();

Donor\_\_c don=[Select Id,Status\_\_c from Donor\_\_c where Id =: testDon.Id];

System.assertEquals('Close',don.Status\_\_c,'Status Should be changed To Close');

Blood\_Inventory\_\_c updatedBank = [SELECT Id, A\_positive\_Quantity\_\_c FROM Blood\_Inventory\_\_c WHERE Id = :testBank.Id];

System.assertEquals(103, updatedBank.A\_positive\_Quantity\_\_c, 'A+ quantity should be increase to 103');

}

}

**LWC**

**Aadhar Card Component:**

<template>

<lightning-card title="Enter Your Aadhar Card" icon-name="custom:custom12">

<div>

<lightning-input label="Aadhar Card" type="text" value={aadharCard} onchange={handleAadharChange} maxlength="12" required >

</lightning-input>

<lightning-button onclick={handleAadharCheck} label="Check"></lightning-button>

<p>{message}</p>

<template if:true={isEligible}>

<div>

<lightning-input label="New Donation Date" type="date" name="newDonationDate" value={newDonationDate} onchange={handleInputChange} required >

</lightning-input>

<lightning-input label="Campaign Name" type="text" name="campaignId" value={campaignId} onchange={handleInputChange} required></lightning-input>

<lightning-input label="Age" type="number" name="age" value={age} onchange={handleInputChange} required></lightning-input>

<lightning-button onclick={handleUpdate} label="Update"></lightning-button>

</div>

</template>

<template if:true={newDonor}>

<c-donor-form></c-donor-form>

</template>

</div>

</lightning-card>

</template>

import { LightningElement,track } from 'lwc';

import getDonorByAadhar from '[@salesforce/apex/DonorController.getDonorByAadhar](mailto:@salesforce/apex/DonorController.getDonorByAadhar)';

import updateDonorDetails from '[@salesforce/apex/DonorController.updateDonorDetails](mailto:@salesforce/apex/DonorController.updateDonorDetails)';

//import { ShowToastEvent } from 'lightning/platformShowToastEvent';

//import {LightningAlert} from 'lightning/alert';

export default class AadharCardComponent extends LightningElement {

@track aadharCard = '';

donor;

//@track message = '';

@track newDonor = false;

@track isEligible = false;

@track newDonationDate = '';

@track campaignId = '';

@track age = '';

aadhaar\_regex = /^[2-9]{1}[0-9]{3}[0-9]{4}[0-9]{4}$/;

handleAadharChange(event) {

this.aadharCard = event.target.value;

console.log(this.aadharCard);

}

async handleAadharCheck() {

if (this.aadharCard.length === 12 && this.aadhaar\_regex.test(this.aadharCard)) {

console.log('Inside');

getDonorByAadhar({ aadharCard: this.aadharCard })

.then((result) => {

console.log(result);

this.donor = result;

console.log(this.donor);

if (this.donor) {

const lastDonationDate = new Date(this.donor.Donation\_Date\_\_c);

console.log(lastDonationDate);

const daysSinceLastDonation = Math.floor((new Date() - lastDonationDate) / (1000 \* 60 \* 60 \* 24));

if (daysSinceLastDonation < 90) {

//this.message = 'You can donate after 90 days.';

console.log(daysSinceLastDonation);

let dateToShow=90-daysSinceLastDonation;

// eslint-disable-next-line no-alert

alert(`You can donate after ${dateToShow} days of previous donation date.`);

// LightningAlert.Open({

// message: `You can donate after 90 days`,

// label: 'Error !',

// theme: 'Error'

// });

this.isEligible = false;

// let evt = new ShowToastEvent({

// title:'Alert',

// message: `You can donate after ${dateToShow} days.`,

// theme: "error",

// mode:'Error'

// });

// this.dispatchEvent(evt);

} else {

// this.message = 'You can donate';

// eslint-disable-next-line no-alert

this.isEligible = true;

// eslint-disable-next-line no-alert

alert('You can donate.');

}

} else {

//this.message = 'Register for donation.';

this.isEligible = false;

this.newDonor = true;

// eslint-disable-next-line no-alert

alert('You are a new user. Please enter your details');

// let evt = new ShowToastEvent({

// title:'Info',

// message: 'You are a new user. Please enter your details',

// mode: 'info',

// });

// this.dispatchEvent(evt);

// LightningAlert.Open({

// message: `You are a new user. Please enter your details`,

// label: 'Info !',

// theme: 'Info'

// });

}

})

.catch((error) => {

console.error(error);

});

} else {

console.log('outside');

// let evte = new ShowToastEvent({

// title:'Error',

// message: 'Please enter a valid 12-digit Aadhar card number.',

// mode: 'error',

// });

// this.dispatchEvent(evte);

// LightningAlert.Open({

// message: `Please enter a valid 12-digit Aadhar card number.`,

// label: 'Error !',

// theme: 'error',

// });

// eslint-disable-next-line no-alert

//alert('Please enter a valid 12-digit Aadhar card number.');

this.isEligible = false;

// eslint-disable-next-line no-alert

alert('Please enter a valid 12-digit Aadhar card number.');

}

}

handleInputChange(event) {

const field = event.target.name;

if (field === 'newDonationDate') {

this.newDonationDate = event.target.value;

} else if (field === 'campaignId') {

this.campaignId = event.target.value;

} else if (field === 'age') {

this.age = event.target.value;

console.log(this.age)

}

console.log('inside handleInput' + this.age + this.newDonationDate + this.campaignId);

}

handleUpdate() {

updateDonorDetails({ donorId: this.donor.Id, newDonationDate: this.newDonationDate, campaignName: this.campaignId, age: parseInt(this.age, 10) })

.then(() => {

// this.message = 'Donor details updated successfully.';

console.log(this.donor);

console.log('inside updateDonorDetails');

// LightningAlert.open({

// message: 'Your details are updated.',

// theme: 'success',

// label:'Success !',

// });

// eslint-disable-next-line no-alert

alert('Your details are updated.');

// let evt = new ShowToastEvent({

// title:'Success',

// message: 'Your details are updated.',

// mode: 'success',

// });

// this.dispatchEvent(evt);

})

.catch((error) => {

console.error(error);

});

}

}

**Blood Request Component:**

<template>

<lightning-card title="Blood Request Details" icon-name="standard:swarm\_request" >

<lightning-record-form object-api-name={objectApiName}

onsuccess={handleSuccessClick}

onerror={handleAlertClick}

fields={selectedFields}

columns="2"

mode="edit">

</lightning-record-form>

</lightning-card>

</template>

import { LightningElement , track} from 'lwc';

// import { ShowToastEvent } from 'lightning/platformShowToastEvent';

import Blood\_Request\_\_c from '@salesforce/schema/Blood\_Request\_\_c';

import {LightningAlert} from 'lightning/alert';

import Patient\_Name\_\_c from '[@salesforce/schema/Blood\_Request\_\_c.Patient\_Name\_\_c](mailto:@salesforce/schema/Blood_Request__c.Patient_Name__c)';

import Email\_Id\_\_c from '[@salesforce/schema/Blood\_Request\_\_c.Email\_Id\_\_c](mailto:@salesforce/schema/Blood_Request__c.Email_Id__c)';

import Gender\_\_c from '[@salesforce/schema/Blood\_Request\_\_c.Gender\_\_c](mailto:@salesforce/schema/Blood_Request__c.Gender__c)';

import Hospital\_Name\_\_c from '[@salesforce/schema/Blood\_Request\_\_c.Hospital\_Name\_\_c](mailto:@salesforce/schema/Blood_Request__c.Hospital_Name__c)';

import Mobile\_Number\_\_c from '[@salesforce/schema/Blood\_Request\_\_c.Mobile\_Number\_\_c](mailto:@salesforce/schema/Blood_Request__c.Mobile_Number__c)';

import State\_\_c from '[@salesforce/schema/Blood\_Request\_\_c.State\_\_c](mailto:@salesforce/schema/Blood_Request__c.State__c)';

import Area\_\_c from '[@salesforce/schema/Blood\_Request\_\_c.Area\_\_c](mailto:@salesforce/schema/Blood_Request__c.Area__c)';

import Blood\_Bank\_Id\_\_c from '[@salesforce/schema/Blood\_Request\_\_c.Blood\_Bank\_Id\_\_c](mailto:@salesforce/schema/Blood_Request__c.Blood_Bank_Id__c)';

import Blood\_Group\_\_c from '[@salesforce/schema/Blood\_Request\_\_c.Blood\_Group\_\_c](mailto:@salesforce/schema/Blood_Request__c.Blood_Group__c)';

import Blood\_Quantity\_Required\_\_c from '[@salesforce/schema/Blood\_Request\_\_c.Blood\_Quantity\_Required\_\_c](mailto:@salesforce/schema/Blood_Request__c.Blood_Quantity_Required__c)';

export default class RequestForm extends LightningElement {

@track objectApiName = Blood\_Request\_\_c;

@track selectedFields = [Patient\_Name\_\_c,Email\_Id\_\_c,Gender\_\_c,Hospital\_Name\_\_c,

Mobile\_Number\_\_c,State\_\_c,Area\_\_c,Blood\_Bank\_Id\_\_c,Blood\_Group\_\_c,

Blood\_Quantity\_Required\_\_c];

// handleSuccess(){

// this.dispatchEvent(

// new ShowToastEvent({

// title: 'Request Made Successfully.',

// message: 'Our Staff will be contacting you shortly.',

// variant: 'success',

// mode: 'pester'

// }

// ));

// }

// handleError(){

// this.dispatchEvent(

// new ShowToastEvent({

// title:'Error Occured Enter Data Properly',

// subject:'Error Occured',

// variant:'error',

// mode: 'pester'

// })

// );

// }

handleAlertClick() {

LightningAlert.open({

message: "Enter the Data Properly.",

theme: "error", // a red theme intended for error states

label: "Error!", // this is the header text

});

// alert notification has been closed

}

handleSuccessClick() {

LightningAlert.open({

message: "Thank You Donation.",

theme: "success", // a red theme intended for error states

label: "Saved!", // this is the header text

});

// alert notification has been closed

}

}

**Donor Component:**

<template>

<lightning-card title="Fill Donor Details" icon-name="custom:custom32">

<lightning-record-form

object-api-name={objectApiName}

mode="edit"

columns="1"

fields={fields}

onsuccess={handleSuccessClick}

onerror={handleAlertClick}>

</lightning-record-form>

</lightning-card>

</template>

import { LightningElement , api} from 'lwc';

//import {ShowToastEvent} from 'lightning/platformShowToastEvent'

import LightningAlert from "lightning/alert";

import Donor\_\_c from '@salesforce/schema/Donor\_\_c';

import Donor\_Name\_\_c from '[@salesforce/schema/Donor\_\_c.Donor\_Name\_\_c](mailto:@salesforce/schema/Donor__c.Donor_Name__c)';

import Donation\_Date\_\_c from '[@salesforce/schema/Donor\_\_c.Donation\_Date\_\_c](mailto:@salesforce/schema/Donor__c.Donation_Date__c)';

import Email\_Id\_\_c from '[@salesforce/schema/Donor\_\_c.Email\_Id\_\_c](mailto:@salesforce/schema/Donor__c.Email_Id__c)';

import Gender\_\_c from '[@salesforce/schema/Donor\_\_c.Gender\_\_c](mailto:@salesforce/schema/Donor__c.Gender__c)';

// import Status\_\_c from '[@salesforce/schema/Donor\_\_c.Status\_\_c](mailto:@salesforce/schema/Donor__c.Status__c)';

import Weight\_\_c from '[@salesforce/schema/Donor\_\_c.Weight\_\_c](mailto:@salesforce/schema/Donor__c.Weight__c)';

import Aadhar\_Card\_Number\_\_c from '[@salesforce/schema/Donor\_\_c.Aadhar\_Card\_Number\_\_c](mailto:@salesforce/schema/Donor__c.Aadhar_Card_Number__c)';

import Blood\_Bank\_Campaign\_Id\_\_c from '[@salesforce/schema/Donor\_\_c.Blood\_Bank\_Campaign\_Id\_\_c](mailto:@salesforce/schema/Donor__c.Blood_Bank_Campaign_Id__c)';

import Age\_\_c from '[@salesforce/schema/Donor\_\_c.Age\_\_c](mailto:@salesforce/schema/Donor__c.Age__c)';

import Contact\_Number\_\_c from '[@salesforce/schema/Donor\_\_c.Contact\_Number\_\_c](mailto:@salesforce/schema/Donor__c.Contact_Number__c)';

export default class DonorForm extends LightningElement {

@api objectApiName=Donor\_\_c;

@api fields=[Donor\_Name\_\_c,Donation\_Date\_\_c,Email\_Id\_\_c,Contact\_Number\_\_c,

Blood\_Bank\_Campaign\_Id\_\_c,Gender\_\_c,

Weight\_\_c,Aadhar\_Card\_Number\_\_c,Age\_\_c,

];

// handleSuccess(){

// this.dispatchEvent(

// new ShowToastEvent({

// title:'Donor Details are Successfully Submitted',

// subject:'Our Staff will contact you Shortly.Thank You',

// variant:'Success'

// }));

// }

// handleError(){

// this.dispatchEvent(

// new ShowToastEvent({

// title:'Error Occured Enter Data Properly',

// subject:'Error Occured',

// variant:'error'

// })

// );

// }

handleAlertClick() {

LightningAlert.open({

message: "Enter the Data Properly.",

theme: "error", // a red theme intended for error states

label: "Error!", // this is the header text

});

// alert notification has been closed

}

handleSuccessClick() {

LightningAlert.open({

message: "Thank You Donation.",

theme: "success", // a red theme intended for error states

label: "Saved!", // this is the header text

});

// alert notification has been closed

}

}

**Blood Campaign Detail Componet:**

<template>

<lightning-card title="Blood Bank Campaign Details">

<template lwc:if={ck}>

<div>

<template if:true={res}>

<lightning-datatable key-field="Id" data={res} columns={columns}>

</lightning-datatable>

<template if:true={err}></template>

</template>

</div>

</template>

<div class="slds-clearfix">

<lightning-button label="Show Details" onclick={handleShow} class="slds-float\_left"></lightning-button>

<lightning-button label="Close Details" onclick={handleClose} class="slds-float\_right"></lightning-button>

</div>

</lightning-card>

</template>

import { LightningElement ,track,wire} from 'lwc';

import getBloodCampaignDetails from '[@salesforce/apex/BloodCampaign.getBloodCampaignDetails](mailto:@salesforce/apex/BloodCampaign.getBloodCampaignDetails)';

export default class WireBloodBankCampaign extends LightningElement {

@track showDetails=false;

@track ck=false;

@track res;

@track err;

@track columns=[

{label:'Campaign Id',fieldName:'Name'},

{label:'Name',fieldName:'Campaign\_Name\_\_c'},

{label:'Email',fieldName:'Email\_Id\_\_c'},

{label:'Start Date',fieldName:'Start\_Date\_\_c'},

{label:'End Date',fieldName:'End\_Date\_\_c'},

{label:'Contact Number',fieldName:'Contact\_Number\_\_c'},

{label:'Venue',fieldName:'Venue\_\_c'}

];

@wire (getBloodCampaignDetails) campaigns({data, error}){

if(data) {

this.res = data;

}

else {

this.err = error;

}

}

handleShow(){

this.ck=true;

}

handleClose(){

this.ck=false;

}

}

public with sharing class BloodCampaign {

@AuraEnabled(cacheable=true)

public static List<Blood\_Bank\_Campaign\_\_c> getBloodCampaignDetails(){

return [Select Name,Campaign\_Name\_\_c,Email\_Id\_\_c,Start\_Date\_\_c,End\_Date\_\_c,

Contact\_Number\_\_c,Venue\_\_c from Blood\_Bank\_Campaign\_\_c

where End\_Date\_\_c>=today];

}

}

**Reports and Dashboards**

Reports and dashboards[provide a visual display of the data, making it easier to understand and analyze business performance at a glance](https://help.salesforce.com/s/articleView?id=000393271&language=en_US&type=1). [**R**eports and dashboards helps to make data-driven decisions by providing insights into metrics.](https://trailhead.salesforce.com/content/learn/modules/reports-dashboards-quick-look/learn-about-reports-and-dashboards) [They save time by automating the process of data collection and presentation, allowing us to focus on analysis rather than data gathering](https://www.forcetalks.com/blog/reports-and-dashboards-in-salesforce-the-ultimate-guide/). They [help identify where to invest resources for the best return, by highlighting areas of strength and opportunities for improvement](https://trailhead.salesforce.com/content/learn/modules/reports-dashboards-quick-look/learn-about-reports-and-dashboards). [Overall, reports and dashboards are key to maximizing the value of the Salesforce platform by enhancing visibility, enabling better decision-making, and driving business growth](https://www.forcetalks.com/blog/reports-and-dashboards-in-salesforce-the-ultimate-guide/)

We have attached the reports in our dashboard and that is displayed in Blood bank manager home page. It is attached below

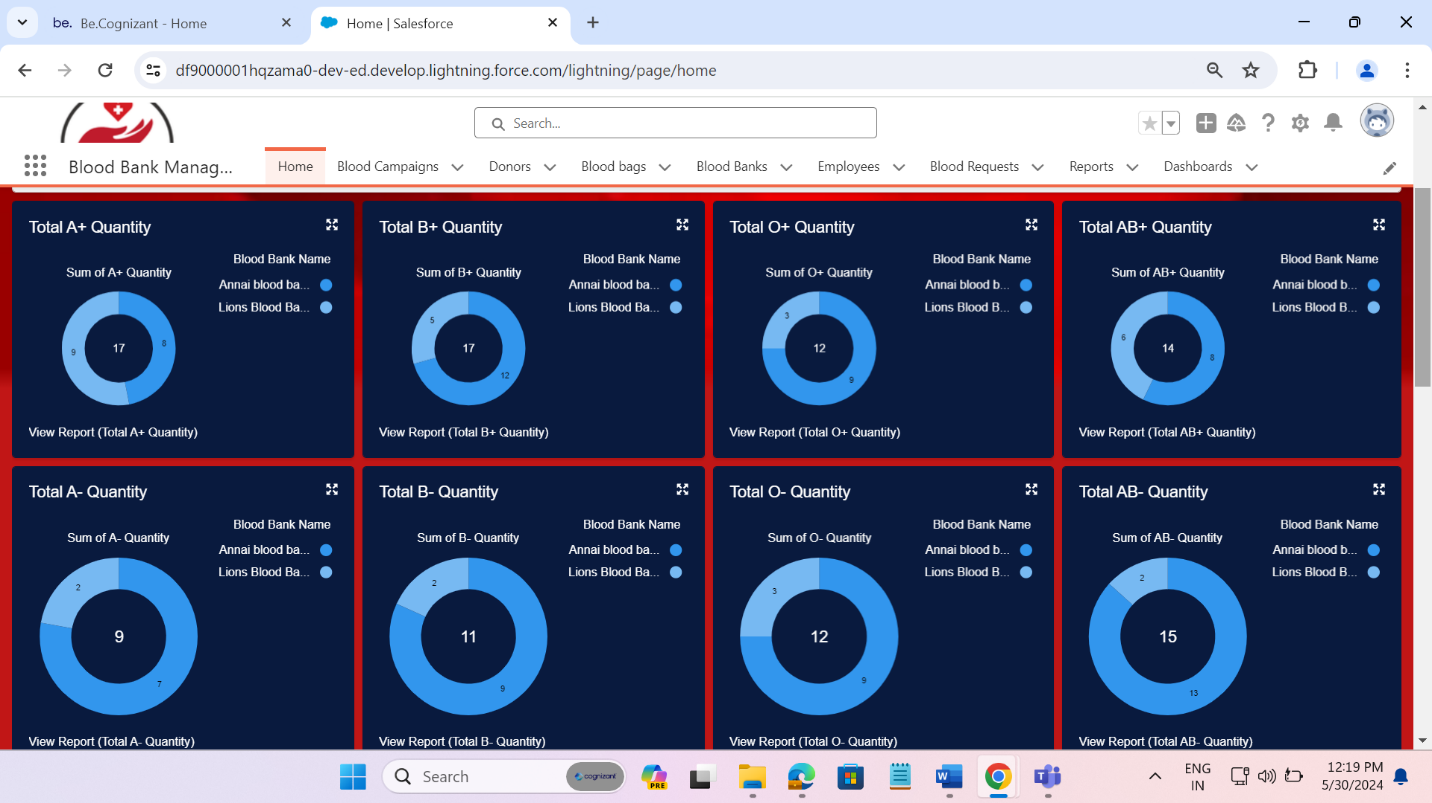


Fig-07 Dashboard (i)

A screenshot of a computer

Description automatically generated

Fig-08 Dashboard (ii)