Table of Contents

Task Decomposition & Abstract Code

Login

Abstract Code

- User enters username('$Username'), password('$Password') input fields.
- When ***Enter*** button is clicked:

```
SELECT password from User WHERE username=$Username;
```

- If User record is not found then:
  - Go back to **Login** form with error message:
    CONCAT('No Username found for: ',$Username)
- Else if User Record is found then
  - If User.password != '$Password'
    - Go back to **Login** form and raise error:
      'Invalid Username/Password Combination'.
    - Else:
      Store Login information as session variable '$UserID
      - Go to **Main Menu** page.
- Else *email* and *password* input fields are invalid, display **Login** form with error message 'Error in processing user credentials – please try again later.'

Main Menu

Abstract Code

- Get '$UserId' from session query
- *Run* ***Get User*** task to pull the user_id, username, and type specific information

```
SELECT u.user_id, u.username,
   case
   when m.municipality_type_id is not null then concat('Municipality Type:
',mt.municipality_type)
   when ga.agency_name is not null then concat('Agency Name: ',ga.agency_name)
   when c.headquarters is not null then concat('HQ: ', c.headquarters, ', ',
c.no_of_employee, ' Employees')
      when iu.job_title is not null then concat('Name: ', u.name)
   end description
FROM User u
left outer join Municipality m on (m.user_id = u.user_id)
left outer join MunicipalityType mt on (mt.municipality_type_id = m.municipality_type_id)
left outer join GovernmentAgency ga on (ga.user_id = u.user_id)
left outer join Company c on (c.user_id = u.user_id)
left outer join IndividualUser iu on (iu.user_id = u.user_id)
WHERE u.user_id = $UserID;
```

- Show the following buttons to navigate to:
  - Click **Add Resource** button to trigger **Add Resource** task
  - Click **Add Emergency Incident** button to trigger **Add Incident** task
  - Click **Search Resources** button to trigger **Search Resource Report** task
  - Click **Resource Status** button to trigger **Populate Resource Status** task
  - Click **Resource Report** button to trigger **View Resource Report** task

Add Resource

Abstract Code

- User clicks on ***Add Resource*** button from **Main Menu**
  - Get $UserID from session query
- Run **Get Resource ID** Task: Pull the next Auto_Increment value for resource

```
SELECT 'AUTO_INCREMENT' FROM INFORMATION_SCHEMA.TABLES
WHERE table_name 'Resource';
```

  - Populate Resource ID in **Add Resource Form**

- Run the **Get ESF** task; query for information about available ESFs.
  - From the ESF table lookup available ESFs

```
SELECT description FROM ESF;
```

  - Populate ESF dropdown in **Add Resource Form**
  - Populate Additional ESF dropdown in **Add Resource Form**
- Run **Get Cost Unit** task: query for information on available cost unit
  - From UnitQuantity table, lookup available cost per units measure.

```
SELECT unit FROM UnitQuantity;
```

  - Populate "cost per **xxx**" dropdown in **Add Resource Form**

- Display ***Add Resource Form***, User enters
  - *Resource_Name('$Resource_Name')*
  - *Model ('$Model') - This is optional*
  - *Capabilities('$Capabilities') -This is optional*
  - *Latitude('$Latitude')*
  - *Longitude ('$Longitude')*
  - *Max_Distance('$MaxDistance')*
  - *Cost('$Cost')*
- User selects
  - *Primary ESF('$Primary_ESF')*
  - *Additional ESF('$Additional_ESF_ID')*
  - *Unit Quantity/Cost Per ('$Unit_Quantity'')* from the respective dropdowns

- If data validation is successful for input fields then:
  - When ***Save*** button is clicked:
  - Insert into Capability

```
For each ($capability as $cap) {
INSERT INTO Capability (resource_id, description)
VALUES ($ResourceID, $cap) }
```

```
INSERT INTO Resource (user_id, name, primary_esf_id, latitude ,longitude ,
max_distance , cost_amount , unit_quantity_id)
VALUES ($UserID, $Resource_Name , $Primary_ESF, $Home.Latitude,
$Home.Longitude, $Max Distance, $Cost, $unit_quantity_id);
```

  - Insert into AdditionalESF

```
For each ($Additional_ESF_IDas $aESF) {
INSERT INTO AdditionalESF (resource_id, esf_id)
VALUES ($ResourceID, $aESF) }
```

- Go to **Main Menu** page.

Add Incident

Abstract Code

- User clicked on ***Add Incident*** button from **Main Menu**
  - Get $UserID from session query.
- Run the **Get Declarations** task; query for information about available incident types
  - From the Declaration table, lookup available incident declarations and descriptions

    SELECT description, abbreviation FROM Declaration;

  - Populate *Declaration* dropdown with Declaration.description on ***New Incident Form***
- Display ***Add Incident Form***
- User enters:
  - *Date('$Date)*
  - *Description('$Description'),*
  - *Latitude('$Incident_Latitude'),*
  - *Longitude('$Incident_Longitude'),*
- User selects *Declaration ('$Incident_Type* from the respective dropdowns
- Validate Parameters:
  - IF $Date > current_date THEN
    - Raise Error "Incident Date must not be in the future."
  - ELSE IF abs($*Incident*_Latitude) > 90 THEN

        - Raise Error "Invalid Latitude Coordinate."
    - ELSE IF abs($*Incident*_Longitude) > 180 THEN
        - Raise Error "Invalid Longitude Coordinate."
- If data validation is successful for input fields then:
    - When ***Save*** button is clicked:
        - Auto-assign the owner of incident to logged-in user
        - Store Incident Information as new entry to Incident table

```
INSERT INTO Incident (user_id, incident_count Latitude, Longitude, Description,
Date, Declaration)
SELECT $UserId, $Incident_Latitude, $Incident_Longitude, $Abbreviation
d.total_count FROM Declaration d where d.description = $Incident_Type;
```

*Note that Declaration.total_count is updated by Trigger Declaration_Count

    - After Insert, return to **Main Menu**.
- Else *input* fields are invalid, display **Add Incident** form with error message

Search Resources Form

Abstract Code

- User clicked on ***Search Resources*** button from **Main Menu**
    - Get $UserId from session query.
- Run the **Get ESF** task;
    - Defined Above in "Add Resource"
- Run the **Get Incidents** task
    - From Incidents table, select all available incidents owned by Current User.

```
SELECT incident_id, description FROM Incident WHERE incident_owner =
$UserId
```

- Display ***Resource Search Form*** with ESF and Incident dropdowns populated
- User enters
    - *keyword('$args) - optional*
    - *ESF('$ESF)*
    - *Incident('$Incident)*
    - *Location('$MaxLoc)*
- User selects ESF and Incident from dropdown menus (optional).
- If data validation is successful for input fields then:
    - When ***Search*** button is clicked:
        - If an incident is selected, calculate the distance between the incident and all resources using the compiled Haversine() function
        - Pass in the input parameters for the ***Display Results*** Task
        - Run the ***Search Incident Resources*** task; query for information about associated resources with a given incident.
            - From Resource select all resources that match the criteria specified in the ***Search Resources*** form including Keyword, ESF, Location and incident ID.

        - Go to ***Search Results*** page/.
    - If no search fields are populated, return all resources in the system.
- Else *input* fields are invalid, display ***Search Resources*** form with error message

Display Results

Abstract Code

- User clicked on ***Search*** button from **Search Resources**
- From Resource select all resources that match the criteria specified in the **Search Resources** form including Keyword, ESF, Location and incident ID.
- Display *Incident Search Form Results* with
- Resource_ID, Resource Name, Resource Owner, Cost, Resource Status, and Resource Action for each resource.
    - For each resource:
        - if max_distance == Null: then display all matched records with distance in the result screen
        - else: then only display record(s) with distance < Resource.max_distance.
    - If Resource.Resource_Status == "In Use" then Display Request.Return_by as the Next Available date, where Request.Resource_ID = Resource.Resource_ID.
    - If Resource.Resource_Status == "Available" then Display "NOW" as Next Available date.
- The search results will be sorted first based on distance in ascending order, then alphabetically by the resource name
    - If Resource.Owner == $UserId && Resource.Resouce_Status == ”Available” then display **Deploy** button.
    - Else:
        - Display **Request** button.

```
SELECT r.resource_id "ID", r.name "Name", u.name "Owner",
concat('$',r.cost_amount,'/',uq.unit) "Cost", rs.status "Status",
concat(cast(round(Haversine(r.latitude,r.longitude,$Latitude,$Longitude),1) AS char), '
km') "Distance",
CASE WHEN u.user_id = $UserID then 'Deploy' ELSE 'Request' END "Action"
FROM Resource r
join User u on (u.user_id = r.user_id)
join UnitQuantity uq on (uq.unit_quantity_id = r.unit_quantity_id)
join ResourceStatus rs on (rs.resource_status_id = r.resource_status_id)
join Capabilities c on (c.capabilities_id = r.resource_id)
join ESF on (esf.esf_id = r.esf_id)
join (
        SELECT aesf.esf_id,
        aesf.resource_id,
        esf.description
```

```
        FROM AdditionalESF aesf
        join ESF on (aesf.esf_id = esf.esf_id)
        ) aesf on (esf.esf_id = r.esf_id)
where 1=1
and Haversine(r.latitude, r.longitude, $Latitude, $Longitude) < $Distance
and (esf.description like concat('%',lower(trim($ESF)),'%') or
(aesf.description like concat('%',lower(trim($ESF)),'%'))
and (lower(trim(r.model)) like concat('%',lower(trim($KeyWord)),'%') or
lower(trim(c.description)) like concat('%',lower(trim($KeyWord)),'%') or
lower(trim(r.name)) like concat('%',lower(trim($KeyWord)),'%'))
order by "Distance" ASC;
```

Create Request

Abstract Code

- User clicked on ***Request*** button from **Search Resource Results**
- Application provides '$ResourceID' and '$IncidentID'
- Run ***"Check Duplicate"*** task to check if user is allowed to proceed

```
SELECT COUNT(*) FROM Request WHERE incident_id = $IncidentID AND
resource_id = $ResourceID;
```

    - If this count > 0, then pop up error message: "Cannot create this request because:
        - 1. This resource was deployed to this same Incident before, or
        - 2. This request is pending now.". Then bring user back to Search Resource Results.
    - If this count = 0, prompt user to enter '$Return_by' date and proceed to ***Create Request*** task
- Run the ***Create Request*** task to create a new resource request

```
INSERT INTO Request (incident_id, resource_id, return_by) Values ($IncidentID,
$ResourceID, $Return_by)
```

Deploy Resource

Abstract Code

- User clicked on ***Deploy*** button from either **Search Resources** or **Resource Status**
- Application provides '$UserId;', '$IncidentID' and reads '$ResourceID'
- Run the ***Update Resource Availability*** task; update resource status to "In Use". (1 = "Available", 2 = "In Use")

```
UPDATE Resource
Set resource_status_id = 2
WHERE Resource_ID = $ResourceID AND UserID = $UserID
```

- Run the ***Update Request Task*** task; update request information

```
UPDATE Request
SET Deployment_Date = CURRENT_DATE()
WHERE Resource_ID = $ResourceID and Incident_ID = $IncidentID
```

Cancel/Reject Resource

Abstract Code

- User clicked on ***Cancel/Reject*** button from **Resource Status page**
- Application provides '$UserID', reads '$ResourceID' and '$IncidentID'
- Run the ***Delete Request*** task; remove resource request

```
DELETE FROM Request WHERE Resource_ID = $ResourceID AND Incident_ID
= $IncidentID
```

Return Resource

Abstract Code

- User clicked on ***Return*** button from **Resource Status**
- Application provides '$UserID', '$ResourceID' and '$IncidentID'
- Run the ***Update Resource Availability*** task; update resource information

```
UPDATE Resource SET Resource_Status = 1 WHERE Resource_ID =
$ResourceID AND Incident_ID = $IncidentID
```

Populate Resource Status

Abstract Code

- User click on the ***Resource Status Screen*** from the **Main Menu**
  - Get '$UserID' from the session query
- Display all resources that are currently in use responding to incidents owned by the current user as the **Resources in Use** table
  - Upon clicking **Return** in the Action column run the **Return Resource** Task

```
SELECT rx.resource_id "ID", rx.name "Resource Name" , i.description "Incident",
  u.name "Owner", rq.deployment_date "Start Date", rq.return_by "Return By",
  rs.status "Action"
FROM Resource rx
  JOIN ResourceStatus rs on (rs.resource_status_id = rx.resource_status_id)
  JOIN User u on (u.user_id = rx.user_id)
  JOIN Request rq on (rq.resource_id = rx.resource_id)
  JOIN Incident i on (i.incident_id = rq.incident_id)
where rs.status = 'In Use'
and i.user_id = $UserID
```

- Display all requests that have been sent by the current user as the ***Resources Requested by Me*** table
    - Upon clicking **Cancel** in the Action column run the ***Cancel/Reject Resource*** Task

```
SELECT rx.resource_id "ID", rx.name "Resource Name" , i.description "Incident",
  u.name "Owner", rq.return_by "Return By", 'Cancel' "Action"
FROM Resource rx
  JOIN ResourceStatus rs on (rs.resource_status_id = rx.resource_status_id)
  JOIN Request rq on (rq.resource_id = rx.resource_id)
  JOIN User u on (u.user_id = rq.user_id)
  JOIN Incident i on (i.incident_id = rq.incident_id)
where u.user_id = $UserID
```

- Display all requests made to the resources owned by the current user awaiting response as the **Resource Requests received by me** table
    - Upon clicking **Deploy** in the Action column run the ***Deploy Resource*** Task
    - Upon clicking **Reject** in the Action column run the ***Cancel/Reject Resource*** Task
    - IF Resource Status is "In Use" THEN
        - Do not display **Deploy**

```
SELECT rx.resource_id "ID", rx.name "Resource Name" , i.description "Incident",
  u.name "Owner", rq.deployment_date "Start Date", rq.return_by "Return By",
  rs.status "Action"
FROM Resource rx
  JOIN ResourceStatus rs on (rs.resource_status_id = rx.resource_status_id)
  JOIN User u on (u.user_id = rx.user_id)
  JOIN Request rq on (rq.resource_id = rx.resource_id)
  JOIN Incident i on (i.incident_id = rq.incident_id)
where
and rx.user_id = $UserID
```

View Resource Report

Abstract Code

- User clicked on ***Resource Report*** button from **Main Menu**
- Application provide '$UserID'
- Run the ***View Report*** task:

```
SELECT e.esf_id "ESF#", e.description "Primary Emergency Support Function",
COUNT(r.resource_status_id) "Total Resources",
SUM(IF(r.resource_status_id=1, 1, 0)) "Resources In Use"
FROM ESF e
LEFT OUTER JOIN Resource r on (e.esf_id = r.primary_esf_id)
WHERE u.user_id = $UserID
GROUP BY e.esf_id
ORDER BY e.esf_id ASC;
```