Swachh Desi

The Service Learning App

**CAPSTONE PROJECT PROPOSAL**

**Group 9**

**Shiva Krishna K Avijit Nandi Sachin Gaurav Srivastava**

**EC2 – Elaboration**

1. **Actors**:

This document identifies use cases for 2 main actors in the system:

* Swachh Desi citizen/activist
* Swachh volunteer

2. **Primary Use cases**:

A. **Swachh Citizen use cases**:

1. Account management
2. Submit a case/issue
3. Accept volunteer for issue
4. Review a work and mark done
5. Share feedback (P2)

B. **Swachh Volunteer use cases**:

* 1. Manage own account
  2. Explore issue
  3. Manage lifecycle of subscribed/volunteered issue
  4. Share feedback (P2)

C. **Horizontal use cases**:

* 1. Login and signup
  2. Search issues by locality

D. **Administrator use cases**:

1. Manage users (P2)
2. Moderate posts/issues/comments (P2)

**Details of the use cases (fully dressed):**

|  |  |
| --- | --- |
| **Use Case** | **Descriptions** |
| **Usecase Name**: Account Management |  |
| **Scope:** | User should be able to:   1. Update account details such as Name, address, phone number 2. Change password 3. Set preference, locality 4. Set type: volunteer or Citizen (issue submitter) |
| **Actor:** | Citizen (Issue submitter) and Volunteer |
| **Precondition:** | User is authenticated and authorized to make changes in account |
| **Success condition (Postcondition):** | Users should be able to   1. View updated details 2. View updated preferences 3. Login with updated password |

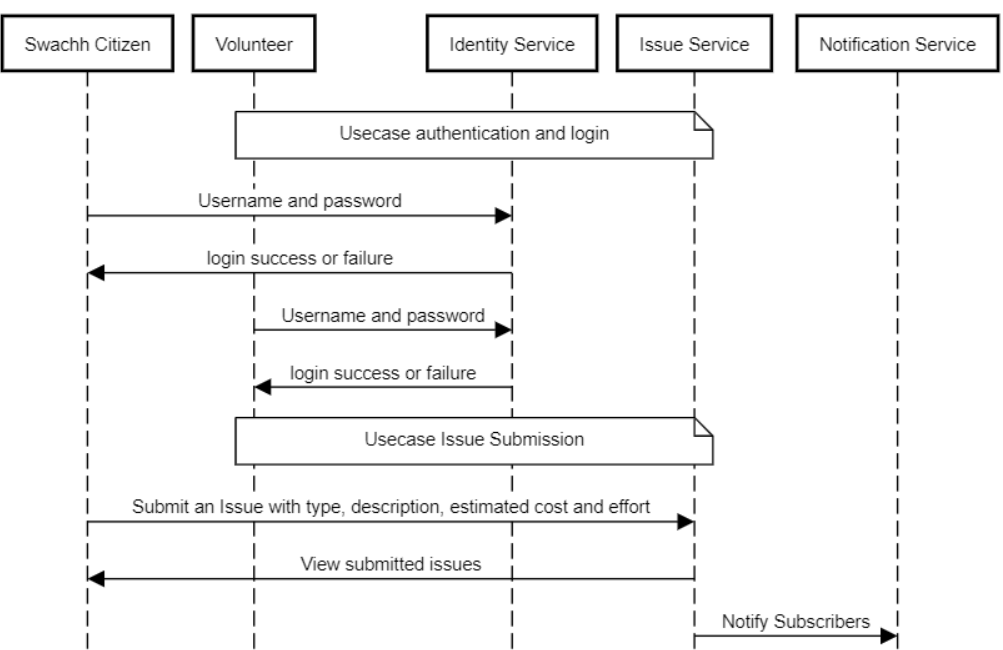
|  |  |
| --- | --- |
| **Use Case** | **Descriptions** |
| **Usecase Name**: Issue Submission |  |
| **Scope:** | Users should be able to create an issue by specifying details like place of the issue, direction to the place, type of issue, picture of the issue, number of people needed to fix the problem, date of resolution, and resources needed.  User should be able to add smaller tasks for an issue and assign interested volunteer for each task. |
| **Actor:** | Citizen (Issue submitter) |
| **Precondition:** | User is authenticated and authorized to submit an issue |
| **Success condition (Postcondition):** | Users should be able to   1. View submitted issue 2. Update an issue submitted by himself or authorized to update an issue. |

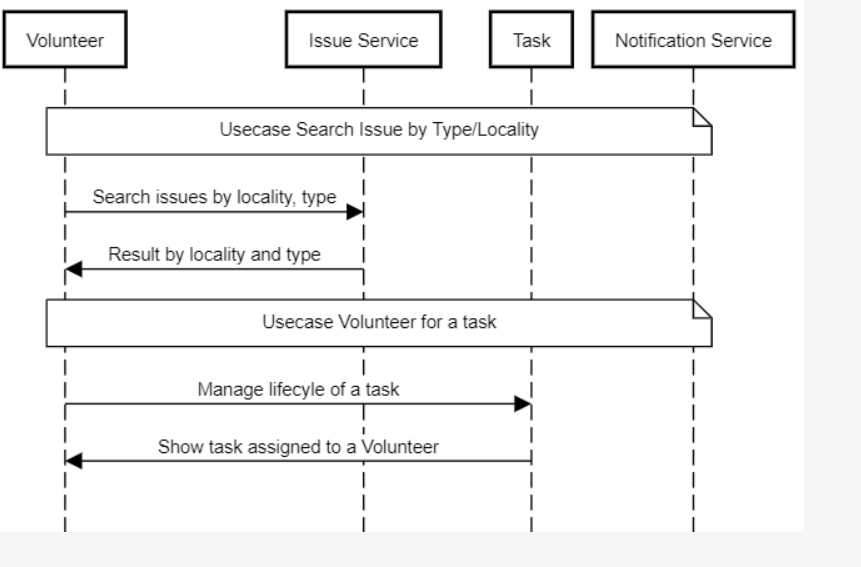
|  |  |
| --- | --- |
| **Use Case** | **Descriptions** |
| **Usecase Name**: Issue Volunteer |  |
| **Scope:** | Any Volunteer can subscribe to an issue he is interetsed. Once the maximum number of people needed is reached no one can sign up for it. Volunteer can mention the resource he is willing to dedicate for the task – number of hour daily, equipments he can bring in to complete the task. |
| **Actor:** | Volunteer |
| **Precondition:** | User is authenticated and authorized to subscribe to an issue |
| **Success condition (Postcondition):** | Users should be able to   1. View subscribed issue 2. Update a subscription 3. Update an issue |

|  |  |
| --- | --- |
| **Use Case** | **Descriptions** |
| **Usecase Name**: Track issue and provide feedback |  |
| **Scope:** | Issue submitter should be able to manage a issue by:   1. Selecting a specific number of volunteers from list of interetsed volunteers. 2. Assign work to each volunteer 3. Submit feedback on the ongoing tasks and if necessary add more resources 4. Upon completion of all tasks, submit final feedback and close the issue. |
| **Actor:** | Swachh citizen |
| **Precondition:** | User is authenticated and authorized to manage an issue |
| **Success condition (Postcondition):** | Users should be able to   1. View issue owned by himself 2. Update an issue |

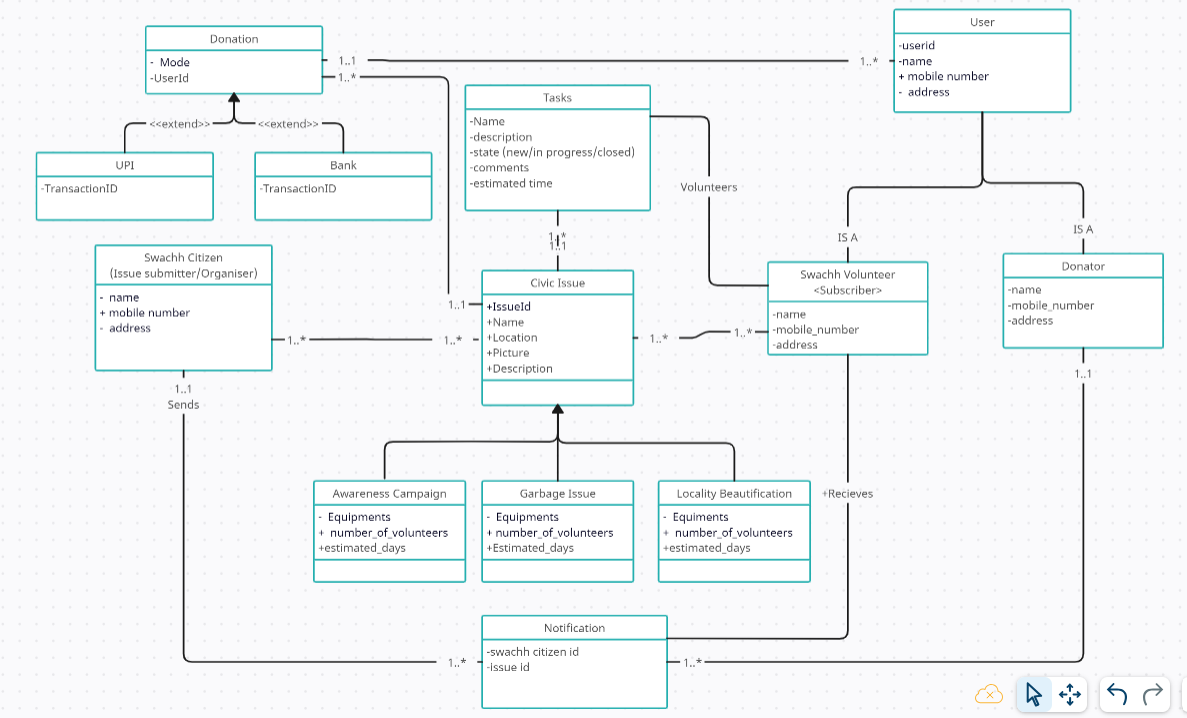
|  |  |
| --- | --- |
| **Use Case** | **Descriptions** |
| **Usecase Name**: Update issue lifesycle |  |
| **Scope:** | Volunteer of an issue should be able to:   1. Update task assigned to him. 2. Upload picture of the task completed. 3. Respond to feedbacks provided by owner of the issue 4. Upon successful completion of the task, mark the task complete.   Typical lifecycle of a task is:  New -> Progress -> Acceptance -> Closed |
| **Actor:** | Swachh Volunteer |
| **Precondition:** | User is authenticated and authorized to manage an issue |
| **Success condition (Postcondition):** | Users should be able to   1. View issue owned by himself 2. Update an issue |

|  |  |
| --- | --- |
| **Use Case** | **Descriptions** |
| **Usecase Name**: Browse Issues |  |
| **Scope:** | Any user (both Swachh Citizen and Volunteer) should be able to browse issues.  They should be able to browse by location or type.  **Swachh Citizen** may want to get notificaton on some issue. Typical usecase for this is if Swachh Citizen finds a similar issue he wants to submit and wants to get notified once completed.  **Swachh Volunteer** may want to be notified when a certain issue type or issue locality is submitted |
| **Actor:** | Swachh Citizen and Volunteer |
| **Precondition:** | User is authenticated and authorized to browse issue |
| **Success condition (Postcondition):** |  |





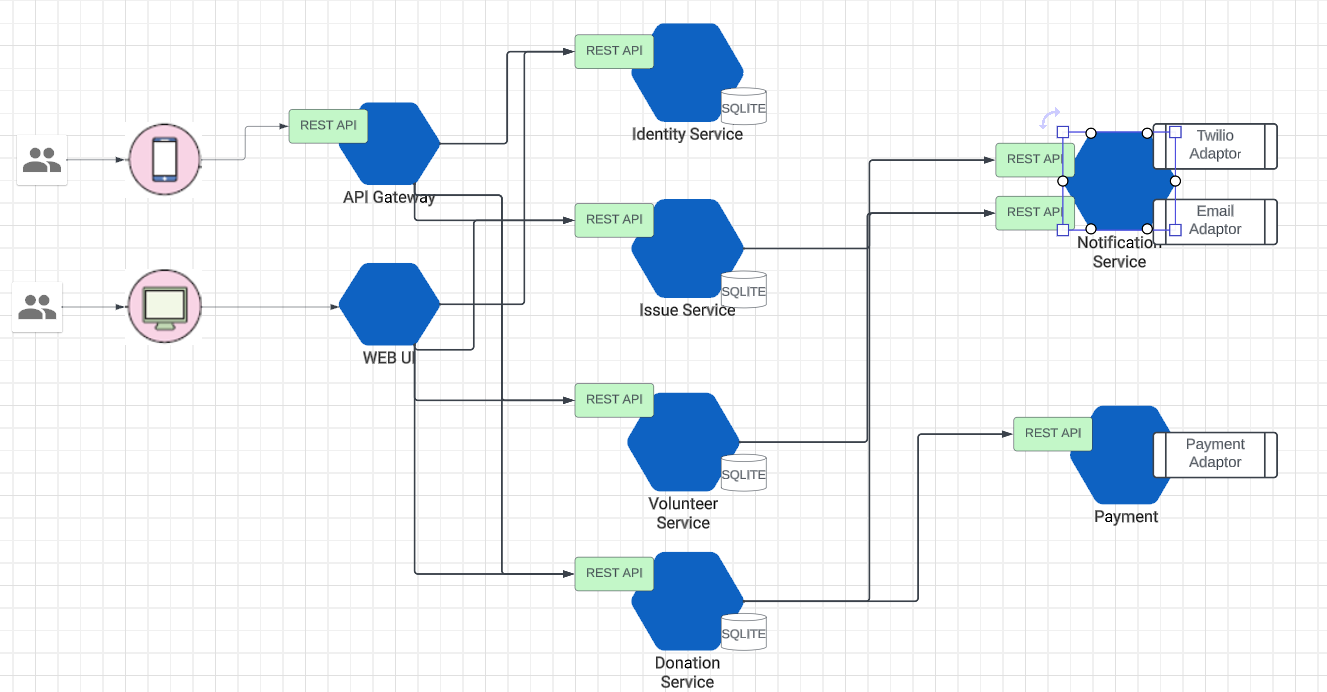
Based on the usecases identified above, the domain diagram is as follows.



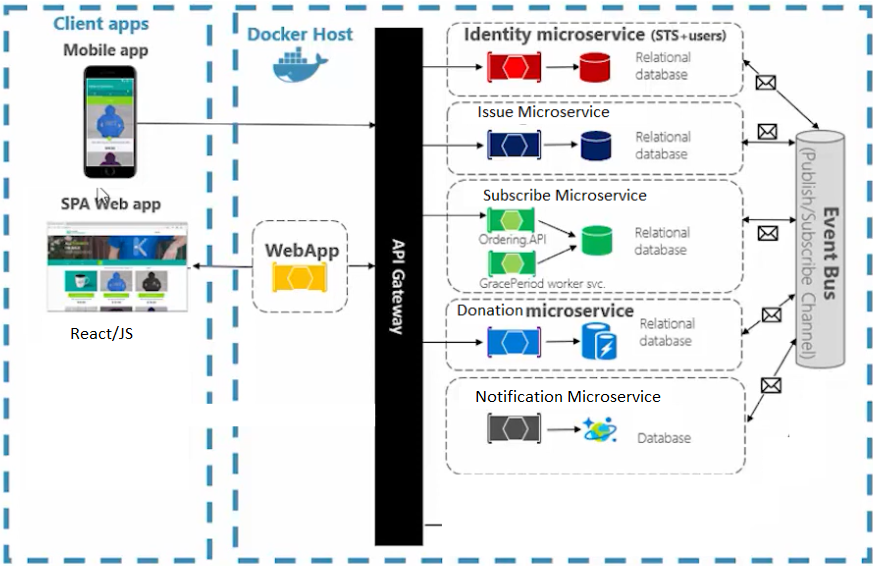
The overall design of the application is depicted in the logical architecture below. It has the following microservices:

1. Identity service: For login, authorization and authentication
2. Issue Service: For creating new issue
3. Volunteer Service: For subscribing to an issue and manage lifecycle of a task assigned for the issue.
4. Donation service: For facilitating donation done for a project
5. Notification Service: For notifying users about different stages of the project.

The architecture is based on RESTAPI framework. All communication between the services happen with RESTFUL APIs.



As a variant of the above architecture, we can also have asynchronous communication between the services by a message-based system. Kafka or RabbitMQ is suited for such broker-based messaging. If time permits, we may explore message-based communication between services.



**Shared or Split database**:

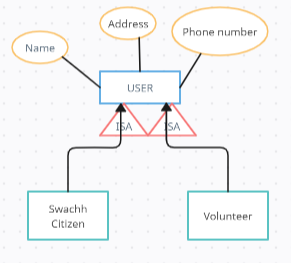
For this microservice based application, there are many choices for data storage, namely Shared data, Database view pattern, database wrapping service pattern or database as a service pattern. However, for this application we choose to use splitting apart database. Each microservice will have its own database.

**SQL or NoSQL database:**

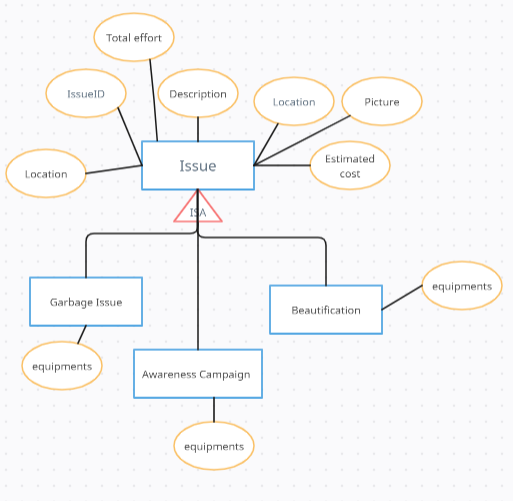
We can choose between SQLITE, or document based DBs such as MongoDB to store the data. For the purpose of simplicity, we chose SQLITE database for the microservices.

Schema for DBs:

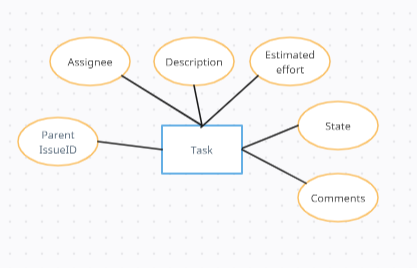
**Identity Service:**



**Issue Service:**



**Volunteer Service:**



**Donation Service:**

