Introduction and background

Problem Scenario

The objective of the application was to collect epidemiological data across the country. The requirement included a mobile application capable of collecting data when not connected to a network and upload data when connected to a network. It was also required to assign users with roles depending on the ground situation such as health officers, medical officers and patients.

Purpose

Purpose of the report is to provide a detailed description on the project architecture and the workflow. The report indicates the optimizations taken in different layers of the application. Furthermore, possible improvements, extension to the system also have been identified by the system.

Background

Since the application is used by a wide range of users ranging from general public to higher level officers in the government, security of data is a huge concern for the application.

Assumptions

- The names of diseases will be unique and different names will not be used to indicate the same disease.
- Most accurate data is most recent data and the the accuracy of data will be independent of the user (medical user updating data)
- Same medical report will be maintained for each patient per disease and the subsequent count will be saved in a table containing diseases along with the districts with the patient count.
- It is assumed that the mobile numbers entered to the application is the number of the functioning connection of the user and the user is assumed to have only a single functioning mobile connection.

Design of the system

Requirements

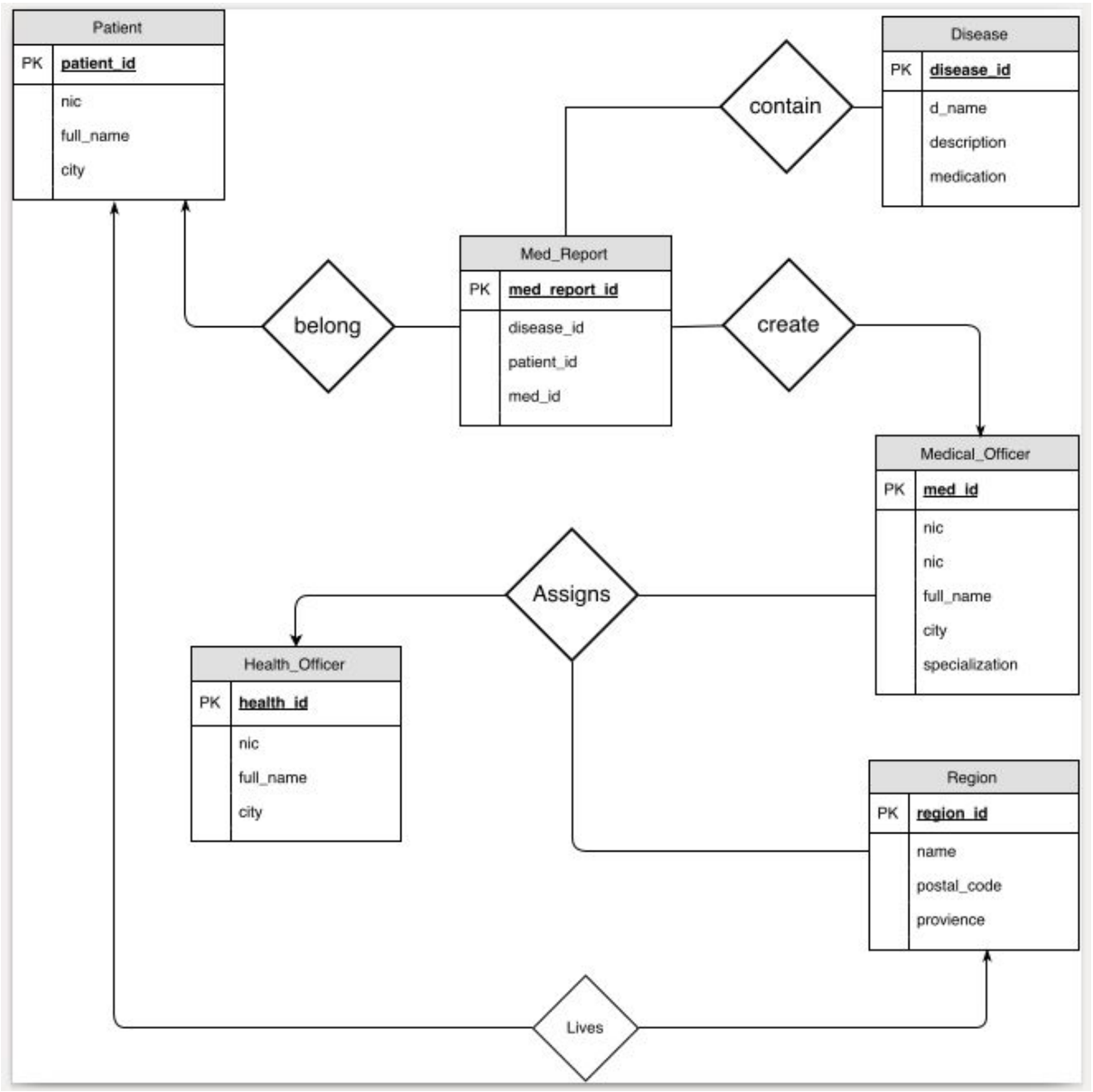
Functional requirements

- Add diseases to both central and embedded database.
- Retrieve disease details from the central database.
- Signup a user in the application.
- Log in to the application.
- Add medical reports to the central database and embedded database.
- Retrieve medical reports and details from the central database.
- Updating data between the mobile application and the central database.

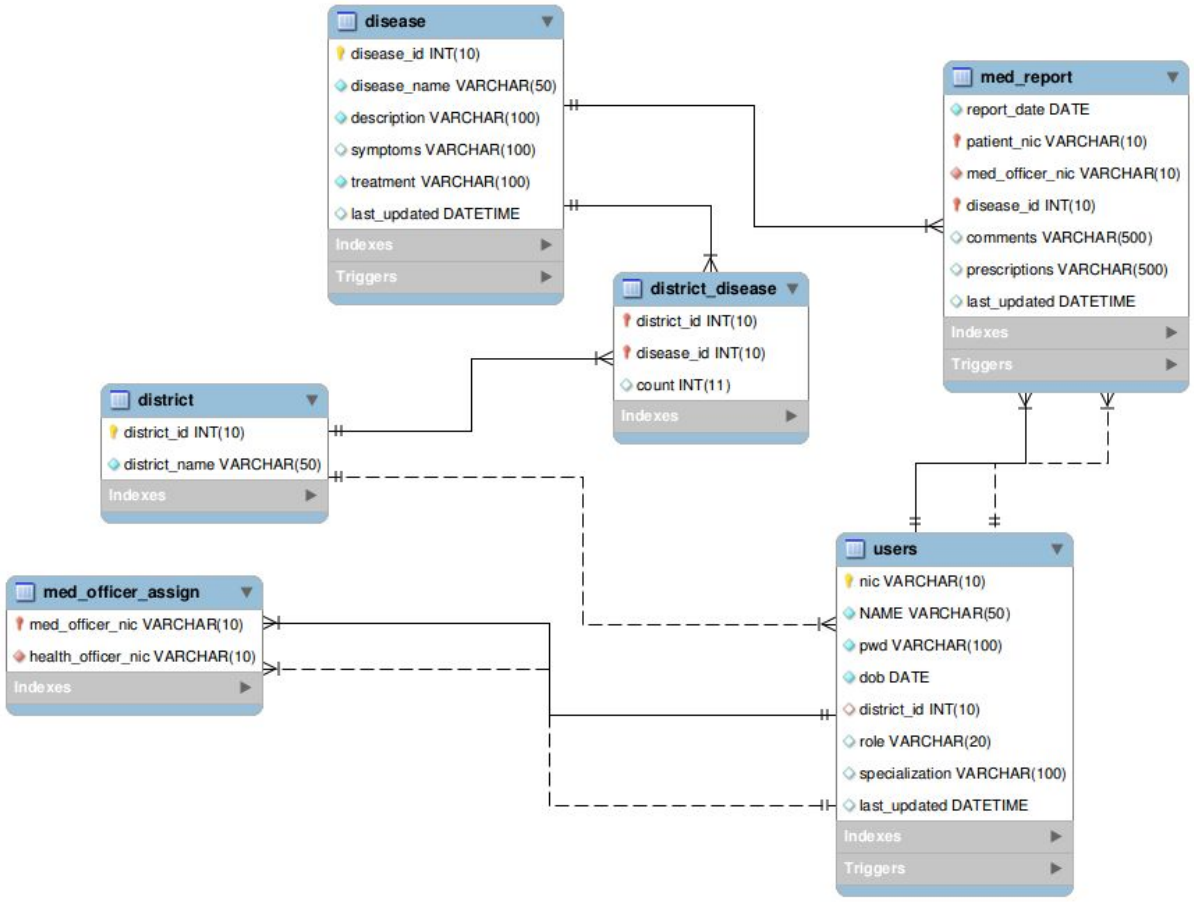
Non functional requirements

- Security and consistency of data.
- Reliability.
- Simple user interface which is easy to use.

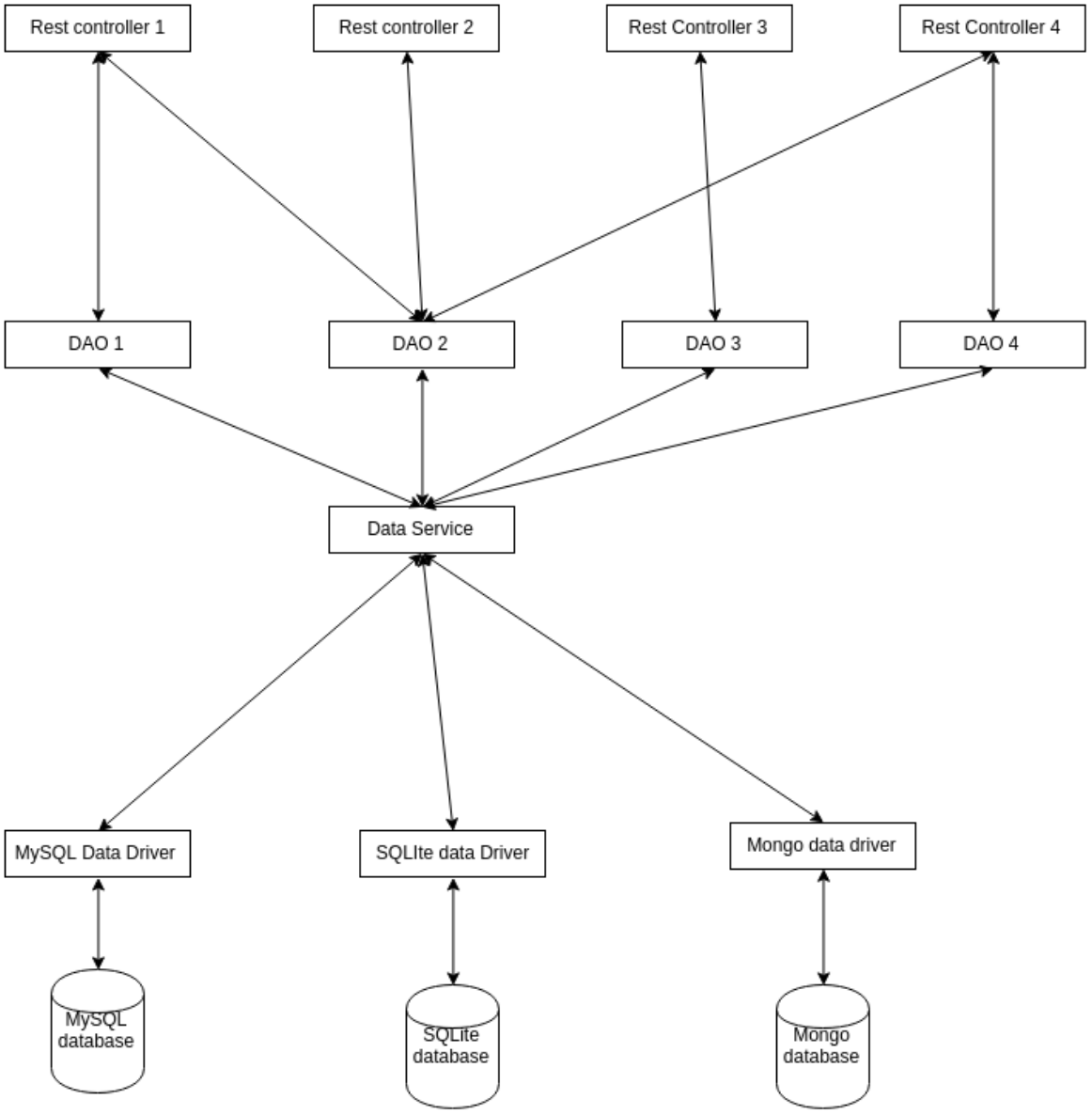
ER-diagram


Patient
PK
patient_id
nic
full_name
city
Disease
PK
disease_id
d_name
description
medication
contain
Med_Report
PK
med_report_id
disease_id
patient_id
med_id
belong
create
Medical_Officer
PK
med_id
nic
nic
full_name
city
specialization
Assigns
Health_Officer
PK
health_id
nic
full_name
city
Region
PK
region_id
name
postal_code
provience
Lives

Database Schema



Software design



Database Design

Superclasses and Subclasses

In the application even though the different levels of users existed, they had to be categorized as patients, medical officers and health officers. The main issue in having the data about users being stored in the database was that solving the issue of generalization and specialization. There were 3 main alternatives for solving the above issue in the application. Consider the scenario of the only patient and the medical officer.

Method 1 :

Form a separate class for the super class (user) and for 2 different classes for patient and medical officer. Here the issue will be performance. Since the table can grow significantly to a large extent, the database operations can cause a significant performance issue in the application. Hence the approach was abandoned.

user(nic, name, address)
patient(nic, patient_specific)
medical Officer(nic, medical_officer_specific)

Method 2 :

Form tables for each class separately. Here the issue will be the duplication of data. For instance a doctor can have an entry in both the tables causing duplication of data. Furthermore it can also introduce inconsistencies to the database.

patient(nic, name, address, patient_speicific)
medical_officer(nic, name, medical_officer_specific)

Method 3:

Use null values with a single table. Here a single table will be used and subclass specific fields will contain null values. For instance the specialization for a medical officer will be null. Compared to method 2, this can eliminate possibility of data inconsistencies and duplication of data.

Furthermore it will provide more performance as everything is available in a single table, rather than requiring to look into several tables in the database.

Maintaining patient count for a disease

Rather than taking taking disease counts individually by browsing through all the patient records in the database, the database of the application maintains a count of the number of patients having a specific disease in a specific region. In the implementation, this will allow the database to have faster performance in the case of retrieval of statistical information.

Normalization

First Normal Form

For a schema to be in first normal form, all the domains should be atomic. None of the attributes not being in the atomic domain were found during the designing of the database for the application.

Boyce-Codd Normal Form
A relational schema R is in Boyce-Codd normal form with respect to with respect to a set F of functional dependencies

Taken all the functional dependencies of F+ such that α -> β, α and β are subsets of F+, the relation schema is in BCNF if one of the following holds:

- α -> β is a trivial functional dependency
- α is a superkey for R

For example - keeping data related to district along with data of the disease will violate BCNF. Hence they have to be separated for different tables.

Second Normal Form
A relation is in second normal form, if it is in 1NF and every non-key attribute is fully functionally dependent on the primary key.

Third Normal Form
A relation is in third normal form, if it is in 2NF and no non-key attribute is transitively dependent on the primary key.

Denormalization for performance
One of the application of denormalization for performance was the approach taken in solving the subclass superclass problem in the application. Instead of having them decomposed into different tables while adhering to normalization guidelines, a weaker normal form was used in order to gain performance improvements in the application.
Since MySQL has no implementation of materialized views, the only option for the application was to use the denormalized form.
Another denormalization application was maintaining a count of the patients in corresponding to a specific disease and specific region in the database.

Indexes
User table

- Nic - the attribute has been used as the primary key and all the requests will retrieve the user using the nic generated by decrypting the token. Hence a significant number of retrievals will be done using the nic.

Disease

- Name - Retrieval of in the disease table will be mainly retrieved using the name. Hence in order to improve the performance, indexing has to be applied to the name field of the disease table.

Reports

- Patient_nic - The retrieval of data in the report table will be mainly retrieved with the using the nic of the patient. Thus in order to improve the performance of the application, the indexing has to be applied to patient_nic.

District Disease

- In district disease relation, the retrieval will be based either on disease_id or district_id. For improving the performance of the application, indexing will have to be applied for both fields individually.

Triggers

User table triggers

- Insert trigger - The user table contains a trigger that will be triggered on insert of values to the user table to set the last_updated time in the table.
- Update trigger - The user table also contains a trigger that will be triggered on update of values in the user table. The trigger will set value to the current time to the last_updated field of the table.
- Before update trigger - The before update trigger in the table will check the role for existence in the allowed categories, if not the trigger will set value to standard to flag the issue of unsolved membership category.
- Before insert trigger - the trigger will set the value of nic to 'invalid' if the user enters an incorrect nic to the system.

Disease table triggers

- Insert trigger - The trigger on add the current date time to the disease table under last_modified field.
- Update trigger - The trigger will set the current date time to the disease table under the last_modified field of the database.

Medical Report triggers

- Insert trigger - The trigger will add the current date time to the table as the last_modified field.
- Update trigger - The Trigger will set the current date time of the server to the last_modified field of the table.

Medical Report Patient

- Insert trigger - the trigger will be used to maintain the count of patients in each district with each disease. The trigger will increment the patient count if the patient subjected to the disease has not been subjected to the disease earlier.

Data Access

The application has been designed in a way such that the application layer runs independent of the of the database layer, meaning that the application has the capability of supporting other database systems such as Oracle, Postgre or even mongoDB. For the application to cope with such modifications, the drivers has to be written at the database layer adhering to the method signatures set by the the database adapter interface

Application design

Users

The application has three types of users as patients, medical officers and health officers. A new registration of a user in the application registers a user as a patient by default. The patients will have only the view permissions throughout the application in order to manage the accuracy of data provided by the application.

Medical officers have the ability to report, add and modify disease related content to the central database which will be shared among users once uploaded to the central server.
The health officers play the role of super users of the system. They have the capability to update a user as a medical officer in addition to the all the functionalities enjoyed by the other users of the system.

Restriction of the patients from entering data to the application will reduce the possible overloading of information to validated in the system as well as will improve the quality and accuracy of the results in the application.

Data Sharing between mobile device and the server

The embedded database will only contain data relating to the entries posted by the user os the application. Upon the retrieval, depending on the availability of the network connection, the application will connect with the server to fetch information relating to the query will be fetched from the central server.

Layered architecture in application

The application follows a layered architecture and higher levels have been shielded from the data access layers through a data service layer. The architectural pattern will provide a flexible database service for the application and the minimal amount of configuration changes will be required for any form of modification in the database server in the application.

Process Flow

Validators

Basic validation for the requests have been implemented in the controllers and is widely available in the controller package of the web application. In addition, each method can specify an access level for the validations to take place for the request and perform actions.

Interception and authorization

All incoming requests to the web application are intercepted by an authenticator in order to perform the authentication. All the requests paths except for request paths designated as guest routes.

Web application

Implementation

Frameworks

Mobile application

The mobile application has been developed using native android, hence the application would be light weight, consume less battery and can be easily optimized. For the development of the embedded database, SQLIte has been used, which is a widely used embedded database system. Compared with other embedded database systems, SQLIte has more wider community than other database systems.

Web application

The application also comprise of a web back end serving as the central data repository. The web backend has been built with Spring framework, which is among the widely used frameworks in the web development.

The application development has been carried out with minimal use of spring components and using custom components for different services required for the web application. For instance the authentication layers has been rebuilt with the essential components.
Unlike the previous versions, the framework release used for the development of the application used pure java in the form of annotations rather than xml based configuration.

Security

The security of the web application has been managed by the use of jwt token based authentication with a non-expiring token being issued with the authenticated user. However the token issue can be extended easily with the use of expiring tokens being issued for a limited time along with refresh tokens.

Project management

Project management for the application was done using trello, a free web based collaboration tool. The application tasks and pending tasks were uploaded to Trello by tagging the with the respective member of the team. Through the use of trello, every member was able to track down the progress of the project.For the purpose of version control, github was used.
Each member was given a main area of the project where the member was responsible to keep tack on while getting others to work on the project. The main areas of project was classified as the embedded component, web application, database design and the documentation. In Addition to the responsibility on the main area, each had to contribute in other areas under the supervision of the member assigned for the specific area.

Deployment

For the deployment of the mobile application, the google Play store can be used, which will provide management of application user groups and version management. Other toolsets for the mobile application are included within the android operating system along with other default tool sets.

Deployment of the web application has to be done to an apache-tomcat server, which is a server software capable of supporting java based web applications and services. For the persistency layer of the web application, a mysql database has to be used and the configuration for the mysql database has to be altered to the mysql server in the web application.

Discussion

Further improvements

One of the main improvements to the application is the development of an application to iOS. Even Though it is possible to develop cross platform applications with different frameworks such as cordova and ionic, they tend to be slow and to consume more battery and computing capacity of the mobile. However independent native development of applications would allow the developers to take control over the parameter of performance and expandability.

Another possible improvement to the application is the development of analytics based on the data collected through the application. The analytics can provide the government with the details on possible epidemics in areas and their patterns. In turn it would also be helpful in determining the allocation plans of resources more accurately.

Another improvement to the application is the caching of the requests such as details on the diseases and most frequently used resources. It will reduce the load on the main server substantially while improving the performance. For the purpose of caching, an embedded database system such as redis, berkeley db can be used depending on the preference and the type of caching required.

The count of number of patients in corresponding to a specific disease can be made to store in a separate table containing statistical data. For instance the statistics can be made available once a month or a day using a trigger for time in the database server.