**Software Design Document**

April 5, 2020

Team 4 (Mariem Bchir, Liam Carriker, Joe Kane, Mary Lasater, Lina McDermott, Caroline Sigl)

**Introduction:**This report describes Team 4’s Software Design for our project to develop an application to connect teachers and schools in the MENA region.

1. Introduction

1.1. Purpose

A number of schools in the African Continent and MENA region suffer a low teacher retention rate and use non-digital recruitment channels to replace and recruit new educators. The purpose of our project is to digitize the recruitment and search process for both teachers and schools by providing an online platform (mobile application) hosting the whole process.

1.2. Scope

This document covers the software design specifications for the entirety of the first version (pilot) of a completely new application. The developers’ main concerns in regards to this document are feasibility and clarity of the system. The developers wish for the system to be readable and maintainable for any future development. The primary concern of any users of the system is the reliability of the system.

1.3. Definitions, Acronyms, and Abbreviations

UDP - User Datagram Protocol

UML - Unified Modelling Language

OO - Object Oriented

1.4. References

JAVA community Process

Android standards

UML Documentation

1.5. Overview

This application aims to connect schools to teachers seeking employment. Schools and teachers will be able to create and edit accounts to display relevant information. Our application will provide a search and filtering engine that will allow both parties to find their best fit as well as potentially provide a matching algorithm that automatically pairs teachers and schools and provides them with suggestions.

2. Software Design Description

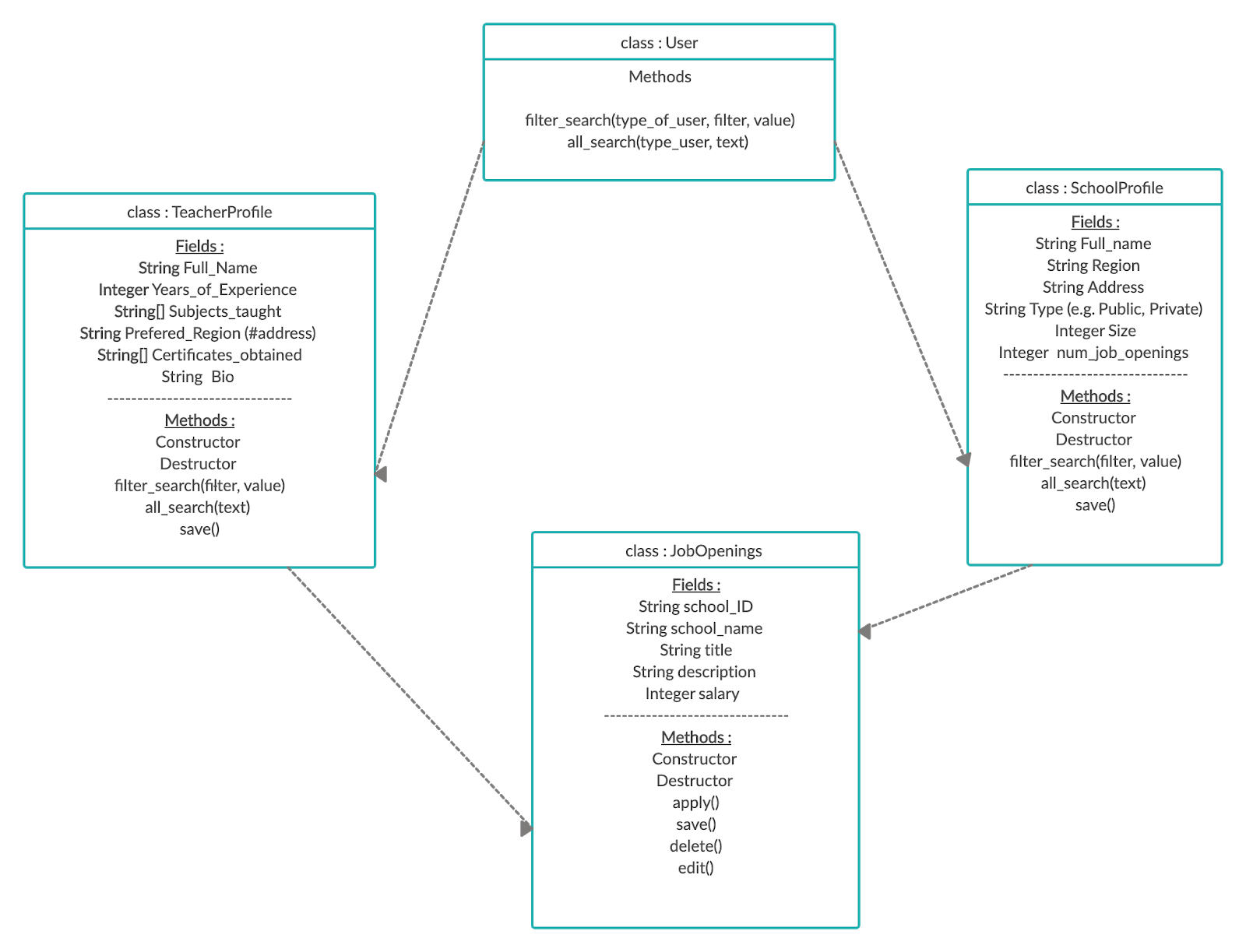
2.1. Client

2.1.1. Design Overview

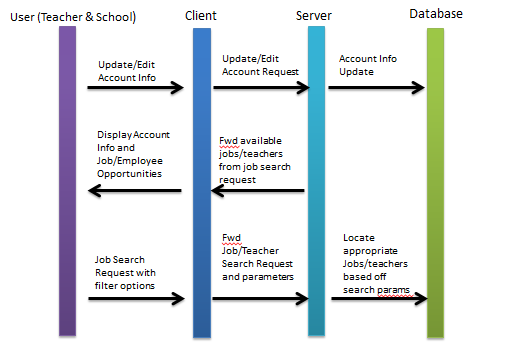
We are performing object-oriented programming. This component will allow the user to make requests to the server and pull information from the server in interaction with our database. The client will display information from the server to the user.

2.1.2. Language and Infrastructure

We will be using the Java programming language.

2.1.3. Class diagram or data structure diagram

2.1.4. Sequence diagram for key use cases



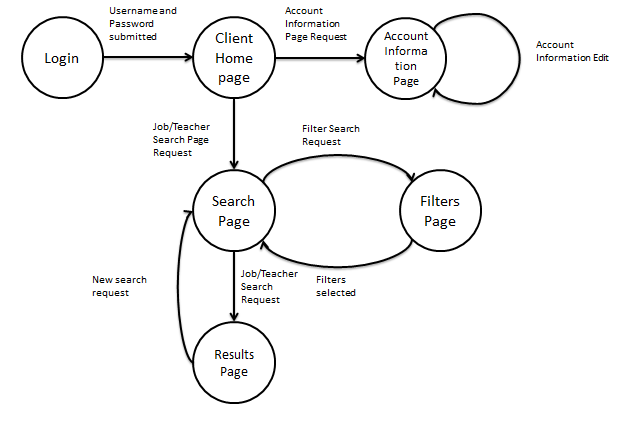
2.1.5. Detailed Design

2.1.5.1. Logic/Algorithm Design

The methods described in the design below are designed to mainly employ loose coupling. That is, we intend to only use methods and their parameters to pass data between classes. For our User class, we will use inheritance coupling. Additionally, many of the functions used in MainActivity and the classes it navigates to employ a procedural level of cohesion.

* MainActivity - Activity that opens when app is opened
  + onCreate() - initializes application upon startup
    - LoginFragment - opens if no user is signed in
      * Navigates to DashboardFragment
    - DashboardFragment
      * Displays buttons to navigate either to SearchFragment or to edit account information
      * Buttons displayed depend on the Type of the User (School or Teacher)
        + Methods for each button are described in further detail below in the User Classes section
    - SearchFragment - can be navigated to using a callback method for a button displayed in MainActivity
      * allSearch() - Uses a request builder class to build SearchRequest objects
      * makeRequest() - makes request to server
      * displayResults() - uses a ViewModel to display search results
        + Shows information from Teacher/School classes in a RecyclerView
        + filterSearch() - filters the list of SearchResponse objects returned by makeRequest()
* User classes
  + TeacherProfile and SchoolProfile are children classes of User, meaning they are inheritance coupled.
  + Any of the methods that involve editing or interacting with the database will make requests to the server in order to do so.
  + editAccount() - used to edit account information.
  + addJobOpening() - used by SchoolProfile class to add job openings.

2.1.5.2. State Diagram



2.2. Server

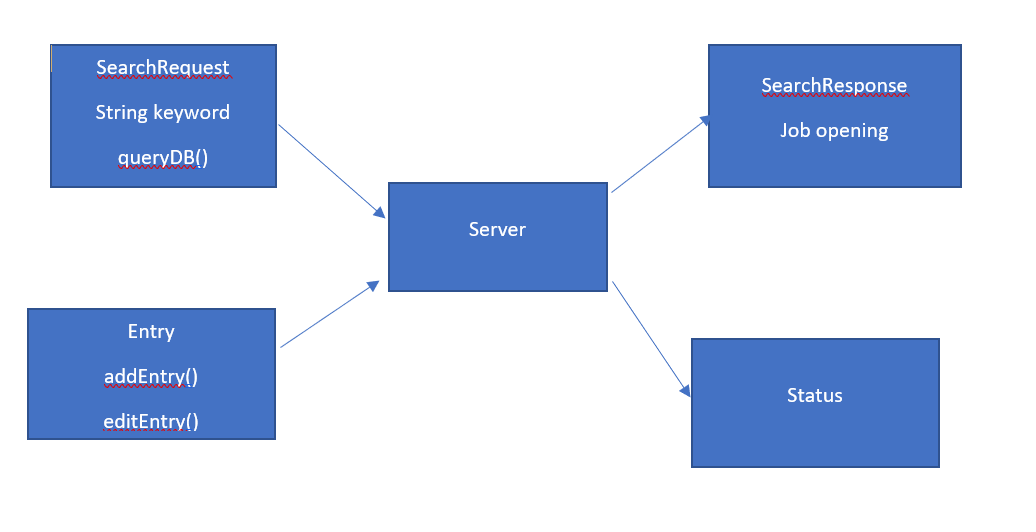
2.2.1. Design Overview

We will be performing an OO design. The server component will allow the client to communicate with the database, and retrieve information for the client. The server accepts SearchRequests and returns SearchResponses. Additionally, when users edit or add information to their accounts, the client makes a request to the server and the server will interact with the database, returning a Status to indicate level of success.

2.2.2. Language and Infrastructure

All client-server interaction will be written in the Java programming language. We will largely be using third party methods to implement the server-database connection.

2.2.3. Class diagram or data structure diagram



2.2.4. Sequence diagram for key use cases

See diagram from section 2.1.4.

2.2.5. Detailed Design

2.2.5.1. Logic/Algorithm Design

* makeRequest() - The client will make a request to the server in the form of a SearchRequest.
  + queryDB() - The server will parse the request and create a query for the database.
    - If there is an error parsing the request, the server will return an error to the client.
  + The server will build a list of SearchResponse objects to pass back to the client.
    - The server will use a builder method to build each SearchResponse object.
* addEntry() - The client will send information to add an account entry to the database.
  + The server will parse information and use a third party method to add an entry to the database.
  + The server will return a Status object indicating level of success.
* editEntry() - The client will send information to edit an account entry in the database.
  + The server will parse information and use a third party method to edit an entry in the database.
  + The server will return a Status object indicating level of success.

2.3. Database

2.3.1. Design Overview

We are using a relational database. The database component will store the information of given schools and teachers. The server will be able to access the database information.

2.3.2. Language and Infrastructure

We will be using SQL for the database component. The database will have three tables, one for School Users and another Teacher Users, as well as a table for job openings.

2.3.3. Class diagram or data structure diagram



2.3.4. Sequence diagram for key use cases

See diagram from section 2.1.4.

2.3.5. Detailed Design

2.3.5.1. Database Design

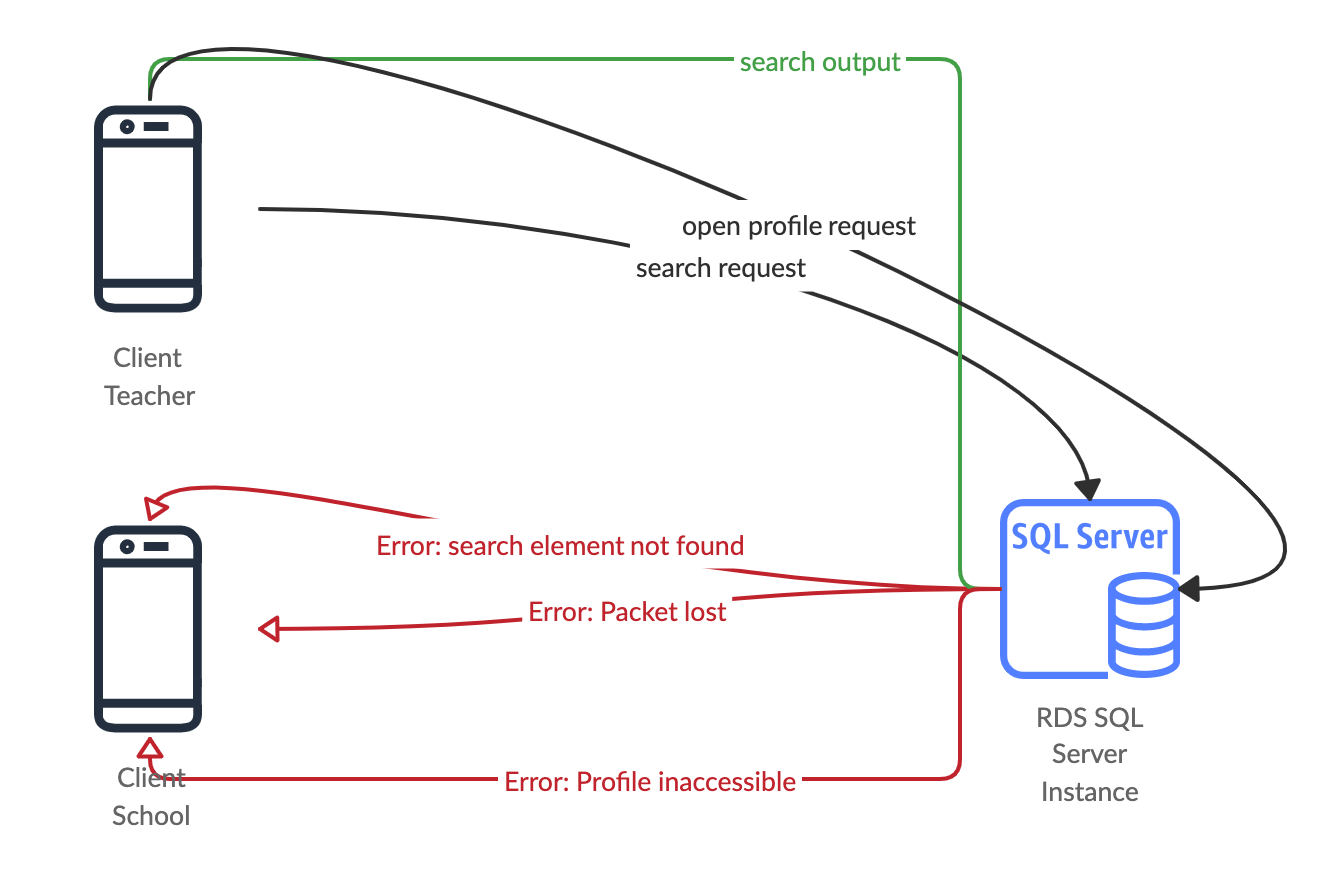
Many of the functions that interact directly with the database will be third-party methods. Below, we detail what an entry for the School table will comprise of and what an entry for the Teacher table will comprise of. We also detail the entries in the JobOpenings table.

* School
  + ID (int)
  + Name (string)
  + Region (string)
  + Type (string)
    - Options are “Public” or “Private”
* Teacher
  + ID (int)
  + Name (string)
  + YearsOfExperience (int)
  + Region (string)
  + Subjects (booleans)

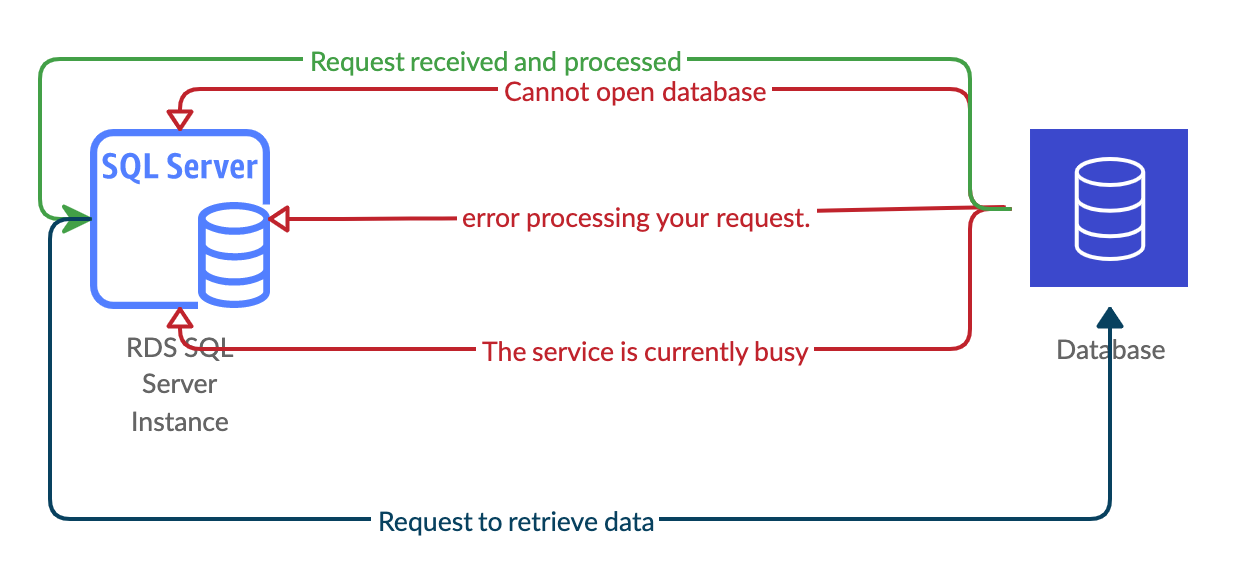
* + - There will be a field for each possible subject, and each Teacher entry will fill these fields in with True or False depending on whether the teacher user is able to teach those subjects.
  + Certificates (booleans)
    - There will be a field for each possible certificate, and each Teacher entry will fill these fields in with True or False depending on whether the teacher user has obtained these certificates.
* JobOpening
  + SchoolID (int)
  + SchoolName (string)
  + Subject (string)
  + Salary (int)
  + Region (string)
  + Description (string)

3. Inter-component or inter-subsystem communications

3.1. Client-Server communications



3.2. Server-Database communications



4. Metrics

4.1. Size information

4.1.1. s(MainActivity) = 200 LOC

4.1.2. s(LoginFragment) = 100 LOC

4.1.3. s( LoginViewModel) = 100 LOC

4.1.4. s( DashboardFragment) = 100 LOC

4.1.5. s(DashboardViewModel) = 100 LOC

4.1.6. s(SearchFragment) = 100 LOC

4.1.7. s(SearchViewModel) = 100 LOC

4.1.8. s(User) = 100 LOC

4.1.9. s(TeacherProfile) = 200 LOC

4.1.10. s(SchoolProfile) = 200 LOC

4.2. Complexity

4.2.1. Weighted methods per class

4.2.1.1. (Using information flow metrics): WMC = 600LOC(6input\*5output)^2 + 200LOC(2input\*2output)^2 + 200LOC(6input\*5output)^2 + 200LOC(5input\*6output)^2 = 540000 + 3200 + 180000 + 180000 = 903200

4.2.2. Depth of inheritance tree

4.2.2.1. Our application has two instances of inheritance; SchoolProfile and TeacherProfile each inherit from User. Thus, *DIT* = 2.

4.2.3. Coupling between classes

4.2.3.1. MainActivity *CBC* = 2

4.2.3.2. LoginFragment *CBC* = 2

4.2.3.3. LoginViewModel *CBC* = 3

4.2.3.4. DashboardFragment *CBC* = 2

4.3.3.5. DashboardViewModel *CBC* = 3

4.3.3.6. SearchFragment *CBC* = 2

4.3.3.7. SearchViewModel *CBC* = 3

4.3.3.8. User *CBC* = 2

4.3.3.9. TeacherProfile *CBC* = 5

4.3.3.10. SchoolProfile *CBC* = 5

4.2.4. Response for a class

4.2.4.1. TeacherProfile Class Response = 7

4.2.4.2. SchoolProfile Class Response = 7