NUST Connect

System Design Document

Project Phase 2

Muhammad Saifullah Khan
12-8-2016

1 Table of Contents

Revision History

Name	**Date**	**Reason For Changes**	**Version**
Initial version	07/12/2016	Initial system design containing minimal details as required by Project Phase 2 assignment.	1.0

2 INTRODUCTION

2.1 PURPOSE AND SCOPE

The purpose of this document is to document the design of NUST Connect system, and to identify different design patterns which may be applied to the system when implemented. It contains *UML Package Diagrams* and *UML Component Diagrams* of the NUST Connect system, and deals with a very high-level and abstract design of the system only.

The design patterns identified fall under the categories of *Creational*, *Behavioral* and *Structural Design Patterns*. At least three design patterns from each of these categories will be identified and applied to the NUST Connect system.

2.2 PRODUCT PERSPECTIVE

This system, named the NUST Connect is a platform where the whole community of National University of Sciences and Technology can come together and interact with each other. The main purpose of the system is to provide the NUST community with a LinkedIn-like platform which they can use to connect with other people in the university. NUST community is defined as the students, faculty members, administration and organizations operating within NUST.

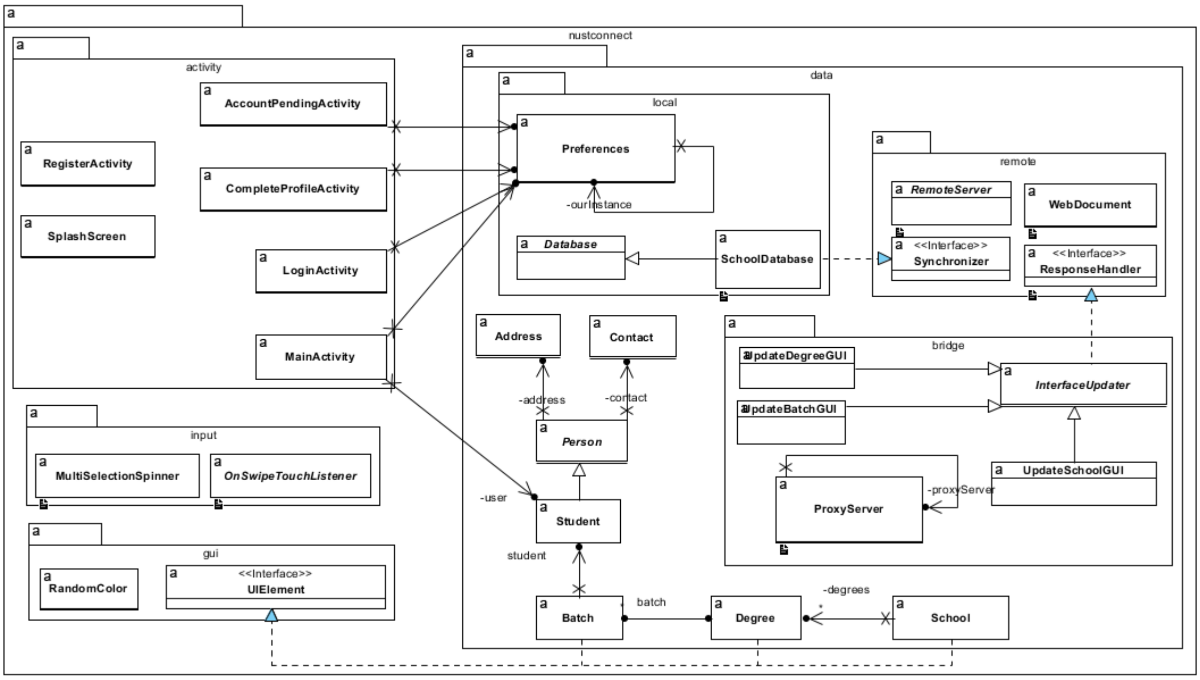
The major functions to be supported by the system are listed below. Detail of the same could be read in Section 3 of the SRS of the system.

1. User accounts
    a. Login
    b. Register
2. Newsfeed
3. Profile page
    a. Timeline
    b. About the user
        i. Personal details
        ii. Work history
        iii. Education history
4. Browse user accounts
5. Search for users
    a. By name
    b. By registration number
6. Posts
    a. Post on your own timeline
    b. Post on someone else's wall
7. Notifications
    a. Subscribe for notifications from users
8. Unfollow users to hide posts from them
9. Send messages to other users

3 PACKAGE DIAGRAMS

3.1 ANDROID APPLICATION

The following diagram shows the packages and classes for Android application (part of the *smartphone applications* sub-system). Activity in Android terminology is, in most simple terms, a screen.



4 COMPONENTS

4.1 HIERARCHY TREE

- System
    - Server
        - Web Server
            - User Management System (UMS)
            - Website Generator (WG)
            - Client Application Request Handler (CARH)
        - Database Server
            - User Database
            - Employee Database
            - Schools Database
    - Admin Panel
        - User Accounts
            - Account Creator
            - Detail Verifier
        - School Records Manager
            - School Manager
            - Department Manager
            - Degree Manager
            - Batch Manager
    - Smartphone Application
        - Local Database
        - Notifications System
            - Notification Manager
            - Notification Generator
        - Messaging
            - Message Sender
            - Recipient Validator
            - User Interface
                - Login UI
                - Registration UI
                - Messaging UI
                - Notifications UI
        - Profile
            - Timeline
                - Wall Posts
                - Status Updates
            - About User
            - Settings

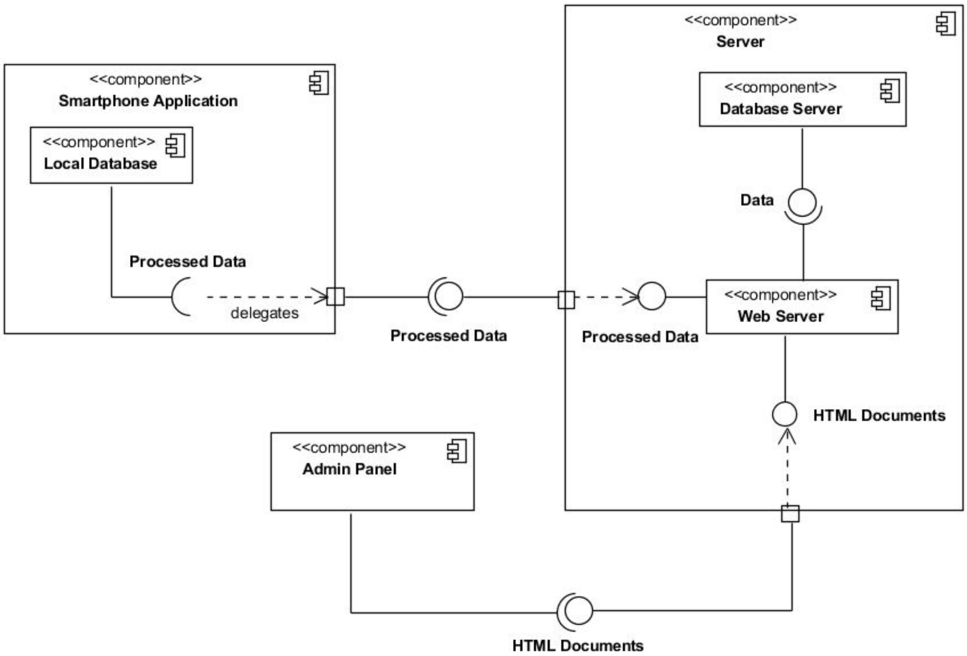
4.2 DIAGRAMS

4.2.1 System Overview

This component diagrams describes the system on a very abstract level by showing only the highest level sub-systems. Each sub-system will be explored in further details in the following diagrams. The system, which has a client/server architecture,

The system has two kinds of clients:

1. **Smartphone Application**
   The smartphone applications for students and employees of National University of Sciences and Technology (NUST). These may be further sub-divided into Android and iOS applications.
2. **Admin Portal**
   A web application the system administrators. It is used for performing administrative tasks, such as accepting or declining user account registration requests after verifying their details.



4.2.2 Server

This component diagram describes the *server* sub-system, exploring it in more depth than the previous component diagram. The server is further divided into two main components; the web server and the database server.

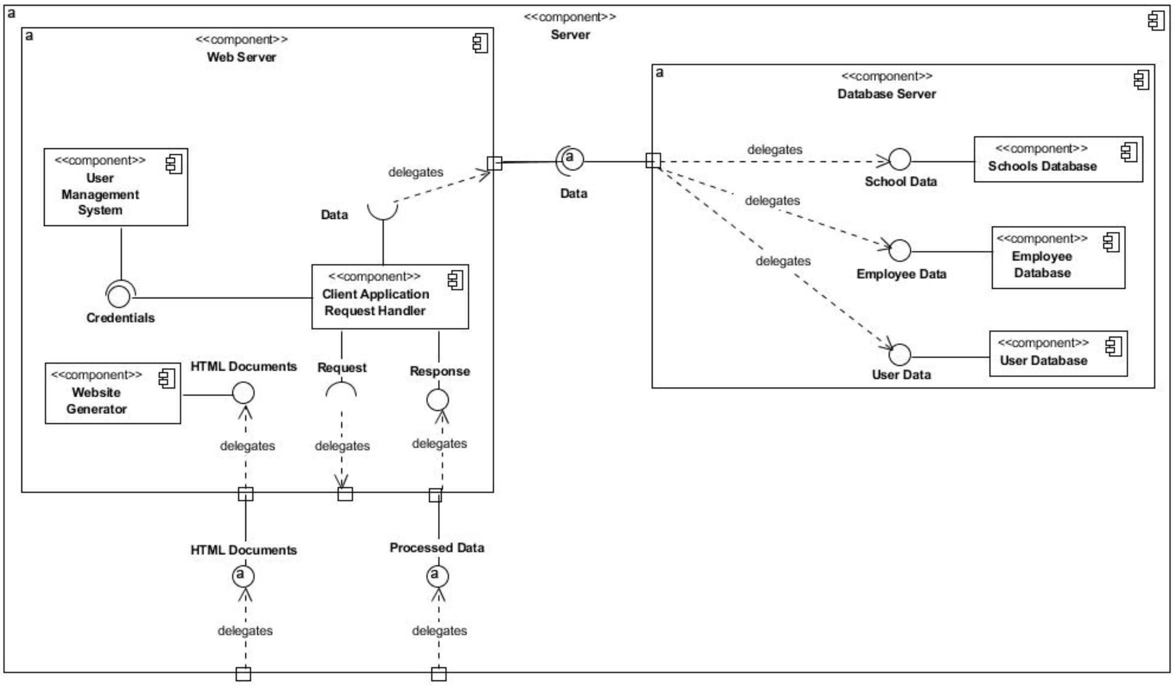
The database server is responsible for storing and serving data to the web server, which acts as an intermediate between the client applications and the database server. No client application can access the data directly from the databases in database server.

There are three types of databases in the database server:

1. **Employee database**
   A database which contains data related to the *Admin Panel* client web application and its users.
2. **User database**

This database contains data related to the *Smartphone Application* client of the system, and its users.

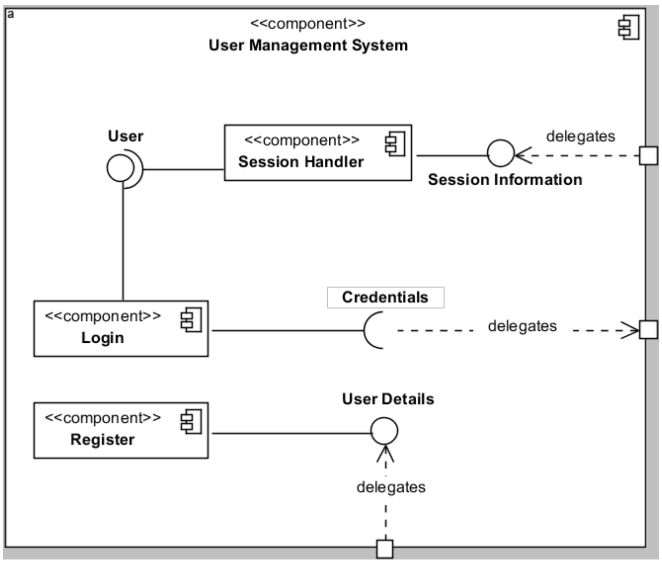
3. **Schools database**
   This database contains details of all the schools of National University of Sciences and Technology (NUST), different departments and degrees being taught at each school and department, and the batches currently studying in different degrees.



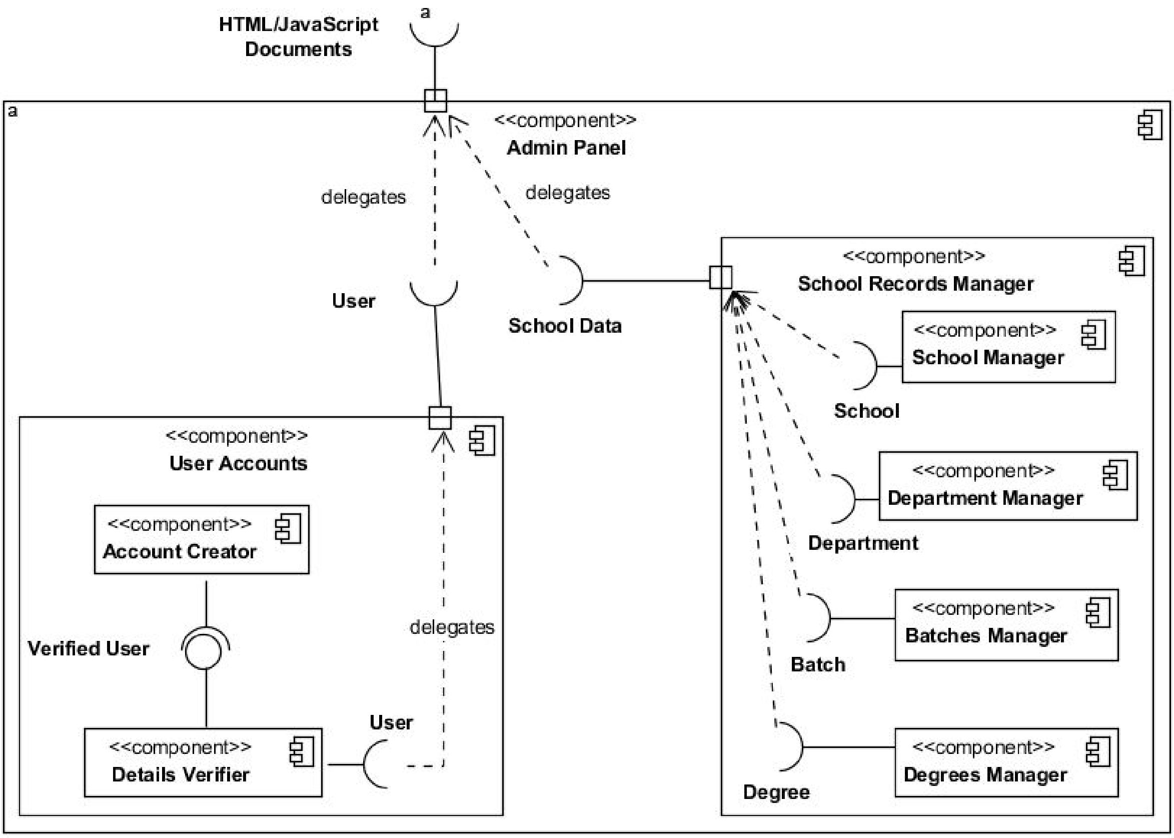
4.2.3 User Management System

The user management system is a component of the web-server, which deals with actions related to managing user data and accounts. Its two major functions are user registration for new accounts and logging users in after checking in the credentials provided against the user data stored on the database server.

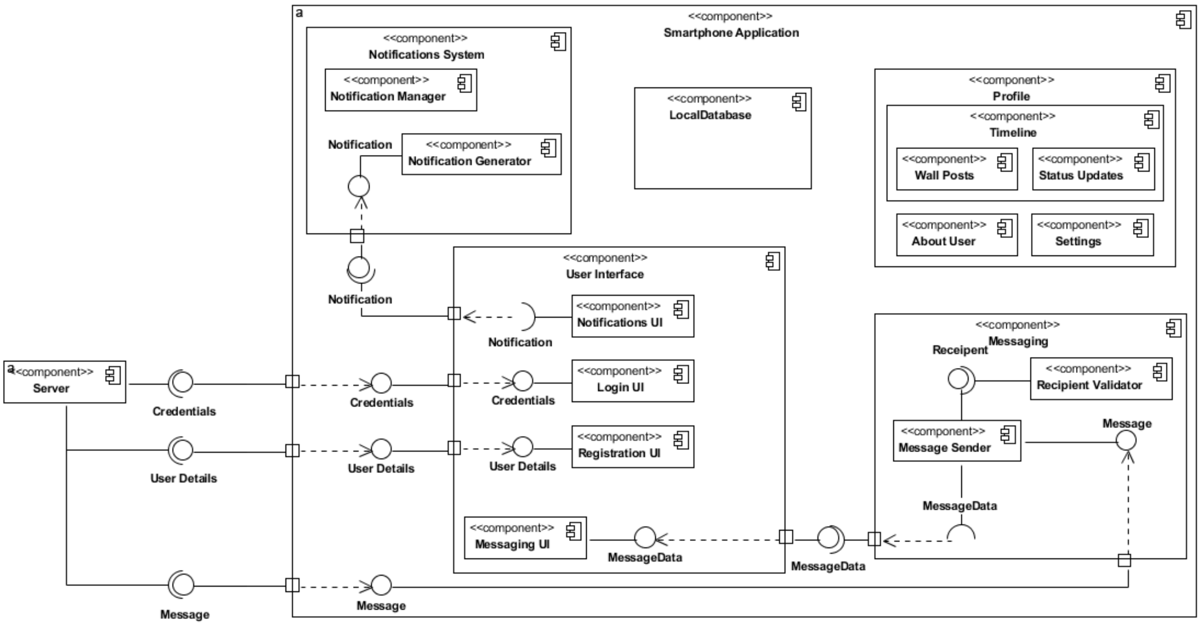
Session handler is also a part of this sub-system, which maintains details of a session when a user is logged in.



4.2.4 Admin Panel



4.2.5 Smartphone Application



5 DESIGN PATTERNS

5.1 CREATIONAL DESIGN PATTERNS

5.1.1 Singleton

From the diagrams given above, Singleton design pattern can be applied at at-least two places in the system.

First, the *Preferences* class in the client application can be made a Singleton, because *Preferences* contains a subset of data from the remote database on the database server which is cached locally and we want it to be globally accessible inside the application. Therefore, making it a Singleton and providing a global access point would provide a solution for what we want to do.

Similarly, the *ProxyServer* class can also be made a Singleton. This class in the client application is used by other classes for all the communications with the web server.

5.1.2 Prototype

Prototype design pattern can be used for creating different *Activities* in the Android application. This pattern allows us to make a prototype object and then clone and modify it to create specialized objects.

In Android, an Activity, in most simple words, represents a single screen of the application and each Activity must have its own class. Since most of the Activities are hugely similar, sometimes differing only in layout which is provided through an XML document, we can safely use prototype pattern here to create prototype Activity and then specialize it by plugging in different types of XML layouts and any other specializations needed.

5.1.3 Factory Method

The system has many different types of users, such as Student, Staff, Employee, Admin, Faculty, etc. Each of the users will be represented by a separate class extending the abstract class User. Creation of different types of users can be simplified for the client by providing an interface using the Factory Method, which would decide which type of User object to create through delegation.

5.2 STRUCTURAL DESIGN PATTERNS

5.2.1 Adapter

The type of database (e.g. MySQL, Oracle, etc.) used in the server may be different from the type of database supported by Android platform (i.e. SQLite). The local database in the Android application is to store a subset of data from remote database locally. Since these two database may provide different interfaces, and adapter class can be used between the MySQLDatabase and SQLiteDatabase compatible. This adapter class would be implemented using the Adapter Design Pattern.

5.2.2 Proxy

Proxy design pattern can be used for the *ProxyServer* class which is an intermediary between the client classes and the web server. In Proxy design pattern, an object represents another object. In our case of *ProxyServer*, it would be representing the *RemoteSever* class.

5.2.3 Façade Design Pattern

Façade design pattern is used when a segment of the client community needs a simplified interface to the overall functionality of a complex subsystem. The Facade defines a unified, higher level interface to a subsystem that makes it easier to use. In our system this design pattern may be used for the Profile component, which has a complex internal structure, and providing a *unified, higher level interface* may make it easier for the other classes and components to interact with the Profile component.

5.3 Behavioral Design Patterns

5.3.1 State

A *Post* object (which represents a status post like in Facebook) has different behaviors in different states. These states are defined by the Visibility controls which can set the target audience of the *Post*. For example, in Public state, the Post should be visible to all users, but in *School* state, the Post should be visible only to people of the same school as the user who posted.

These states and changing behaviors with each state can use the state design pattern.

5.3.2 Chain of Responsibility

As can be seen from the Component Diagrams above, a *Message* object is passes through different classes which process it on its journey from one user's application to another user's application. To avoid coupling the sender with the receiver, we can use the chain of responsibility design pattern here which encapsulates the processing elements inside a "pipeline" abstraction; and have clients "launch and leave" their requests at the entrance to the pipeline.

In this way, *Message* object could be processed by different nodes which the *Sender* would not need to worry about.

5.3.3 Observer

Observer design pattern is a way of notifying change to a number of classes. This pattern can be used in many different places in the system. For example, the notifications system could use this pattern. Whenever a new notification arrives, this design pattern could be used to notify all the relevant classes who need to know about this notification.

Or, if multithreading is implemented anywhere in the system, especially on the web server, this design pattern can be employed.