TAPS Design

Prepared by Moti Begna

University of Minnesota

Computer Science Department

April 1st, 2019

1. Introduction

1.1. System Overview

TAPS will be a backend system that will process various user requests submitted by University campus community members such as students, faculty, staff, and administrators. This system is designed to help faculty members organize and handle large amounts of TA recommendations and allow prospective TA's to enter their course preferences as well as enter their own personal and academic details. TAPS will also allow administrators to assign TAs to courses, sort their preferences based on different attributes, as well as make announcements and status updates to notify TAs, faculty, and payroll. In addition, TAPs will allow payroll managers to view appointments and percentage types, as well as a method for the system itself to utilize third-party software to acquire external data.

1.2. Design Objective

The objective of this design document is as follows:

- To provide a method of design for the system to be reliable, usable, and modifiable.
- To provide a method of design to ensure that the system meets performance and security requirements.
- To provide enough information for programmers to implement the system.
- To provide enough detail to formulate and implement test cases for various subsystems before any code is developed.

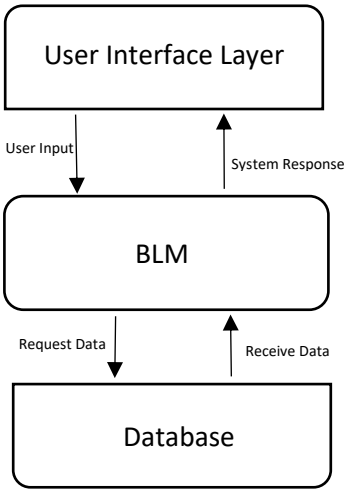
2. Glossary

BLM: Business Layer Module, which interacts with the User Interface Layer and the Database. Through this layer, all requests of the system is handled.

3. Design Overview

3.1. System Architecture

The software will receive a user input from the user, and the BLM will extract pessary data from the database (if necessary), and produce a corresponding response based on what was requested.



3.2. System Interfaces

3.2.1. Database Interface

The interface to the Database is utilized to obtain necessary data that must be transported to the BLM. In the current system, the Database contains all TA applications that have been submitted, as well as information regarding those individuals that have been appointed to a TA position. In this way, the Database can be described as two sub-databases: a TA application database, and an Appointed TA database. The BLM utilizes the Database in these manners:

1. When a prospective TA submits an application, the BLM sends the application to the TA application database.
2. When requested by an Administrator, the BLM retrieves all TA applications from the TA application database.
3. When an Administrator assigns a TA, the BLM sends the assignment to the Appointed TA database.
4. When requested by an Administrator, the BLM retrieves all TA applications from the TA application database.

3.2.2. User Interface

The user interface is utilized to extract a request from the user in order for the BLM to produce a corresponding response. The BLM determines the credentials of the user and allows a limited range of requests to the system based on those credentials.

3.3. Constraints and Assumptions

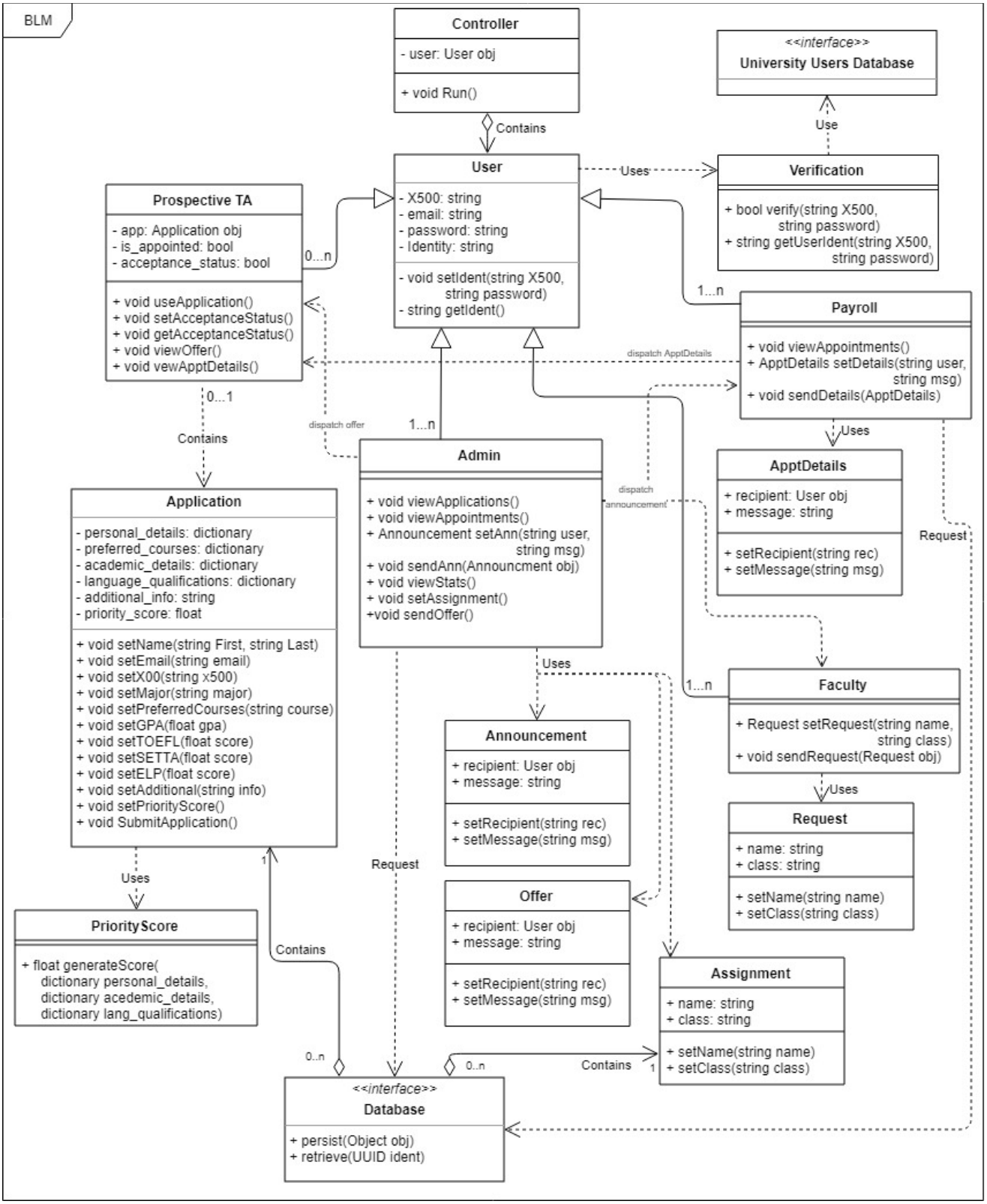
We make the following assumptions:

- The language used to implement the design allows for the use of all of the defined data types within the UML design.

- The User Interface and Database Interface will be implemented in such a way that communication with the BLM is functional and effective.

  Because the TAPS architecture utilizes a layered model, and we have no way of knowing how the UI Layer and Database Layer will be implemented, we must assume that any choices that are made for those interfaces ensures that communication to the BLM is maintained.

4. UML Design


BLM
Controller
- user: User obj
+ void Run()
Contains
<<interface>>
University Users Database
Use
User
- X500: string
- email: string
- password: string
- Identity: string
- void setIdent(string X500, string password)
- string getIdent()
Uses
Verification
+ bool verify(string X500, string password)
+ string getUserIdent(string X500, string password)
Prospective TA
- app: Application obj
- is_appointed: bool
- acceptance_status: bool
+ void useApplication()
+ void setAcceptanceStatus()
+ void getAcceptanceStatus()
+ void viewOffer()
+ void vewApptDetails()
0...n
1...n
Payroll
+ void viewAppointments()
+ ApptDetails setDetails(string user, string msg)
+ void sendDetails(ApptDetails)
dispatch ApptDetails
0...1
Contains
dispatch offer
1...n
Admin
+ void viewApplications()
+ void viewAppointments()
+ Announcement setAnn(string user, string msg)
+ void sendAnn(Announcment obj)
+ void viewStats()
+ void setAssignment()
+void sendOffer()
Uses
ApptDetails
+ recipient: User obj
+ message: string
+ setRecipient(string rec)
+ setMessage(string msg)
dispatch announcement
Request
Application
- personal_details: dictionary
- preferred_courses: dictionary
- academic_details: dictionary
- language_qualifications: dictionary
- additional_info: string
- priority_score: float
+ void setName(string First, string Last)
+ void setEmail(string email)
+ void setX00(string x500)
+ void setMajor(string major)
+ void setPreferredCourses(string course)
+ void setGPA(float gpa)
+ void setTOEFL(float score)
+ void setSETTA(float score)
+ void setELP(float score)
+ void setAdditional(string info)
+ void setPriorityScore()
+ void SubmitApplication()
Uses
1...n
Faculty
+ Request setRequest(string name, string class)
+ void sendRequest(Request obj)
Uses
Announcement
+ recipient: User obj
+ message: string
+ setRecipient(string rec)
+ setMessage(string msg)
Request
Request
+ name: string
+ class: string
+ setName(string name)
+ setClass(string class)
1
Uses
PriorityScore
+ float generateScore(dictionary personal_details, dictionary acedemic_details, dictionary lang_qualifications)
Contains
Offer
+ recipient: User obj
+ message: string
+ setRecipient(string rec)
+ setMessage(string msg)
Assignment
+ name: string
+ class: string
+ setName(string name)
+ setClass(string class)
0..n
0..n
Contains
1
<<interface>>
Database
+ persist(Object obj)
+ retrieve(UUID ident)