TAPS
Design Document

Author:
Tasha Bornemann
Mai Nguyen
Guanpin Zhong
Xinyi Yan

Group: 17

Document Revision History

Date	Version	Description	Author
4/15/2019	1	Initial Draft	Group 17
4/27/2019	2	Final Design	Group 17

Contents

1. Introduction

1.1 Purpose

The purpose of this document is to describe the proposed design of the TAPS system. It is intended for use by a development team to implement the system as described. First, an overview of the design is described, followed by descriptions of the various layers of the design and a more in-depth description of the Business Layer Module.

1.2 System Overview

The TA Processing System (TAPS) is intended to be a system for assigning graduate TAs to all Computer Science and related courses at the University of Minnesota. The system will be used by prospective TAs, appointed TAs, faculty members, administrative staff, and payroll staff for their respective objectives.

1.3 Design Objectives

This design intends to describe the basic organization and functions of the system. Primarily, it is focused on how the system will fulfill user requests, as much of the functionality of the system is based on user interactions. This will include how the system responds to various user requests by creating, storing, and modifying information in internal and external data sources. Meanwhile, the user interface and the data layer are described in general terms and the details of these are left to another team.

In detail, the document will specify how the system should let users complete the following actions:

1. The system shall require users to enter credentials before using it.
2. The system shall allow prospective TAs to fill out and submit a TA application.
3. The system shall allow faculty members to recommend TAs.
4. The system shall allow faculty members to request TAs.
5. The system shall allow relevant information to be stored and allow administrative staff to view this information.
6. The system shall allow administrative staff to assign a TA to a course.
7. The system shall allow payroll staff to view the list of TA appointments.
8. The system shall allow payroll staff to send appointment offers to appointed TAs.
9. The system shall allow appointment TA to accept or decline a TA offer.
10. The system shall provide a way to store the appointment status of a TA and allow administrative staff to view and update it.
11. The system shall provide a way for users to receive notifications. Users shall be able to receive notifications that are relevant to them.

1.4 References

The TAPS Requirements Document contains the User Requirements and System Requirements Specification that this design intends to fulfill.

1.5 Definitions, Acronyms, and Abbreviations

TA: Teaching Assistant
TAPS: TA Processing System
BLM: Business Layer Module
UI: User Interface
OO: Object-Oriented

UUID: Universally Unique Identifier

2 Design Overview

2.1 Introduction

This design uses an object-oriented (OO) approach, where parts of the system are represented as objects connected through relationships and objects are further described by their attributes and operations. A UML class diagram is used to depict the OO design. The system architecture follows a multitier client-server architecture with three layers: the UI/Presentation layer, Business Logic layer, and Data Persistence/Infrastructure layer.

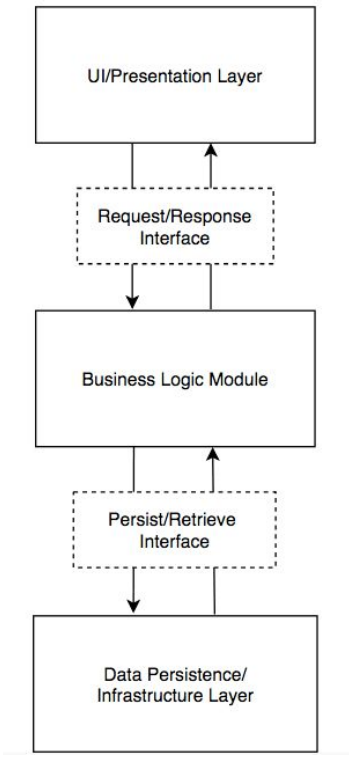
2.2 Environment Overview

Users will interact with the system via a web browser. Further environmental factors TBD.

2.3 System Architecture

The system architecture is split into three layers: the UI/Presentation layer, Business Logic Module, and Data Persistence/Infrastructure layer. The UI layer defines the presentation of the system to the user and the direct user interactions that occur. The Data Persistence layer defines the underlying structure of how data is persistently stored, such as logging mechanisms and/or database organization. The Business Logic module connects these two layers and defines the system's main functionality: how the system fulfills user requests by storing and accessing data. There are two interfaces that define how the layers interact: the Request/Response Interface, which details how the BLM handles requests from the UI layer, and the Persist/Retrieve Interface, which details how the BLM stores and accesses data in the data layer.

2.3.1 Three-tier Architecture



2.4 Constraints and Assumptions

Constraints:
1. There are multiple users and different users have different access to some data or take some actions. The system is supposed to recognize the type of user for each user that logs in, and it should allow a user to only complete the corresponding legal actions.

In order to accommodate the constraint, we use the system data. It stores all the user information that are allowed to use the software. Whenever a user tried to log in, the system will find out the specific information and match it with the user. Once the user is verified as a legal user, the system will record it, and in the future the user's behavior will be constrained within a certain scope.

2. For the notification and announcement part, there should be a standard structure for the message. There can be a platform or software with featured functionality that is able to provide convenient guidance when users want to write some words as the notification.

3. The third party database has an interface with TAPS, and the user should have the authority to access the third party database in order to get history data from the third party. Since there would be history data stored in the third party database and the user needs to be able to get into the database, the entry of it would be connected with TAPS.

Assumptions:
1. It is assumed that the application will be used via a web browser. Users must log in to the system before using it. It is assumed that much of the information is collected through separate forms, such as the TA application, faculty request, and faculty recommendation forms.

2. There is enough space for the implementation based on the given data amount and processes, so that the performance will not be limited.

3. The data from the external databases is correct and have been checked for compliance with any constraints. That means the system does not need to analyze the data in order to make sure it is usable and accurate.

4. All changes will be kept inside the system database, in case there is a need to get historical data. However, the old data will not be shown to users if it is unnecessary, and usually the most updated version will be displayed.

3 Interfaces and Data Stores

3.1 System Interfaces

3.1.1 Request/Response Interface

This interface is used to process requests from the UI layer. The UI layer acts as the point of contact with the user and the interface assembles user requests in a format that the BLM can interpret. A request is a function to be performed by the BLM, which may

include data to be stored or updated, or a request for data to be retrieved. This interface is also responsible for converting the results from the BLM into a format that is meaningful to the user.

3.1.2 Persist/Retrieve Interface

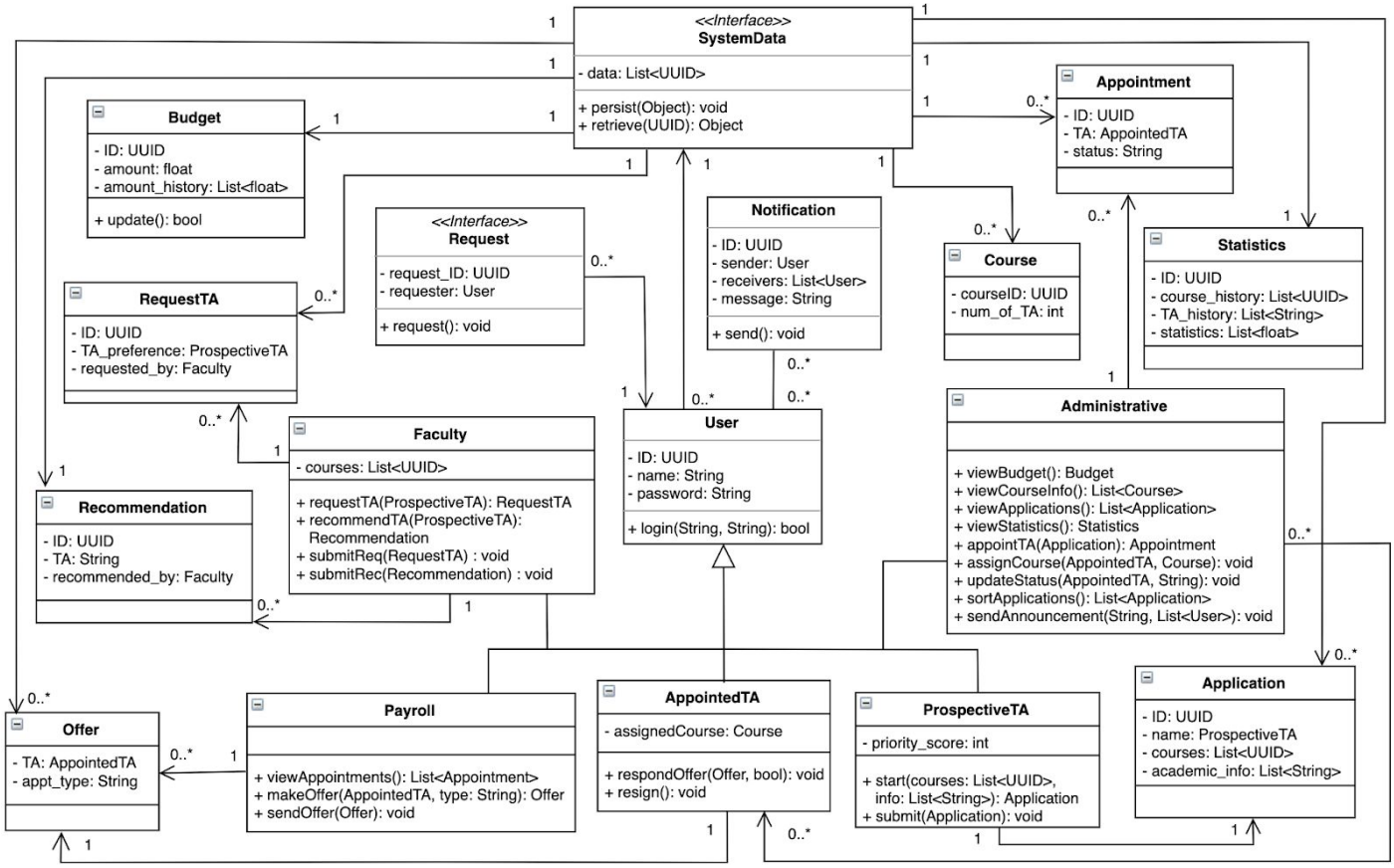
This interface is used to interact with the data layer, including retrieving stored data and sending data to be stored persistently. The interface translates the function produced by the BLM into a query that will retrieve, store, or update the corresponding data from the database. The interface is also responsible for converting the results of the query into a format that the BLM can interpret.

3.2 Data Stores

Most Objects, such as Applications, Requests/Recommendation, Notifications, and Offers are stored temporarily in the BLM before being delivered either to the Interface with the UI layer or the Interface with the Data Persistence layer. Active/relevant Users are stored persistently.

4 Structural Design

4.1 Class Diagram



4.2 Class Descriptions

4.2.1 Class: Appointment

- Purpose: This class stores the necessary information of an appointment.
- Constraints: None
- Persistent: None

4.2.1.1 Attribute Descriptions

1. Attribute: ID
   Type: UUID
   Description: Specifies an identifier for each appointment.
   Constraints: Each ID of an appointment must be unique.

2. Attribute: TA
   Type: AppointedTA
   Description: Specifies the TA that was appointed for an appointment.
   Constraints: None

3. Attribute: CourseID
   Type: UUID
   Description: Specifies the assigned course for the appointed TA for an appointment.
   Constraints: CourseID must be unique.

4. Attribute: status
   Type: String
   Description: Specifies the active status of the appointment.
   Constraints: Each ID of an appointment must be unique

4.2.1.2 Method Descriptions

None

4.2.2 Class: RequestTA

- Purpose: to represent a request from a faculty and store relative information
- Constraints:
- Persistent:

4.2.2.1 Attribute Descriptions

1. Attribute: ID
   Type: UUID
   Description: Specifies an identifier for each request.
   Constraints: Each ID of a request must be unique.

2. Attribute: TA_preference
   Type: Prospective TA
   Description: Specifies the prospective TA that the faculty wants to request for
   Constraints: None

3. Attribute: TA_requested
   Type: Faculty
   Description: Specifies the faculty member that makes this request

Constraints: None

4.2.3 Class: Faculty

- Purpose: This class keeps track of different faculty members and their permitted actions.
- Constraints: None
- Persistent: None

4.2.3.1 Attribute Descriptions

1. Attribute: courses
   Type: List<UUID>
   Description: This represents a list of courses the faculty is teaching or is in charge of.
   Constraints: Each identifier must be unique.

4.2.3.2 Method Descriptions

1. Method: requestTA(ProspectiveTA TA)
   Return Type: Object
   Parameters: TA - the requested TA
   Return value: RequestTA
   Pre-condition: None
   Post-condition: A RequestTA object is created.
   Attributes read/used: It will set the ProspectiveTA's UUID as a TA_preference and a Faculty's UUID as the TA_requested_by from the requestTA class.
   Methods called: The constructor of RequestTA class, and the persist(Object obj) method of Database interface.
   Processing logic: This method initialized a RequestTA object, sets its attributes and return it.

2. Method: recommendTA(ProspectiveTA TA)
   Return Type: Object
   Parameters: TA - the recommended TA
   Return value: Recommendation
   Pre-condition: None
   Post-condition: A Recommendation object is created.
   Attributes read/used: It will set the ProspectiveTA's UUID as a TA and a Faculty's UUID as the cecommended_by from the Recommendation class.
   Methods called: The constructor of Recommendation class, and the persist(Object obj) method of Database interface.
   Processing logic: This method initializes a Recommendation object, sets its attributes and return it.

3. Method: submitReq(RequestTA form)
   Return Type: Object
   Parameters: TA - the requested object
   Return value: None
   Pre-condition: None
   Post-condition: A RequestTA object is stored in the database.
   Attributes read/used: None
   Methods called: persist(RequetedTA)
   Processing logic: After the RequestedTA object is created, this method will store it in the system database.

4. Method: submitRec(Recommendation rec)
   Return Type: Object
   Parameters: rec - the recommendation object
   Return value: None
   Pre-condition: None
   Post-condition: A Recommendation object is stored in the database.
   Attributes read/used: None
   Methods called: persist(Recommendation)
   Processing logic: After the Recommendation object is created, this method will store it in the system database.

4.2.4 Class: Recommendation

- Purpose: This class is theTA recommendation filled by faculty staff
- Constraints: None
- Persistent: None

4.2.4.1 Attribute Descriptions

1. Attribute: ID
   Type: UUID
   Description: A unique ID for each recommendation
   Constraints: Each identifier must be unique

2. Attribute: TA
   Type: String
   Description: The TA who is recommended in this recommendation
   Constraints: None

3. Attribute: recommended_by
   Type: Faculty
   Description: Specifies the faculty who makes this recommendation
   Constraints: None

4.2.4.2 Method Descriptions

None

4.2.5 Class: ProspectiveTA

- Purpose: This class is the prospective TA type of user, contains a certain movement that prospective TA can do inside the system.
- Constraints: None
- Persistent: None

4.2.5.1 Attribute Descriptions

1. Attribute: priority_score
   Type: int
   Description: A score that decides priority order of TA
   Constraints: None

4.2.5.2 Method Descriptions

1. Method: start(List<UUID> courses, List<String> info)

Return Type: Application
Parameters: courses, info
Return value: application
Pre-condition: None
Post-condition: There is a new application started
Attributes read/used: None
Methods called: None
Processing logic: An application is returned

4.2.6 Class: Application

- Purpose: This class is the application form which needs to be filled if a person wants to be TA.
- Constraints: None
- Persistent: None

4.2.6.1 Attribute Descriptions

1. Attribute: ID
   Type: UUID
   Description: A unique ID for each application
   Constraints: Each identifier must be unique

2. Attribute: name
   Type: ProspectiveTA
   Description: The name of the prospectiveTA
   Constraints: None

3. Attribute: course
   Type: List<UUID>
   Description: A list of classes that the person wants to apply for
   Constraints: None

4. Attribute: academic_info
   Type: List<String>
   Description: The necessary information for further sorting
   Constraints: All the necessary blank should be filled

4.2.6.2 Method Descriptions

None

4.2.7 Class: Statistics

- Purpose: This class is the statistics saved or will be saved into database
- Constraints: None
- Persistent: None

4.2.7.1 Attribute Descriptions

1. Attribute: ID
   Type: UUID
   Description: A unique number to identify the statistics

Constraints: Each identifier must be unique

2. Attribute: course_history
   Type: List<UUID>
   Description: The history information of the class
   Constraints: None

3. Attribute: TA_history
   Type: List<String>
   Description: The history information of the TA
   Constraints: None

4. Attribute: statistics
   Type: List<float>
   Description: Any other statistics needed
   Constraints: None

4.2.7.2 Method Descriptions

None

4.2.8 Class: CourseInfo

- Purpose: to represent detailed information of courses
- Constraints: None
- Persistent: None

4.2.8.1 Attribute Descriptions

1. Attribute: courseID
   Type: UUID
   Description: The unique class ID for each class
   Constraints: Each identifier should be unique

2. Attribute: num_of_TA
   Type: int
   Description: The number of TA needed for this class
   Constraints: Should be nonnegative

4.2.8.2 Method Descriptions

None

4.2.9 Class: Budget

- Purpose: to represent a user who is an administrative staff
- Constraints: None
- Persistent: None

4.2.9.1 Attribute Descriptions

1. Attribute: ID
   Type: UUID
   Description: The unique budget ID for each budget
   Constraints: Each identifier should be unique

2. Attribute: amount
   Type: float
   Description: The amount of money of a budget
   Constraints: Should be nonnegative

3. Attribute: amount_history
   Type: List<float>
   Description: The list of amount of money for budgets
   Constraints: Each value in the list should be nonnegative

4.2.9.2 Method Descriptions

1. Method: update()
   Return Type: bool
   Parameters: None
   Return value: update success or failure
   Pre-condition: There is a new budget information
   Post-condition: The new budget is added to the list of historical budget
   Attributes read/used: ID, amount and amount_history
   Methods called: None
   Processing logic: This method updates the amount_history, adding a new budget to the list.

4.2.10 Class: Administrative

- Purpose: to represent a user who is an administrative staff
- Constraints: None
- Persistent: None

4.2.10.1 Attribute Descriptions

None

4.2.10.2 Method Descriptions

1. Method: viewBudget()
   Return Type: Budget
   Parameters: None
   Return value: a Budget object containing the budget wanted to be seen
   Pre-condition: None
   Post-condition: A budget can be seen by the administrative staff
   Attributes read/used: The attribute ID, amount and amount_history of the returned Budget object will be seen
   Methods called: None
   Processing logic:
   A request is received from the Request interface for the administrative staff to view the budget information, which uses the System Data to retrieve the information from the Budget class. This is then returned back to System Data and System Data will return that information back to the administrative staff.

2. Method: viewCourseInfo()
   Return Type: CourseInfo
   Parameters: None
   Return value: a CourseInfo object containing the information for a certain course
   Pre-condition: None

Post-condition: The course information can be seen by the administrative staff
Attributes read/used: The attributes courseID and num_of_TA will be read.
Methods called: retrieve(UUID)
Processing logic:

A request is received from the Request interface for the administrative staff to view the course information, which uses the System Data to retrieve the information in the CourseInfo class. This is then returned back to System Data and System Data will return the course information back to the administrative staff.

3. Method: viewApplications()
   Return Type: List<Application>
   Parameters: None
   Return value: A list of Application objects.
   Pre-condition: None
   Post-condition: The list of applications can be viewed by the administrative staff.
   Attributes read/used: The attributes of name, courses and academic_info (from Application) will be read.
   Methods called: retrieve(UUID)
   Processing logic:

   A request is received from the Request interface for the administrative staff to view the list of applications, which uses the System Data to retrieve the information in the Application class. This is then returned back to System Data and System Data will return the list of applications back to the administrative staff.

4. Method: viewStatistics()
   Return Type: Statistics
   Parameters: None
   Return value: a Statistics object containing the statistics and history.
   Pre-condition: None
   Post-condition: The statistics and history can be viewed by the administrative staff.
   Attributes read/used: The attributes of the ID, course_history, TA_history, and statistics (from Statistics) will be read.
   Methods called: retrieve(UUID)
   Processing logic:

   A request is received from the Request interface for the administrative staff to view the statistics and history, which uses the System Data to retrieve the information in the Statistics class. This is then returned back to System Data and System Data will return the statistics information back to the administrative staff.

5. Method: appointTA(Application app)
   Return Type: Appointment
   Parameters: app
   Return value: An Appointment object
   Pre-condition: None
   Post-condition: A new Appointment is created and attributes are set.
   Attributes read/used: Attributes in the Application will be read, such as the ID of the application, name, courses and academic_info. Other attributes used will be the ID of the appointment, TA, courseID, and status.
   Methods called: persist(Object), retrieve(UUID), viewApplications(), sortApplications
   Processing logic:

A request is received from the Request interface for the administrative staff to appoint TAs, which uses the System Data to retrieve the information in the Application class. This is then returned back to System Data and System Data will return the list of Application back to the administrative staff. The Administrative class will sort the list of the TAs and appoint the TAs. The System Data will persist this information in the database.

6. Method: assignCourse(AppointedTA app_ta, UUID course)
   Return Type: void
   Parameters: app_ta, course
   Return value: None
   Pre-condition: None
   Post-condition: The appointed TA will have a course assigned to them.
   Attributes read/used: The attributes used read will the assignedCourse in AppointedTA
   Methods called: viewCourseInfo()
   Processing logic:
   A request is received from the Request interface for the administrative staff to assign courses to the appointed TA, which uses the System Data to retrieve the information in the CourseInfo class and return it back to the administrative staff. The administrative staff chooses a course (UUID) by setting the assignedCourse attribute for the specific appointed TA passed in as a parameter.

7. Method: updateStatus(AppointedTA app_ta, String status)
   Return Type: bool
   Parameters: app_ta, status
   Return value: true for success, false for failure
   Pre-condition: None
   Post-condition: The status of the appointed TA is changed/updated.
   Attributes read/used: The attribute status in AppointedTA
   Methods called: None
   Processing logic:
   The administrative staff sets the status of the appointed TA to something else. For example, this string could be "active" or "inactive" to signify if the appointed TA is still working or resigned.

8. Method: sortApplications()
   Return Type: List<Application>
   Parameters: None
   Return value: Sorted list of application
   Pre-condition: An unsorted list of applications
   Post-condition: A sorted list of applications based on the priority scores (ascending).
   Attributes read/used: priority_score in ProspectiveTA
   Methods called: viewApplication(), retrieve(UUID), persist(Object)
   Processing logic:
   A request is received from the Request interface for the administrative staff to sort Applications. It will call the viewApplication() function and will sort the applications accordingly to the ProspectiveTA's priority scores in ascending order. The System Data will persist this sorted list in the database.

9. Method: sendAnnouncement(String str, List<User> recipients)
   Return Type: void

Parameters: str, recipients
Return value: None
Pre-condition: None
Post-condition: The announcement is sent to all the users initiated in the list.
Attributes read/used: None
Methods called: persist(Object)
Processing logic:

The administrative staff writes something and send it out to the user specified in the recipients parameter. The content of the announcement is specifies within the parameter str. The System Data will persist them in the database.

4.2.11 Class: Notification

- Purpose: This class is the notification sending between users
- Constraints: None
- Persistent: None

4.2.11.1 Attribute Descriptions

1. Attribute: ID
   Type: UUID
   Description: The unique budget ID for each notification
   Constraints: Each identifier should be unique

2. Attribute: sender
   Type: User
   Description: The sender of this notification
   Constraints: The type and content should match the existing user

3. Attribute: receivers
   Type: List<user>
   Description: The receiver of this notification
   Constraints: The type and content should match the existing user

4. Attribute: message
   Type: String
   Description: The content of this notification
   Constraints: None

4.2.11.2 Method Descriptions

1. Method: send()
   Return Type: void
   Parameters: offer: None
   Return value: N/A
   Pre-condition: None
   Post-condition: A notification made by sender user is sent to all receiver user
   Attributes read/used: sender, receiver and message
   Methods called: None
   Processing logic:

   A request is received from the Request Interface for a user to send a notification to other users. A Notification object is created calling the Notification's send() method to send this notification.

4.2.12 Class: AppointedTA

- Purpose: This class represents a user who is an Appointed TA
- Constraints: None
- Persistent: None

4.2.12.1 Attribute Descriptions

1. Attribute: assignedCourse
   Type: UUID
   Description: This is the course that the TA has been assigned to.
   Constraints:

2. Attribute: status
   Type: String
   Description: This represents the appointments status of an appointed TA. For example, it may signify whether the TA has been appointed or resigned.
   Constraints:

4.2.12.2 Method Descriptions

1. Method: respondOffer(Offer offer, Bool response)
   Return Type: void
   Parameters: offer: the Offer that the TA is responding to, response: whether the TA is accepting or rejecting the Offer.
   Return value: N/A
   Pre-condition: A request is received for an Appointed TA to accept an offer they have been sent.
   Post-condition: A notification of the Appointed TA’s response is sent to administrative staff.
   Attributes read/used: None
   Methods called: A notification will get created and its send() method will be called.
   Processing logic:
   A request is received from the Request Interface for an appointed TA to resign from their appointment. A Notification object is created about the TA's response and sent to administrative staff by calling the Notification’s send() method.

2. Method: resign()
   Return Type: void
   Parameters: None
   Return value: N/A
   Pre-condition: A request is received for an Appointed TA to resign from their appointment.
   Post-condition: A notification of the Appointed TA’s resignation is sent to administrative staff.
   Attributes read/used: None
   Methods called: A notification will get created and its send() method will be called.
   Processing logic:
   A request is received from the Request Interface for an appointed TA to respond to an offer. A Notification object is created about the TA's resignation and sent to administrative staff by calling the Notification’s send() method.

4.2.13 Class: Offer

- Purpose: This class represents an appointment offer that payroll sends to appointed TAs.
- Constraints: None
- Persistent: None

4.2.13.1 Attribute Descriptions

1. Attribute: TA
   Type: AppointedTA
   Description: the appointed TA that the offer is for
   Constraints:

2. Attribute: appt_type
   Type: String
   Description: the type of appointment that the offer is for, defined as a percentage.
   Constraints:

4.2.13.2 Method Descriptions

None

4.2.14 Class: Payroll

- Purpose: This class represents a user who is a payroll staff member.
- Constraints: None
- Persistent: None

4.2.14.1 Attribute Descriptions

None

4.2.14.2 Method Descriptions

1. Method: viewAppointments()
   Return Type: List<Appointment>
   Parameters: None
   Return value: list of Appointments that have been created by Administrative staff.
   Pre-condition: Request received for Payroll to view the list of appointments.
   Post-condition: A list of Appointments is available to the Payroll user who requested it.
   Attributes read/used: None
   Methods called: retrieve() from System Data Interface
   Processing logic:
   Request is received from the Request Interface for a payroll staff member to view the appointments list. The retrieve() function in System Data Interface is called for each appointment UUID, which returns an Appointment object for each appointment. A list of all Appointments is then returned to Payroll.

2. Method: makeOffer(AppointedTA TA, String type)
   Return Type: Offer
   Parameters: TA: Appointed TA that will receive the offer, type: type of appointment that the offer is for.
   Return value: an Offer object containing the Appointed TA and the appointment type
   Pre-condition: Request received for Payroll to send an offer to an appointed TA.

Post-condition: An Offer has been created.
Attributes read/used: None
Methods called: the constructor for an Offer
Processing logic:
Request is received from the Request Interface for a payroll staff member to send an offer to an appointed TA. An Offer object is created containing the Appointed TA and the appointment type.

3. Method: sendOffer(Offer offer)
   Return Type: void
   Parameters: offer: the Offer object created by makeOffer.
   Return value: N/A
   Pre-condition: Request received for Payroll to send an offer to an appointed TA, offer has been created.
   Post-condition: Offer has been sent to the AppointedTA in the Offer.
   Attributes read/used: None
   Methods called: None
   Processing logic: After makeOffer() has been called and an Offer object has been returned to Payroll, sendOffer() is called, which sends the offer to the Appointed TA.

4.2.15 Class: User

- Purpose: This class represents a generic user of the system. Has subclasses for each type of user.
- Constraints: None
- Persistent: None

4.2.15.1 Attribute Descriptions

1. Attribute: ident
   Type: UUID
   Description: unique system identifier for each user
   Constraints: unique

2. Attribute: username
   Type: String
   Description: each user needs a username to log into the system
   Constraints: unique

3. Attribute: password
   Type: String
   Description: password for logging in
   Constraints:

4.2.15.2 Method Descriptions

1. Method: login(String username, String password)
   Return Type: bool
   Parameters: username and password entered
   Return value: whether the login was successful or not
   Pre-condition: Request received for a user to log in.
   Post-condition: User is logged in or not
   Attributes read/used: username, password
   Methods called: None

Processing logic:

User enters a username and password to log in returns True upon success and False upon failure of logging in.

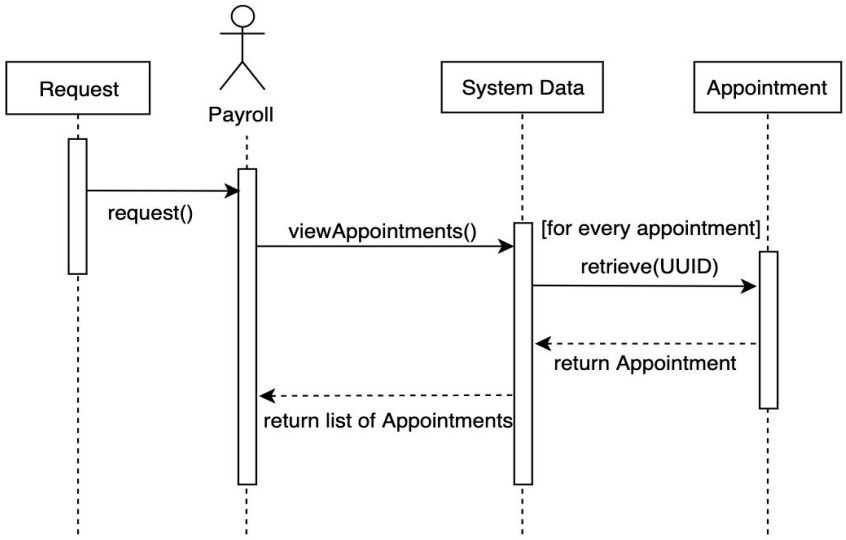
5 Dynamic Model

The purpose of this section is to model how the system responds to various events, i.e., model the system's behavior. We do this using UML sequence diagrams.
The first step is to identify different scenarios (e.g. Fuel Level Overshoots), making sure you address each use case in your requirements document. Do not invent scenarios, rather a general guideline is to include scenarios that would make sense to the customer. For example, for the course enrollment system, logging in is a valid scenario.

5.1 Scenarios

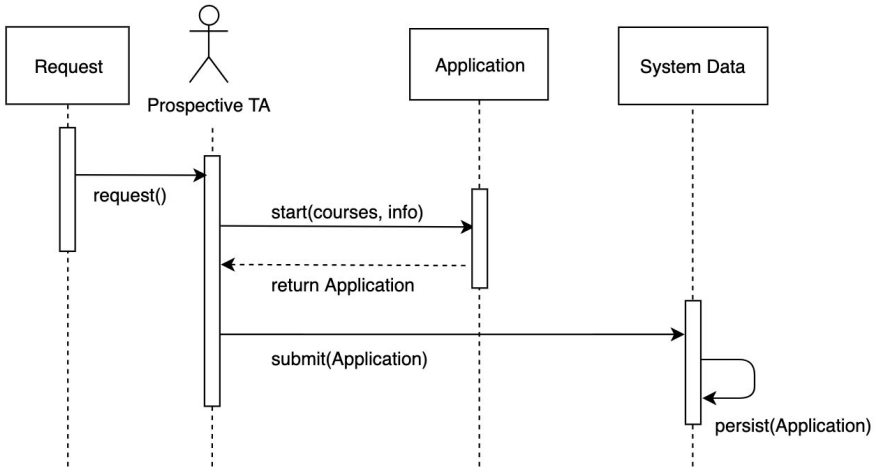
5.1.1 View appointments

This sequence diagram shows the series of interactions that occur when a payroll staff member views the list of appointments. After receiving a request from the Request Interface about this event, the Payroll object calls the viewAppointments() function, which then calls the retrieve() function in System Data Interface to produce an Appointment object for each appointment in the database. Once all appointments have been retrieved, the list of Appointments is returned to the Payroll staff.



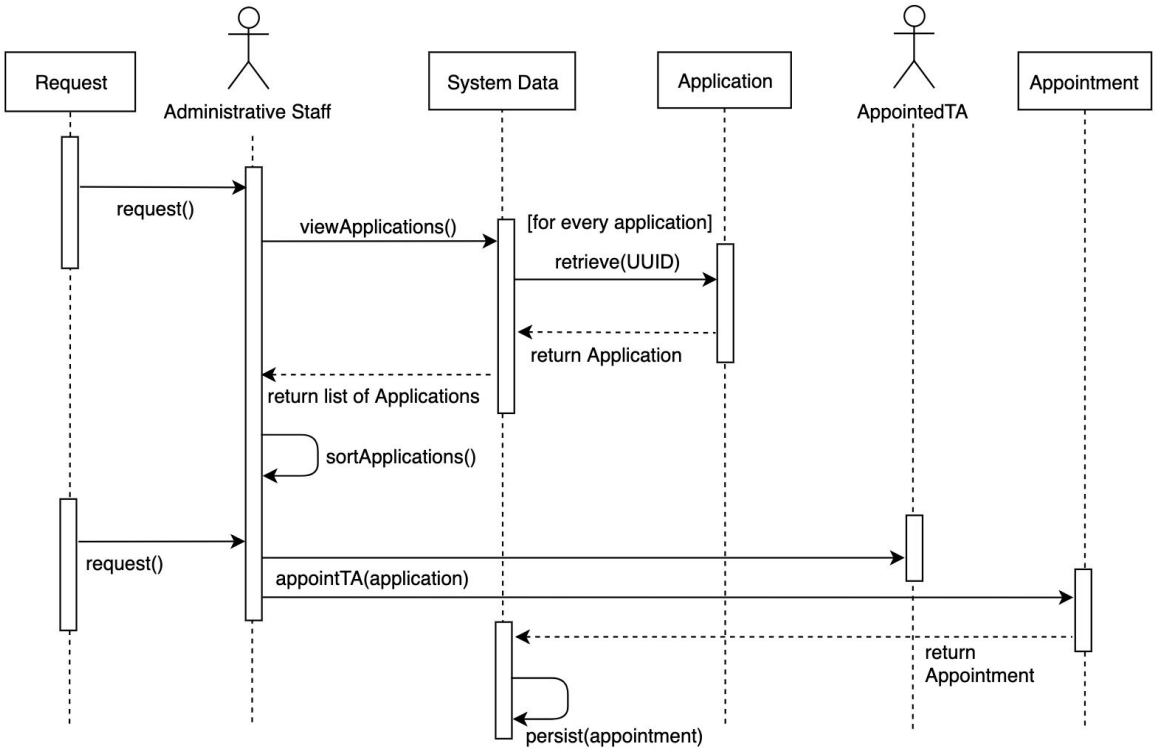
5.1.2 Submit Application

This sequence diagram shows the series of interactions that occur when a prospective TA submits his/her application. After receiving a request from the Request Interface about this event, the Prospective TA object calls the function start(), which returns an Application object. Then, the Prospective TA calls the function submit() on the Application, and the persist() function in System Data Interface is called to store that Application in the external database.



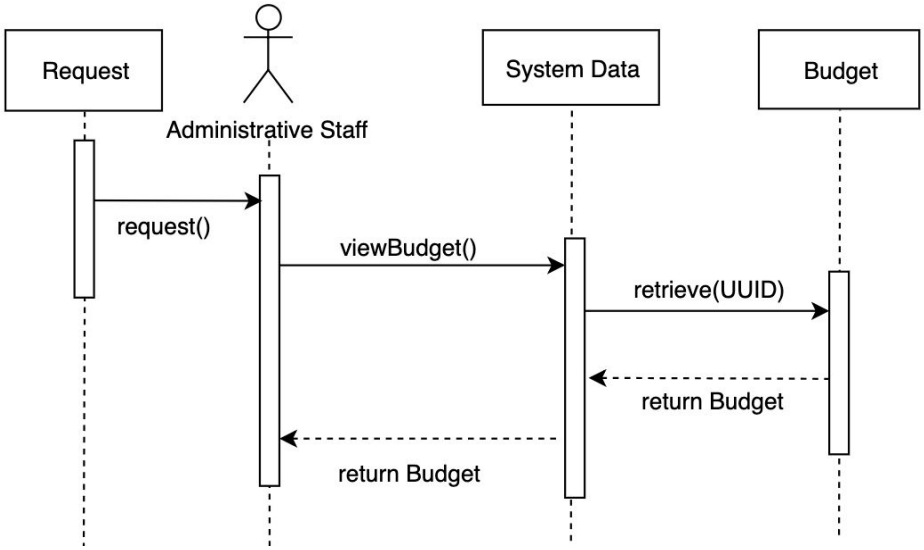
5.1.3 View Applications and Appoint TA

This sequence diagram shows the series of interactions that occur when an administrative staff member views the applications and appoints a TA. After receiving a request from the Request Interface about this event, the Administrative staff object calls the function viewApplications(), which then calls the retrieve() function in System Data Interface to produce an Application object for each application in the database. Once all applications have been retrieved, the list of Applications is returned to the Administrative staff. The Administrative staff can call the function sortApplications() to sort applications by priority score.Then, the Administrative staff calls the function appointTA() on a chosen Application, which creates a new Appointment and a new Appointed TA, and then calls the persist() function in System Data Interface that stores the appointment in the database.



5.1.4 View budget information

This sequence diagram shows the series of interactions that occur when an administrative staff member views the budget. After receiving a request from the Request Interface about this event, the Administrative Staff object calls the viewBudget() function, which then calls the retrieve() function in System Data Interface to produce a Budget object in the database. The Budget returns to the Administrative staff.



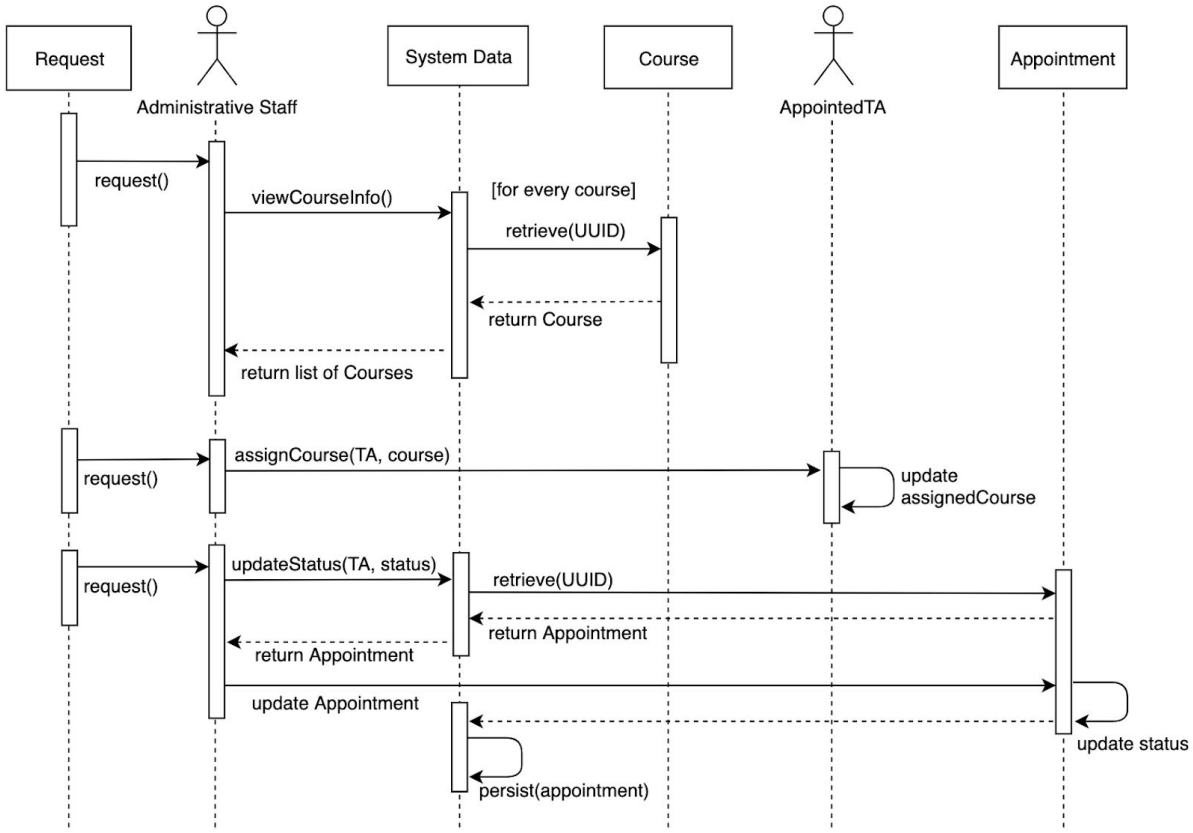
5.1.5 View course/position information, Assign course to appointed TA, and Update appointment status

This sequence diagram shows the series of interactions that occur in three situations:

- When an administrative staff member views the course or position information. After receiving a request from the Request Interface about this event, the Administrative Staff object calls the viewCourseInfo() function, which then calls the retrieve() function in System Data Interface to produce a Course object in the database. This process will be done for each course. Finally the list of Course returns to the Administrative staff.

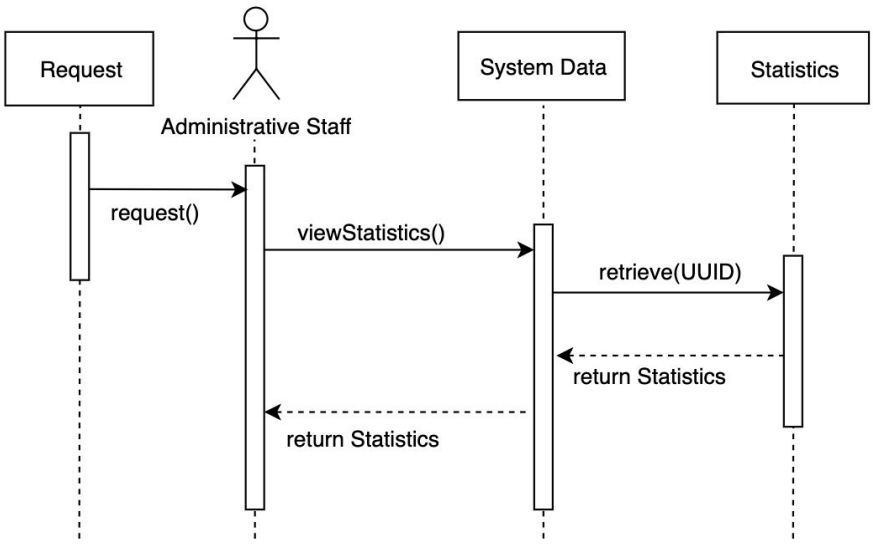
- When an administrative staff member assigns courses to appointed TAs. After receiving a request from the Request Interface about this event, the Administrative Staff object calls the assignCourse(TA, course) function for the appointed TA. Then the assignedCourse of that Appointed TA will be updated as a result.

- When an administrative staff member updates appointment status.
  After receiving a request from the Request Interface about this event, the Administrative Staff object calls the retrieve() function in System Data Interface to produce an Appointment object in the database, and return it to the administrative staff. Then the Administrative Staff are able to change the information of that appointment. Afterwards the status of that appointment will be updated, and it will be stored in the system data through calling persist(Appointment) function.



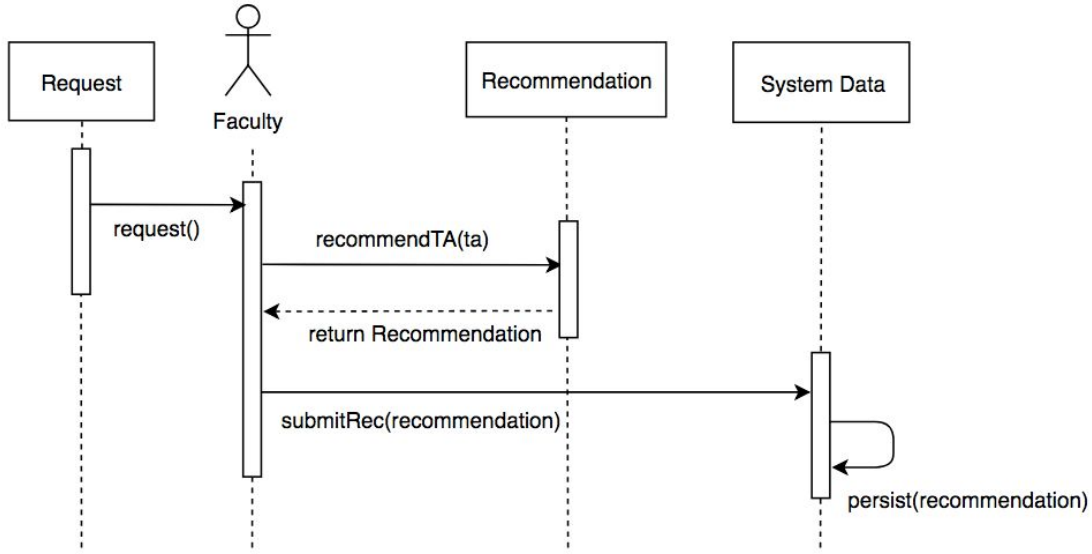
5.1.6 View Statistics

This sequence diagram shows the series of interactions that occur when an administrative staff member views the Statistics. After receiving a request from the Request Interface about this event, the Administrative Staff object calls the viewStatistics() function, which then calls the retrieve() function in System Data Interface to produce a Statistics object in the database. The Statistics returns to the Administrative staff.



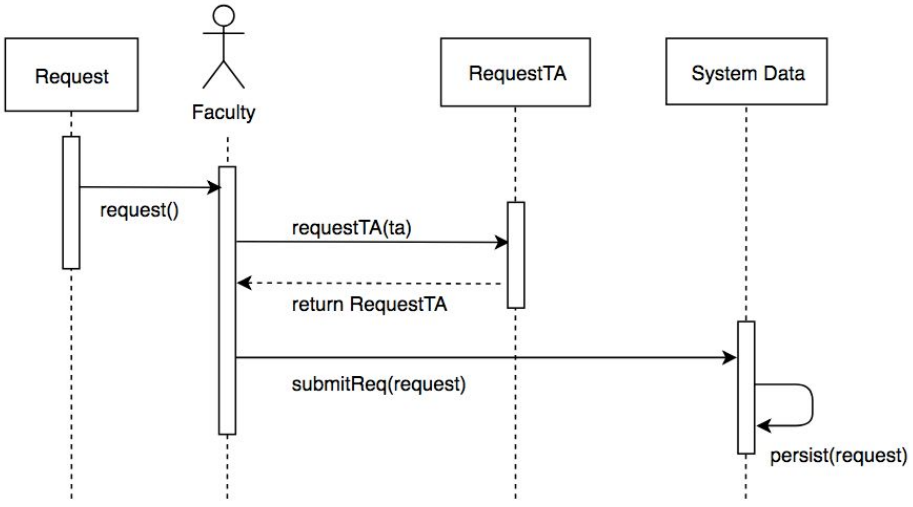
5.1.7 Recommend TA

This sequence diagram shows the series of interactions that occur when a faculty member recommends a TA. The Faculty can call the function recommendTA(), which creates a new Recommendation. The Faculty then can call submitRec(recommendation) which calls the persist() function in System Data Interface that stores the recommendation in the database.



5.1.8 Request TA

This sequence diagram shows the series of interactions that occur when a faculty member requests a TA. After receiving a request from the Request Interface about this event, the Faculty object calls the function requestTA(), which creates a new RequestTA object. Then, the Faculty calls the function submitReq() on the RequestTA, which calls the persist() function in System Data Interface to store that RequestTA in the external database.



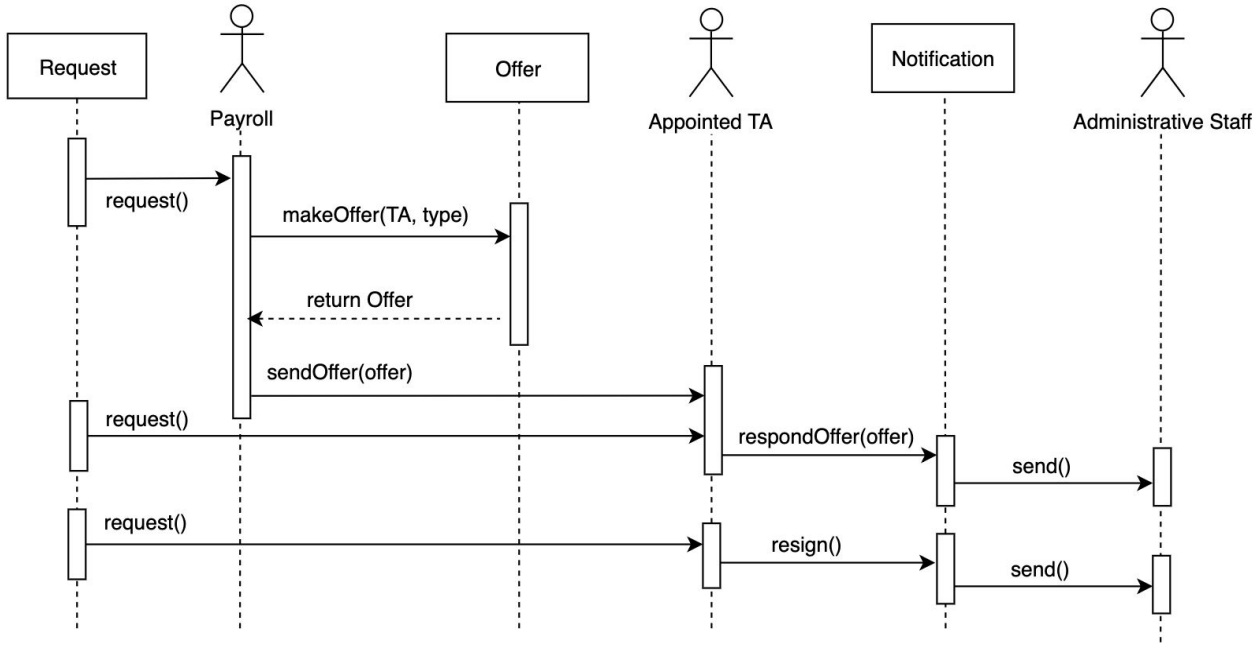
5.1.9 Send appointment offer, Respond to offer, and Resign from appointment

This sequence diagram shows the series of interactions that occur during 3 related use cases:

- When a payroll staff sends an appointment offer. After receiving a request from the Request Interface about payroll sending an appointment offer, the Payroll object calls the function makeOffer() with the specific TA and type of appointment, which returns an Offer object. Payroll then calls the function sendOffer() on the Offer, which sends the Offer to the Appointed TA.

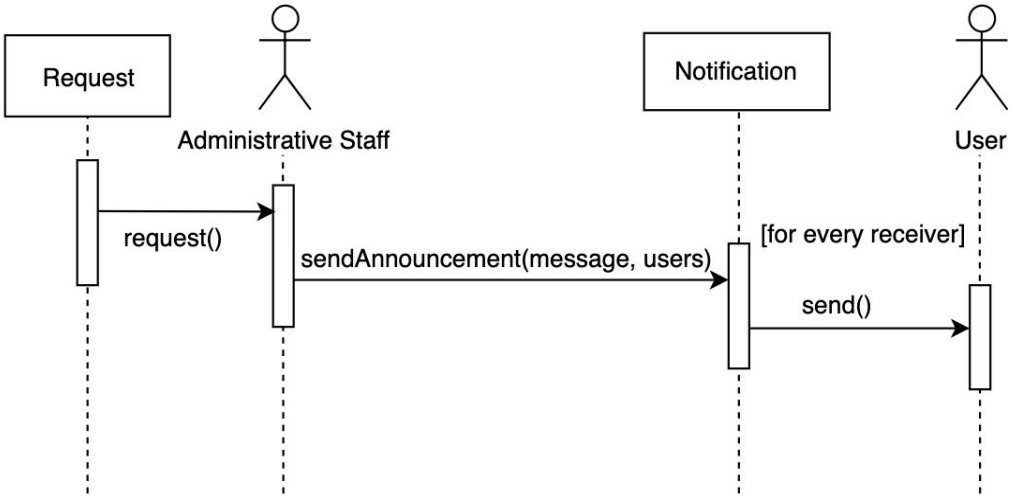
- When the TA responds to the offer. After receiving a request from the Request Interface about a TA responding to an offer, the Appointed TA object calls the function respondOffer() on the Offer they have received. This creates a Notification object containing a message about the TA's response and sends it to Administrative staff using the send() function.

- When a TA resigns from an appointment. After receiving a request from the Request Interface about a TA resigning from an appointment, the Appointed TA object calls the function resign(). This creates a Notification object containing a message about the TA's resignation and sends it to Administrative staff using the send() function.



5.1.10 Make announcement

This sequence diagram shows the series of interactions that occur when an administrative staff member wants to send announcement to other users. After receiving a request from the Request Interface about this event, the Administrative staff calls the function sendAnnouncement(message, users), which creates a Notification object for each receiving user. Once all the announcements are created, it calls send() to send them to all receivers.



6 Non-functional requirements

The design of the system allows for extensibility. Since all requests are handled by the Request Interface, new functionality can be added by simply adding new classes to represent new domain objects, such as new types of forms. These can follow the existing pattern by connecting to both the relevant user and the System Data Interface. New types of users can also be added by inheriting from the existing User base class. Adding new functionality to existing classes will require making changes to those classes, however.

With many of the requirements, we expect them to all be testable. We should be able to test the functionality of each requirement as well as error cases to ensure the performance and results are as expected. This allows us to examine the brittleness of our design and helps us detect places in our design that are still weak and need more work.