# CSCE 740 BILL System

Design Document

Authors:
Aaron Hein
Jeremy Lewis
Peter Mourfield

Group: 5

This page intentionally left blank.

# Document Revision History

| Date | Version | Description | Author |
|---|---|---|---|
| 10/20/2017 | 0.0 | Initial draft | Provided |
| 10/25/2017 | 0.1 | Team updates | Team |
| 11/14/2017 | 1.0 | Edits made based on feedback | Peter Mourfield |

This page intentionally left blank.

# Contents

# 1 Introduction

*This section addresses the purpose of this document including the intended audience, an introduction to the problem and a detailed view of the project's design. In the discussion, the design of the final system including several detailed diagrams will be described in detail.*

## 1.1 Purpose

*This document is meant to outline the design of the Billing Interface Linkable Library (BILL) at both a high level and at a more detailed class level. The BILL API will be designed in an Object Oriented manner and will follow a client/server design model with this document being addressing the server design for the system. The server will follow a hybrid control model where one module contains the business logic for the system and interacts with other modules that control the individual data interactions.*

## 1.2 System Overview

*The BILL system will serve as the back-end, or server, of a client/server system by offering an API that is accessible by a variety of other systems (and user interfaces). This back-end API will offer services such as viewing and editing of student profiles, viewing of bills, and payment of bills in a university setting where multiple students are enrolled in classes and use this system to pay their account.*

## 1.3 Design Objectives

*This system is being designed to allow students to manage their account at the university. More specifically, it will allow students to view or edit their profile information, pay their current bill, or view current or past bills. The system will also support administrators who are associate with a college or the graduate school and allow them to make payments or modifications to a bill. Finally the system will generate a students bill at the start of the term. All of the currency amounts that the system uses are assumed to be in US Dollars and are accurate to two decimal places.*

*This design does not encompass any of the front end design or user interface aspects of whatever program might use the API. It also does not address any interactions with other systems or databases that might hold the student and user information. It assumes that all of the data is provided in text files in a JSON format and can be stored in memory. Changes should be persisted in the files.*

*This design also does not address user authentication or underlying security concerns (encryption of PII Data at rest, encryption in transit, etc.). It is assumed that authentication is*

*performed outside of this system and only handles basic authorization as it pertains to modifying data within the BILL system. It is also assumed that payments and payment processing are handled outside of this system in a PCI compliant manner. This system does not take in any PCI data or perform any authorizations.*

*This API will be written in Java.*

## 1.4 References

*The requirements document for the BILL API can be accessed online at https://drive.google.com/open?id=0B-2hy6hB4QNcS0RwZzZlb01nNVU.*

## 1.5 Definitions, Acronyms, and Abbreviations

***API:*** *An Application Programming Interface is the means by which other applications can access this data and perform operations on the data.*

***GUI:*** *The Graphical User Interface is how a user would interact with the API. This is outside the scope of this document.*

***JSON:*** *JavaScript Object Notation is a data format that is commonly used to transfer data online because of it's lightweight and easy to read nature.*

***PCI:*** *Payment Card Industry data includes credit card numbers, any data stored on a payment card and any numbers (like PIN) assigned by the payment card issuer.*

***PII:*** *Personally Identifiable Information is any data that could identify a specific individual. In the BILL system, this includes name, address, email, etc. Industry best practice is to protect this data with encryption or anonymization. Both are outside the scope of this project.*

***PIN:*** *Personal Identification Numbers are used to validate some payment card transactions.*

# 2 Design Overview

## 2.1 Introduction

*The design of this API system is a data-centric architecture it could also be described as a layer architecture, hiding the database interaction behind a set of API calls. Internally, the system uses an OO design with classes representing the different users, student records, bills, and transaction.*

## 2.2 Environment Overview

*The API will be accessed as a Java Library archive (jar). Though the system is designed for multi-user interactions, all files are local to the calling environment. To provide shared access to files, the front-end must manage multiple in-coming user connections. The API assumes user roles and login verification are handled by this front-end system. In addition, payment transactions must be handled by third party systems.*

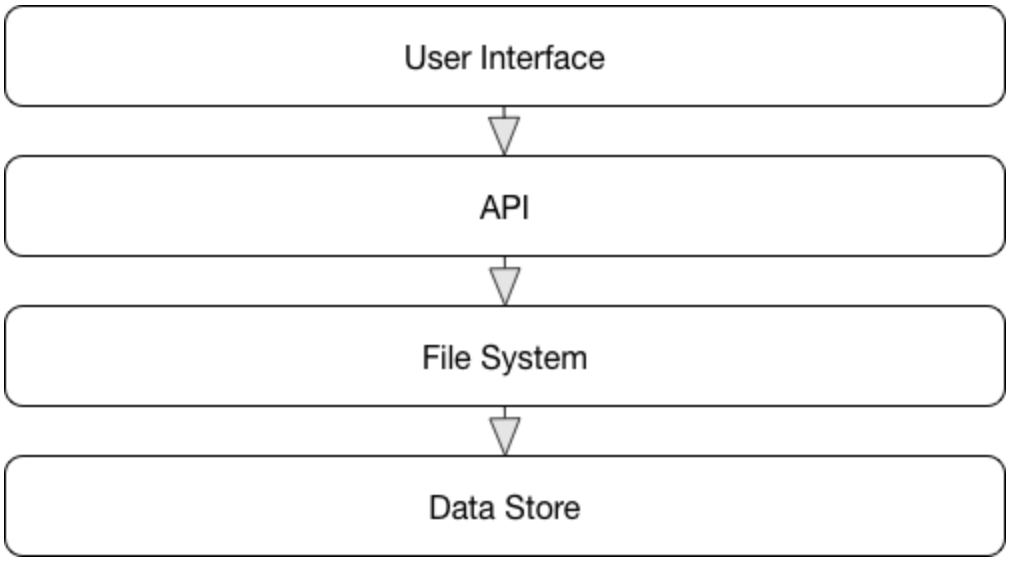
## 2.3 System Architecture

### 2.3.1 Top-level system structure of a system using the BILL API

*The system follows a layered architectural model as the User Interface is segregated in it's own layer and runs on top of the API, which is the focus of this document. The API layer is also segregated from the file that contains the data which is separated from the actual data stored.*

*The layered approach was chosen for the following reasons:*

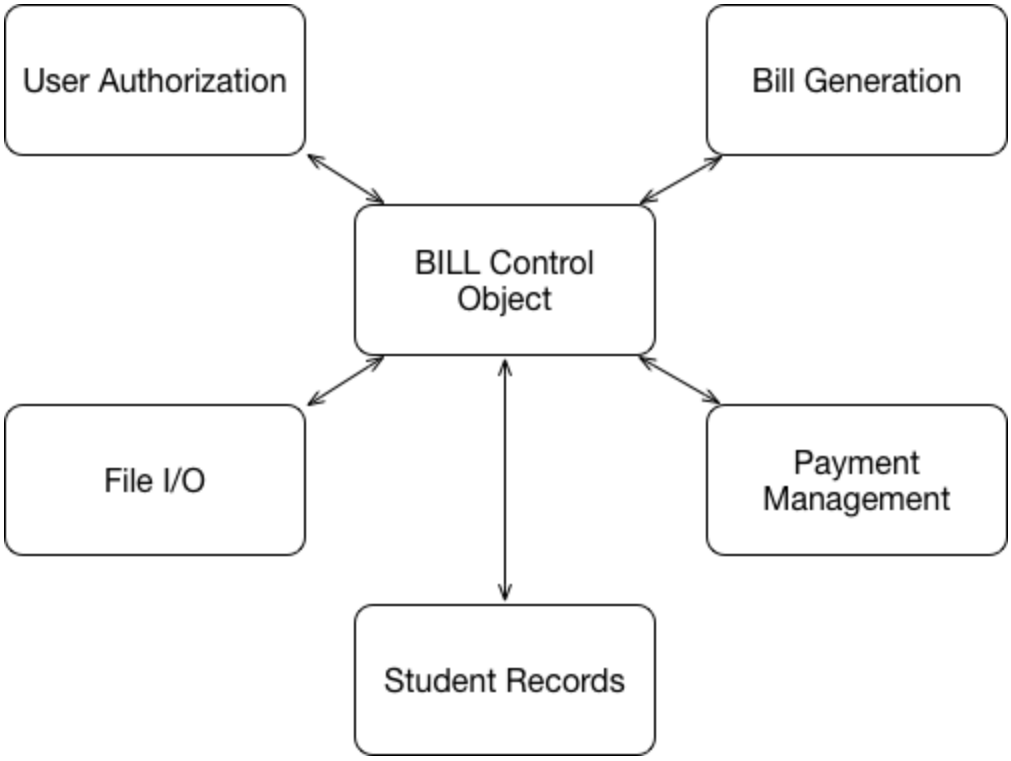
- *It is not clear what front end will use the system and so the API must be designed for usage without requiring details of the underlying system.*
- *Based on the current file storage, user growth will probably require a database in the near future and so a modular data store connection will make that transition possible.*



### 2.3.2 Control Model

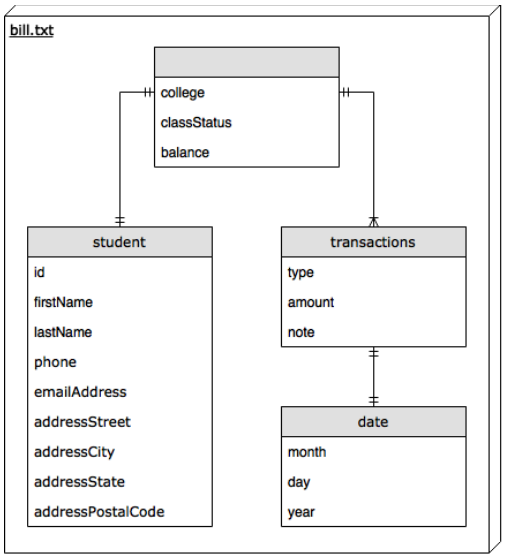
*A centralized control model is followed for the API Layer where the business logic is contained in a few small "helper" classes that then interact with the student records, perform the bill generation and any bill related activities, perform the file IO and also applies payments to the appropriate student account after validating that the user has sufficient authorization to perform each action.*

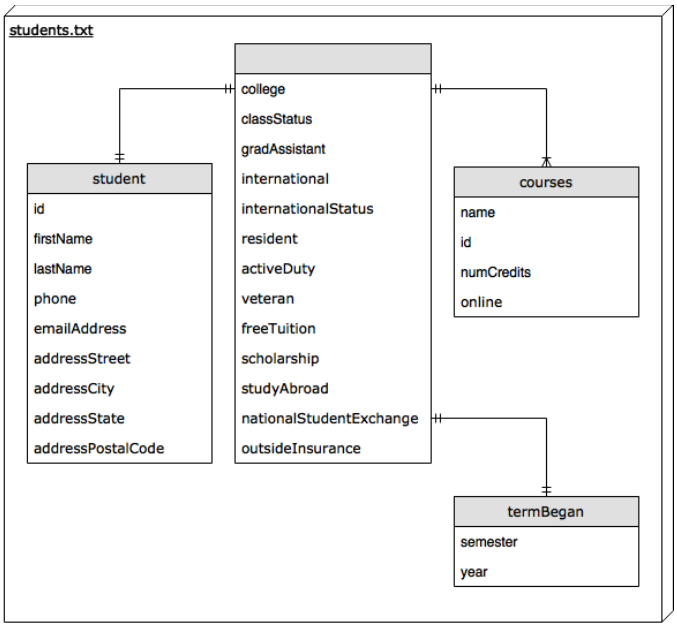
*The centralized model was chosen because it most directly fits the API. Most of the logic of the system is centered around a few user tasks. By placing our business logic in the a few class es with which the API is realized, we are both able to readily separate our architecture layers and locate possibly changing code to a single class. In this way, as rules change, only the relevant classes need be updated.*

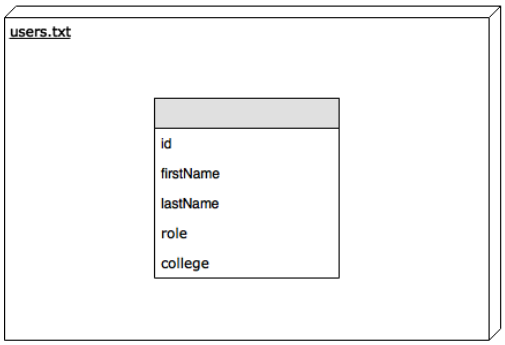


### 2.3.3 Data Sources

*The following three local JSON-encoded data files store all data for BILL. The unnamed entities are top-level JSON objects identified, as expected, by indices:*



## 2.4 Constraints and Assumptions

*Design constraints and assumptions are as follows:*

1. *This system will be implemented in Java.*

2. *The data used by the system will be passed in via files in a JSON format.*
    a. *The number of users, students and payments will be constrained by the memory of the machine since all of the data will need to be stored in memory. Should the system need to support more users, this design should be re-evaluated.*
    b. *Scalability will also be impacted as multiple instances of the same API cannot operate on the same data at the same time. This limits the number of concurrent users that the system will support in addition to potentially impacting the system response times.*
    c. *Multiple users cannot be logged in at the same time.*

3. *The system will not handle user authentication. Users are assumed to be authenticated by a third party system before they are logged in.*

4. *The system will not handle payment authorization. Payments are assumed to be authorized by a third-party before they are submitted.*

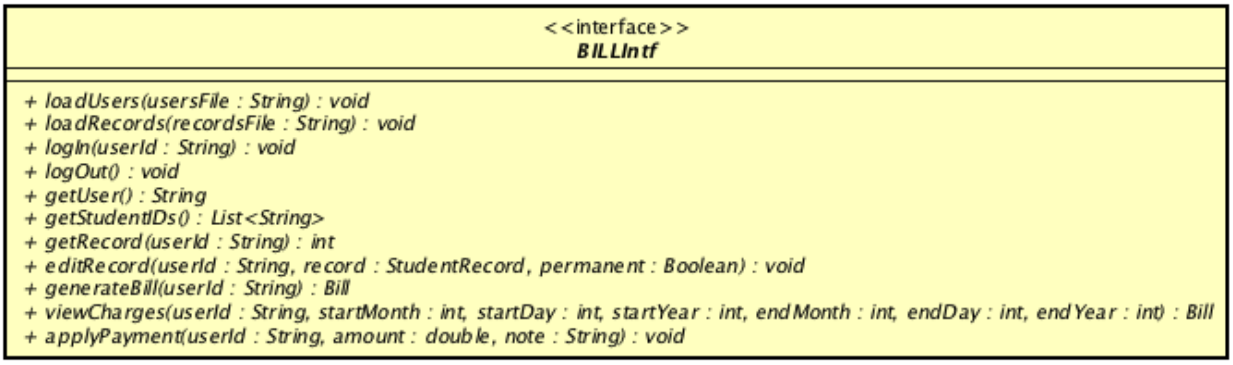
5. *No data encryption is required.*

6. *Data sent to the system is assumed to have arrived securely.*
7. *The front end / GUI for the complete system is out of scope of this document.*

# 3 Interfaces

## 3.1 System Interfaces

The system consists of a class library where interaction is defined by the interface BillInf (see below). Additionally, the system uses files from the file system as data repositories.




### 3.1.1 BILL Interface

This interface is used allow external systems to interact with the BILL Student Billing System. It provides these functions:

1. loadUsers(String usersFile)
    Description: Loads the list of system usernames and permissions.
    Parameters:
        usersFile – the filename of the users file
    Return Type: void
    Return value: nothing
    Throws: UserDataLoadException

2. loadRecords(String recordsFile)
    Description: Loads the list of system transcripts.
    Parameters:
        recordsFile – the filename of the records file
    Return Type: void
    Return value: nothing
    Throws: RecordDataLoadException

3. logIn(String userId)
    Description: Sets the user id of the user currently using the system to the provided user id. NOTE: this method does not perform the authentication step. That functionality is handled by a system external to BILL and occurs before this method is called.
    Parameters:
        userId – the Id of the user to login
    Return Type: void
    Return value: nothing
    Throws: UserSessionException

4. logOut()
    Description: Closes the current session, logs the user out, and clears any session data.
    Parameters: None
    Return Type: void
    Return value: nothing
    Throws: UserSessionException

5. getUser()
    Description: Returns the user id of the user currently using the system.
    Parameters: None
    Return Type: String
    Return value: the user id of the user currently using the system

6. getStudentIDs()
    Description: Returns a list of user ids of the students that an admin can view.
    Parameters: None
    Return Type: List<String>
    Return value: return a list containing the userId of each student in the college belonging to the current user.
    Throws: UnauthorizedUserException

7. getRecord(String userId)
    Description: Returns the raw student record data for a given user id.
    Parameters:
        userId – the identifier of the student
    Return Type: StudentRecord
    Return value: the student record data
    Throws: UnauthorizedUserException, DataRetrievalException

8. editRecord(String userId, StudentRecord record, Boolean permanent)
    Description: Saves a new set of student data to the records data.
    Parameters:
        userId – the identifier of the student
        Record - the new student record
        Permanent - a status flag indicating (if false) to make a temporary edit to the in-memory structure or (if true) a permanent edit.
    Return Type: void
    Return value: nothing
    Throws: UnauthorizedUserException, DataSaveException

9. generateBill(String userId)
    Description: Generates and returns the current bill.
    Parameters:
        userId – the identifier of the student
    Return Type: Bill
    Return value: the students bill in a data class matching the I/O file
    Throws: UnauthorizedUserException, BillGenerationException

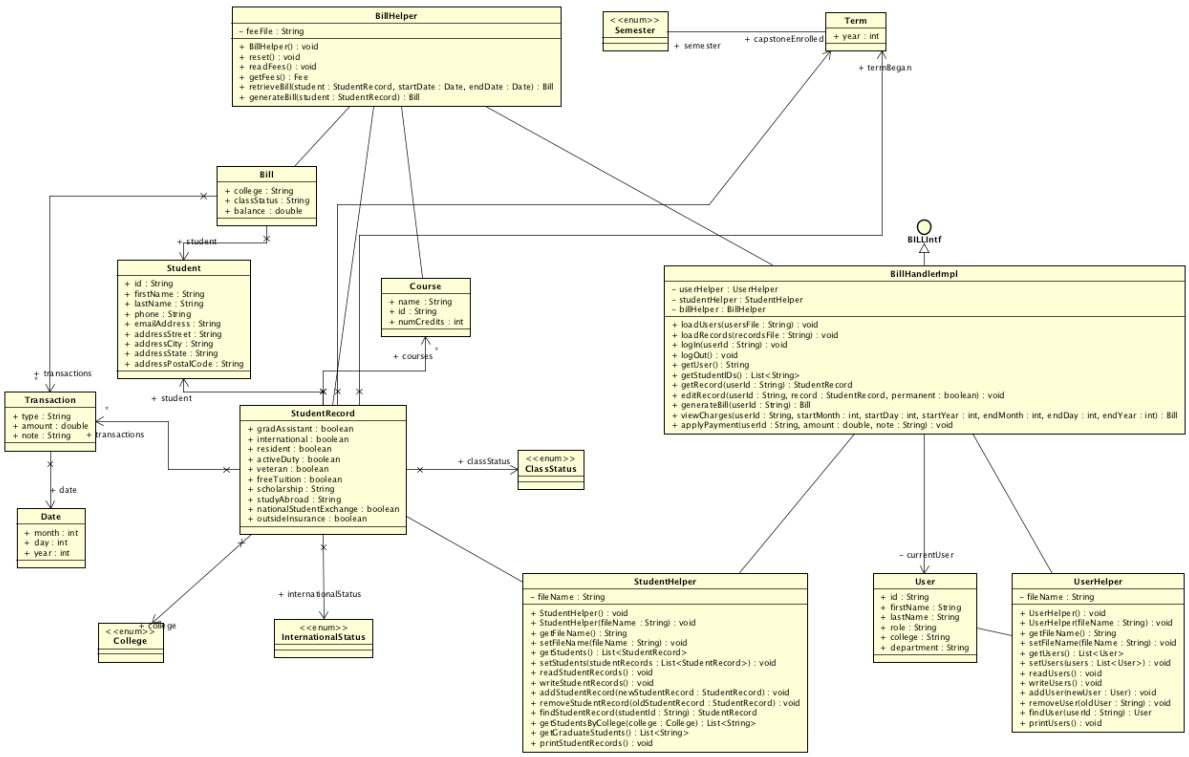
10. viewCharges(String userId, int startMonth, int startDay, int startYear, int endMonth, int endDay, int endYear)
    Description: Generates and returns a list of transactions for a chosen period.
    Parameters:
        userId – the identifier of the student
        startMonth – the month of the start date
        startDay – the day of the start date
        startYear – the year of the start date
        endMonth – the month of the end date
        endDay – the day of the end date
        endYear – the year of the end date
    Return Type: Bill
    Return value: the students bill in a data class matching the I/O file
    Throws: UnauthorizedUserException, BillRetrievalException

11. applyPayment(String userId, double amount, String note)
    Description: Makes a payment for the student.
    Parameters:
        userId – the identifier of the student
        amount – amount to apply to the balance
        note – a string indicating the reason for the payment
    Return Type: void
    Return value: success or failure
    Throws: UnauthorizedUserException, PaymentException, PaymentSaveException

# 4 Structural Design

The class structure in represented by a class that covers the BillInf implementation (BillHandlerImpl). The BillHandlerImpl interacts with several helper classes that are responsible for providing the business logic of the system. Finally, there are a number of Data Transfer Objects (DTOs) that represent the data either consumed or provided as a part of the overall system.

## 4.1 Class Diagram



## 4.2 Classes in the BILL Subsystem

### 4.2.1 Class: StudentHelper

Attributes:

| Name | Type | Constraints |
|---|---|---|
| fileName | String | |

Operations:

| Name | Parameter(s) | Returns |
|---|---|---|
| getStudents | | List<StudentRecord> |
| addStudentRecord | newStudentRecord : StudentRecord | |
| removeStudentRecord | studentId : String | |
| findStudentRecord | studentId : String | StudentRecord |

### 4.2.2 Class: UserHelper

Attributes:

| Name | Type | Constraints |
|---|---|---|
| fileName | String | FALL; SPRING; SUMMER |

Operations:

| Name | Parameter (s) | Returns |
|---|---|---|
| getUsers | | List<User> |
| addUser | newUser : User | |
| removeUser | userId : String | |
| findUser | userId : String | User |

### 4.2.3 Class: BillHelper

Attributes:

| Name | Type | Constraints |
|---|---|---|

| | | |
|---|---|---|
| fileName | String | FALL; SPRING; SUMMER |

Operations:

| **Name** | **Parameter (s)** | **Returns** |
|---|---|---|
| getFees | | List<Fee> |
| retrieveBill | Student : StudentRecord, startDate : Date, endDate : Date | Bill |
| generateBill | Student ; StudentRecord | Bill |
| makePayment | studentId : String, amount, double | |

## 4.3 Data Transfer Objects

### 4.3.1 DTO Class: User

Attributes:

| Name | Type | Constraints |
|---|---|---|
| id | string | |
| firstName | string | |
| lastName | string | |
| role | string | STUDENT, GRADUATE_PROGRAM_COORDINATOR |
| department | string | All uppercase with underscores instead of spaces |

### 4.3.2 DTO Class: Course

Attributes:

| Name | Type | Constraints |
|---|---|---|
| name | string | |
| id | string | Four lowercase letters followed by 3 numbers |
| numCredits | string | Either a single number or a range |

### 4.3.3 DTO Class: Date

Attributes:

| Name | Type | Constraints |
|---|---|---|
| month | int | Two digit number; 1 - 12 |
| day | int | Number; 1 - 31 |
| year | int | Four digit number |

### 4.3.4 DTO Class: Transaction

Attributes:

| Name | Type | Constraints |
|---|---|---|
| type | string | PAYMENT or CHARGE |
| transactionDate | Date | |
| amount | double | |
| note | String | Intended to explain the transaction |

### 4.3.5 DTO Class: Student

Attributes:

| Name | Type | Constraints |
|---|---|---|
| id | String | |
| firstName | String | |
| lastName | String | |
| phone | String | ddd-ddd-dddd |
| emailAddress | String | |
| addressStreet | String | |
| addressCity | String | |
| addressState | String | 2 character representation |
| addressPostalCode | String | |

### 4.3.6 DTO Class: Term

Attributes:

| Name | Type | Constraints |
|---|---|---|
| semester | String | FALL; SPRING; SUMMER |
| year | int | Four digit number |

### 4.3.7 DTO Class: StudentRecord

Attributes:

| Name | Type | Contraints |
|---|---|---|
| student | Student | |
| college | String | |
| termBegan | Term | |
| capstoneEnrolled | Term | |
| classStatus | String | FRESHMAN; SOPHOMORE; JUNIOR; SENIOR; MASTERS; PHD; GRADUATED |
| gradAssistant | boolean | |
| international | boolean | |
| internationalStatus | String | SHORT_TERM; SPONSORED; NONE |
| resident | boolean | |
| activeDuty | boolean | |
| veteran | boolean | |
| freeTuition | boolean | |
| scholarship | String | WOODROW; DEPARTMENTAL; GENERAL; ATHLETIC; SIMS; NONE |
| studyAbroad | String | REGULAR; COHORT; NONE |
| nationalStudentExchange | boolean | |
| outsideInsurance | boolean | |
| couses | List<Course> | |
| transactions | List<Transaction> | |

### 4.3.7 DTO Class: Bill

Attributes:

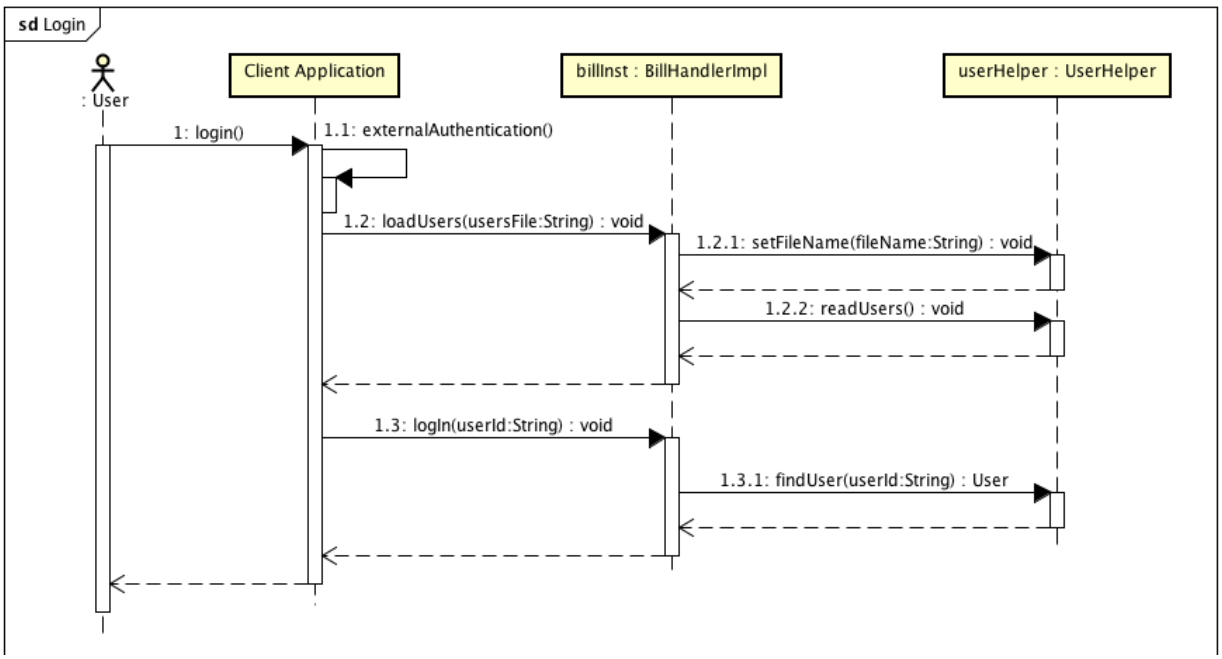
| Name | Type | Contraints |
|---|---|---|
| student | Student | |
| college | String | ARTS_AND_SCIENCES; ENGINEERING_AND_COMPUTING; GRADUATE_SCHOOL |
| classStatus | String | FRESHMAN; SOPHOMORE; JUNIOR; SENIOR; MASTERS; PHD; GRADUATED |
| transactions | List<Transaction> | |

# 5 Dynamic Model

## 5.1 Scenarios

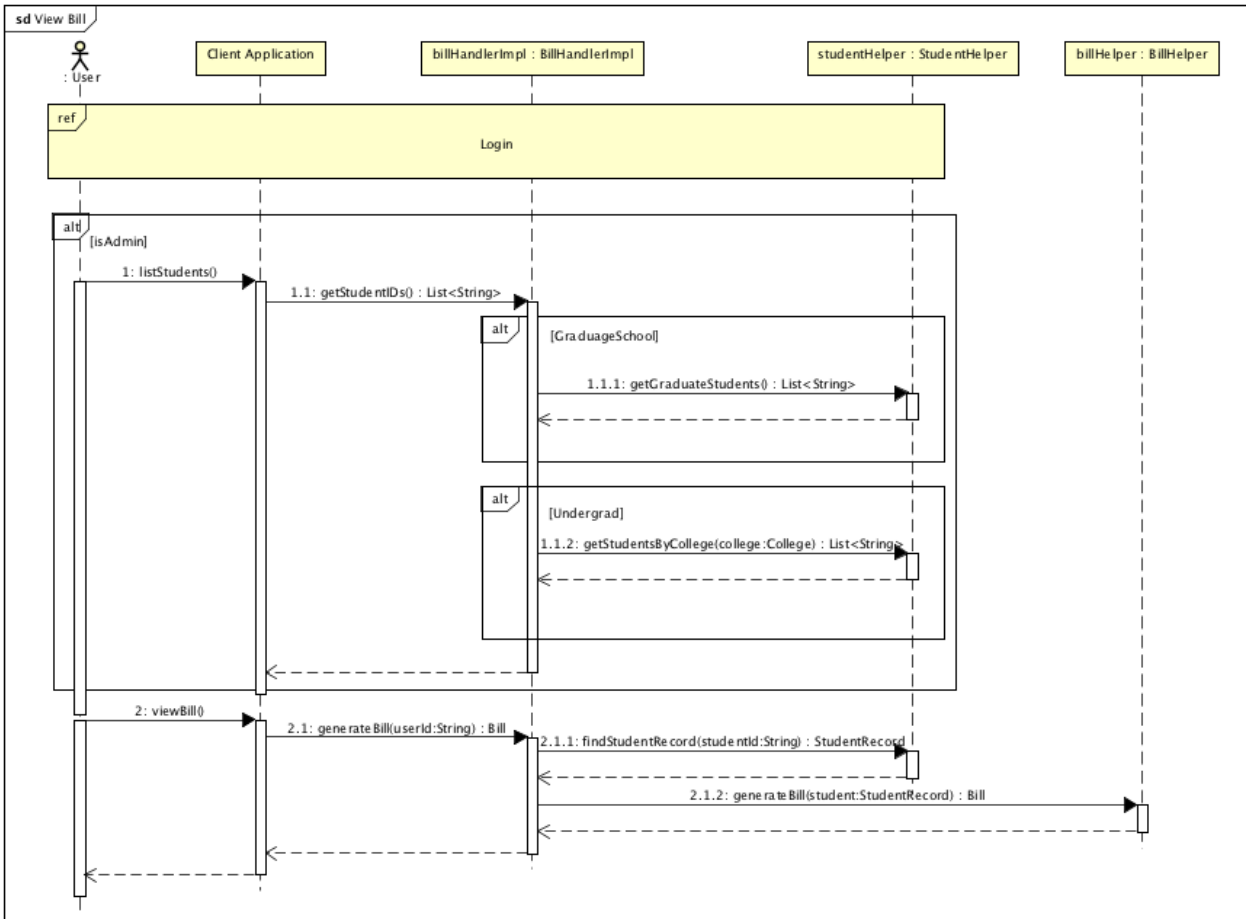
### 5.1.1 User Login

This scenario occurs when a user logs in through some client application.



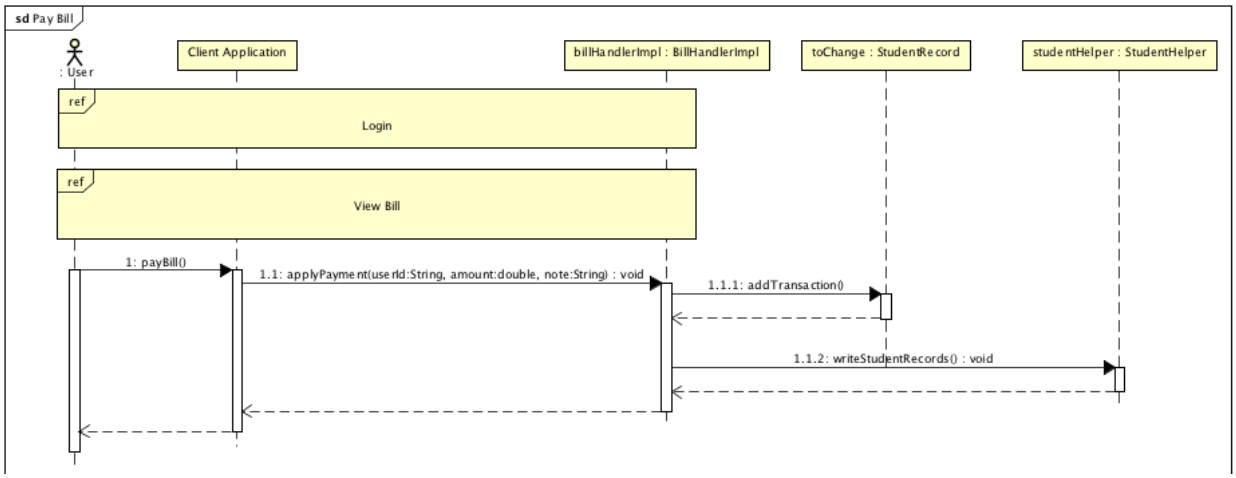
## 5.1.2 View Bill

When a user logs in and views a bill. In the case of a student, their own bill; graduate administrator, any graduate student bill; or college administrator, any student in their college. References the login (5.1.1) sequence diagram.



## 5.1.3 Pay Bill

When a user logs in, views, then pays a bill. In the case of a student, their own bill; graduate administrator, any graduate student bill; or college administrator, any student in their college. References the login (5.1.1) and view bill (5.1.2) sequence diagrams.



# 6 Non-functional requirements

1. Security:
    a. The system shall ensure that the student information can only be accessed by authorized users.
2. Availability:
    a. The system shall achieve 99.5% uptime.
3. Efficiency:
    a. The system shall have a maximum response time of 2 seconds.
4. Integrity:
    a. All currency amounts will be in US dollars.
    b. All currency amounts shall be accurate to 2 decimal places.
5. Scalability:
    a. System should be able to generate a bill within 2 seconds.
    b. System should be able to handle 1000 requests per minute.

# 7 Supplementary Documentation

Provide any other relevant documentation that may help understanding the design.