Requirements for

CSCE 740, Fall 2017 B.I.L.L.
10/10/2017

Project Team:

- Aaron Hein
- Jeremy Lewis
- Peter Mourfield

Table of Contents

1. Introduction

1.1. Purpose

1.1.1. The system will provide the user the ability to view and pay their student account bill and to view and update their student account information.

1.2. Scope

1.2.1. The document specifies all functions and external interactions that the system must provide.

1.3. Definitions and Acronyms

1.3.1. BILL - Billing Interface Linkable Library

1.4. References

1.4.1. Dropbox.cse.sc.edu. (2017). CSE Dropbox 2.0: Requirements Elicitation [online] Available at: https://dropbox.cse.sc.edu/mod/forum/discuss.php?d=139 [Accessed 27 Sep. 2017].

1.5. Overview

1.5.1. The following sections provide, first, an overall description which outlines the product specifications as understood from both the product's perspective as well as the user's perspective. The following section outlines the functional requirements of the system. In this section, a complete and correct product is defined.

2. Overall Description

2.1. Product Perspective

2.1.1. The system shall interface with database systems that contain student account information. The system shall interface with web browser software and allow the user to query, view, and manipulate their user profile within the calling web browser.

2.2. Product Functions

2.2.1. The system shall allow the user to view and edit their profile information.
2.2.2. The system shall allow the user to view and pay their current bill.
2.2.3. The system shall allow the user to view previous bills.
2.2.4. The system shall allow administrators to assign profiles to students and distribute bills.

2.3. User Characteristics

2.3.1. Student - a student at the university
2.3.2. Administrator - distributes bills and processes payments

2.4. Constraints

2.4.1. The system must be written in Java.

2.5. Assumptions

2.5.1. The system shall assume that the student has credentials to log in to the system (i.e. password) and those credentials have been validated before the API is called.
2.5.2. The system shall assume that payments are handled outside the system and have been validated before the API is called.

3. Functional Requirements

3.1. College Maintenance Requirements

3.1.1. Read in Colleges

3.1.1.1. The API shall provide the ability to read in supported Colleges from an input file.

Description: Read in all of the colleges that are to be supported by the API from a file.
Rationale: The rationale is that the colleges are fairly static and are being maintained in an external system. The colleges in that system will be exported in a flat file.
Assumptions: The file will be in a valid JSON format.
Precondition: None
Inputs: Filename of file with colleges in there
Outputs: Success/Failure
Persistent Changes: The list of colleges will be available to the API in memory. The filename inputted should be retained so changes to the colleges can be saved.
Related Requirements: None
Error Scenarios:

- If the file is not available, an error shall be returned stating that the file could not be opened.
- If the file is empty, an error shall be returned stating that the file was empty.
- If the JSON in the file is invalid, an error shall be returned stating that the file format is invalid.

Test Cases: See Attached "Test Cases" Document

3.1.2. Retrieve College

3.1.2.1. The API shall provide the ability to retrieve the detail of a college that was read in from the input file.

Description: Given a college ID, retrieve the details for that college.
Rationale: The rationale is that the colleges in memory might need to be retrieved and viewed.
Assumptions: None
Precondition:

- Requirement 3.1.1.1: Colleges have been loaded into the API.

Inputs: College ID
Outputs: College ID, College Name
Persistent Changes: None
Related Requirements: None
Error Scenarios:

- If the college ID is not found, an error shall be returned stating that college ID is not found.

Test Cases: See Attached "Test Cases" Document

3.1.2.2. The API shall provide the ability to retrieve a list of all of the supported colleges

Description: Return a list of all of the colleges that are supported by the system.
Rationale: The rationale is that this list could be retrieved and provided by the front end to allow a user to choose the correct college or see all of the colleges.
Assumptions: None
Precondition: Requirement 3.1.1.1: Colleges have been loaded into the API.
Inputs: None
Outputs: A list of supported College names and the corresponding IDs
Persistent Changes: None
Related Requirements:

- Requirement 3.1.1.1: Colleges have been loaded into the API.

Error Scenarios: None
Test Cases: See Attached "Test Cases" Document

3.1.3. Edit College

3.1.3.1. The API shall allow an Admin for the College to edit a College's details

Description: A college Admin should be allowed to modify the name of the college in the system and have that change persisted.
Rationale: The rationale is that in the event of the change to a college's details, an admin for the college should be able to update that in the system.
Assumptions: None
Precondition:

- Requirement 3.1.1.1: Colleges have been loaded into the API.

Inputs: College ID
Outputs: Success/Failure of the change.
Persistent Changes: The modified college name is saved in the file and changed in memory.
Related Requirements:

- Requirement 3.1.1.1: Colleges have been loaded into the API.
- (Optional) Requirement 3.1.2.1: College details are retrieved.

Error Scenarios:

- If the college ID is not found, an error shall be returned stating that college ID is not found.
- If the updated data cannot be written to the file, an error shall be returned stating that the change could not be made and the data in memory will remain unchanged.
- If the user is not an admin for the college being changed, an error shall be returned stating the user does not have access to make the change and the change will not be made.

Test Cases: See Attached "Test Cases" Document

3.2. Term Requirements

3.2.1. Read in Terms

3.2.1.1. The API shall provide the ability to read in supported Terms for each college from an input file.

Description: Read in all of the terms that are valid for a college.
Rationale: The rationale is that the active terms for a colleges are being maintained in an external system. The terms by college in that system will be exported in a flat file.
Assumptions: The file will be in a valid JSON format.
Precondition: None
Inputs: Path and filename of file containing terms by college
Outputs: Success/Failure
Persistent Changes: The list of terms by college will be available to the API in memory. The filename provided should be retained so changes to the terms can be saved.
Related Requirements: None
Error Scenarios:

- If the file is not available, an error shall be returned stating that the file could not be opened.
- If the file is empty, an error shall be returned stating that the file was empty.
- If the JSON in the file is invalid, an error shall be returned stating that the file format is invalid.

Test Cases: See Attached "Test Cases" Document

3.3. User Profile Information Requirements

3.3.1. Read in Users

3.3.1.1. The API shall provide the ability to read in supported Users for each college from an input file.

Description: Read in all of the users that are valid for the API.
Rationale: The rationale is that the active users are being maintained in an external system. The users in that system will be exported in a flat file.
Assumptions: The file will be in a valid JSON format.
Precondition: None
Inputs: Filename of file with users in there
Outputs: Success/Failure
Persistent Changes: The list users will be available to the API in memory. The filename inputted should be retained so changes to the terms can be saved.
Related Requirements: None
Error Scenarios:

- If the file is not available, an error shall be returned stating that the file could not be opened.
- If the file is empty, an error shall be returned stating that the file was empty.
- If the JSON in the file is invalid, an error shall be returned stating that the file format is invalid.

Test Cases: See Attached "Test Cases" Document

3.3.2. Retrieve Users

3.3.2.1. The API shall provide the ability to retrieve the detail of a student user that was read in from the input file.

Description: Given a user ID, retrieve the details for that user to include first/last/middle name, e-mail address, address, class status (freshman, sophomore, etc.), college, major, and role.

Rationale: The rationale is that the user details in memory might need to be viewed or retrieved.

Assumptions: Only allowed/successful if the User ID and the Target User ID are the same or if the User ID is an admin and the User's College and the Target User's College are the same. Or if the User ID is a grad school admin and the Target User is in Grad School.

Precondition:

- Requirement 3.1.1.1: Users have been loaded into the API.

Inputs: User ID, Target User ID

Outputs: First/Last/Middle Name, E-mail Address, Address, Class Status (Freshman, Sophomore, etc.), College, Major.

Persistent Changes: None

Related Requirements:

- Requirement 3.1.1.1: Users have been loaded into the API.

Error Scenarios:

- If the user ID is not found, an error shall be returned stating that user ID is not found.
- If the user ID is not authorized (see assumptions), an error shall be returned stating that the user is not authorized.

Test Cases: See Attached "Test Cases" Document

3.3.2.2. The API shall provide the ability to retrieve the detail of an administrative user that was read in from the input file.

Description: Given a user ID, retrieve the details for that user to include first/last/middle name, e-mail address, address, college, and role.

Rationale: The rationale is that the user details in memory might need to be viewed or retrieved.

Assumptions: Only allowed/successful if the User ID and the Target User ID are the same or if the User ID is an admin and the User's College and the Target User's College are the same. Or if the User ID is a grad school admin and the Target User is in Grad School.

Precondition:

- Requirement 3.1.1.1: Users have been loaded into the API.

Inputs: User ID, Target User ID

Outputs: First/Last/Middle Name, E-mail Address, Address, College

Persistent Changes: None

Related Requirements:

- Requirement 3.1.1.1: Users have been loaded into the API.

Error Scenarios:

- If the user ID is not found, an error shall be returned stating that user ID is not found.
- If the user ID is not authorized (see assumptions), an error shall be returned stating that the user is not authorized.

Test Cases: See Attached "Test Cases" Document

3.3.2.3. The API shall provide the ability to retrieve a list of all of the users for a supported college

Description: Retrieve a list of all users who are enrolled in that college.

Rationale: The rationale is that the this list would be displayed and used by an administrator to select the correct student to add a fee to if the User ID were not known.

Assumptions:

- The college will be college the admin is associated with.
- One user is not the admin of two schools.

Precondition:

- Requirement 3.1.1.1: Users have been loaded into the API.

Inputs: User ID

Outputs: List of Names and User ID's

Persistent Changes: None

Related Requirements:

- Requirement 3.1.1.1: Users have been loaded into the API.

Error Scenarios:

- If the user ID is not found, an error shall be returned stating that user ID is not found.

Test Cases: See Attached "Test Cases" Document

3.3.2.4. The API shall provide the ability to retrieve a list of all of the student users for the graduate school

Description: Retrieve a list of all users who are enrolled in the graduate school.

Rationale: The rationale is that the this list would be displayed and used by an administrator to select the correct student to add a fee to if the User ID were not known.

Assumptions:

- If the admin is a graduate school admin, then they will get a list of graduate students.
- One user is not the admin of two schools.

Precondition:

- Requirement 3.1.1.1: Users have been loaded into the API.

Inputs: User ID

Outputs: List of Names and User ID's

Persistent Changes: None

Related Requirements:

- Requirement 3.1.1.1: Users have been loaded into the API.

Error Scenarios:

- If the user ID is not found, an error shall be returned stating that user ID is not found.

Test Cases: See Attached "Test Cases" Document

3.3.2.5. Retrieve a list of all Students who are part of a College for a given Term

Description: Given a College and a Term, retrieve a list of all Students Users who are enrolled in the College for that term.
Rationale: The rationale is that when adding a fee, and admin might need to have a way of selecting a user who they are authorized to add the fee to.
Assumptions: Only allowed/successful if the admin user is associated with the graduate school and the target/student user is also in graduate school.
Precondition:

- Requirement 3.1.1.1: Colleges have been loaded into the API.
- Requirement 3.2.1.1: Terms have been loaded into the API.
- Requirement 3.3.1.1: Users have been loaded into the API.

Inputs: College, Term
Outputs: List of students
Persistent Changes: None.
Related Requirements:

- Requirement 3.1.1.1: Colleges have been loaded into the API.
- Requirement 3.2.1.1: Terms have been loaded into the API.
- Requirement 3.3.1.1: Users have been loaded into the API.

Error Scenarios:

- If the user ID is not found, an error shall be returned stating that user ID is not found.
- If the user ID is not authorized (see assumptions), an error shall be returned stating that the user is not authorized and the data will remain unchanged.

Test Cases: See Attached "Test Cases" Document

3.3.3. Edit User Profile

3.3.3.1. The API shall provide the ability to for a Student User to edit some information in their profile.

Description: A student user should be able to edit the name, email address and address fields in their profile.

Rationale: The rationale is that the user information might change and it is easiest to maintain if the users are able to update it themselves.

Assumptions: Only allowed/successful if the User who is logged in and the Target User are the same.

Precondition:

- Requirement 3.3.1.1: Users have been loaded into the API.

Inputs: User ID, Target User ID, Firstname, Middlename, Lastname, email address, home address.

Outputs: Success/Failure

Persistent Changes: User data is changed in memory and written to the user JSON file

Related Requirements:

- Requirement 3.3.1.1: Users have been loaded into the API.

Error Scenarios:

- If the user ID is not found, an error shall be returned stating that user ID is not found.
- If the user ID is not authorized (see assumptions), an error shall be returned stating that the user is not authorized and the data will remain unchanged.
- If the user attempts to edit a field that cannot be edited by students, an error shall be returned stating that that field cannot be modified and the data will remain unchanged.

Test Cases: See Attached "Test Cases" Document

3.3.3.2. The API shall provide the ability to for a College Admin to edit information in a Student User's profile.

Description: An admin user to edit all fields except ID in a student user's profile.

Rationale: The rationale is that the user information might change and an admin might be required to update that information.

Assumptions: Only allowed/successful if the College of the admin user who is logged in and the College of the target/student user are the same.

Precondition:

- Requirement 3.3.1.1: Users have been loaded into the API.

Inputs: Admin User ID, Target User ID, Firstname, Middlename, Lastname, email address, home address, College, Major, International Status, Scholarship, Study Abroad, In-State, Military, and Degree Program

Outputs: Success/Failure

Persistent Changes: User data is changed in memory and written to the user JSON file.

Related Requirements:

- Requirement 3.3.1.1: Users have been loaded into the API.

Error Scenarios:

- If the user ID is not found, an error shall be returned stating that user ID is not found.
- If the user ID is not authorized (see assumptions), an error shall be returned stating that the user is not authorized and the data will remain unchanged.

Test Cases: See Attached "Test Cases" Document

3.3.3.3. The API shall provide the ability to for a Graduate School Admin to edit information in a Graduate Student User's profile.

Description: A graduate admin user will use this to edit all fields except ID in a student user's profile.

Rationale: The rationale is that the user information might change and an admin might be required to update that information.

Assumptions: Only allowed/successful if the admin user is associated with the graduate school and the target/student user is also in graduate school.

Precondition:

- Requirement 3.3.1.1: Users have been loaded into the API.

Inputs: Admin User ID, Target User ID, modified fields

Outputs: Success/Failure

Persistent Changes: User data is changed in memory and written to the user JSON file.

Related Requirements:

- Requirement 3.3.1.1: Users have been loaded into the API.

Error Scenarios:

- If the user ID is not found, an error shall be returned stating that user ID is not found.
- If the user ID is not authorized (see assumptions), an error shall be returned stating that the user is not authorized and the data will remain unchanged.

Test Cases: See Attached "Test Cases" Document

3.4. Fee Requirements

3.4.1. Read in Fees

3.4.1.1. The API shall provide the ability to read in possible fees from an input file.

Description: Read in all of the fees that are valid. A list of possible fees are detailed in Appendix B.
Rationale: The rationale is that the fees will be static and should be the same for all students. They are also maintained by another system. The fees in that system will be exported in a flat file.
Assumptions: The file will be in a valid JSON format.
Precondition: None
Inputs: Filename of file with fees in there
Outputs: Success/Failure
Persistent Changes: The list of fees will be available to the API in memory. The filename inputted should be retained so changes to the fees can be saved.
Related Requirements: None
Error Scenarios:

- If the file is not available, an error shall be returned stating that the file could not be opened.
- If the file is empty, an error shall be returned stating that the file was empty.
- If the JSON in the file is invalid, an error shall be returned stating that the file format is invalid.

Test Cases: See Attached "Test Cases" Document

3.4.2. Retrieve Fee

3.4.2.1. The API shall provide the ability to retrieve the detail of a fee that was read in from the input file.

Description: Given a fee, retrieve the details for that fee.
Rationale: The rationale is that the fee details in memory might need to be viewed or retrieved for information purposes.
Assumptions: None
Precondition:

- Requirement 3.4.1.1: Fees have been loaded into the API.

Inputs: Fee
Outputs: Fee details
Persistent Changes: None
Related Requirements:

- Requirement 3.4.1.1: Fees have been loaded into the API.

Error Scenarios:

- If the fee is not found, an error shall be returned stating that the fee is not found.

Test Cases: See Attached "Test Cases" Document

3.4.3. Assign Fee

3.4.3.1. Associate a Student as part of a College for a Term

Description: Allow an admin user to enroll a student in a College for a Term.

Rationale: The rationale is administrators enroll students and then all of the fees for that college and term student are added to the student's bill.

Assumptions: Only allowed/successful if the College of the admin user who is logged in and the College of the target/student user are the same.

Precondition:

- Requirement 3.1.1.1: Colleges have been loaded into the API.
- Requirement 3.2.1.1: Terms have been loaded into the API.
- Requirement 3.3.1.1: Users have been loaded into the API.
- Requirement 3.4.1.1: Fees have been loaded into the API.

Inputs: Admin User, Target User, Term

Outputs: Success/Failure

Persistent Changes: Fees for the term for the student's college are assigned to the student and the student's amount due is updated. The history and amount are saved in the student file.

Related Requirements:

- Requirement 3.1.1.1: Colleges have been loaded into the API.
- Requirement 3.2.1.1: Terms have been loaded into the API.
- Requirement 3.3.1.1: Users have been loaded into the API.
- Requirement 3.4.1.1: Fees have been loaded into the API.

Error Scenarios:

- If the admin user ID is not found, an error shall be returned stating that admin user ID is not found.
- If the student user ID is not found, an error shall be returned stating that student user ID is not found.
- If the admin user ID is not authorized (see assumptions), an error shall be returned stating that the user is not authorized and the data will remain unchanged.
- If the changes are unable to be saved, and error shall be returned stating that the fee cannot be added because the fail cannot be saved.

Test Cases: See Attached "Test Cases" Document

3.4.3.2. Assign a Fee to a Single Students

Description: Allow an admin user to assign an individual fee to a student in a College for a Term.

Rationale: The rationale is administrators might have a one-off or special fee that needs to be added to a student's bill for a particular term.

Assumptions: Only allowed/successful if the College of the admin user who is logged in and the College of the target/student user are the same.

Precondition:

- Requirement 3.1.1: Colleges have been loaded into the API.
- Requirement 3.2.1.1: Terms have been loaded into the API.
- Requirement 3.3.1.1: Users have been loaded into the API.
- Requirement 3.4.1.1: Fees have been loaded into the API.

Inputs: Admin User, Target User, Term, Fee

Outputs: Success/Failure

Persistent Changes: The added fee is assigned to the student and the student's amount due is updated. The history and amount are saved in the student file.

Related Requirements:

- Requirement 3.1.1.1: Colleges have been loaded into the API.
- Requirement 3.2.1.1: Terms have been loaded into the API.
- Requirement 3.3.1.1: Users have been loaded into the API.
- Requirement 3.4.1.1: Fees have been loaded into the API.

Error Scenarios:

- If the admin user is not found, an error shall be returned stating that admin user is not found.
- If the student user is not found, an error shall be returned stating that student user is not found.
- If the admin user is not authorized (see assumptions), an error shall be returned stating that the user is not authorized and the data will remain unchanged.
- If the fee is not found, an error shall be returned stating that the fee is not found.
- If the changes are unable to be saved, and error shall be returned stating that the fee cannot be added because the fail cannot be saved.

Test Cases: See Attached "Test Cases" Document

3.4.3.3. Assign a Fee to all Students by College and Term

Description: Allow an admin user to assign an individual fee to all students in a College for a Term.

Rationale: The rationale is administrators might have a one-off or special fee that needs to be added to all student's bill for a particular term.

Assumptions: Only allowed/successful if the College of the admin user who is logged in and the College of the target/student user are the same.

Precondition:

- Requirement 3.1.1.1: Colleges have been loaded into the API.
- Requirement 3.2.1.1: Terms have been loaded into the API.
- Requirement 3.3.1.1: Users have been loaded into the API.
- Requirement 3.4.1.1: Fees have been loaded into the API.

Inputs: Admin User, Term, Fee

Outputs: Success/Failure

Persistent Changes: The added fee is assigned to all of the student for the college and each student's amount due is updated. The history and amount are saved in the student file.

Related Requirements:

- Requirement 3.1.1.1: Colleges have been loaded into the API.
- Requirement 3.2.1.1: Terms have been loaded into the API.
- Requirement 3.3.1.1: Users have been loaded into the API.
- Requirement 3.4.1.1: Fees have been loaded into the API.

Error Scenarios:

- If the admin user is not found, an error shall be returned stating that admin user is not found.
- If the admin user is not authorized (see assumptions), an error shall be returned stating that the user is not authorized and the data will remain unchanged.
- If the fee is not found, an error shall be returned stating that the fee is not found.
- If the changes are unable to be saved, an error shall be returned stating that the fee cannot be added because the fail cannot be saved.

Test Cases: See Attached "Test Cases" Document

3.5. Payment Requirements

3.5.1. Create Payment

3.5.1.1. The system shall allow a student user to make a payment on their account.

Description: Allow a student user to make a payment on their own account for a particular term.
Rationale: The rationale is students need to pay their bills and might want to be able to affect those payments themselves.
Assumptions: A student user can make payment only to their own accounts
Precondition:

- Requirement 3.3.1.1: Users have been loaded into the API.
- Requirement 3.4.1.1: Fees have been loaded into the API.

Inputs: User, Payment amount, Term
Outputs: Success/Failure
Persistent Changes: The new payment is assigned to the student and the student's amount due is updated. The payment is added to the history with the date and the history and amount are saved in the student file.
Related Requirements:

- Requirement 3.3.1.1: Users have been loaded into the API.
- Requirement 3.4.1.1: Fees have been loaded into the API.

Error Scenarios:

- If the student user is not found, an error shall be returned stating that student user is not found.
- If the student user is not authorized (see assumptions), an error shall be returned stating that the user is not authorized and the data will remain unchanged.
- If the payment amount is more than the balance due, and error shall be returned that states that only a payment for the balance due or less is possible.
- If the payment amount is not positive (i.e. zero or negative), an error shall be returned stating that the amount is invalid.
- If the changes are unable to be saved, an error shall be returned stating that the payment cannot be added because the fail cannot be saved.

Test Cases: See Attached "Test Cases" Document

3.5.1.2. The system shall allow a user admin to make a payment on a student user's in the admin's college's account.

Description: Allow an admin user to make a payment on the account of a student who is in that admin's own college for a particular term.

Rationale: The rationale is administrators may need to post credits or payments to accounts.

Assumptions: An admin user can make payments only to accounts within their college.

Precondition:

- Requirement 3.3.1.1: Users have been loaded into the API.
- Requirement 3.4.1.1: Fees have been loaded into the API.

Inputs: Admin User, Target (Student) User, Payment amount, Term

Outputs: Success/Failure

Persistent Changes: The new payment is assigned to the student and the student's amount due is updated. The payment is added to the history with the date and the history and amount are saved in the student file.

Related Requirements:

- Requirement 3.3.1.1: Users have been loaded into the API.
- Requirement 3.4.1.1: Fees have been loaded into the API.

Error Scenarios:

- If the student user is not found, an error shall be returned stating that student user is not found.
- If the admin user is not authorized (see assumptions), an error shall be returned stating that the user is not authorized and the data will remain unchanged.
- If the payment amount is more than the balance due, and error shall be returned that states that only a payment for the balance due or less is possible.
- If the payment amount is not positive (i.e. zero or negative), an error shall be returned stating that the amount is invalid.
- If the changes are unable to be saved, an error shall be returned stating that the payment cannot be added because the fail cannot be saved.

Test Cases: See Attached "Test Cases" Document

3.5.1.3. The system shall allow a user graduate admin to make a payment on a graduate student user's account.

Description: Allow an graduate school admin user to make a payment on the account of a graduate student for a particular term.

Rationale: The rationale is administrators may need to post credits or payments to accounts.

Assumptions: A graduate admin user can make payments only to accounts of graduate students.

Precondition:

- Requirement 3.3.1.1: Users have been loaded into the API.
- Requirement 3.4.1.1: Fees have been loaded into the API.

Inputs: Admin User, Target (Student) User, Payment amount, term

Outputs: Success/Failure

Persistent Changes: The new payment is assigned to the student and the student's amount due is updated. The payment is added to the history with the date and the history and amount are saved in the student file.

Related Requirements:

- Requirement 3.3.1.1: Users have been loaded into the API.
- Requirement 3.4.1.1: Fees have been loaded into the API.

Error Scenarios:

- If the student user is not found, an error shall be returned stating that student user is not found.
- If the admin user is not authorized (see assumptions), an error shall be returned stating that the user is not authorized and the data will remain unchanged.
- If the payment amount is more than the balance due, and error shall be returned that states that only a payment for the balance due or less is possible.
- If the payment amount is not positive (i.e. zero or negative), an error shall be returned stating that the amount is invalid.
- If the changes are unable to be saved, an error shall be returned stating that the payment cannot be added because the fail cannot be saved.

Test Cases: See Attached "Test Cases" Document

3.5.2. Retrieve Payments

3.5.2.1. The system shall allow a student user to view the payment history of their account.

Description: Allow a student user to retrieve a list of all payments made on their account for a term.

Rationale: The rationale is students will want to view a list of all of the payments that have been made to their account.

Assumptions: A student user can only view payments made to their own account

Precondition:

- Requirement 3.3.1.1: Users have been loaded into the API.

Inputs: User, Term

Outputs: A list of payments

Persistent Changes: None

Related Requirements:

- Requirement 3.3.1.1: Users have been loaded into the API.

Error Scenarios:

- If the student user is not found, an error shall be returned stating that student user is not found.
- If the student user is not authorized (see assumptions), an error shall be returned stating that the user is not authorized and the data will remain unchanged.

Test Cases: See Attached "Test Cases" Document

3.5.2.2. The system shall allow a user admin to view the payment history of a student user account in their college.

Description: Allow an admin user to retrieve a list of all payments made on the account of a student in their college for a term.
Rationale: The rationale is administrators may need view a student's payment history.
Assumptions: An admin user can make payments only to accounts within their college.
Precondition:

- Requirement 3.3.1.1: Users have been loaded into the API.

Inputs: Admin User, Target (Student) User, Term
Outputs: A list of payments
Persistent Changes: None
Related Requirements:

- Requirement 3.3.1.1: Users have been loaded into the API.

Error Scenarios:

- If the student user is not found, an error shall be returned stating that student user is not found.
- If the admin user is not authorized (see assumptions), an error shall be returned stating that the user is not authorized and the data will remain unchanged.

Test Cases: See Attached "Test Cases" Document

3.5.2.3. The system shall allow a user graduate admin to view the payment history of a graduate student user account.

Description: Allow an graduate admin user to retrieve a list of all payments made on the account of a graduate student.

Rationale: The rationale is administrators may need view a student's payment history.

Assumptions: An admin user can make payments only to accounts within their college.

Precondition:

- Requirement 3.3.1.1: Users have been loaded into the API.

Inputs: Admin User, Target (Student) User, Term

Outputs: A list of payments

Persistent Changes: None

Related Requirements:

- Requirement 3.3.1.1: Users have been loaded into the API.

Error Scenarios:

- If the student user is not found, an error shall be returned stating that student user is not found.
- If the admin user is not authorized (see assumptions), an error shall be returned stating that the user is not authorized and the data will remain unchanged.

Test Cases: See Attached "Test Cases" Document

3.5.2.4. The system shall allow a student user to view the details of a single payment made to their account.

Description: Allow a student user to retrieve the details of a particular payment made to their account.

Rationale: The rationale is students will want to view more details about a particular payment.

Assumptions: A student user can only view payment details of a payment made to their own account

Precondition:

- Requirement 3.3.1.1: Users have been loaded into the API.
- Requirement 3.5.1.1-3.5.1.3: A Payment has been made.

Inputs: User, Payment ID

Outputs: Payment Amount, Payment Date, User who made the Payment

Persistent Changes: None

Related Requirements:

- Requirement 3.3.1.1: Users have been loaded into the API.
- Requirement 3.5.1.1-3.5.1.3: A Payment has been made.

Error Scenarios:

- If the student user is not found, an error shall be returned stating that student user is not found.
- If the student user is not authorized (see assumptions), an error shall be returned stating that the user is not authorized.

Test Cases: See Attached "Test Cases" Document

3.5.2.5. The system shall allow an admin user to view the payment details of a single payment on a student user account in their college.

Description: Allow an admin user to retrieve the details of a particular payment made to a student in their college's account.
Rationale: The rationale is that admins might need to view details about a particular payment that was made to a student account.
Assumptions: A admin user can only view payment details of a payment made to an account for a student in their college.
Precondition:

- Requirement 3.3.1.1: Users have been loaded into the API.
- Requirement 3.5.1.1-3.5.1.3: A Payment has been made.

Inputs: Admin User, Target/Student User, Payment ID
Outputs: Payment Amount, Payment Date, User who made the Payment
Persistent Changes: None
Related Requirements:

- Requirement 3.3.1.1: Users have been loaded into the API.
- Requirement 3.5.1.1-3.5.1.3: A Payment has been made.

Error Scenarios:

- If the student user is not found, an error shall be returned stating that student user is not found.
- If the admin user is not authorized (see assumptions), an error shall be returned stating that the user is not authorized.

Test Cases: See Attached "Test Cases" Document

3.5.2.6. The system shall allow a user graduate admin to view the payment details of a single payment on a graduate student user account.

Description: Allow a graduate admin user to retrieve the details of a particular payment made to a graduate student's account.
Rationale: The rationale is that graduate admins might need to view details about a particular payment that was made to a student account.
Assumptions: A graduate admin user can only view payment details of a payment made to a graduate student's account.
Precondition:

- Requirement 3.3.1.1: Users have been loaded into the API.
- Requirement 3.5.1.1-3.5.1.3: A Payment has been made.

Inputs: Admin User, Target/Student User, Payment ID
Outputs: Payment Amount, Payment Date, User who made the Payment
Persistent Changes: None
Related Requirements:

- Requirement 3.3.1.1: Users have been loaded into the API.
- Requirement 3.5.1.1-3.5.1.3: A Payment has been made.

Error Scenarios:

- If the student user is not found, an error shall be returned stating that student user is not found.
- If the admin user is not authorized (see assumptions), an error shall be returned stating that the user is not authorized.

Test Cases: See Attached "Test Cases" Document

3.5.3. Edit Payment

3.5.3.1. The system shall allow a user admin to edit the payment details of a single payment on a student user account in their college.

Description: Allow an admin user to edit a payment on the account of a student who is in that admin's own college for a particular term.

Rationale: The rationale is administrators may need to modify credits or payments to accounts.

Assumptions: An admin user can edit payments only to accounts within their college.

Precondition:

- Requirement 3.3.1.1: Users have been loaded into the API.
- Requirement 3.5.1.1-3.5.1.3: A Payment has been made.

Inputs: Admin User, Target (Student) User, Payment, Term

Outputs: Success/Failure

Persistent Changes: The modified payment is assigned to the student and the student's amount due is updated. The modified payment is added to the history with the date and the history and amount are saved in the student file.

Related Requirements:

- Requirement 3.3.1.1: Users have been loaded into the API.
- Requirement 3.5.1.1-3.5.1.3: A Payment has been made.

Error Scenarios:

- If the student user is not found, an error shall be returned stating that student user is not found.
- If the admin user is not authorized (see assumptions), an error shall be returned stating that the user is not authorized and the data will remain unchanged.
- If the payment amount is more than the balance due, and error shall be returned that states that only a payment for the balance due or less is possible.
- If the payment amount is not positive (i.e. zero or negative), an error shall be returned stating that the amount is invalid.
- If the changes are unable to be saved, an error shall be returned stating that the payment cannot be added because the fail cannot be saved.

Test Cases: See Attached "Test Cases" Document

3.5.3.2. The system shall allow a user graduate admin to edit the payment details of a single payment on a graduate student user account.

Description: Allow a graduate admin user to edit a payment on the account of a graduate student.

Rationale: The rationale is administrators may need to modify credits or payments to accounts.

Assumptions: A graduate admin user can edit payments only to accounts within the graduate school.

Precondition:

- Requirement 3.3.1.1: Users have been loaded into the API.
- Requirement 3.5.1.1-3.5.1.3: A Payment has been made.

Inputs: Admin User, Target (Student) User, Payment, Term

Outputs: Success/Failure

Persistent Changes: The modified payment is assigned to the student and the student's amount due is updated. The modified payment is added to the history with the date and the history and amount are saved in the student file.

Related Requirements:

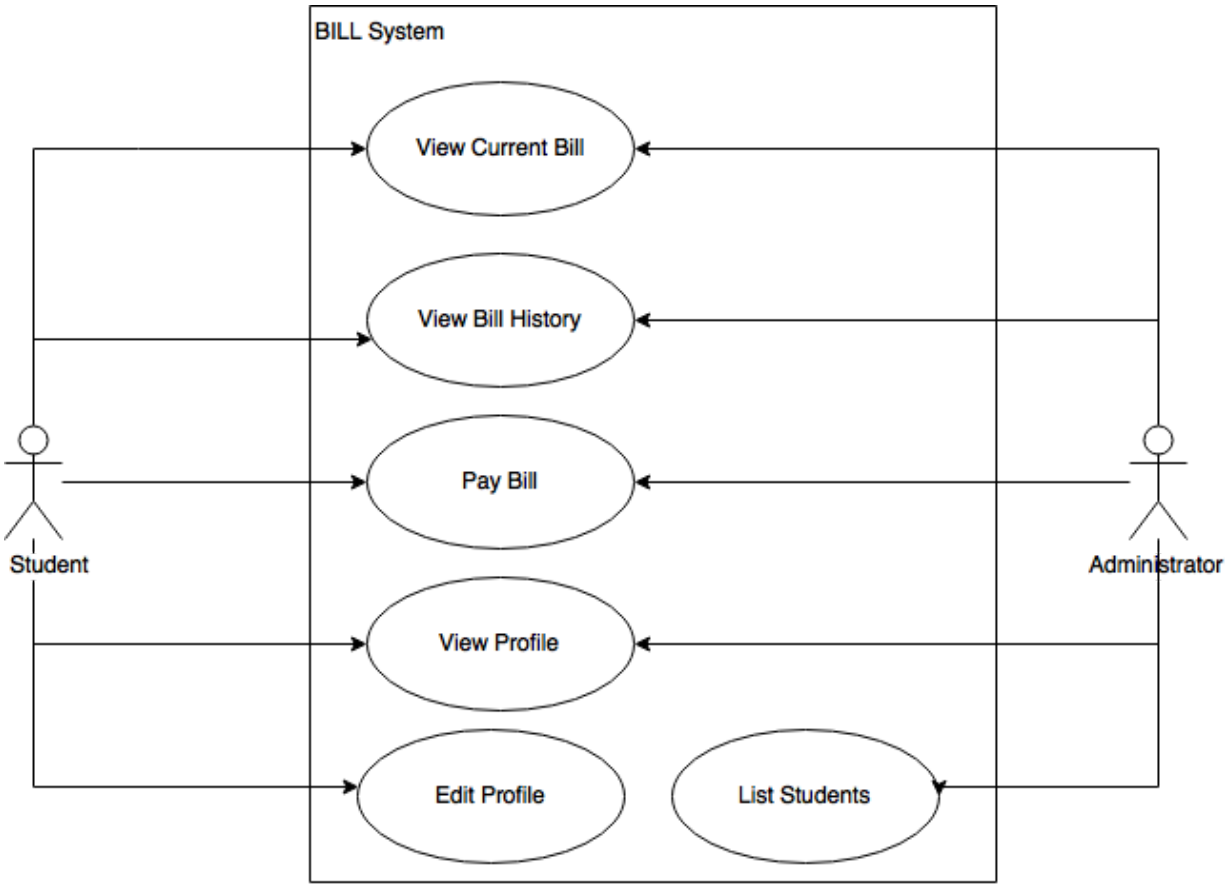
- Requirement 3.3.1.1: Users have been loaded into the API.
- Requirement 3.5.1.1-3.5.1.3: A Payment has been made.

Error Scenarios:

- If the student user is not found, an error shall be returned stating that student user is not found.
- If the admin user is not authorized (see assumptions), an error shall be returned stating that the user is not authorized and the data will remain unchanged.
- If the payment amount is more than the balance due, and error shall be returned that states that only a payment for the balance due or less is possible.
- If the payment amount is not positive (i.e. zero or negative), an error shall be returned stating that the amount is invalid.
- If the changes are unable to be saved, an error shall be returned stating that the payment cannot be added because the fail cannot be saved.

Test Cases: See Attached "Test Cases" Document

Appendix A : Use Case Diagrams and Descriptions for BILL System



Use Case: *View Current Bill*

Actor: Student, Administrator

Description: When the student requests to view their current bill, the system shall:

- Returns current bill data, accounting for all credits and debits

Exception Paths:

- System shall return Access Denied if student attempts to view a bill that is not their own.
- System shall return Access Denied if a college administrator attempts to view the bill of a student who is not in their college.
- System shall return Access Denied if an administrator in the graduate school attempts to view the bill of any undergraduate student.

Preconditions: The requestor has been authenticated and is authorized to view the bill

Use Case: *View Bill History*

Actor: Student, Administrator

Description: When the student requests to view their bill history, the system shall:

- Load bill history (debits and credits) between (inclusive) the dates provided by the student
- Return the data to the requestor

Exception Paths:

- If the provided date range is invalid and error is returned
- System shall return Access Denied if student attempts to view the bill history that is not their own.
- System shall return Access Denied if a college administrator attempts to view the bill history of an student who is not in their college.
- System shall return Access Denied if an administrator in the graduate school attempts to view the bill history of any undergraduate student.

Preconditions:

- The requestor has been authenticated and is authorized to view the bill
- The requestor has provided a date range (start and end date) of billing history

Use Case: *Pay Bill*

Actor: Student

Description: When the student requests to make a payment, the system shall:

- System sends billing details to payment provider
- Payment provider returns the result of the payment (succeeded, failed) to system

Exception Paths:

- System shall return Access Denied if student attempts to pay a bill that is not their own.
- System shall return Access Denied if a college administrator attempts to pay the bill of a student who is not in their college.
- System shall return Access Denied if an administrator in the graduate school attempts to pay the bill of any undergraduate student.

Preconditions: The student has been authenticated and is authorized to pay the bill.

Use Case: *View Profile*

Actor: Student, Administrator

Description: When the student requests to view their profile, the system shall:

- Load profile information
- Return the data to the requestor

Exception Paths:

- System returns Access Denied if a student attempts to view the profile of another student
- System returns Access Denied if an administrator attempts to view the profile of a student outside their college
- System shall return Access Denied if an administrator in the graduate school attempts to view the profile of any undergraduate student.

Preconditions: The requestor has been authenticated and is authorized to view their profile.

Use Case: *Edit Profile*

Actor: Student

Description: When the student requests to edit their profile, the system shall:

- Update the profile information.
- Students can update their: Firstname, Middlename, Lastname, email address, home address.
- Administrators can update a profiles: College, Major, International Status, Scholarship, Study Abroad, In-State, Military, and Degree Program.

Exception Paths:

- Student provides data that does not conform to the data types (strings in date fields, etc)
- Student attempts to edit read-only data (username, student id, etc)
- System returns Access Denied if a student attempts to post information to another's profile.
- System shall return Access Denied if an administrator in the graduate school attempts to edit the profile of any undergraduate student.

Preconditions: The student has been authenticated and is authorized to edit their profile.

Use Case: *List Students*

Actor: Administrator

Description: When the administrator requests list students, the system shall:

- Return a list of all the students in their college.

Exception Paths:

- System returns Access Denied if Administrator attempts to view students not in their college.

Preconditions: The administrator has been authenticated and is authorized to list students for their college.

Appendix B : Tuition and Fees

Source (http://www.sc.edu/about/offices_and_divisions/bursar/tuition_and_required_fees/index.php)

Undergraduate Student Tuition

Student Type	Full-Time	Part-Time
Resident	5931	494.25
Non-Resident	15981	1331.75
Non-Resident with a Woodrow or Departmental Scholarship	8802	733.50
Non-Resident with a General University Scholarship	5931	494.25
Non-Resident with an Athletic Scholarship	8802	733.50
Non-Resident with a Sims Scholarship	11352	946
Active Duty Military Student	3474	289.50

Graduate Student Tuition

Student Type	Full-Time	Part-Time
Resident	6627	522.25
Non-Resident	14184	1182

Fees

Fee Description	Full-Time	Part-Time
Technology Fee	200	17 / credit hour
Health Insurance	2020	
Health Center (6-8 hours)	123	

Health Center (9-11 hours)	184	
Capstone Scholar	150	
International Student Enrollment - One Time Charge	750	
Short-Term International Student	200	
Sponsored International Student	250	
Study Abroad	150	
Cohort Study Abroad	300	

Requirements Based Test Cases for

CSCE 740, Fall 2017 B.I.L.L.
Revision 1.0
10/1/2017

Project Team:

- Aaron Hein
- Jeremy Lewis
- Peter Mourfield

Table of Contents

Test Cases

College Test Cases (A)

Tests Requirements from 3.1

good_colleges.json:

```
{
  "colleges": [
    {
      "id": "1",
      "name": "College of Engineering and Computing"
    }, {
      "id": "2",
      "name": "College of Arts and Sciences"
    }
  ]
}
```

corrupt_colleges.json:

```
{
  "colleges":
    {
      "id": "1",
      "name": "College of Engineering and Computing"
    }, {
      "id": "2",
      "name": "College of Arts and Sciences"
    }
}
```

Traceability Matrix - A

Requirement \ Test Case	3.1.1.1	3.1.2.1	3.1.2.2	3.1.3.1
A.1	X			
A.2	X			
A.3	X			
A.4	X			
A.5	X			
A.6		X		
A.7		X		
A.8			X	
A.9				X
A.10				X
A.11				X
A.12				X

A.1 Read Valid JSON Input File From Correct Path And Filename

Test Inputs: A string containing a path to and filename of a non-empty valid JSON text file containing the colleges supported by the API. See file above.
Expected Results: A list of colleges read in from the file.
Dependencies: None
Initialization: None
Test Steps:

1. The string “good_colleges.json” is passed to the API.
2. Check to make sure that an exception was not thrown.
3. Display the list of loaded colleges from the API
4. The list returned by the API will match the data as described by good_colleges.json above.

A.2 Read JSON Input File From Incorrect Path And Filename

Test Inputs: A string that does not reference any file or path in the test system.
Expected Results: Throws a FileNotFoundException.
Dependencies: None
Initialization: None
Test Steps:

1. The string “/this/is/not/a/file” is passed to the API.
2. A FileNotFoundException is caught.

A.3 Read Empty JSON Input File From Correct Path And Filename

Test Inputs: A string containing a path to an empty file.
Expected Results: Throws a IOException.
Dependencies: None
Initialization: None
Test Steps:

1. The string “empty_colleges.json” is passed to the API.
2. A FileNotFoundException is caught with getMessage() equal to “The file empty_colleges.json” is empty.

A.4 Read Invalid JSON Input File From Correct Path And Filename

Test Inputs: A string containing a path to and filename of a non-empty JSON text file containing invalid json. See corrupt_colleges.json.
Expected Results: Throws Exception. To avoid forced inclusion of JSON packages, API rethrows JsonParsingException to Exception.
Dependencies: None
Initialization: None
Test Steps:

1. The string “corrupt_colleges.json” is passed to the API.

2. An exception is caught with getMessage() equal to “The file corrupt_colleges.json is corrupt.”.

A.5 Ensure nothing loaded after an error is thrown

Test Inputs: None
Expected Results: Nothing is displayed to the screen.
Dependencies: None
Initialization: Execute a test resulting in an exception:

1. A.2
2. A.3
3. A.4

Test Steps:

1. Attempt to display the list of loaded colleges. Nothing should be displayed

A.6 Retrieve College Using Correct ID

Test Inputs: The ID number of a college from good_colleges.json.
Expected Results: None.
Dependencies: The file good_colleges.json has been loaded into the API
Initialization: None
Test Steps:

1. The string 1 is passed to API as the ID to retrieve college.
2. The output for college 1 is displayed as per the file above.

A.7 Retrieve College Using Incorrect ID

Test Inputs: An ID number which is not found in good_colleges.json.
Expected Results: Throws IndexOutOfBounds exception.
Dependencies: The file good_colleges.json has been loaded into the API
Initialization: None
Test Steps:

1. The string 3 is passed as the ID to the API to retrieve the college.
2. An IndexOutOfBoundsException is thrown with getMessage() equal to “There is no College with ID 3.”

A.8 Retrieve List of All Colleges

Test Inputs: none
Expected Results: A list of colleges
Dependencies: The file good_colleges.json has been loaded into the API
Initialization: None
Test Steps:

1. The list is requested from API
2. The list returned by the API matches the file as described above.

A.9 Edit College with allowed Administrator ID

Test Inputs:

1. Administrator ID
2. College ID
3. New College Name

Expected Results: None

Dependencies:

1. The file good_colleges.json has been loaded into the API
2. The file good_admin.json has been loaded into the API

Initialization: None

Test Steps:

1. Pass in a string containing a college ID that is in the system (example: “1”), the modified elements (“Awesome College of Engineering and Computing”) and any valid administrator.
2. Display the modified list.
3. The list returned by the API matches the file as described above.

A.10 Edit College error because of an unknown College ID

Test Inputs:

1. Administrator ID
2. College ID
3. New College Name

Expected Results: Throws IndexOutOfBoundsException

Dependencies:

1. The file good_colleges.json has been loaded into the API
2. The file good_admin.json has been loaded into the API

Initialization: None

Test Steps:

1. Pass in a string containing a college ID that is not in the system (example: “3”), the modified elements (“New_Name”) and any valid administrator.
2. Check to ensure the exception is thrown
3. Examine the both records to ensure the college list was not modified.

A.11 Edit College with failed save to the file

Test Inputs:

1. Administrator ID
2. College ID
3. New College Name

Expected Results: Throws a IOException.

Dependencies:

1. The file good_nowrite_colleges.json has been loaded into the API
2. The file good_admin.json has been loaded into the API

Initialization: Ensure permissions on good_nowrite_colleges.json are such that the file cannot be written to.

Test Steps:

1. Pass in a string containing a college ID that is in the system (example: “2”), the modified elements (“New_Name”) and any valid administrator.
2. Check to ensure the exception is thrown
3. Examine the record to ensure it was not modified.

A.12 Edit College with disallowed Administrator ID

Test Inputs:

1. Administrator ID
2. College ID
3. New College Name

Expected Results: None

Dependencies:

1. The file good_colleges.json has been loaded into the API
2. The file good_admin.json has been loaded into the API

Initialization: None

Test Steps:

1. Pass in a string containing a college ID that is in the system (example: “1”), the modified elements (“Awesome College of Engineering and Computing”) and any invalid administrator.
2. Check to ensure the exception is thrown
3. Examine the record to ensure it was not modified

Term Test Cases (B)

Tests Requirements from 3.2

Good_terms.json:

```
[ {
       "id":"1",
       "terms":[  {
             "SP18":[  {
                   "name":"CSCE211-001",
                   "credits":3,
                   "online":false
                }  ]  },
          {
             "SU18":[   {
                   "name":"CSCE311-001",
                   "credits":3,
                   "online":true
                } ] }
       ] },
    {
       "id":"2",
       "Terms":[  {
             "SP18":[ {
                   "name":"CSCE240-001",
                   "credits":3,
                   "online":true
                } ]
          } ]
    } ]
```

Traceability Matrix - B

Requirement \ Test Case	3.2.1.1
B.1	X
B.2	X
B.3	X
B.4	X
B.5	X

B.1 Read Valid Term JSON Input File From Correct Path And Filename

Test Inputs: A string: a path to and filename of a non-empty valid JSON text file containing the terms available for each college as described above.
Expected Results: A list of courses for each college.
Dependencies: None
Initialization: Colleges from good_colleges.json successfully read into system.
Test Steps:

1. The string “good_terms.json” is passed to the API.
2. The list returned by the API will have three elements,
    a. A course--name:CSCE211-001, college:1, term:SP18, credits:3, online:false.
    b. A course--name:CSCE311-001, college:1, term:SU18, credits:3, online:true.”
    c. A course--name:CSCE240-001, college:2, term:SP18, credits:3, online:true.

B.2 Read Invalid JSON Input File From Correct Path And Filename

Test Inputs: A string containing a path to and filename of a non-empty JSON text file containing containing the text: [{"id":"1","terms":[{"SP18":[{"name":"CSCE211-001",
Expected Results: Throws Exception. To avoid forced inclusion of JSON packages, API rethrows JsonParsingException to Exception.
Dependencies: None
Initialization: None
Test Steps:

1. The string “corrupt_terms.json” is passed to the API.
2. An exception is caught with getMessage() equal to “The file corrupt_colleges.json is corrupt.”.

B.3 Read JSON Input File From Incorrect Path And Filename

Test Inputs: A string that does not reference any file or path in the test system.
Expected Results: Throws a FileNotFoundException.
Dependencies: None
Initialization: None
Test Steps:

1. The string “/this/is/not/a/file” is passed to the API.
2. A FileNotFoundException is caught.

B.4 Read Empty JSON Input File From Correct Path And Filename

Test Inputs: A string containing a path to an empty file.
Expected Results: Throws a IOException.
Dependencies: None
Initialization: None
Test Steps:

1. The string “empty_terms.json” is passed to the API.
2. A FileNotFoundException is caught with getMessage() equal to “The file empty_colleges.json” is empty.

B.5 Read Valid Term JSON Input File From Correct Path And Filename For Non-Existent College

Test Inputs: A string: a path to and filename of a non-empty valid JSON text file containing the terms available for each college. The file should contain the text:
[{"id":"99","terms":[{"SP18":[{"name":"CSCE740-001","credits":3,"online":false}]}]}]
Expected Results: A list of courses for each college.
Dependencies: None
Initialization: Colleges from good_colleges.json successfully read into system.
Test Steps:

1. The string “missing_college_terms.json” is passed to the API.
2. An exception is caught with getMessage() equal to “The file missing_college_terms.json references college id 3. Ensure this college exits in your input file and reload colleges”.

User Test Cases (C)

Tests Requirements from 3.3

good_students.json:


```
{
  "students": [
    {
      "id": “101”,
      "first_name": "John",
      "middle_name": "Q",
      "last_name": "Doe",
      "email": "john.doe@email.sc.edu",
      "address": "102 Main Street",
      "city": "Columbia",
      "state": "SC",
      "zip": "29208",
      "status": "Junior",
      "college_id": "1",
      "enrollment": [
        {
          "term": "SP18",
          "classes": ["CSCE211-001"],
        }, {
          "term": "FA17",
          "classes": ["CSCE101-002"],
        }
      ],
      "int_status": "NONE",
      "scholarship": "NONE",
      "abroad": "NONE",
      "degree": "BS",
      "in_state": true,
      "full_time": true,
      "military": false,
      "int_fee_paid": true,
      "placement": true,
    }, {
      "id": “102”,
      "first_name": "Jane",
      "middle_name": "Q",
      "last_name": "Grad",
```

```
  "email": "jane.grad@email.sc.edu",
  "address": "202 Main Street",
  "city": "Columbia",
  "state": "SC",
  "zip": "29208",
  "status": "MASTERS",
  "college_id": "1",
  "enrollment": [
    {
      "term": "SP18",
      "classes": ["CSCE511-001"],
    }, {
      "term": "FA17",
      "classes": ["CSCE722-002"],
    }
  ],
  "int_status": "SPONSORED",
  "scholarship": "ATHLETIC",
  "abroad": "REGULAR",
  "degree": "MS",
  "in_state": false,
  "full_time": true,
  "military": false,
  "int_fee_paid": false,
  "placement": true,
}, {
  "id": "103",
  "first_name": "Jane",
  "middle_name": "Q",
  "last_name": "Doe",
  "email": "jane.doe@email.sc.edu",
  "address": "1202 Main Street",
  "city": "Columbia",
  "state": "SC",
  "zip": "29209",
  "status": "Freshman",
  "college_id": "2",
  "enrollment": [
    {
      "term": "SP18",
      "classes": ["CSCE211-001"],
    }, {
      "term": "FA17",
```

```
      "classes": ["CSCE101-002"],
    }
  ],
  "int_status": "NONE",
  "scholarship": "NONE",
  "abroad": "REGULAR",
  "degree": "BA",
  "in_state": true,
  "full_time": true,
  "military": false,
  "int_fee_paid": true,
  "placement": true,
}, {
  "id": "104",
  "first_name": "John",
  "middle_name": "Q",
  "last_name": "Grad",
  "email": "john.doe@email.sc.edu",
  "address": "102 Main Street",
  "city": "Columbia",
  "state": "SC",
  "zip": "29208",
  "status": "PHD",
  "college_id": "1",
  "enrollment": [
    {
      "term": "SP18",
      "classes": ["CSCE511-001"],
    }, {
      "term": "FA17",
      "classes": ["CSCE722-002"],
    }
  ],
  "int_status": "NONE",
  "scholarship": "NONE",
  "abroad": "NONE",
  "degree": "PHD",
  "in_state": true,
  "full_time": true,
  "military": false,
  "int_fee_paid": false,
  "placement": true,
}
```

```
  ]
}
```

corrupt_students.json:

```
{
  "students":[{]
}
```

good_admins.json:

```
{
  "administrators": [
    {
      "id": "A0001",
      "first_name": "John",
      "middle_name": "Q",
      "last_name": "Admin",
      "email": "john.admin@email.sc.edu",
      "college_id": "1",
      "status": "UNDERGRAD",
    }, {
      "id": "A0002",
      "first_name": "Jane",
      "middle_name": "Q",
      "last_name": "Admin",
      "email": "jane.admin@email.sc.edu",
      "college_id": "2",
      "status": "UNDERGRAD",
    }, {
      "id": "A0003",
      "first_name": "Bob",
      "middle_name": "Q",
      "last_name": "Admin",
      "email": "bob.admin@email.sc.edu",
      "college_id": "1",
      "status": "GRAD",
    }
  ]
}
```

corrupt_admins.json:

```
{
  "administrators":[testing"this"{]
```

```
}
```

Traceability Matrix - C

Requirement \ Test Case	3.3.1.1	3.3.2.1	3.3.2.2	3.3.2.3	3.3.2.4	3.3.2.5	3.3.3.1	3.3.3.2	3.3.3.3
C.1	X								
C.2	X								
C.3	X								
C.4	X								
C.5	X								
C.6	X								
C.7	X								
C.8	X								
C.9	X								
C.10			X						
C.11			X						
C.12			X						
C.13		X							
C.14		X							
C.15		X							
C.16		X							
C.17		X							
C.18				X					
C.19				X					
C.20					X				
C.21					X				
C.22						X			
C.23						X			

C.24						X			
C.25						X			
C.26							X		
C.27								X	
C.28							X		
C.29							X		
C.30								X	
C.31							X		
C.32							X		
C.33									X
C.34									X

C.1 Read Valid Administrators JSON Input File From Correct Path And Filename

Test Inputs: A string containing a path to and filename of a non-empty valid JSON text file containing administrators supported by the API. See good_administrators above.
Expected Results: A list of administrators read in from the file.
Dependencies: None
Initialization: None
Test Steps:

1. The string “good_admins.json” is passed to the API.
2. Check to make sure that an exception was not thrown.
3. Display the list of loaded administrators from the API.
4. The list returned by the API matches the file as described above.

C.2 Read Administrators JSON Input File From Incorrect Path And Filename

Test Inputs: A string that does not reference any file or path in the test system.
Expected Results: Throws a FileNotFoundException.
Dependencies: None
Initialization: None
Test Steps:

1. The string “/this/is/not/a/file” is passed to the API.
2. A FileNotFoundException is caught.

C.3 Read Empty Administrators JSON Input File From Correct Path And Filename

Test Inputs: A string containing a path to an empty file.
Expected Results: Throws a IOException.
Dependencies: None
Initialization: None
Test Steps:

1. The string “empty_admins.json” is passed to the API.
2. A FileNotFoundException is caught with getMessage() equal to “The file empty_admins.json” is empty.

C.4 Read Invalid Administrators JSON Input File From Correct Path And Filename

Test Inputs: A string containing a path to and filename of a non-empty JSON text file containing poorly formed JSON. See corrupt_admins.json above.
Expected Results: Throws Exception. To avoid forced inclusion of JSON packages, API rethrows JsonParsingException to Exception.
Dependencies: None
Initialization: None
Test Steps:

1. The string “corrupt_admins.json” is passed to the API.
2. An exception is caught with getMessage() equal to “The file corrupt_admins.json is corrupt.”

C.5 Read Valid Students JSON Input File From Correct Path And Filename

Test Inputs: A string containing a path to and filename of a non-empty valid JSON text file containing the colleges supported by the API. See good_students above
Expected Results: A list of students read in from the file.
Dependencies: None
Initialization: None
Test Steps:

1. The string “good_students.json” is passed to the API.
2. Check to make sure that an exception was not thrown.
3. Display the list of loaded students from the API
4. The list returned by the API matches the file as described above.

C.6 Read Students JSON Input File From Incorrect Path And Filename

Test Inputs: A string that does not reference any file or path in the test system.
Expected Results: Throws a FileNotFoundException.
Dependencies: None
Initialization: None
Test Steps:

1. The string “/this/is/not/a/file” is passed to the API.
2. A FileNotFoundException is caught.

C.7 Read Empty Students JSON Input File From Correct Path And Filename

Test Inputs: A string containing a path to an empty file.
Expected Results: Throws a IOException.
Dependencies: None
Initialization: None
Test Steps:

1. The string “empty_students.json” is passed to the API.
2. A FileNotFoundException is caught with getMessage() equal to “The file empty_students.json” is empty.

C.8 Read Invalid Students JSON Input File From Correct Path And Filename

Test Inputs: A string containing a path to and filename of a non-empty JSON text file containing poorly formed JSON. See corrupt_students.json above.
Expected Results: Throws Exception. To avoid forced inclusion of JSON packages, API rethrows JsonParsingException to Exception.
Dependencies: None
Initialization: None
Test Steps:

1. The string “corrupt_admins.json” is passed to the API.

2. An exception is caught with getMessage() equal to “The file corrupt_admins.json is corrupt.”

C.9 Ensure nothing loaded after an error is thrown

Test Inputs: None

Expected Results: Nothing is displayed to the screen.

Dependencies: None

Initialization: Execute a test resulting in an exception:

1. C.2
2. C.3
3. C.4
4. C.6
5. C.7
6. C.8

Test Steps:

1. Attempt to display the list of loaded colleges. Nothing should be displayed

C.10 Retrieve Admin Using Correct Administrator ID

Test Inputs:

1. User: The ID number of an administrator from good_admins.json.
2. Target: The same ID number as above

Expected Results: Return the record for the chosen administrator.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.

Initialization: None

Test Steps:

1. The string “A001” is passed to the API as the ID to retrieve an administrator.
2. The details for the administrator with id “A0001” are returned with the details per the file above.

C.11 Retrieve Admin Using Incorrect ID

Test Inputs:

1. User: The ID number of an administrator from good_admins.json.
2. Target: An ID number which is not found in good_admins.json.

Expected Results: Throws UnknownUser exception.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.

Initialization: None

Test Steps:

1. The string “BAD” is passed as the ID to the API to retrieve the administrator.
2. An UnknownException is thrown with getMessage() equal to “There is no Administrator with ID BAD.”

C.12 Retrieve Admin Using Invalid ID

Test Inputs:

1. User: The ID number of an administrator from good_admins.json.
2. Target: An ID number different from the above user ID.

Expected Results: Throws InvalidUser exception.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.

Initialization: None

Test Steps:

1. The string “A0001” is passed to the API as the user ID
2. The string “A0003” is passed to the API as the ID to the administrator to retrieve.
3. An InvalidUserException is thrown with getMessage() equal to “Insufficient permission to view User <ID>.”

C.13 Retrieve Student Using Correct Student ID

Test Inputs:

1. User: The ID number of a student from good_students.json.
2. Target: The same ID number as above

Expected Results: Return the record for the chosen student.

Dependencies:

3. The file good_colleges.json has been loaded into the API.
4. The file good_students.json has been loaded into the API.

Initialization: None

Test Steps:

1. The string “101” is passed to the API as the user ID and the target ID to retrieve a student.
2. The details for the student with id “101” are returned with the details per the file above.

C.14 Retrieve Student Using Correct Administrator ID

Test Inputs:

1. Target: The ID number of a student from good_students.json.
2. User: An Admin ID from the same college.

Expected Results: Return the record for the chosen student.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.

Initialization: None

Test Steps:

1. The string “101” is passed to the API as the target ID to retrieve a student.
2. The string “A0001” is passed to the API as the user ID.
3. The details for the student with id “101” are returned with the details per the file above.

C.15 Retrieve Student Using Incorrect ID

Test Inputs:

1. User: The ID number of a student from good_students.json.
2. Target: An ID number not found in good_students.json.

Expected Results: Throws UnknownUser exception.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_students.json has been loaded into the API.

Initialization: None

Test Steps:

1. The string “101” is passed to the API as the user.
2. The string “BAD” is passed to the API as the student to retrieve.
3. An UnknownUserException is thrown with getMessage() equal to “There is no Student with ID BAD.”

C.16 Retrieve Student Using Invalid Student ID

Test Inputs:

1. User: The ID number of a student from good_students.json.
2. Target: An ID number of a different student from good_students.json.

Expected Results: Throws InvalidUser exception.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.

Initialization: None

Test Steps:

1. The string “101” is passed to the API as the user.
2. The string “103” is passed to the API as the student to retrieve.
3. An InvalidUserException is thrown with getMessage() equal to “Permissions are not sufficient to view Student <ID>”

C.17 Retrieve Student Using Invalid Administrator ID

Test Inputs:

1. User: The ID number of a student from good_students.json.
2. Target: An ID number of an administrator from a different college.

Expected Results: Throws InvalidUser exception.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.

Initialization: None

Test Steps:

1. The string “A0001” is passed to the API as the user.
2. The string “103” is passed to the API as the student to retrieve.
3. An InvalidUserException is thrown with getMessage() equal to “Permissions are not sufficient to view Student <ID>”

C.18 Retrieve List of All Students

Test Inputs: An Administrator ID

Expected Results: A list of students for that Administrator’s college

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.

Initialization: None

Test Steps:

1. The string “A001” is passed to the API as the ID to retrieve an administrator.
2. The list contains all of the students whose college ID matches the college ID of the admin per the file described above.

C.19 Retrieve List of All Students by Term

Test Inputs:

1. An Administrator ID
2. A Term

Expected Results: A list of students for that Administrator’s college

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file good_terms.json has been loaded into the API.

Initialization: None

Test Steps:

1. The string “A001” is passed to the API as the ID to retrieve an administrator.

2. A Term from the terms file.
3. The list contains all of the students whose college ID matches the college ID of the admin per the file described above and who have a non-empty enrollment array for the specified term.

C.20 Retrieve List of All Graduate Students

Test Inputs: An Administrator ID for Grad School

Expected Results: A list of students for that Administrator's college

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.

Initialization: None

Test Steps:

1. The string "A003" is passed to the API as the ID to retrieve an administrator.
2. The list contains all of the students whose status indicated that they are in graduate school.

C.21 Retrieve List of All Graduate Students by Term

Test Inputs:

1. An Administrator ID for Grad School
2. A Term

Expected Results: A list of students for that Administrator's college

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file good_terms.json has been loaded into the API.

Initialization: None

Test Steps:

1. The string "A003" is passed to the API as the ID to retrieve an administrator.
2. The list contains all of the students whose status indicated that they are in graduate school and are enrolled in a class for the given term.

C.22 Edit Admin Using Correct Administrator ID

Test Inputs:

1. User: The ID number of an administrator from good_admins.json.
2. Target: The same ID number as above
3. Modifications except ID, college_id or status.

Expected Results: Return the modified record for the chosen administrator.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.

Initialization: None

Test Steps:

1. The string “A001” is passed to the API as the ID to edit the administrator/user.
2. Pass in the modified details.
3. The details for the administrator with id “A0001” are returned per the file above with the appropriate details modified.

C.23 Edit Admin Using Incorrect ID

Test Inputs:

1. User: The ID number of an administrator from good_admins.json.
2. Target: An ID number which is not found in good_admins.json.
3. Modifications except ID, college_id or status.

Expected Results: Throws UnknownUser exception.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.

Initialization: None

Test Steps:

1. The string “BAD” is passed as the ID to the API to edit the administrator.
2. Pass in the modified details.
3. An UnknownException is thrown with getMessage() equal to “There is no Administrator with ID BAD.”
4. Ensure that no administrators were changed or added.

C.24 Edit Admin Using Invalid ID

Test Inputs:

1. User: The ID number of an administrator from good_admins.json.
2. Target: An ID number different from the above user ID.
3. Modifications except ID, college_id or status.

Expected Results: Throws InvalidUser exception.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.

Initialization: None

Test Steps:

1. The string “A0001” is passed to the API as the user ID
2. The string “A0003” is passed to the API as the ID to the administrator to retrieve.
3. Pass in the modified details.
4. An InvalidUserException is thrown with getMessage() equal to “Insufficient permission to edit User <ID>.”
5. Ensure that administrator A0003 was not modified.

C.25 Edit Admin with failed save to the file

Test Inputs:

1. User: The ID number of an administrator from good_admins.json.
2. Target: The same ID number as above
3. Modifications except ID, college_id or status.

Expected Results: Throws a IOException.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_nowrite_admins.json has been loaded into the API
3. The file good_admin.json has been loaded into the API

Initialization: Ensure permissions on good_nowrite_admins.json are such that the file cannot be written to.

Test Steps:

1. Pass in a string containing a administrator ID that is in the system (example: “A0002”) and the modified elements (“New_email”).
2. Check to ensure the exception is thrown
3. Examine the record to ensure it was not modified.

C.26 Edit Student Using Correct Student ID

Test Inputs:

1. User: The ID number of a student from good_students.json.
2. Target: The same ID number as above
3. At least one modified field: first_name, middle_name, last_name, email, address, city, state or zip.

Expected Results: Return the modified record for the chosen student.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_students.json has been loaded into the API.

Initialization: None

Test Steps:

1. The string “101” is passed to the API as the user ID and the target ID to retrieve a student.
2. Pass in the modified details.
3. A details for the student with id “101” are returned with the details per the file above but with the modifications.

C.27 Edit Student Using Correct Administrator ID

Test Inputs:

1. Target: The ID number of a student from good_students.json.
2. User: An Admin ID from the same college.
3. At least one modified field except ID

Expected Results: Return the modified record for the chosen student.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.

Initialization: None

Test Steps:

1. The string “101” is passed to the API as the target ID to retrieve a student.
2. The string “A0001” is passed to the API as the user ID.
3. Pass in the modified data.
4. The details for the student with id “101” are returned with the details per the file above but with the modified data.

C.28 Edit Student Using Incorrect ID

Test Inputs:

1. User: The ID number of a student from good_students.json.
2. Target: An ID number not found in good_students.json.
3. At least one modified field: first_name, middle_name, last_name, email, address, city, state or zip.

Expected Results: Throws UnknownUser exception.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_students.json has been loaded into the API.

Initialization: None

Test Steps:

1. The string “101” is passed to the API as the user.
2. The string “BAD” is passed to the API as the student to retrieve.
3. Pass in the modified data.
4. An UnknownUserException is thrown with getMessage() equal to “There is no Student with ID BAD.”
5. Examine the record to ensure it was not modified.

C.29 Edit Student Using Invalid Student ID

Test Inputs:

1. User: The ID number of a student from good_students.json.
2. Target: An ID number of a different student from good_students.json.
3. At least one modified field: first_name, middle_name, last_name, email, address, city, state or zip.

Expected Results: Throws InvalidUser exception.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.

Initialization: None

Test Steps:

1. The string “101” is passed to the API as the user.
2. The string “103” is passed to the API as the student to retrieve.
3. Pass in the modified data.
4. An InvalidUserException is thrown with getMessage() equal to “Permissions are not sufficient to edit Student <ID>.”
5. Examine the record to ensure it was not modified.

C.30 Edit Student Using Invalid Administrator ID

Test Inputs:

1. Target: The ID number of a student from good_students.json.
2. User: An ID number of an administrator from a different college.
3. At least one modified field.

Expected Results: Throws InvalidUser exception.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.

Initialization: None

Test Steps:

1. The string “A0001” is passed to the API as the user.
2. The string “103” is passed to the API as the student to retrieve.
3. Pass in the modified data.
4. An InvalidUserException is thrown with getMessage() equal to “Permissions are not sufficient to edit Student <ID>”
5. Examine the record to ensure it was not modified.

C.31 Edit Student fails because student doesn't have access to that field

Test Inputs:

1. User: The ID number of a student from good_students.json.
2. Target: The same ID number as above
3. At least one modified field not from the following: first_name, middle_name, last_name, email, address, city, state or zip.

Expected Results: Throws InvalidPermissions Exception

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_students.json has been loaded into the API.

Initialization: None

Test Steps:

1. The string "101" is passed to the API as the user ID and the target ID to retrieve a student.
2. Pass in the modified details.
3. An InvalidPermissionsException is thrown with getMessage() equal to "Permissions are not sufficient to edit this field"
4. Examine the record to ensure it was not modified.

C.32 Edit Student with failed save to the file

Test Inputs:

1. User: The ID number of student from good_admins.json.
2. Target: The same ID number as above
3. At least one modified field: first_name, middle_name, last_name, email, address, city, state or zip.

Expected Results: Throws a IOException.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_nowrite_admins.json has been loaded into the API
3. The file good_admin.json has been loaded into the API

Initialization: Ensure permissions on good_nowrite_students.json are such that the file cannot be written to.

Test Steps:

1. Pass in a string containing a administrator ID that is in the system (example: "A0002") and the modified elements ("New_email").
2. Check to ensure the exception is thrown
3. Examine the record to ensure it was not modified.

C.33 Edit Student Using Correct Graduate Administrator ID

Test Inputs:

1. Target: The ID number of a graduatestudent from good_students.json.
2. User: An Admin ID from the same college.
3. At least one modified field except ID

Expected Results: Return the modified record for the chosen student.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.

Initialization: None

Test Steps:

1. The string “102” is passed to the API as the target ID to retrieve a student.
2. The string “A0003” is passed to the API as the user ID.
3. Pass in the modified data.
4. The details for the student with id “101” are returned with the details per the file above but with the modified data.

C.34 Edit Student Using Invalid Graduate Administrator ID

Test Inputs:

1. Target: The ID number of a student from good_students.json.
2. User: An ID number of an administrator from a different college.
3. At least one modified field.

Expected Results: Throws InvalidUser exception.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.

Initialization: None

Test Steps:

1. The string “A0001” is passed to the API as the user.
2. The string “102” is passed to the API as the student to retrieve.
3. Pass in the modified data.
4. An InvalidUserException is thrown with getMessage() equal to “Permissions are not sufficient to edit Student <ID>”
5. Examine the record to ensure it was not modified.

Fee Test Cases (D)

Tests Requirements from 3.4

good_fees.json:

```
{
  "under_grad" : {
    "resident" : {
       "base" : 494.25,
       "over" : 80.0
    },
    "nonresident" : {
       "base" : 1331.75,
       "over" : 208.00
    },
    "technology" : {
      "full" : 200.00,
      "part" : 17.00
    },
    "health" : {
      "insurance" : 2020.00,
      "part" : 123.00
    },
    "enrichment" : 250
  },
  "grad" : {
    "resident" : {
       "base" : 552.25,
       "over" : 80.0
    },
    "nonresident" : {
       "base" : 1182.00,
       "over" : 170.00
    },
    "health" : {
      "insurance" : 2020.00,
      "part1" : 123.00,
      "part2" : 184.00,
      "part_ga" : 184.00
    },
    "application" : 50,
    "app_readmit" : 15,
```

```
    "status_change" : 15,
    "bench_fees" : 3750
  },
  "college" : {
    "1" : {
      "apogee" : {
        "part" : 220
      },
      "program" : {
        "full" : 918,
        "part" : 76.5
      },
      "mhit" : {
        "full" : 900,
        "part" : 75
      },
      "lab_CSCE_101_102" : {
        "full" : 148
      },
      "exec_master" : {
        "part" : 412
      },
      "sys_design" : {
        "part" : 292
      }
    },
    "2" : {
      "language" : 130,
      "art_education_lab" : 80,
      "art_history_lab" : 80,
      "dance_lab" : 150,
      "media_lab" : 200,
      "sudio_lab" : 200,
      "marine_sci" : 300,
      "math_lab" : 210,
      "field" : 75,
      "high_school_drama" : 2500
    }
  }
}
```

activity.json:


```
{
  "account_activity": [
      {
      "activity_id": "1",
      "user_id": "A0001",
      "payment_date": "2017-10-01",
      "amount": "200",
      "type": "debit",
      "description": "technology fee",
      "account_id": "101",
      "term": "FA2017"
      },
      {
      "activity_id": "2",
      "user_id": "A0001",
      "payment_date": "2017-10-01",
      "amount": "250",
      "type": "debit",
      "description": "enrichment",
      "account_id": "101",
      "term": "FA2017"
      },
      {
      "activity_id": "3",
      "user_id": "101",
      "payment_date": "2017-10-03",
      "amount": "100",
      "type": "credit",
      "description": "payment",
      "account_id": "101",
      "term": "FA2017"
      }
  ]
}
```

Traceability Matrix - Shared

Requirement \ Test Case	3.5.2.1	3.5.2.2	3.5.2.3	3.5.2.4	3.5.2.5	3.5.2.6
S.1	X					
S.2	X					
S.3		X				
S.4		X				
S.5			X			
S.6			X			
S.7				X		
S.8				X		
S.9					X	
S.10					X	
S.11						X
S.12						X

S.1 Student Can View Their Payment History

Test Inputs:

1. User: The ID number of a student from good_students.json.
2. Account: The same ID as above.
3. Term: The term's being queried.

Expected Results: A List of all the payments for the given account and term.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file good_fees.json has been loaded into the API.
5. The file activity.json is accessible by the API.

Initialization: Some fees have been added to the account.

Test Steps:

1. The string "101" is passed to the API as the user.
2. The string "101" is passed to the API as the account.
3. The string "SP18" is passed to the API as the term.
4. Validate that all of the payments for student "101" and term "SP18" are listed as above.

S.2 Student Cannot View Others Payment History

Test Inputs:

1. User: The ID number of a student from good_students.json.
2. Account: The same ID as above.
3. Term: The term's being queried.

Expected Results: Throws access denied exception

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file good_fees.json has been loaded into the API.
5. The file activity.json is accessible by the API.

Initialization: Some fees have been added to the account.

Test Steps:

1. The string "101" is passed to the API as the user.
2. The string "102" is passed to the API as the account.
3. The string "SP18" is passed to the API as the term.

S.3 Administrator Can View Payment History of Student in Their College

Test Inputs:

1. User: The ID number of a admin from good_admins.json.
2. Account: The ID number of a student from good_students.json who is in the same college as the admin.
3. Term: The term's being queried.

Expected Results: A List of all the payments for the given account and term.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file good_fees.json has been loaded into the API.
5. The file activity.json is accessible by the API.

Initialization: Some fees have been added to the account.

Test Steps:

1. The string "501" is passed to the API as the administrator.
2. The string "101" is passed to the API as the account.
3. The string "SP18" is passed to the API as the term.
4. Validate that all of the payments for student "101" and term "SP18" are listed as above.

S.4 Administrator Cannot View Payment History of Student NOT in their College

Test Inputs:

1. User: The ID number of an admin from good_admins.json.
2. Account: The ID number of a student from good_students.json who is NOT in the same college as the admin.
3. Term: The term's being queried.

Expected Results: Throws access denied exception

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file good_fees.json has been loaded into the API.
5. The file activity.json is accessible by the API.

Initialization: Some fees have been added to the account.

Test Steps:

1. The string "501" is passed to the API as the user.
2. The string "102" is passed to the API as the account.
3. The string "SP18" is passed to the API as the term.

S.5 Graduate Administrator Can View Payment History of Any Graduate Student

Test Inputs:

1. User: The ID number of a graduate admin from good_admins.json.
2. Account: The ID number of a graduate student from good_students.json.
3. Term: The term's being queried.

Expected Results: A List of all the payments for the given account and term.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file good_fees.json has been loaded into the API.
5. The file activity.json is accessible by the API.

Initialization: Some fees have been added to the account.

Test Steps:

1. The string "501" is passed to the API as the administrator.
2. The string "101" is passed to the API as the account.
3. The string "SP18" is passed to the API as the term.
4. Validate that all of the payments for student "101" and term "SP18" are listed as above.

S.6 Graduate Administrator Cannot View Payment History of Student Who is not a Graduate Student

Test Inputs:

1. User: The ID number of a graduate admin from good_admins.json.
2. Account: The ID number of a undergrad student from good_students.json who is NOT in the same college as the admin.
3. Term: The term's being queried.

Expected Results: Throws access denied exception

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file good_fees.json has been loaded into the API.
5. The file activity.json is accessible by the API.

Initialization: Some fees have been added to the account.

Test Steps:

1. The string "501" is passed to the API as the user.
2. The string "102" is passed to the API as the account.
3. The string "SP18" is passed to the API as the term.

S.7 Student Can View The Details of A Single Payment

Test Inputs:

1. User: The ID number of a student from good_students.json.
2. Payment Id.

Expected Results: A Details of the payment specified by PaymentId.

Dependencies:

1. The file good_students.json has been loaded into the API.
2. The file activity.json is accessible by the API.

Initialization: Some fees have been added to the account.

Test Steps:

1. The string “101” is passed to the API as the user.
2. The string “234” is passed to the API as the payment id.
3. Validate the payment details for student “101” and payment “234” are listed as above.

S.8 Student Cannot View The Details of A Payment The is Not Theirs

Test Inputs:

1. User: The ID number of a student from good_students.json.
2. Payment Id that does NOT belong to the student.

Expected Results: Access Denied Exception

Dependencies:

1. The file good_students.json has been loaded into the API.
2. The file activity.json is accessible by the API.

Initialization: Some fees have been added to the account.

Test Steps:

1. The string “101” is passed to the API as the user.
2. The string “234” is passed to the API as the payment id.

S.9 Administrator Can View Payment Details of Student in Their College

Test Inputs:

1. User: The ID number of a admin from good_admins.json.
2. Payment Id

Expected Results: A List of the payment details for the given student and payment id.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file activity.json is accessible by the API.

Initialization: Some fees have been added to the account.

Test Steps:

1. The string “501” is passed to the API as the administrator.
2. The string “101” is passed to the API as the payment id.
3. Validate that the payment details for payment id “101” are listed as above.

S.10 Administrator Cannot View Payment Details of Student NOT in their College

Test Inputs:

1. User: The ID number of an admin from good_admins.json.
2. Payment Id

Expected Results: Throws access denied exception

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file activity.json is accessible by the API.

Initialization: Some fees have been added to the account.

Test Steps:

4. The string “501” is passed to the API as the user.
5. The string “101” is passed to the API as the payment id.

S.11 Graduate Administrator Can View Payment Details of Any Graduate Student

Test Inputs:

1. User: The ID number of a graduate admin from good_admins.json.
2. Payment Id

Expected Results: A List of the payment details for the given payment id.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file activity.json is accessible by the API.

Initialization: Some fees have been added to the account.

Test Steps:

1. The string “501” is passed to the API as the administrator.
2. The string “101” is passed to the API as the payment id.
3. Validate that the payment details for payment id are listed as above.

S.12 Graduate Administrator Cannot View Payment History of Student Who is NOT a Graduate Student

Test Inputs:

1. User: The ID number of a graduate admin from good_admins.json.
2. Payment Id

Expected Results: Throws access denied exception

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file good_fees.json has been loaded into the API.
5. The file activity.json is accessible by the API.

Initialization: Some fees have been added to the account.

Test Steps:

1. The string “501” is passed to the API as the user.
2. The string “102” is passed to the API as the payment id.

S.13 Any user cannot view a payment that doesn't exist

Test Inputs:

1. User: The ID number of a graduate admin from good_admins.json.
2. Payment ID

Expected Results: Throws UnknownPayment exception

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file good_fees.json has been loaded into the API.
5. The file activity.json is accessible by the API.

Initialization: Some fees have been added to the account.

Test Steps:

1. The string "501" is passed to the API as the user.
2. The string "BAD" is passed to the API as the payment id.
3. Validate an UnknownPaymentException is thrown

S.14 Any user cannot view a payment history that doesn't exist

Test Inputs:

3. User: The ID number of a graduate admin from good_admins.json.
4. Target: A non-existent student ID

Expected Results: Throws UnknownStudent exception

Dependencies:

6. The file good_colleges.json has been loaded into the API.
7. The file good_admins.json has been loaded into the API.
8. The file good_students.json has been loaded into the API.
9. The file good_fees.json has been loaded into the API.
10. The file activity.json is accessible by the API.

Initialization: Some fees have been added to the account.

Test Steps:

4. The string "501" is passed to the API as the user.
5. The string "BAD" is passed to the API as the payment id.
6. Validate an UnknownStudentException is thrown

S.15 Read Valid Activity JSON Input File From Correct Path And Filename

Test Inputs: A string containing a path to and filename of a non-empty valid JSON text file containing activity made on the accounts including fees and payments. See activity above.
Expected Results: A list of administrators read in from the file.
Dependencies: None
Initialization: None
Test Steps:

1. The string “good_admins.json” is passed to the API.
2. Check to make sure that an exception was not thrown.
3. Display the list of payment activity from the API for a student with activity.

S.16 Read Activity JSON Input File From Incorrect Path And Filename

Test Inputs: A string that does not reference any file or path in the test system.
Expected Results: Throws a FileNotFoundException.
Dependencies: None
Initialization: None
Test Steps:

1. The string “/this/is/not/a/file” is passed to the API.
2. A FileNotFoundException is caught.

S.17 Read Empty Activity JSON Input File From Correct Path And Filename

Test Inputs: A string containing a path to an empty file.
Expected Results: A new activity file is opened..
Dependencies: None
Initialization: None
Test Steps:

1. The string “empty_admins.json” is passed to the API.
2. There should not be activity for a given student.

S.18 Read Invalid Activity JSON Input File From Correct Path And Filename

Test Inputs: A string containing a path to and filename of a non-empty JSON text file containing poorly formed JSON. See corrupt_activity.json above.
Expected Results: Throws Exception. To avoid forced inclusion of JSON packages, API rethrows JsonParsingException to Exception.
Dependencies: None
Initialization: None
Test Steps:

3. The string “corrupt_admins.json” is passed to the API.
4. An exception is caught with getMessage() equal to “The file corrupt_admins.json is corrupt.”

Traceability Matrix - D

Requirement \ Test Case	3.4.1.1	3.4.2.1	3.4.3.1	3.4.3.2	3.4.3.3
D.1	X				
D.2	X				
D.3	X				
D.4	X				
D.5		X			
D.6		X			
D.7			X		
D.8			X		
D.9				X	
D.10				X	
D.11				X	
D.12					X
D.13					X
D.14					X
D.15					X
D.16					X

D.1 Read Valid Fees JSON Input File From Correct Path And Filename

Test Inputs: A string containing a path to and filename of a non-empty valid JSON text file containing administrators supported by the API. See good_fees.json above.
Expected Results: A list of administrators read in from the file.
Dependencies: None
Initialization: User logged in as admin.
Test Steps:

1. The string “good_fee.json” is passed to the API.
2. List of fees returned from the API.
3. The list returned by the API matches the file as described above.

D.2 Read JSON Input File From Incorrect Path And Filename

Test Inputs: A string that does not reference any file or path in the test system.
Expected Results: Throws a FileNotFoundException.
Dependencies: None
Initialization: User is admin.
Test Steps:

1. The string “/this/is/not/a/file” is passed to the API.
2. A FileNotFoundException is caught.

D.3 Read Empty JSON Input File From Correct Path And Filename

Test Inputs: A string containing a path to an empty file.
Expected Results: Throws a IOException.
Dependencies: None
Initialization: User is admin.
Test Steps:

1. The string “empty_fees.json” is passed to the API.
2. A FileNotFoundException is caught with getMessage() equal to “The file empty_fees.json” is empty.

D.4 Read Valid Fees JSON Input File From Correct Path And Filename For Non-Existent College

Test Inputs: A string: a path to and filename of a non-empty valid JSON text file containing the terms available for each college. The file should contain the text: {"college":{"NO":{"fee":130}}}

Expected Results: A list of courses for each college.

Dependencies: None

Initialization: User is admin. Colleges from good_colleges.json successfully read into system.

Test Steps:

1. The string “missing_college_fees.json” is passed to the API.
2. An exception is caught with getMessage() equal to “The file missing_college_fees.json references college id NO. Ensure this college exits in your input file and reload colleges”.

D.5 Retrieve Fee With Class, Category, and Name

Test Inputs: "under_grad” "resident" and "base"

Expected Results: The floating point value 494.25.

Dependencies: None

Initialization: User is admin. Fees from good_fees.json successfully read into system.

Test Steps:

1. The three inputs are passed to the API.
2. The value returned is compared to 494.25.

D.6 Retrieve Missing Fee With Class, Category, and Name

Test Inputs: "under_grad” "resident" and "missing"

Expected Results: IllegalArgumentException is thrown.

Dependencies: None

Initialization: User is admin. Fees from good_fees.json successfully read into system.

Test Steps:

1. The three inputs are passed to the API.
2. An IllegalArgumentException is thrown with message, “There is no fee for under_grad, resident, missing. Update the file good_fees.json and reload file.

D.7 Assign Student Profile to College For Term

Test Inputs: Student id “A0001”, college_id, term
Expected Results: A set of new activities for each fee is added to file activity.json.
Dependencies: None
Initialization:

1. User is admin.
2. Fees from good_fees.json successfully read into system.
3. Users have been successfully read into system.
4. Student profile successfully retrieved

Test Steps:

1. Student data is used to select correct fees for a college_id, and term.
2. A new activity is added to activity.json for the student.

D.8 Assign Student Profile to College For Duplicate Term

Test Inputs: Student id “A0001”, college_id, term
Expected Results: IllegalArgumentException is thrown.
Dependencies: None
Initialization:

1. User is admin.
2. Fees from good_fees.json successfully read into system.
3. Users have been successfully read into system.
4. Student profile successfully retrieved.

Test Steps:

1. Student data is used to select correct fees for a college_id, and term.
2. A new activity is added to activity.json for the student.
3. Student data is used to select correct fees for a college_id, and term as in step 1.
4. An IllegalArgumentException is thrown with error “Student student_id already has an assigned term for term_id.”

D.9 Unsuccessful Save After Successfully Adding Term to Student Profile

Test Inputs: Student id, college_id, term

Expected Results: IOException is thrown.
Dependencies: None
Initialization:

1. User is admin.
2. Fees from good_fees.json successfully read into system.
3. Users have been successfully read into system.
4. Student profile successfully retrieved.

Test Steps:

1. Student data is used to select correct fees for a college_id, and term.
2. File activity.json is set to read-only.
3. A new activity is added to activity.json for the student.
4. IOException is thrown with message “Unable to update file activity.json with this term. Please check that file and try again.”

D.10 Assign Student Profile to College For Term With Incorrect Student Id

Test Inputs: Student id “Z0013”, college_id, term
Expected Results: An IllegalArgumentException.
Dependencies: None
Initialization:

1. User is admin.
2. Fees from good_fees.json successfully read into system.
3. Users have been successfully read into system.
4. Student profile successfully retrieved

Test Steps:

1. Student data is used to select correct fees for a college_id, and term.
2. An IllegalArgumentException is caught with message “There is no student with id Z0013. Please verify and reload your student file and try again.

D.11 Assign Fee To Single Student

Test Inputs: Student id “A0001”, college_id, fee term, fee class, fee category, and fee name

Expected Results: File activity.json updated to include fee for A0001.

Dependencies: None

Initialization:

1. User is admin.
2. Fees from good_fees.json successfully read into system.
3. Users have been successfully read into system.

Test Steps:

1. Student id A0001 and the fee details are passed to API.
2. File activity.json updated to reflect step 1.

D.12 Assign Fee To Single Student Not Found

Test Inputs: Student id “Z0013”, college_id, SP18, undergrad, resident, and fee name

Expected Results: IllegalArgumentException thrown.

Dependencies: None

Initialization:

1. User is admin.
2. Fees from good_fees.json successfully read into system.
3. Users have been successfully read into system.

Test Steps:

1. Student id Z0013 and the fee details are passed to API.
2. An IllegalArgumentException is caught with message “There is no student with id Z0013. Please verify and reload your student file and try again.

D.13 Assign Fee To Single Student Fee Not Found

Test Inputs: Student id “Z0013”, college_id “1”, fee term “SP18, fee class college_id, fee category “apogee”, and fee name “not_part”

Expected Results: IllegalArgumentException thrown.

Dependencies: None

Initialization:

1. User is admin.
2. Fees from good_fees.json successfully read into system.
3. Users have been successfully read into system.

Test Steps:

1. Student id Z0013 and the fee details are passed to API.
2. An IllegalArgumentException is caught with message “There is no fee for college 1, term SP18, category apogee, name not_part. Please verify and reload your fee file and try again.”

D.14 Assign Fee To All Students

Test Inputs: Fee term “SP18, fee class college_id, fee category “undergrad”, and fee name “enrichment”

Expected Results: File activity.json updated to include fee for all students.

Dependencies: None

Initialization:

1. User is admin.
2. Terms have been successfully read into system.
3. Fees from good_fees.json successfully read into system.
4. Users have been successfully read into system.

Test Steps:

1. Fee details are passed to API.
2. All students receive new activity.
3. File activity.json updated to reflect step 1.

D.15 Assign Fee To All Students Incorrect Fee

Test Inputs: Fee term “SP18, fee class college_id, fee category “undergrad”, and fee name “renrichment”

Expected Results: IllegalArgumentException thrown

Dependencies: None

Initialization:

1. User is admin.
2. Terms have been successfully read into system.
3. Fees from good_fees.json successfully read into system.
4. Users have been successfully read into system.

Test Steps:

4. Fee details are passed to API.
5. An IllegalArgumentException is caught with message “There is no fee for term SP18, category undergrad, name renrichment. Please verify and reload your fee file and try again.”

D.16 Assign Fee To All Students File Cannot Be Saved

Test Inputs: Fee category “undergrad”, and fee name “enrichment”

Expected Results: File activity.json updated to include fee for all students.

Dependencies: None

Initialization:

1. User is admin.
2. Terms have been successfully read into system.
3. Fees from good_fees.json successfully read into system.
4. Users have been successfully read into system.

Test Steps:

1. File activity.json is set to read-only.
2. Fee details are passed to API.
3. IOException is thrown with message “Unable to update file activity.json. Please check that file and try again.”

Payment Test Cases (E)

Tests Requirements from 3.5

Traceability Matrix - E

Requirement \ Test Case	3.5.1.1	3.5.1.2	3.5.1.3	3.5.3.1	3.5.3.2
E.1	X				
E.2	X				
E.3	X				
E.4	X				
E.5	X				
E.6	X				
E.7		X			
E.8		X			
E.9		X			
E.10		X			
E.11		X			
E.12		X			
E.13			X		
E.14			X		
E.15			X		
E.16			X		
E.17			X		
E.18			X		
E.19				X	
E.20				X	

E.21				X	
E.22				X	
E.23				X	
E.24				X	
E.25					X
E.26					X
E.27					X
E.28					X
E.29					X
E.30					X

E.1 Create Payment using Student ID

Test Inputs:

1. User: The ID number of a student from good_students.json.
2. Account: The same ID as above.
3. Amount: The amount to pay
4. Term: The term's account to pay

Expected Results: The new payment record.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file good_fees.json has been loaded into the API.
5. The file activity.json is accessible by the API.

Initialization: Some fees have been added to the account.

Test Steps:

1. The string "101" is passed to the API as the user_id.
2. The string "101" is passed to the API as the account_id.
3. The string "100" is passed to the API as the amount.
4. The string "FA17" is passed to the API as the term.
5. The string "credit" is passed to the API as the type.
6. The string "payment" is passed to the API as the description
7. Run test Case S6 (Get Account History) and validate that the payment has been added with the correct details.

E.2 Create Payment using a Student's account that's not found.

Test Inputs:

1. User: The ID number of a student from good_students.json.
2. Account: an invalid/not found ID.
3. Amount: The amount to pay
4. Term: The term's account to pay

Expected Results: Throws UnknownUser exception.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file good_fees.json has been loaded into the API.
5. The file activity.json is accessible by the API.

Initialization: Some fees have been added to the account.

Test Steps:

1. The string "101" is passed to the API as the user_id.
2. The string "BAD" is passed to the API as the account_id.
3. The string "110" is passed to the API as the amount.
4. The string "FA17" is passed to the API as the term.
5. The string "credit" is passed to the API as the type.
6. The string "payment" is passed to the API as the description.
7. An UnknownUserException is thrown with getMessage() equal to "There is no Student with ID BAD."
8. Run test Case S6 (Get Account History) and validate that the payment has not been added.

E.3 Create Payment using a Student ID's account that's not authorized.

Test Inputs:

1. User: The ID number of a student from good_students.json.
2. Account: Another student's account.
3. Amount: The amount to pay
4. Term: The term's account to pay

Expected Results: Throws InvalidUser exception.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file good_fees.json has been loaded into the API.
5. The file activity.json is accessible by the API.

Initialization: Some fees have been added to the account.

Test Steps:

1. The string "101" is passed to the API as the user_id.
2. The string "102" is passed to the API as the account_id.
3. The string "110" is passed to the API as the amount.
4. The string "FA17" is passed to the API as the term.
5. The string "credit" is passed to the API as the type.
6. The string "payment" is passed to the API as the description.
7. An InvalidUserException is thrown with getMessage() equal to "User is not authorized to make payments on account "102"."
8. Run test Case S6 (Get Account History) and validate that the payment has not been added.

E.4 Create Payment for more than the balance due.

Test Inputs:

1. User: The ID number of a student from good_students.json.
2. Account: The same ID as above.
3. Amount: The amount to pay
4. Term: The term's account to pay

Expected Results: Throws InvalidAmount exception.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file good_fees.json has been loaded into the API.
5. The file activity.json is accessible by the API.

Initialization: Some fees have been added to the account that total less than $1,000,000.00.

Test Steps:

1. The string "101" is passed to the API as the user_id.
2. The string "101" is passed to the API as the account_id.
3. The string "1,000,000.00" is passed to the API as the amount.
4. The string "FA17" is passed to the API as the term.
5. The string "credit" is passed to the API as the type.
6. The string "payment" is passed to the API as the description.
7. An InvalidAmountException is thrown with getMessage() equal to "Unable to make a payment for more than is owed."
8. Run test Case S6 (Get Account History) and validate that the payment has not been added.

E.5 Create Payment using a negative amount.

Test Inputs:

1. User: The ID number of a student from good_students.json.
2. Account: The same ID as above.
3. Amount: The amount to pay
4. Term: The term's account to pay

Expected Results: Throws InvalidAmount exception.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file good_fees.json has been loaded into the API.
5. The file activity.json is accessible by the API.

Initialization: None

Test Steps:

1. The string "101" is passed to the API as the user_id.
2. The string "101" is passed to the API as the account_id.
3. The string "-100" is passed to the API as the amount.
4. The string "FA17" is passed to the API as the term.
5. The string "credit" is passed to the API as the type.
6. The string "payment" is passed to the API as the description.
7. An InvalidAmountException is thrown with getMessage() equal to "Unable to make a payment for a negative amount."
8. Run test Case S6 (Get Account History) and validate that the payment has not been added.

E.6 Create Payment where the payment is unable to be recorded.

Test Inputs:

1. User: The ID number of a student from good_students.json.
2. Account: The same ID as above.
3. Amount: The amount to pay
4. Term: The term's account to pay

Expected Results: Throws a IOException.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file good_fees.json has been loaded into the API.
5. The file nowrite_activity.json is accessible by the API.

Initialization: Ensure permissions on nowrite_activity.json are such that the file cannot be written to.

Test Steps:

1. The string "101" is passed to the API as the user_id.
2. The string "101" is passed to the API as the account_id.
3. The string "100" is passed to the API as the amount.
4. The string "FA17" is passed to the API as the term.
5. The string "credit" is passed to the API as the type.
6. The string "payment" is passed to the API as the description.
7. An IOException is thrown with getMessage() equal to "Unable to save changes to nowrite_activity.json."
8. Run test Case S6 (Get Account History) and validate that the payment has not been added.

E.7 Create Payment by Administrator

Test Inputs:

1. User: The ID number of an administrator from good_admins.json.
2. Account: The ID number of a student from good_students.json whose college matches the admin.
3. Amount: The amount to pay
4. Term: The term's account to pay

Expected Results: The new payment record.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file good_fees.json has been loaded into the API.
5. The file activity.json is accessible by the API.

Initialization: Some fees have been added to the account.

Test Steps:

1. The string "A001" is passed to the API as the user_id.
2. The string "101" is passed to the API as the account_id.
3. The string "100" is passed to the API as the amount.
4. The string "FA17" is passed to the API as the term.
5. The string "credit" is passed to the API as the type.
6. The string "payment" is passed to the API as the description
7. Run test Case S6 (Get Account History) and validate that the payment has been added with the correct details.

E.8 Create Payment by an Administrator using a Student's account that's not found.

Test Inputs:

1. User: The ID number of an administrator from good_admins.json.
2. Account: The ID number of a non-existent student.
3. Amount: The amount to pay
4. Term: The term's account to pay

Expected Results: Throws UnknownUser exception.

Dependencies:

6. The file good_colleges.json has been loaded into the API.
7. The file good_admins.json has been loaded into the API.
8. The file good_students.json has been loaded into the API.
9. The file good_fees.json has been loaded into the API.
10. The file activity.json is accessible by the API.

Initialization: Some fees have been added to the account.

Test Steps:

9. The string "A0001" is passed to the API as the user_id.
10. The string "BAD" is passed to the API as the account_id.
11. The string "110" is passed to the API as the amount.
12. The string "FA17" is passed to the API as the term.
13. The string "credit" is passed to the API as the type.
14. The string "payment" is passed to the API as the description.
15. An UnknownUserException is thrown with getMessage() equal to "There is no Student with ID BAD."
16. Run test Case S6 (Get Account History) and validate that the payment has not been added.

E.9 Create Payment using an Administrator's account that's not authorized.

Test Inputs:

1. User: The ID number of an administrator from good_admins.json.
2. Account: The ID number of a student from good_students.json whose college does not match the admin.
3. Amount: The amount to pay
4. Term: The term's account to pay

Expected Results: Throws InvalidUser exception.

Dependencies:

6. The file good_colleges.json has been loaded into the API.
7. The file good_admins.json has been loaded into the API.
8. The file good_students.json has been loaded into the API.
9. The file good_fees.json has been loaded into the API.
10. The file activity.json is accessible by the API.

Initialization: Some fees have been added to the account.

Test Steps:

9. The string "A001" is passed to the API as the user_id.
10. The string "104" is passed to the API as the account_id.
11. The string "110" is passed to the API as the amount.
12. The string "FA17" is passed to the API as the term.
13. The string "credit" is passed to the API as the type.
14. The string "payment" is passed to the API as the description.
15. An InvalidUserException is thrown with getMessage() equal to "User is not authorized to make payments on account "102"."
16. Run test Case S6 (Get Account History) and validate that the payment has not been added.

E.10 Create Payment by an Administrator for more than the balance due.

Test Inputs:

1. User: The ID number of an administrator from good_admins.json.
2. Account: The ID number of a student from good_students.json whose college matches the admin.
3. Amount: The amount to pay
4. Term: The term's account to pay

Expected Results: Throws InvalidAmount exception.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file good_fees.json has been loaded into the API.
5. The file activity.json is accessible by the API.

Initialization: Some fees have been added to the account that total less than $1,000,000.00.

Test Steps:

1. The string "A001" is passed to the API as the user_id.
2. The string "101" is passed to the API as the account_id.
3. The string "1,000,000.00" is passed to the API as the amount.
4. The string "FA17" is passed to the API as the term.
5. The string "credit" is passed to the API as the type.
6. The string "payment" is passed to the API as the description.
7. An InvalidAmountException is thrown with getMessage() equal to "Unable to make a payment for more than is owed."
8. Run test Case S6 (Get Account History) and validate that the payment has not been added.

E.11 Create Payment by an Administrator using a negative amount.

Test Inputs:

1. User: The ID number of an administrator from good_admins.json.
2. Account: The ID number of a student from good_students.json whose college matches the admin.
3. Amount: The amount to pay
4. Term: The term's account to pay

Expected Results: Throws InvalidAmount exception.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file good_fees.json has been loaded into the API.
5. The file activity.json is accessible by the API.

Initialization: None

Test Steps:

1. The string "A001" is passed to the API as the user_id.
2. The string "101" is passed to the API as the account_id.
3. The string "-100" is passed to the API as the amount.
4. The string "FA17" is passed to the API as the term.
5. The string "credit" is passed to the API as the type.
6. The string "payment" is passed to the API as the description.
7. An InvalidAmountException is thrown with getMessage() equal to "Unable to make a payment for a negative amount."
8. Run test Case S6 (Get Account History) and validate that the payment has not been added.

E.12 Create Payment by an Administrator where the payment is unable to be recorded.

Test Inputs:

1. User: The ID number of an administrator from good_admins.json.
2. Account: The ID number of a student from good_students.json whose college matches the admin.
3. Amount: The amount to pay
4. Term: The term's account to pay

Expected Results: Throws a IOException.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file good_fees.json has been loaded into the API.
5. The file nowrite_activity.json is accessible by the API.

Initialization: Ensure permissions on nowrite_activity.json are such that the file cannot be written to.

Test Steps:

1. The string "A001" is passed to the API as the user_id.
2. The string "101" is passed to the API as the account_id.
3. The string "100" is passed to the API as the amount.
4. The string "FA17" is passed to the API as the term.
5. The string "credit" is passed to the API as the type.
6. The string "payment" is passed to the API as the description.
7. An IOException is thrown with getMessage() equal to "Unable to save changes to nowrite_activity.json."
8. Run test Case S6 (Get Account History) and validate that the payment has not been added.

E.13 Create Payment by Graduate Administrator

Test Inputs:

1. User: The ID number of an administrator from good_admins.json.
2. Account: The ID number of a graduate student from good_students.json whose college matches the admin.
3. Amount: The amount to pay
4. Term: The term's account to pay

Expected Results: The new payment record.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file good_fees.json has been loaded into the API.
5. The file activity.json is accessible by the API.

Initialization: Some fees have been added to the account.

Test Steps:

1. The string "A003" is passed to the API as the user_id.
2. The string "102" is passed to the API as the account_id.
3. The string "100" is passed to the API as the amount.
4. The string "FA17" is passed to the API as the term.
5. The string "credit" is passed to the API as the type.
6. The string "payment" is passed to the API as the description
7. Run test Case S6 (Get Account History) and validate that the payment has been added with the correct details.

E.14 Create Payment by a Graduate Administrator using a Student's account that's not found.

Test Inputs:

1. User: The ID number of a graduate administrator from good_admins.json.
2. Account: The ID number of a non-existent student.
3. Amount: The amount to pay
4. Term: The term's account to pay

Expected Results: Throws UnknownUser exception.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file good_fees.json has been loaded into the API.
5. The file activity.json is accessible by the API.

Initialization: Some fees have been added to the account.

Test Steps:

1. The string "A0003" is passed to the API as the user_id.
2. The string "BAD" is passed to the API as the account_id.
3. The string "110" is passed to the API as the amount.
4. The string "FA17" is passed to the API as the term.
5. The string "credit" is passed to the API as the type.
6. The string "payment" is passed to the API as the description.
7. An UnknownUserException is thrown with getMessage() equal to "There is no Student with ID BAD."
8. Run test Case S6 (Get Account History) and validate that the payment has not been added.

E.15 Create Payment using a Graduate Administrator's account that's not authorized.

Test Inputs:

1. User: The ID number of a graduate administrator from good_admins.json.
2. Account: The ID number of a student from good_students.json whose college does not match the admin.
3. Amount: The amount to pay
4. Term: The term's account to pay

Expected Results: Throws InvalidUser exception.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file good_fees.json has been loaded into the API.
5. The file activity.json is accessible by the API.

Initialization: Some fees have been added to the account.

Test Steps:

1. The string "A001" is passed to the API as the user_id.
2. The string "104" is passed to the API as the account_id.
3. The string "110" is passed to the API as the amount.
4. The string "FA17" is passed to the API as the term.
5. The string "credit" is passed to the API as the type.
6. The string "payment" is passed to the API as the description.
7. An InvalidUserException is thrown with getMessage() equal to "User is not authorized to make payments on account "102"."
8. Run test Case S6 (Get Account History) and validate that the payment has not been added.

E.16 Create Payment by a Graduate Administrator for more than the balance due.

Test Inputs:

1. User: The ID number of a graduate administrator from good_admins.json.
2. Account: The ID number of a student from good_students.json whose college matches the admin.
3. Amount: The amount to pay
4. Term: The term's account to pay

Expected Results: Throws InvalidAmount exception.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file good_fees.json has been loaded into the API.
5. The file activity.json is accessible by the API.

Initialization: Some fees have been added to the account that total less than $1,000,000.00.

Test Steps:

1. The string "A003" is passed to the API as the user_id.
2. The string "101" is passed to the API as the account_id.
3. The string "1,000,000.00" is passed to the API as the amount.
4. The string "FA17" is passed to the API as the term.
5. The string "credit" is passed to the API as the type.
6. The string "payment" is passed to the API as the description.
7. An InvalidAmountException is thrown with getMessage() equal to "Unable to make a payment for more than is owed."
8. Run test Case S6 (Get Account History) and validate that the payment has not been added.

E.17 Create Payment by a Graduate Administrator using a negative amount.

Test Inputs:

1. User: The ID number of a graduate administrator from good_admins.json.
2. Account: The ID number of a student from good_students.json whose college matches the admin.
3. Amount: The amount to pay
4. Term: The term's account to pay

Expected Results: Throws InvalidAmount exception.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file good_fees.json has been loaded into the API.
5. The file activity.json is accessible by the API.

Initialization: None

Test Steps:

1. The string "A003" is passed to the API as the user_id.
2. The string "101" is passed to the API as the account_id.
3. The string "-100" is passed to the API as the amount.
4. The string "FA17" is passed to the API as the term.
5. The string "credit" is passed to the API as the type.
6. The string "payment" is passed to the API as the description.
7. An InvalidAmountException is thrown with getMessage() equal to "Unable to make a payment for a negative amount."
8. Run test Case S6 (Get Account History) and validate that the payment has not been added.

E.18 Create Payment by a Graduate Administrator where the payment is unable to be recorded.

Test Inputs:

1. User: The ID number of a graduate administrator from good_admins.json.
2. Account: The ID number of a graduate student from good_students.json whose college matches the admin.
3. Amount: The amount to pay
4. Term: The term's account to pay

Expected Results: Throws a IOException.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file good_fees.json has been loaded into the API.
5. The file nowrite_activity.json is accessible by the API.

Initialization: Ensure permissions on nowrite_activity.json are such that the file cannot be written to.

Test Steps:

1. The string "A003" is passed to the API as the user_id.
2. The string "101" is passed to the API as the account_id.
3. The string "100" is passed to the API as the amount.
4. The string "FA17" is passed to the API as the term.
5. The string "credit" is passed to the API as the type.
6. The string "payment" is passed to the API as the description.
7. An IOException is thrown with getMessage() equal to "Unable to save changes to nowrite_activity.json."
8. Run test Case S6 (Get Account History) and validate that the payment has not been added.

E.19 Edit Payment by Administrator

Test Inputs:

1. User: The ID number of an administrator from good_admins.json.
2. Activity ID: The ID of the original payment.
3. Amount: The amount to pay
4. Term: The term's account to pay

Expected Results: The new payment record.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file good_fees.json has been loaded into the API.
5. The file activity.json is accessible by the API.

Initialization: Some fees have been added to the account.

Test Steps:

1. The string "A001" is passed to the API as the user_id.
2. The string "1" is passed to the API as the activity_id.
3. The string "100" is passed to the API as the amount.
4. The string "FA17" is passed to the API as the term.
5. The string "credit" is passed to the API as the type.
6. The string "payment" is passed to the API as the description
7. Run test Case S6 (Get Account History) and validate that the payment has been added with the correct details.

E.20 Edit Payment by an Administrator using a Payment ID that's not found.

Test Inputs:

1. User: The ID number of an administrator from good_admins.json.
2. Activity ID: The ID of the original payment.
3. Amount: The amount to pay
4. Term: The term's account to pay

Expected Results: Throws UnknownUser exception.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file good_fees.json has been loaded into the API.
5. The file activity.json is accessible by the API.

Initialization: Some fees have been added to the account.

Test Steps:

1. The string "A0001" is passed to the API as the user_id.
2. The string "BAD" is passed to the API as the account_id.
3. The string "110" is passed to the API as the amount.
4. The string "FA17" is passed to the API as the term.
5. The string "credit" is passed to the API as the type.
6. The string "payment" is passed to the API as the description.
7. An UnknownUserException is thrown with getMessage() equal to "There is no Student with ID BAD."
8. Run test Case S6 (Get Account History) and validate that the payment has not been added.

E.21 Edit Payment using an Administrator's account that's not authorized.

Test Inputs:

1. User: The ID number of an administrator from good_admins.json.
2. Activity ID: The ID of the original payment.
3. Amount: The amount to pay
4. Term: The term's account to pay

Expected Results: Throws InvalidUser exception.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file good_fees.json has been loaded into the API.
5. The file activity.json is accessible by the API.

Initialization: Some fees have been added to the account.

Test Steps:

1. The string "A002" is passed to the API as the user_id.
2. The string "1" is passed to the API as the activity_id.
3. The string "110" is passed to the API as the amount.
4. The string "FA17" is passed to the API as the term.
5. The string "credit" is passed to the API as the type.
6. The string "payment" is passed to the API as the description.
7. An InvalidUserException is thrown with getMessage() equal to "User is not authorized to make payments on account "102"."
8. Run test Case S6 (Get Account History) and validate that the payment has not been added.

E.22 Edit Payment by an Administrator for more than the balance due.

Test Inputs:

1. User: The ID number of an administrator from good_admins.json.
2. Activity ID: The ID of the original payment.
3. Amount: The amount to pay
4. Term: The term's account to pay

Expected Results: Throws InvalidAmount exception.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file good_fees.json has been loaded into the API.
5. The file activity.json is accessible by the API.

Initialization: Some fees have been added to the account that total less than $1,000,000.00.

Test Steps:

1. The string "A001" is passed to the API as the user_id.
2. The string "1" is passed to the API as the activity_id.
3. The string "1,000,000.00" is passed to the API as the amount.
4. The string "FA17" is passed to the API as the term.
5. The string "credit" is passed to the API as the type.
6. The string "payment" is passed to the API as the description.
7. An InvalidAmountException is thrown with getMessage() equal to "Unable to make a payment for more than is owed."
8. Run test Case S6 (Get Account History) and validate that the payment has not been added.

E.23 Edit Payment by an Administrator using a negative amount.

Test Inputs:

1. User: The ID number of an administrator from good_admins.json.
2. Activity ID: The ID of the original payment.
3. Amount: The amount to pay
4. Term: The term's account to pay

Expected Results: Throws InvalidAmount exception.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file good_fees.json has been loaded into the API.
5. The file activity.json is accessible by the API.

Initialization: None

Test Steps:

1. The string "A001" is passed to the API as the user_id.
2. The string "1" is passed to the API as the activity_id.
3. The string "-100" is passed to the API as the amount.
4. The string "FA17" is passed to the API as the term.
5. The string "credit" is passed to the API as the type.
6. The string "payment" is passed to the API as the description.
7. An InvalidAmountException is thrown with getMessage() equal to "Unable to make a payment for a negative amount."
8. Run test Case S6 (Get Account History) and validate that the payment has not been added.

E.24 Edit Payment by an Administrator where the payment is unable to be recorded.

Test Inputs:

5. User: The ID number of an administrator from good_admins.json.
6. Activity ID: The ID of the original payment.
7. Amount: The amount to pay
8. Term: The term's account to pay

Expected Results: Throws a IOException.

Dependencies:

6. The file good_colleges.json has been loaded into the API.
7. The file good_admins.json has been loaded into the API.
8. The file good_students.json has been loaded into the API.
9. The file good_fees.json has been loaded into the API.
10. The file nowrite_activity.json is accessible by the API.

Initialization: Ensure permissions on nowrite_activity.json are such that the file cannot be written to.

Test Steps:

9. The string "A001" is passed to the API as the user_id.
10. The string "1" is passed to the API as the activity_id.
11. The string "100" is passed to the API as the amount.
12. The string "FA17" is passed to the API as the term.
13. The string "credit" is passed to the API as the type.
14. The string "payment" is passed to the API as the description.
15. An IOException is thrown with getMessage() equal to "Unable to save changes to nowrite_activity.json."
16. Run test Case S6 (Get Account History) and validate that the payment has not been added.

E.25 Edit Payment by Graduate Administrator

Test Inputs:

1. User: The ID number of an administrator from good_admins.json.
2. Activity ID: The ID of the original payment.
3. Amount: The amount to pay
4. Term: The term's account to pay

Expected Results: The new payment record.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file good_fees.json has been loaded into the API.
5. The file activity.json is accessible by the API.

Initialization: Some fees have been added to the account.

Test Steps:

1. The string "A003" is passed to the API as the user_id.
2. The string "1" is passed to the API as the activity_id.
3. The string "100" is passed to the API as the amount.
4. The string "FA17" is passed to the API as the term.
5. The string "credit" is passed to the API as the type.
6. The string "payment" is passed to the API as the description
7. Run test Case S6 (Get Account History) and validate that the payment has been added with the correct details.

E.26 Edit Payment by a Graduate Administrator using a Payment ID that's not found.

Test Inputs:

1. User: The ID number of a graduate administrator from good_admins.json.
2. Activity ID: A non-existent ID for the original payment.
3. Amount: The amount to pay
4. Term: The term's account to pay

Expected Results: Throws UnknownUser exception.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file good_fees.json has been loaded into the API.
5. The file activity.json is accessible by the API.

Initialization: Some fees have been added to the account.

Test Steps:

1. The string "A0003" is passed to the API as the user_id.
2. The string "BAD" is passed to the API as the account_id.
3. The string "110" is passed to the API as the amount.
4. The string "FA17" is passed to the API as the term.
5. The string "credit" is passed to the API as the type.
6. The string "payment" is passed to the API as the description.
7. An UnknownUserException is thrown with getMessage() equal to "There is no Student with ID BAD."
8. Run test Case S6 (Get Account History) and validate that the payment has not been added.

E.27 Edit Payment using a Graduate Administrator's account that's not authorized.

Test Inputs:

1. User: The ID number of a graduate administrator from good_admins.json.
2. Activity ID: The ID of the original payment.
3. Amount: The amount to pay
4. Term: The term's account to pay

Expected Results: Throws InvalidUser exception.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file good_fees.json has been loaded into the API.
5. The file activity.json is accessible by the API.

Initialization: Some fees have been added to the account.

Test Steps:

1. The string "A002" is passed to the API as the user_id.
2. The string "1" is passed to the API as the activity_id.
3. The string "110" is passed to the API as the amount.
4. The string "FA17" is passed to the API as the term.
5. The string "credit" is passed to the API as the type.
6. The string "payment" is passed to the API as the description.
7. An InvalidUserException is thrown with getMessage() equal to "User is not authorized to make payments on account "102"."
8. Run test Case S6 (Get Account History) and validate that the payment has not been added.

E.28 Edit Payment by a Graduate Administrator for more than the balance due.

Test Inputs:

1. User: The ID number of a graduate administrator from good_admins.json.
2. Activity ID: The ID of the original payment.
3. Amount: The amount to pay
4. Term: The term's account to pay

Expected Results: Throws InvalidAmount exception.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file good_fees.json has been loaded into the API.
5. The file activity.json is accessible by the API.

Initialization: Some fees have been added to the account that total less than $1,000,000.00.

Test Steps:

1. The string "A003" is passed to the API as the user_id.
2. The string "1" is passed to the API as the activity_id.
3. The string "1,000,000.00" is passed to the API as the amount.
4. The string "FA17" is passed to the API as the term.
5. The string "credit" is passed to the API as the type.
6. The string "payment" is passed to the API as the description.
7. An InvalidAmountException is thrown with getMessage() equal to "Unable to make a payment for more than is owed."
8. Run test Case S6 (Get Account History) and validate that the payment has not been added.

E.29 Edit Payment by a Graduate Administrator using a negative amount.

Test Inputs:

1. User: The ID number of a graduate administrator from good_admins.json.
2. Activity ID: The ID of the original payment.
3. Amount: The amount to pay
4. Term: The term's account to pay

Expected Results: Throws InvalidAmount exception.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file good_fees.json has been loaded into the API.
5. The file activity.json is accessible by the API.

Initialization: None

Test Steps:

1. The string "A003" is passed to the API as the user_id.
2. The string "1" is passed to the API as the activity_id.
3. The string "-100" is passed to the API as the amount.
4. The string "FA17" is passed to the API as the term.
5. The string "credit" is passed to the API as the type.
6. The string "payment" is passed to the API as the description.
7. An InvalidAmountException is thrown with getMessage() equal to "Unable to make a payment for a negative amount."
8. Run test Case S6 (Get Account History) and validate that the payment has not been added.

E.30 Edit Payment by a Graduate Administrator where the payment is unable to be recorded.

Test Inputs:

1. User: The ID number of a graduate administrator from good_admins.json.
2. Activity ID: The ID of the original payment.
3. Amount: The amount to pay
4. Term: The term's account to pay

Expected Results: Throws a IOException.

Dependencies:

1. The file good_colleges.json has been loaded into the API.
2. The file good_admins.json has been loaded into the API.
3. The file good_students.json has been loaded into the API.
4. The file good_fees.json has been loaded into the API.
5. The file nowrite_activity.json is accessible by the API.

Initialization: Ensure permissions on nowrite_activity.json are such that the file cannot be written to.

Test Steps:

1. The string "A003" is passed to the API as the user_id.
2. The string "1" is passed to the API as the activity_id.
3. The string "100" is passed to the API as the amount.
4. The string "FA17" is passed to the API as the term.
5. The string "credit" is passed to the API as the type.
6. The string "payment" is passed to the API as the description.
7. An IOException is thrown with getMessage() equal to "Unable to save changes to nowrite_activity.json."
8. Run test Case S6 (Get Account History) and validate that the payment has not been added.