50.003:
Elements of Software Construction
C3SI2: ConcurSolutionz

Group Members:
Jiang Hongbei (1005870)
Muthuvel Vimal (1005990)
Toh Hong Jing (1006056)
Chandrasekar Akash (1006228)
Kwok Keith (1006344)
Shaun Phua Wee Jin (1006345)
Jon Koo Jia Jun (1006388)
Siddharth Ganesh (1006407)

Index

Introduction

"ConcurSolutionz: Streamline Your Claims, Seamlessly!"

Students at the Singapore University of Technology and Design (SUTD) are always working on a myriad of projects, assignments and even fifth-row (Extra-curricular) activities. These activities usually require students to submit monetary claims for any items or services that were procured, allowing the school to reimburse them. It is thus inevitable that a large percentage of students would be forced to patronise the ConcurSolutionz platform that SUTD uses.

The issue many individuals may face is that purchases accumulate over a term. Creating claims is viewed by many as a tedious, slow and particularly difficult process due to the Concur System, which includes the slow loading of pages. Furthermore, some people may choose to neglect the maintenance of their claims due to this cumbersome process, choosing instead to deal with their claims at the end of the project. This approach often results in lost receipts and frustration regarding locating these receipts and navigating the Concur System's interface.

Expanding beyond the scope of SUTD, it is commonplace to find claim systems in companies, such as claiming for ones wellness funds, project claims, etc. No doubt, there will be claim submission processes that function similarly to those faced by students in SUTD. Moreover, some claim systems may utilise third-party services like Concur System, which could potentially offer a slow service.

But what if there was a solution that would allow everyone to store and update their monetary claims seamlessly, while having the capabilities to automate the submitting of claims.

ConcurSolutionz is your one-stop ultimate solution to streamline and enhance your claiming experience! Say goodbye to tedious claim creation and slow loading pages. With ConcurSolutionz, you can effortlessly manage your monetary claims, ensuring no receipt is ever lost along the way. A trusted companion, it is sure to guide you towards swift reimbursements and a journey filled with organisation and ease.

Unlock the power to seamless efficiency today with ConcurSolutionz!

Project Management

To ensure that the project is delivered on time, a Gantt chart is created to track our progress and for team members to understand their roles for the sprints.

Link to Gantt Chart:
https://docs.google.com/spreadsheets/d/1_ytsg2Dhm1KJpH1kjGOPW3PCbWpbx44IzV1gX-WNmxM/edit?usp=sharing

Requirements

In the process of developing an application aimed at streamlining the student expense claim process, the team initially gained a comprehensive understanding of the procedure involved in claim creation. Over the course of a single academic term, students frequently navigate multiple projects concurrently. The challenge of misplaced receipts leading to organisational difficulties is a prevailing concern. Given this context, the team has identified the ensuing project prerequisites as follows:

Explicit Requirements:

- Help users proactively manage their receipts and claims
- Cross-platform compatibility
- Accelerate the claiming process for students:
    - Receipt scanning and form auto-fill

Implicit Requirements:

- Help streamline any uncertainty in the process of claiming
- Ease of use for a first-time user of this product

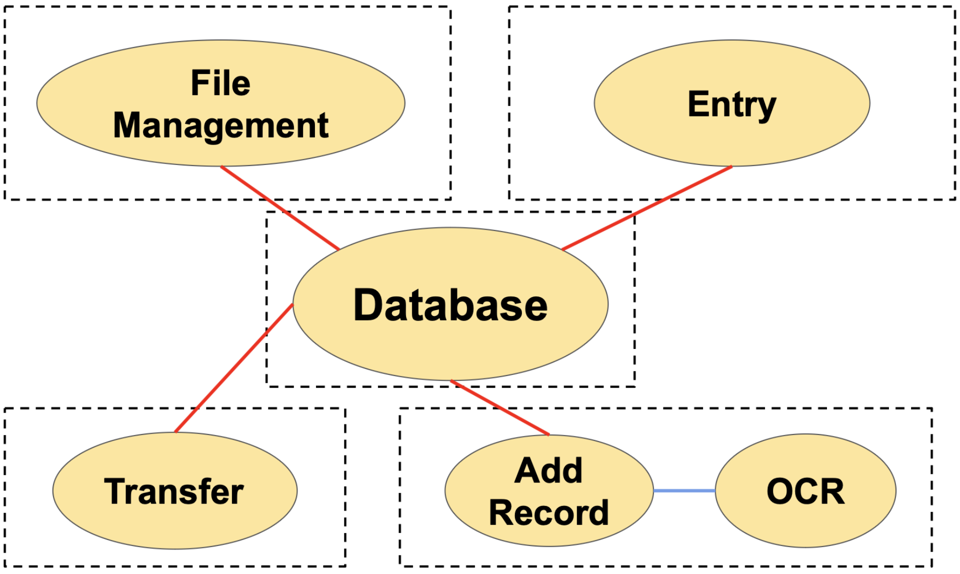


Fig. 1: Schematic of Systems involved

Derived from the subsequent requisites, the team has devised five distinct systems (refer to Fig. 1) for the system architecture of the application:

1. **Add Record System**:
   Positioned as a component that manages adding of receipt data. Together with the OCR system, this system aims to speedup the creation of receipts.

2. **Entry System:**
   Encompassing all utilities required for the conversion of receipts into coherent entries within the system. Represents a claim instance in SAP Concur.

3. **File Management System:**
   Enabling the navigation of the stored Entries in the database as well as the ability to create folders that helps users organise their Entries.

4. **Transfer System:**
   Facilitating users in the process of claim generation within the SAP Concur platform.

5. **Database:**
   Serving as the pivotal heart of the whole system for all interface interactions, ensuring seamless input and output operations. The Database would be the main communication channel for all systems to ensure maximise cohesion and minimise coupling of the systems.

Design

Programming Language

C# Programming language was chosen for this project. It was chosen because of its remarkable scalability and ease of maintenance. It encourages type safety as well as ease of integration of Object-Oriented Designs because of its strongly-typed nature. It uses a just-in-time compiler allowing it to be typically executed more quickly than Java. It also has the added advantage of being more memory efficient than Java.

Use Case Diagrams

File Management System

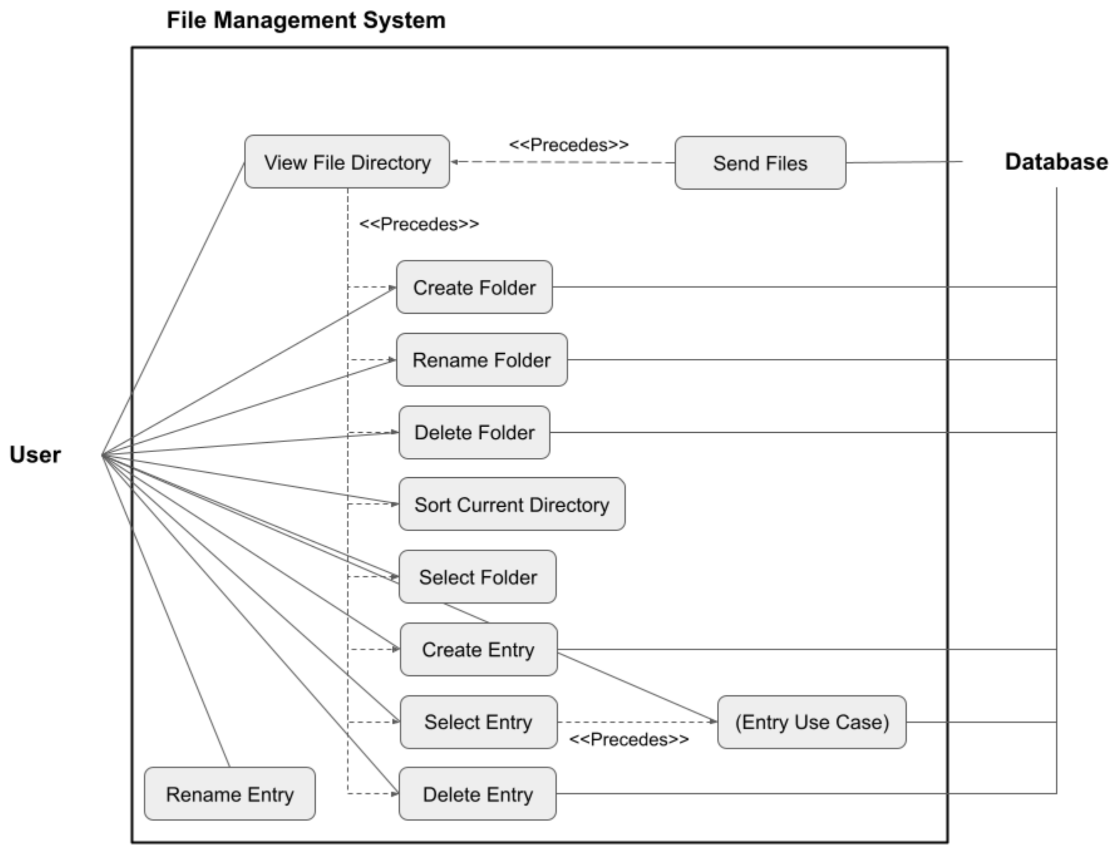


Fig. 2: File Management System Use Case

The file management system is designed to facilitate efficient organisation and manipulation of files and folders. Users can navigate through their directory structure, view and interact with both files and folders, and perform various operations such as creating new folders, adding new files, renaming existing items, and deleting items. The system offers intuitive interactions, such as single and double taps for selection and opening, and provides sorting options based on alphabetical order or creation date, designed to mimic the Finder/Explorer on the user's computer.

Entry System

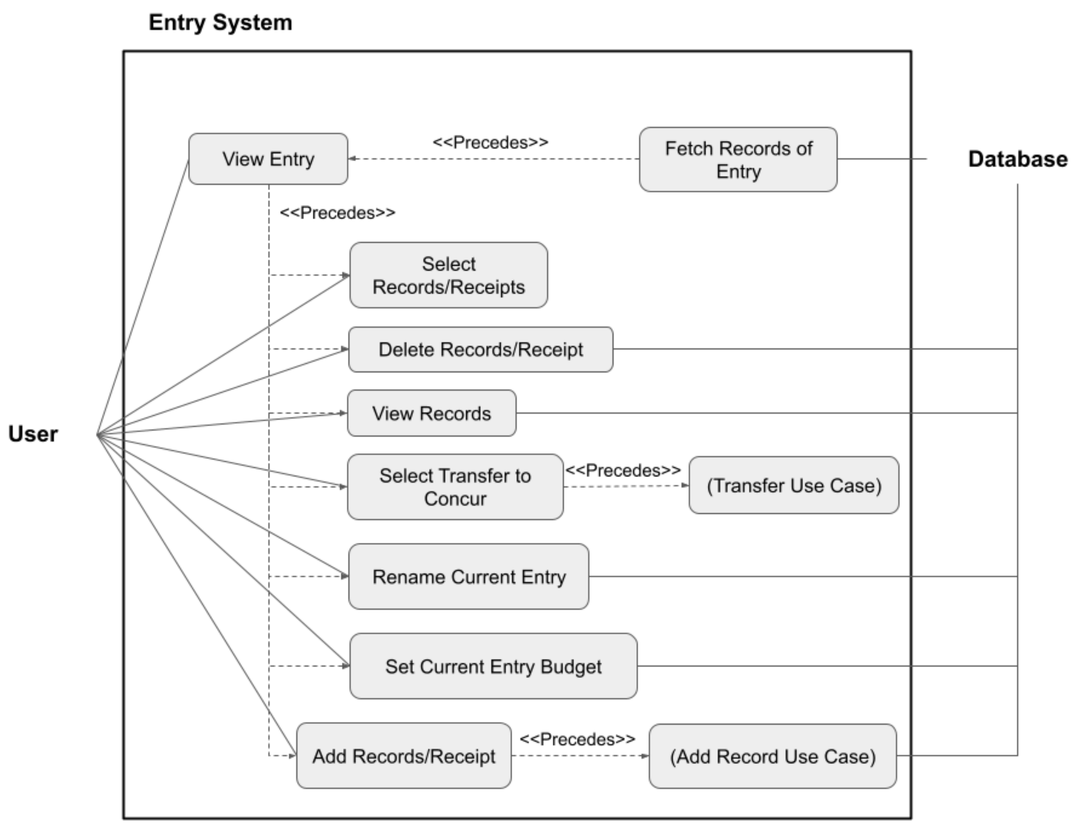


Fig. 3: Entry System Use Case

An entry is equivalent to a claim report in the Concur system, and thus the Entry system facilitates the user in filling in and updating the relevant details of the claim and eventually creating a claim in Concur.

The entry system gives the user a single-paged view of the claim, and they can easily update the details of the claim and store receipt images and details along with the entry. This allows the user to efficiently double check if all the details are correct, and create a claim in one go, as compared to the Concur site where the details are scattered throughout different displays, and the way to edit claim details can be quite obscure (by clicking on a “Report Number:XXXXX” link, which says nothing about editing the details).

The entry system also has an additional feature, which is keeping track of the budget. Each project has its own allocated budget, and students normally have to keep track of their spending manually. With the Entry system, users are required to key in the budget of the claim, and when they add a receipt, the system will automatically calculate the remaining budget left for the claim, allowing users to easily keep track of their spending.

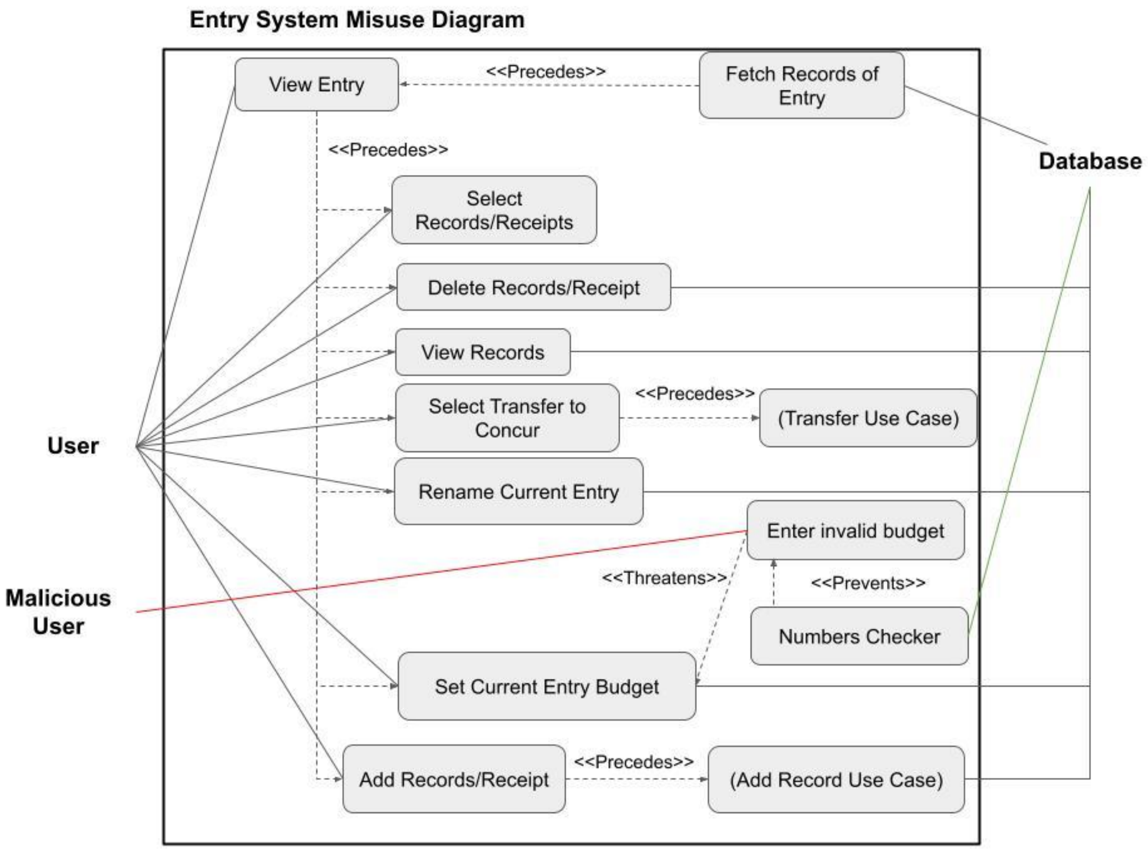


Fig. 4: Entry System Misuse Case

In this misuse scenario, a malicious user might input an invalid budget into the Entry system. An invalid budget can include non-numeric characters or symbols. To prevent this, we've incorporated a numerical validator for the budget field to ensure that only numbers are entered.

Add Record System

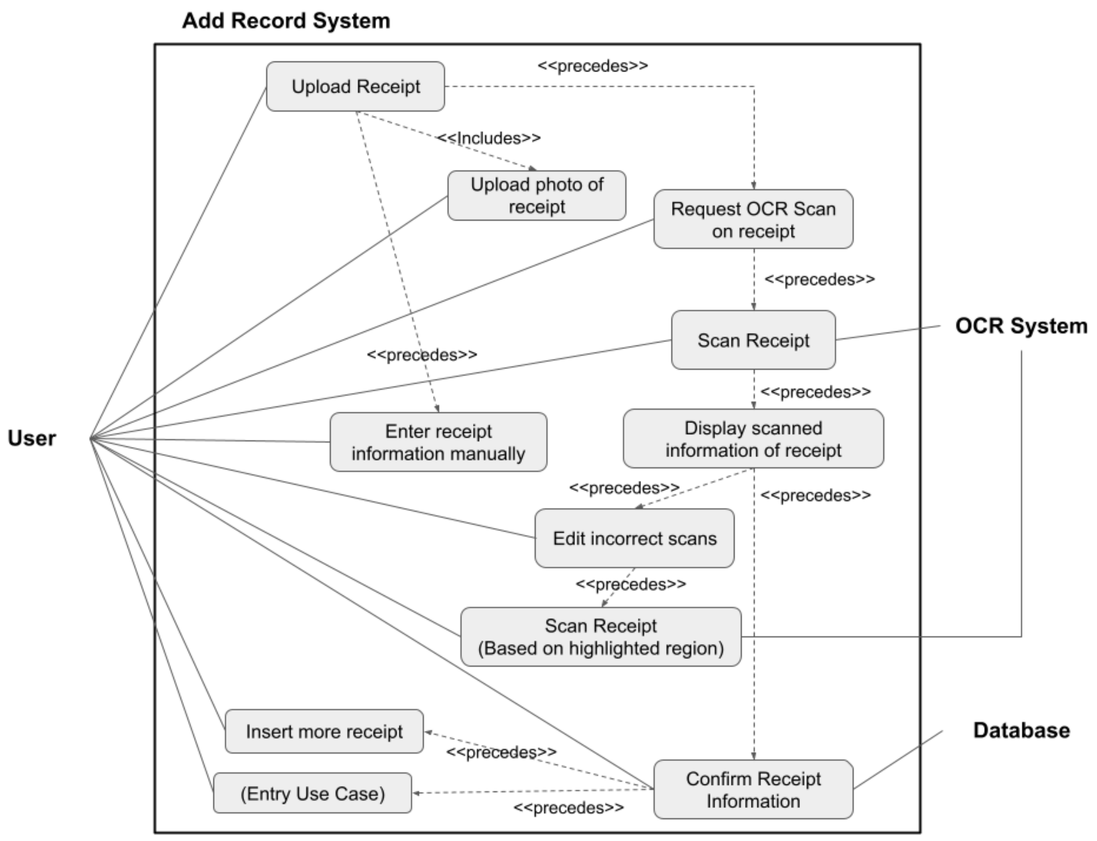


Fig. 5: Add Record System Case

The Add Record system is designed to allow the users to upload Receipts, evoking OCR scans on the receipt for auto filling and editing receipt information. The system serves as a bridge between the frontend and the backend (Database) when it comes to adding of receipts to a claim entry.

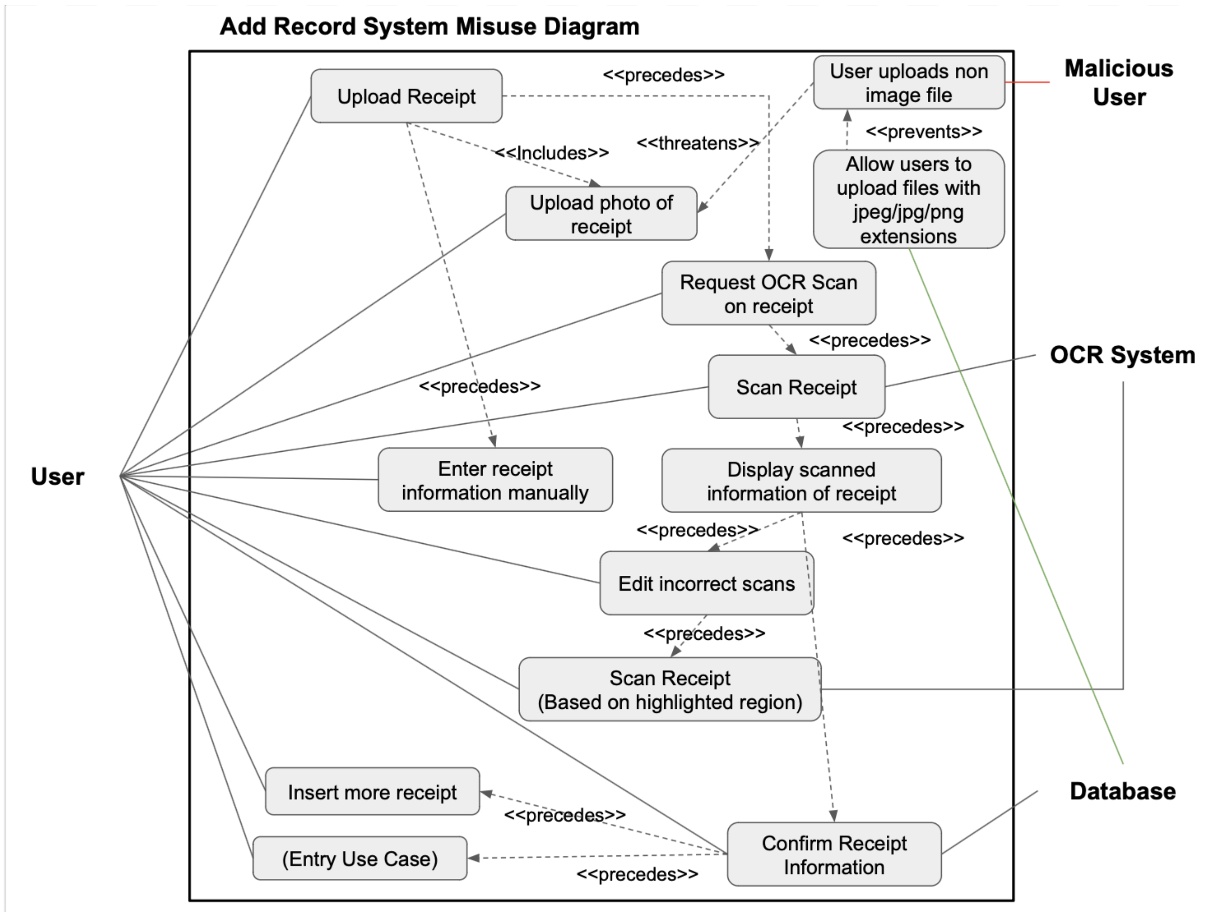


Fig. 6: Add Record System Misuse Case

In this misuse scenario, a malicious user might upload a file that isn't an image, causing the upload receipt functionality to malfunction. To address this, we've added a file extension checker that ensures users upload only files with the extensions .jpeg, .jpg, or .png, ensuring they are uploading an image.

Transfer System

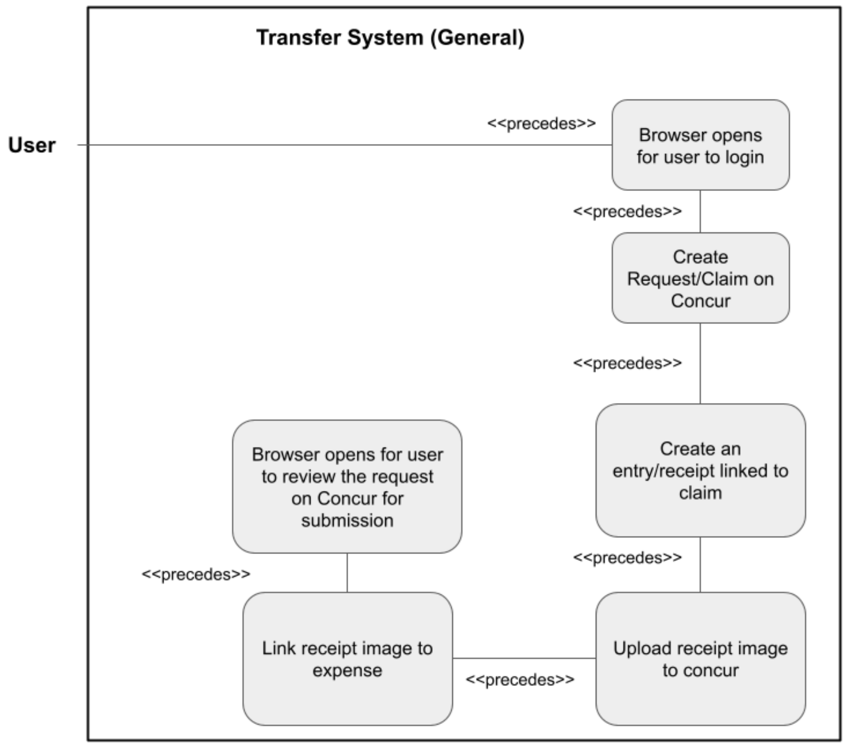

Transfer System (General)
User
<<precedes>>
Browser opens for user to login
<<precedes>>
Create Request/Claim on Concur
<<precedes>>
Create an entry/receipt linked to claim
<<precedes>>
Upload receipt image to concur
<<precedes>>
Link receipt image to expense
<<precedes>>
Browser opens for user to review the request on Concur for submission

Fig. 7: Transfer System Use Case

The Transfer System was initially designed to be a system that will act as the interface between the user and Concur, assisting the users in creating their project claims in a more convenient fashion whether it be done through macros or HTTP requests. It was then finalised to utilise an external application to run Selenium to extract the user's Concur web session, which is then used to create HTTP requests to the website to asynchronously create the user's claim.

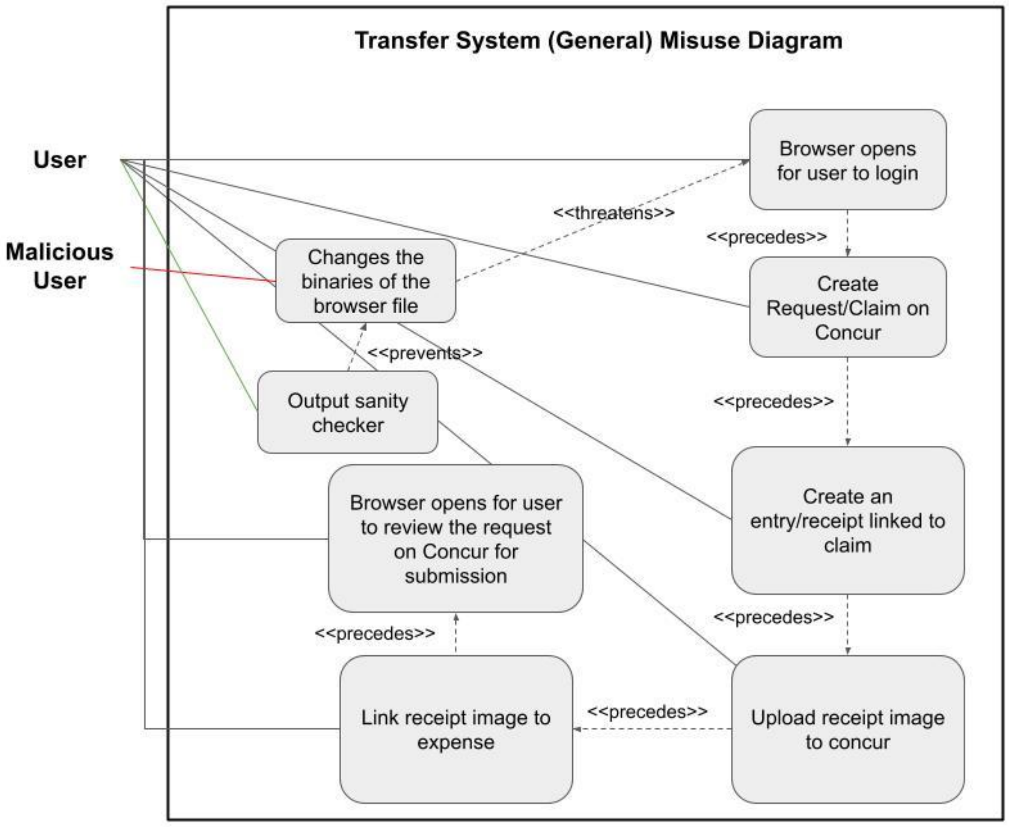

Transfer System (General) Misuse Diagram
User
Malicious User
Browser opens for user to login
<<threatens>>
<<precedes>>
Changes the binaries of the browser file
Create Request/Claim on Concur
<<prevents>>
Output sanity checker
<<precedes>>
Create an entry/receipt linked to claim
Browser opens for user to review the request on Concur for submission
<<precedes>>
<<precedes>>
Link receipt image to expense
<<precedes>>
Upload receipt image to concur

Fig. 8: Transfer System Misuse Case

In this misuse scenario, a malicious user might alter the browser's binary files, preventing users from opening the browser to log in. To prevent this, we've introduced an output sanity checker to ensure the browser files remain unmodified.

Class Diagrams

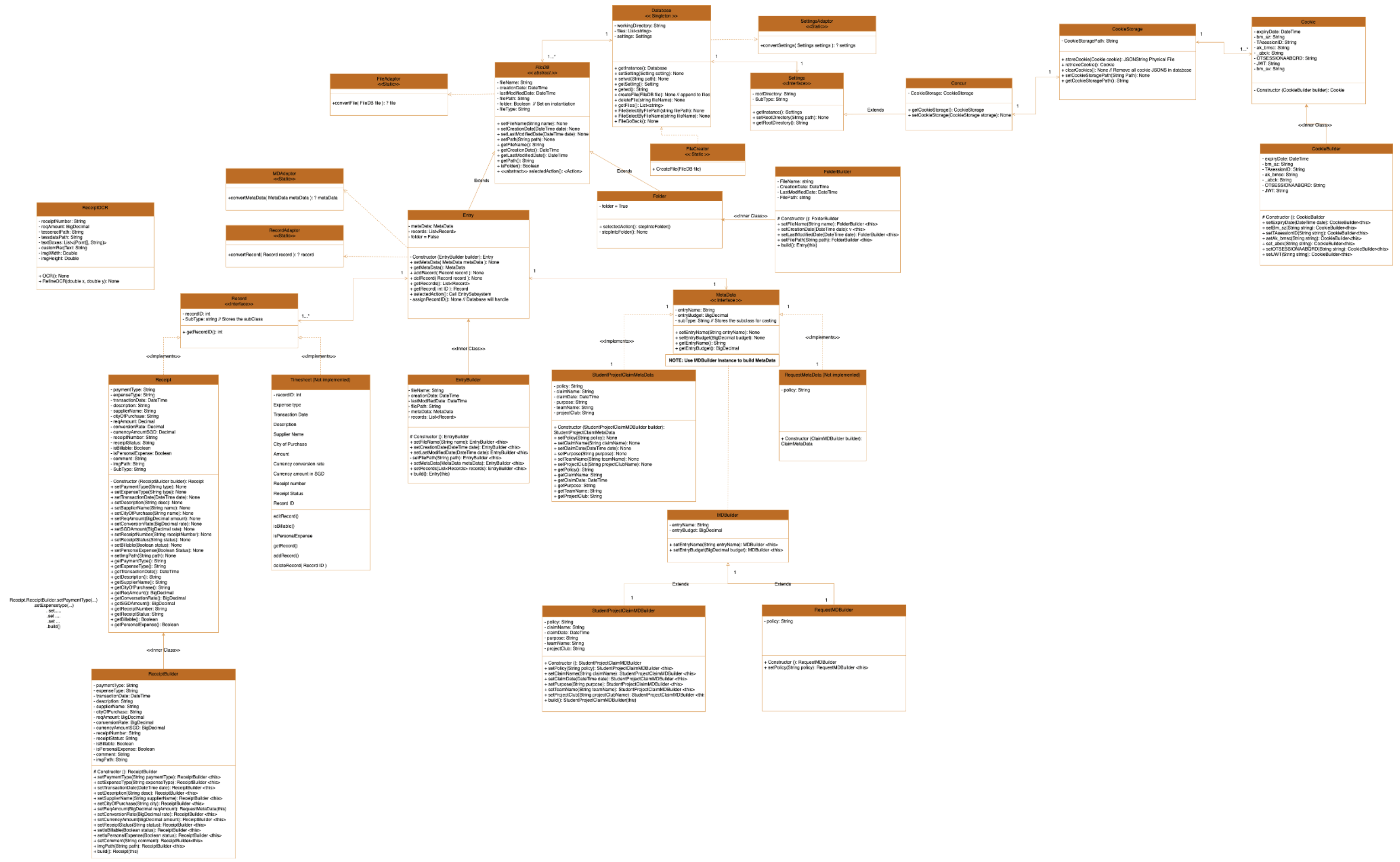


Fig. 9: Database Class Diagram

Transfer System

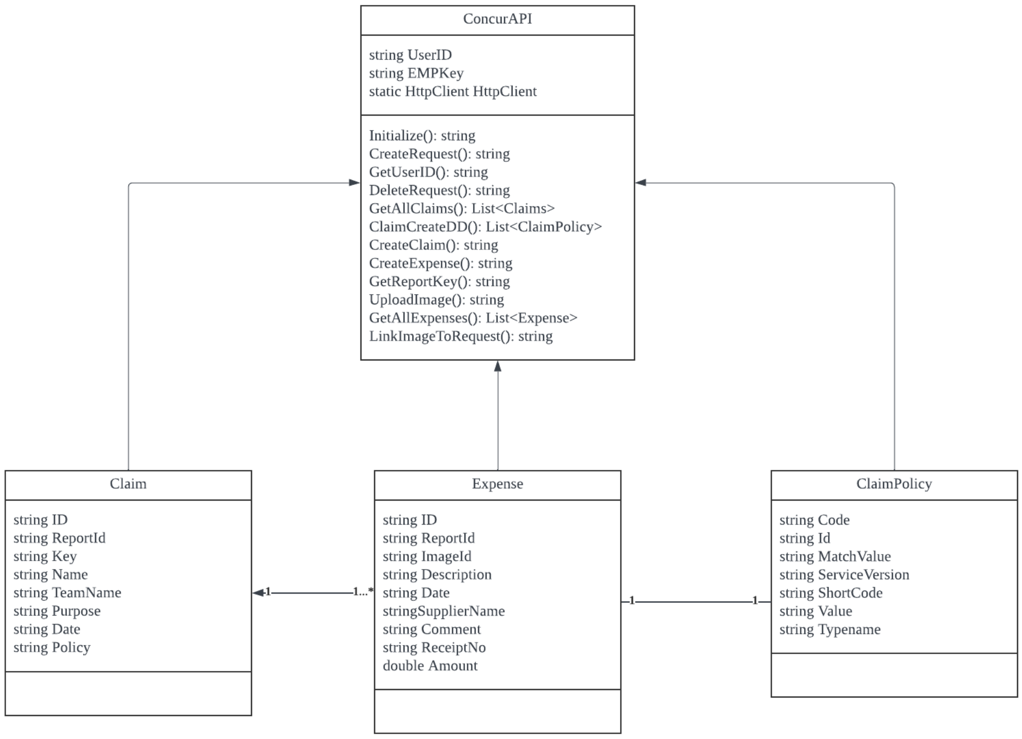


Fig. 10: Transfer System Class Diagram

Sequence Diagrams

<u>User first starts up the application</u>
When the user opens up the application, the system will attempt to retrieve the root directory of the user from the database.

If a root directory is already specified (existing users), the path to the root directory will be returned to the system.

If a root directory has not been specified (for new users/users with corrupted root directory), a "ChooseRootDirectory" page will be shown, prompting users to select their root directory from their native file directory. On successful selection, the file path of the root directory will be saved in the database.

Once the root directory is verified to exist, the "FileDirectory" page will be shown to the users, showing the contents of the root directory.

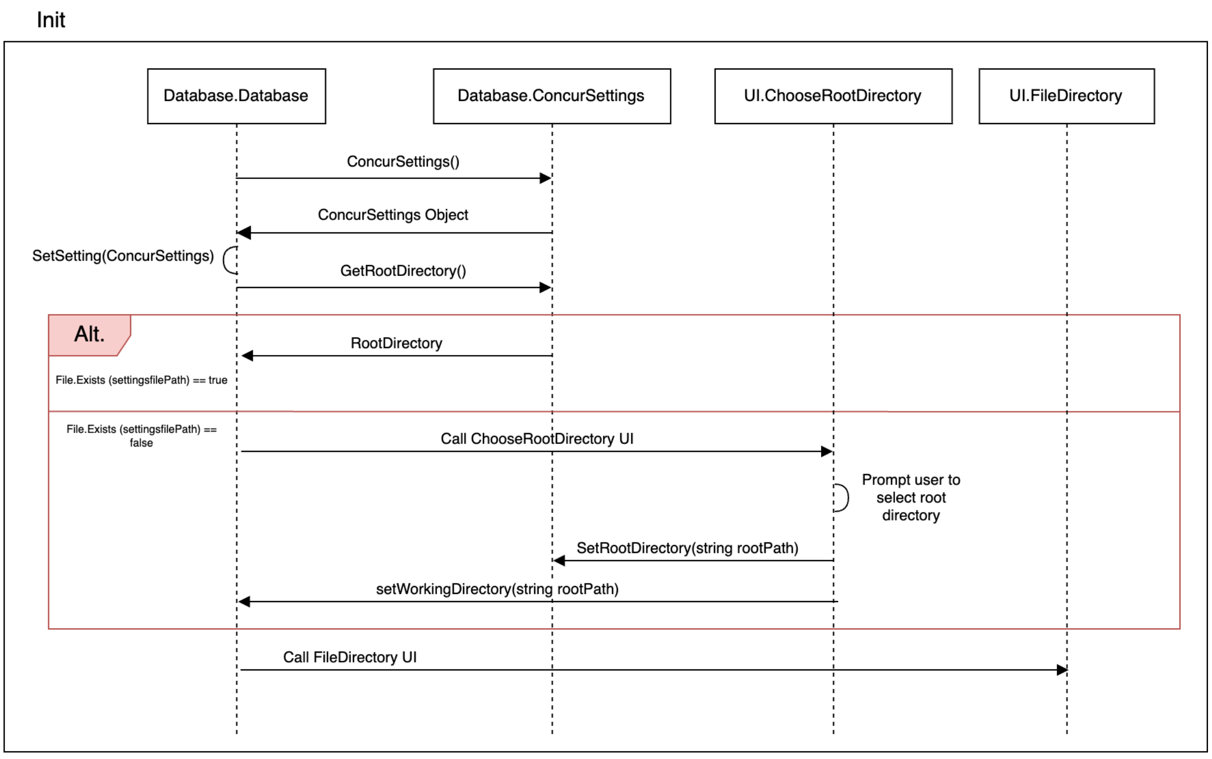

Init
Database.Database
Database.ConcurSettings
UI.ChooseRootDirectory
UI.FileDirectory
ConcurSettings()
ConcurSettings Object
SetSetting(ConcurSettings)
GetRootDirectory()
Alt.
RootDirectory
File.Exists (settingsfilePath) == true
File.Exists (settingsfilePath) == false
Call ChooseRootDirectory UI
Prompt user to select root directory
SetRootDirectory(string rootPath)
setWorkingDirectory(string rootPath)
Call FileDirectory UI

Fig. 11: Init Sequence Diagram

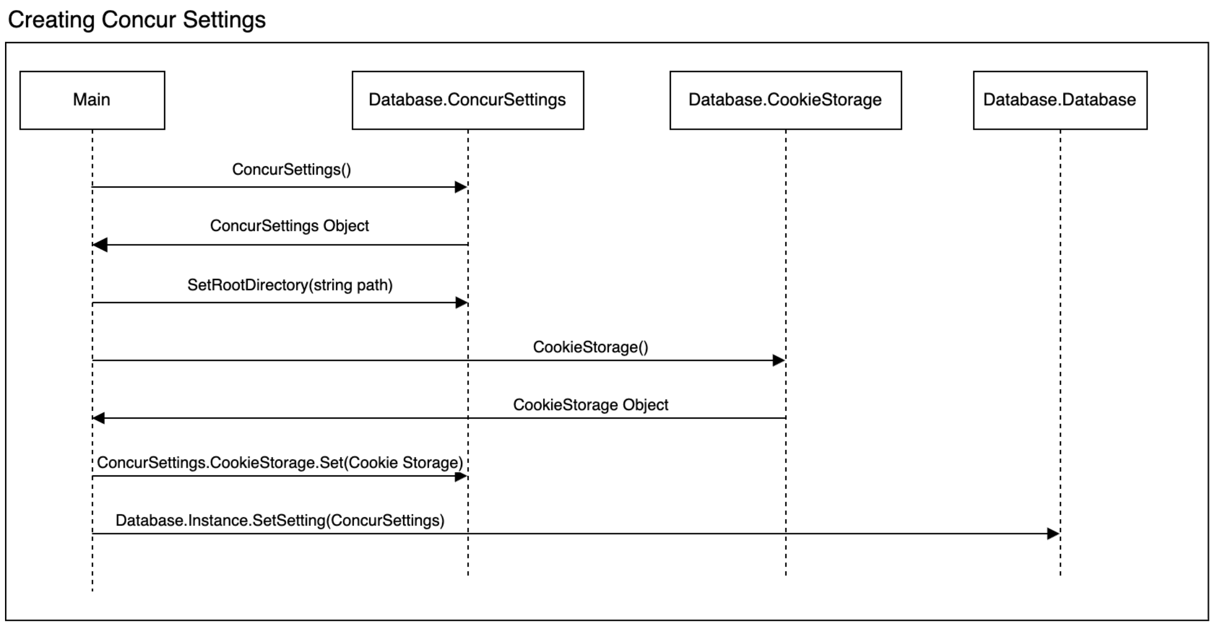


Fig. 12: Creating Concur Settings Sequence Diagram

User actions in "FileDirectory" page

A few sequence diagrams in this section will invoke these sequence actions, which includes repopulating the list of files in the file directory whenever the working directory, or the files in the working directory changes.

User creating new folder

When the user clicks on the "Add Folder" button, they are prompted to key in the name of the new folder. The database will then build and create the folder in the user's current working directory. On successful creation of the folder, the page will then repopulate to show the new folder in the current working directory.

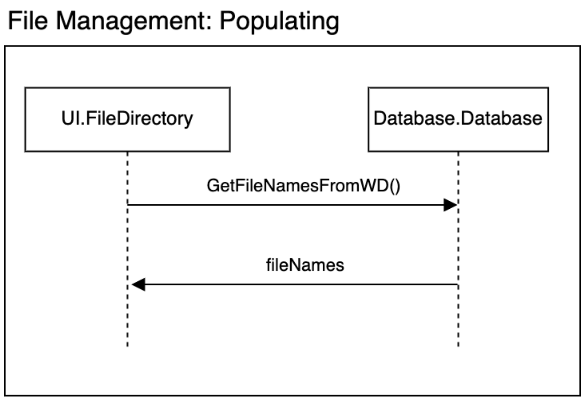


Fig. 13:File Management (Populating) Sequence Diagram

User creating new entry

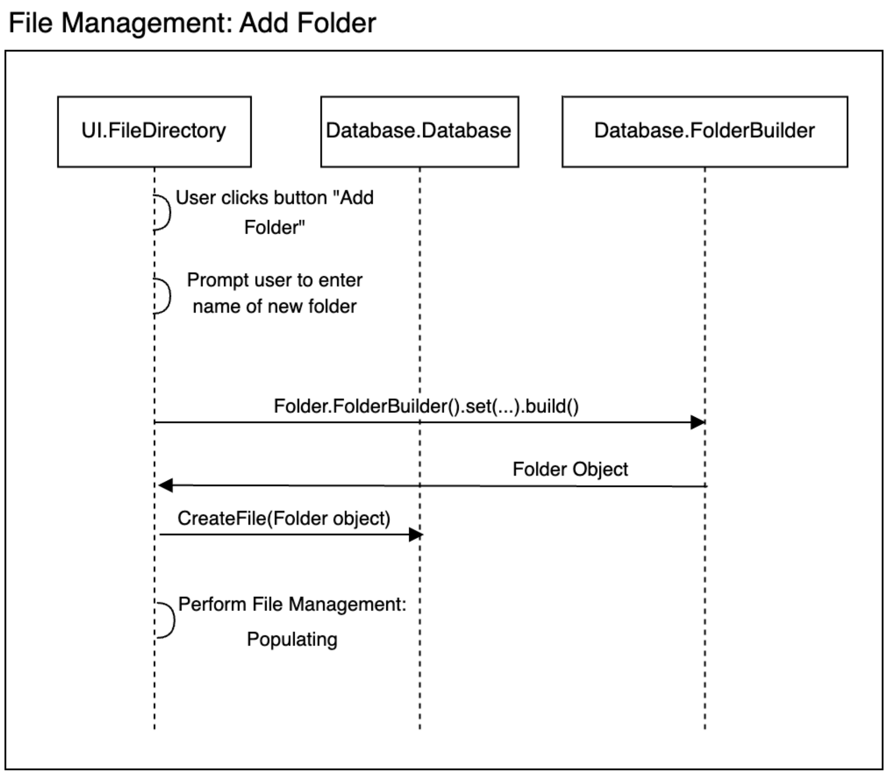

File Management: Add Folder
UI.FileDirectory
Database.Database
Database.FolderBuilder
User clicks button "Add Folder"
Prompt user to enter name of new folder
Folder.FolderBuilder().set(...).build()
Folder Object
CreateFile(Folder object)
Perform File Management: Populating

Fig. 14: File Management (Add Folder) Sequence Diagram

When the user clicks on the "Add Entry" button, they are prompted to key in the name of the new entry. They will then be redirected to the "Entry" page to finish the rest of the entry creation process.

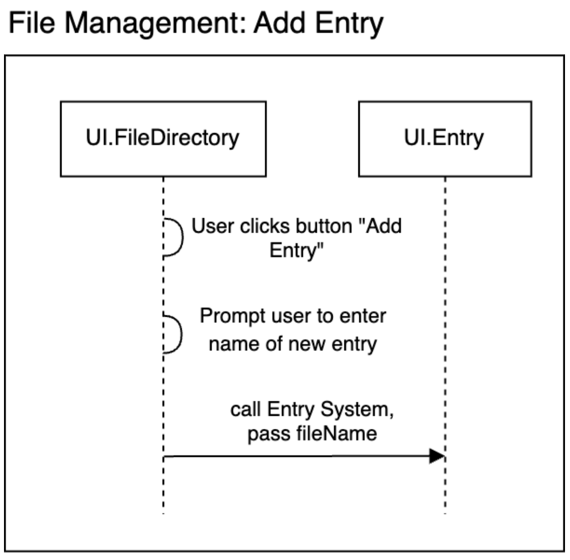

File Management: Add Entry
UI.FileDirectory
UI.Entry
User clicks button "Add Entry"
Prompt user to enter name of new entry
call Entry System, pass fileName

Fig. 15: File Management (Add Entry) Sequence Diagram

In the “Entry” page, users are required to fill in all the metadata fields to facilitate the claim creation process in Concur. If all fields are filled in, inputs are valid and the “Save Metadata” button is clicked, the database will build the metadata of the entry, then create the entry file in the working directory.

If not all fields are filled in or there are invalid inputs, an alert will be prompted to inform the user that the entry creation was not successful.

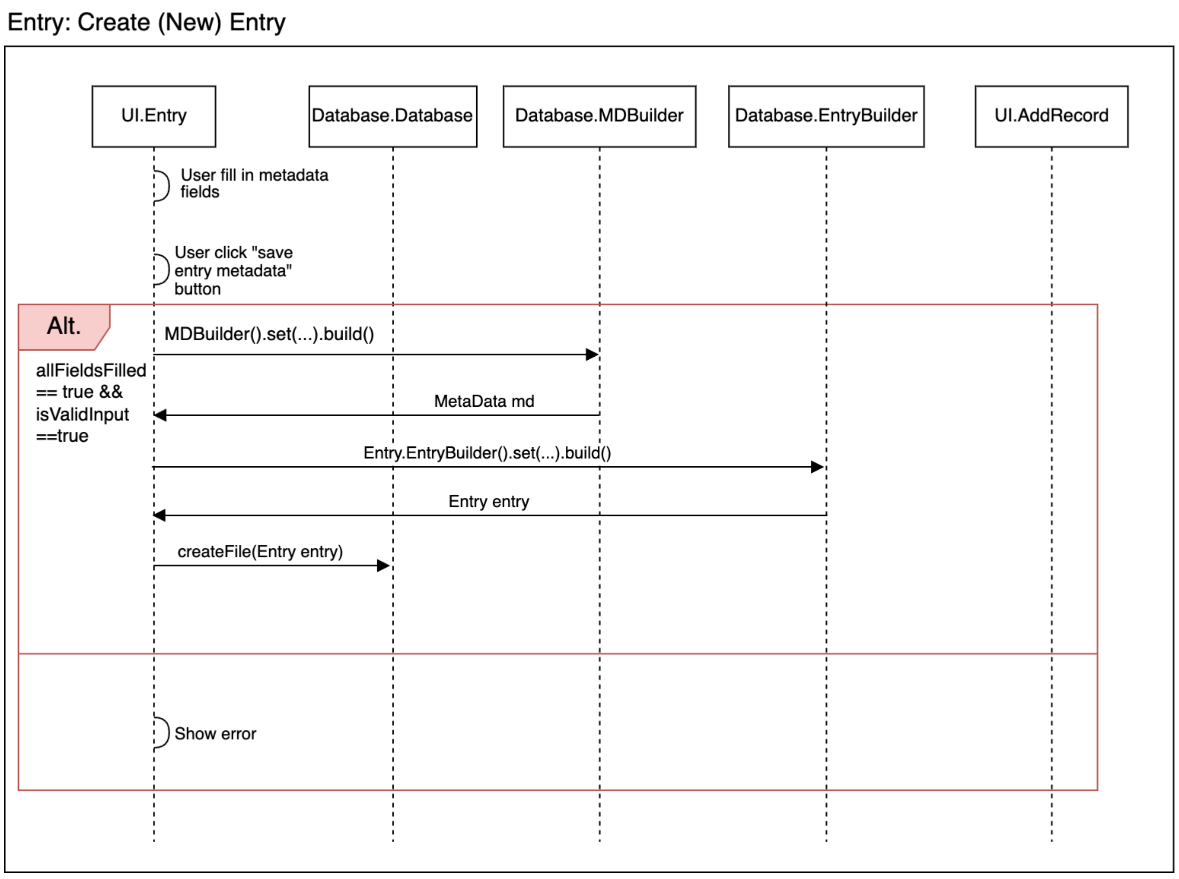

Entry: Create (New) Entry
UI.Entry
Database.Database
Database.MDBuilder
Database.EntryBuilder
UI.AddRecord
User fill in metadata fields
User click "save entry metadata" button
Alt.
allFieldsFilled == true && isValidInput ==true
MDBuilder().set(...).build()
MetaData md
Entry.EntryBuilder().set(...).build()
Entry entry
createFile(Entry entry)
Show error

Fig. 16: Create Entry Sequence Diagram

User deleting file (folder/entry)
To delete a file, the user has to first select a file by single clicking it, and then clicking on the "Delete" button. The file path of the selected file will be passed to the database system for deletion. The page will then repopulate with the remaining files of the current working directory, to show that the selected file has been removed.

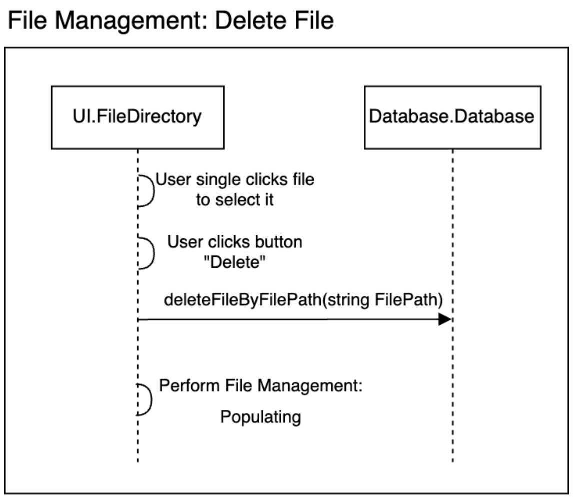

File Management: Delete File
UI.FileDirectory
Database.Database
User single clicks file to select it
User clicks button "Delete"
deleteFileByFilePath(string FilePath)
Perform File Management: Populating

Fig. 17: File Management (Delete File) Sequence Diagram

User navigating into folders/entries
To open folders/entries, users will have to double click on the file in the file directory. If a folder is selected, the working directory in the database will be updated to be the path to the selected folder, and the page will be repopulated with the contents of the selected folder. If an entry is selected, the system will pass the name of the entry to the database, and extract the entry metadata and list of records. Then, the user will be redirected to the "Entry" page, where the metadata and records of the specified entry are shown.

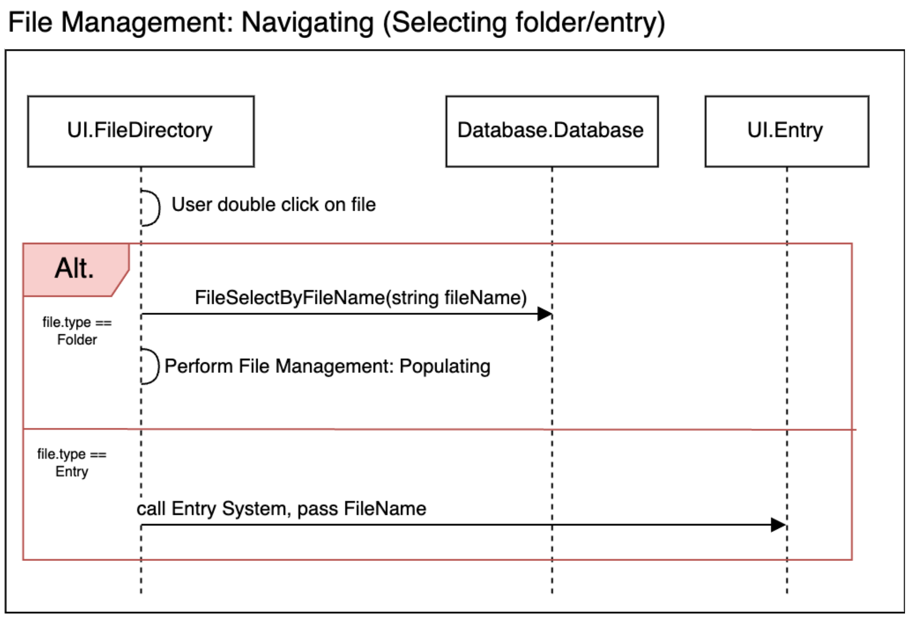

File Management: Navigating (Selecting folder/entry)
UI.FileDirectory
Database.Database
UI.Entry
User double click on file
Alt.
file.type == Folder
FileSelectByFileName(string fileName)
Perform File Management: Populating
file.type == Entry
call Entry System, pass FileName

Fig. 18: File Management (Navigating) Sequence Diagram

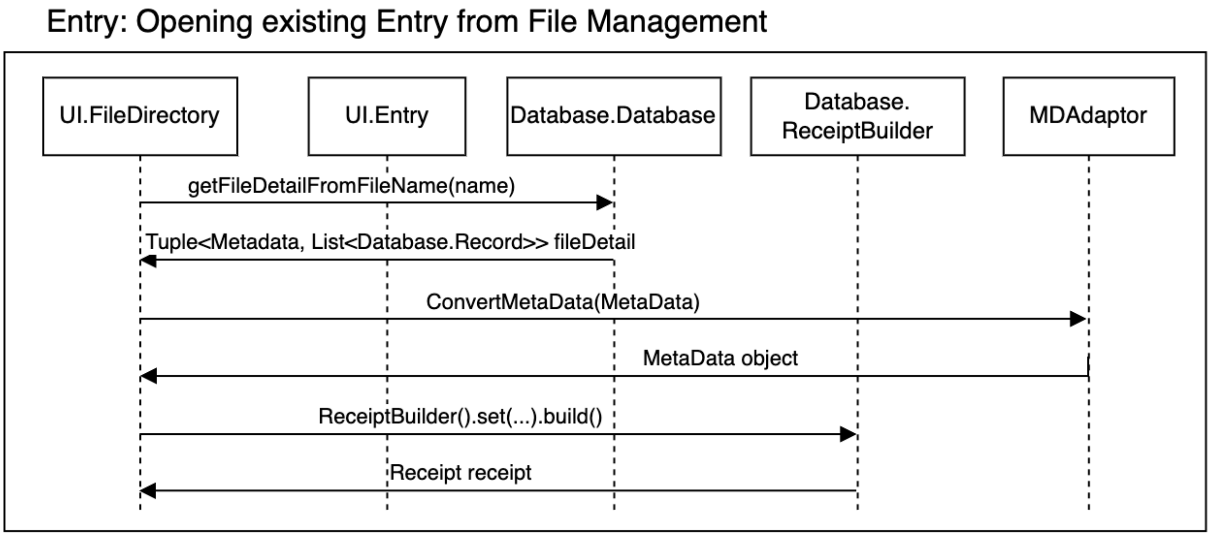

Entry: Opening existing Entry from File Management
UI.FileDirectory
UI.Entry
Database.Database
Database.ReceiptBuilder
MDAdaptor
getFileDetailFromFileName(name)
Tuple<Metadata, List<Database.Record>> fileDetail
ConvertMetaData(MetaData)
MetaData object
ReceiptBuilder().set(...).build()
Receipt receipt

Fig. 19: File Management (Open Entry) Sequence Diagram

User navigating out of folders

When the user clicks on the "Back" button, the working directory saved in the database will be updated to be the parent directory of the current directory, and the page will be repopulated with the contents of the parent directory.

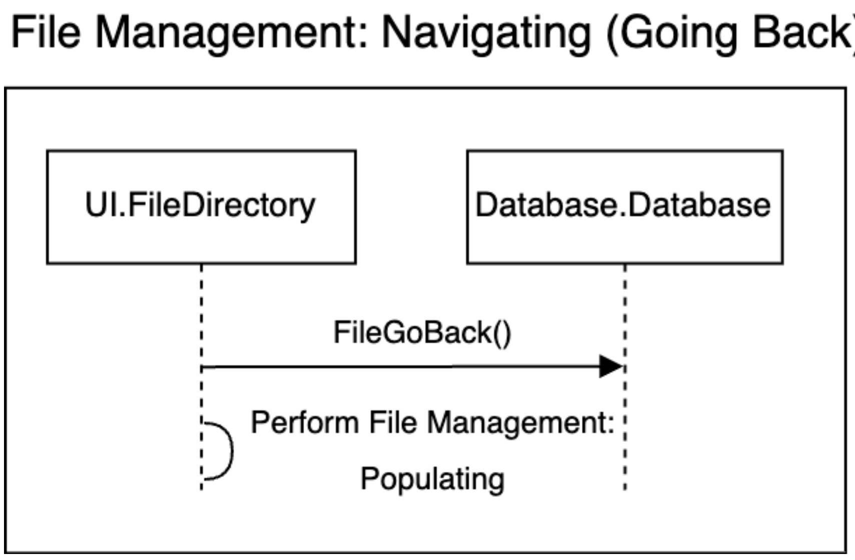

File Management: Navigating (Going Back)
UI.FileDirectory
Database.Database
FileGoBack()
Perform File Management: Populating

Fig. 20: File Management (Back Navigation) Sequence Diagram

User actions in "Entry" page

User updates metadata of entry

If the user wants to update the metadata of the entry, they will have to first edit one of the metadata fields to enable the "Save metadata" button, then click on it.

If the fields have valid inputs, the database will recreate the metadata for the entry, and return the updated metadata to the system.

If there are invalid inputs, the user will be alerted that the metadata was not updated.

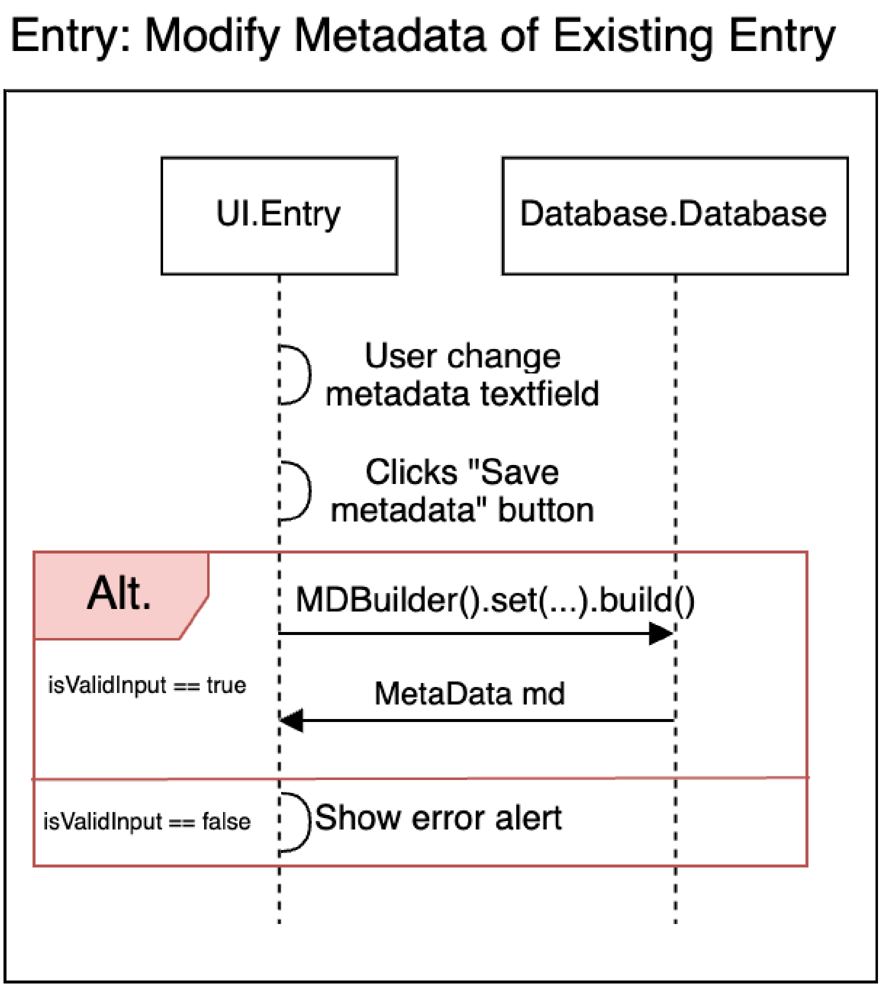

Entry: Modify Metadata of Existing Entry
UI.Entry
Database.Database
User change metadata textfield
Clicks "Save metadata" button
Alt.
isValidInput == true
MDBuilder().set(...).build()
MetaData md
isValidInput == false
Show error alert

Fig. 21: Entry (Modify Metadata) Sequence Diagram

User deleting record from entry
Users have to first select the record they want to delete by single clicking the record from the list, then clicking on the "Delete record" button. A prompt will appear, asking from confirmation to delete the selected record. On confirmation, the database will delete the record from the entry, and the system will repopulate the record list to reflect the deletion.

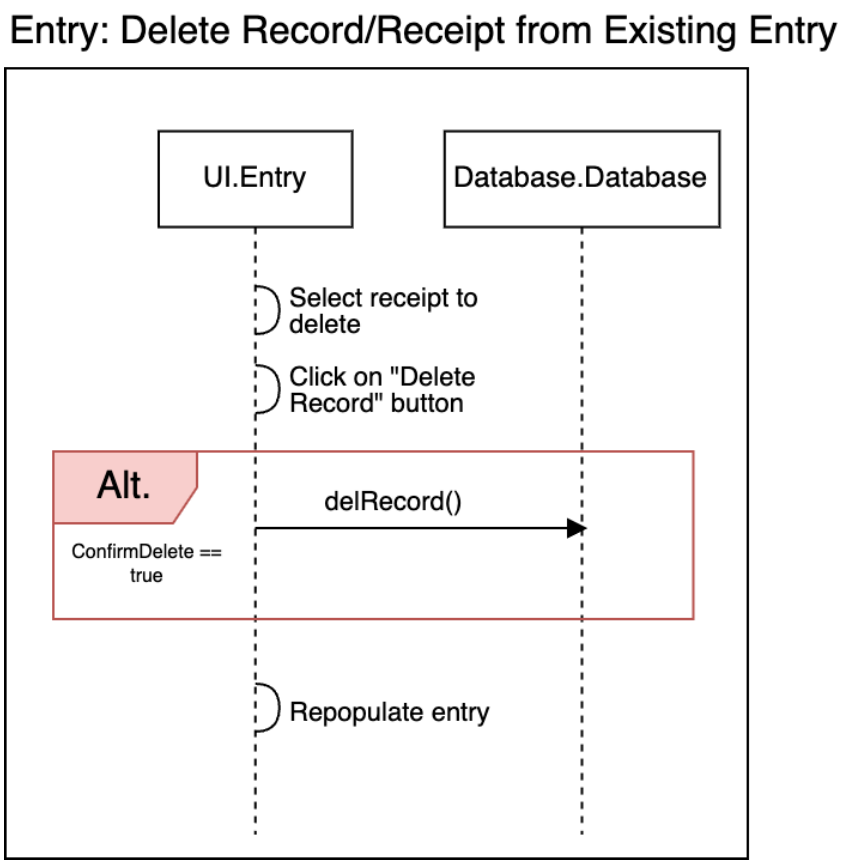


Fig. 22: Entry (Delete Record) Sequence Diagram

User adding record to entry
When the user clicks on the “Add Record” button, a file picker will appear to allow the user to upload their receipt image (only .jpg, .jpeg, .png). When chosen, the user is redirected to the “AddRecord” page. This will be followed by the user’s actions in the “AddRecord” page.

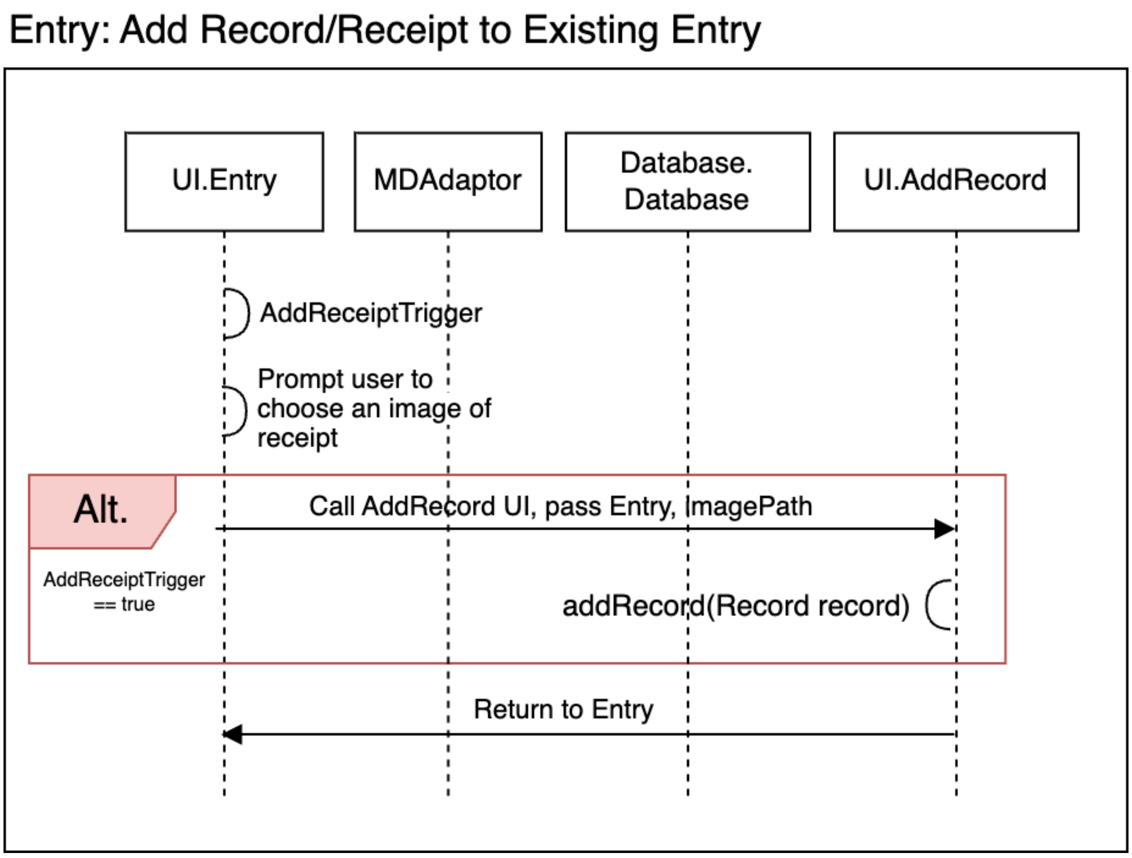

Entry: Add Record/Receipt to Existing Entry
UI.Entry
MDAdaptor
Database. Database
UI.AddRecord
AddReceiptTrigger
Prompt user to choose an image of receipt
Alt.
AddReceiptTrigger == true
Call AddRecord UI, pass Entry, ImagePath
addRecord(Record record)
Return to Entry

Fig. 23: Entry (Add Record) Sequence Diagram

User creating expense report in Concur

When the user clicks on the “Transfer to Concur” button, a selenium wrapper browser will open, showing the login page to the Concur website. Upon logging in, the browser will return to the system the cookie string of the user’s login session. Using the metadata and record list in the entry, the system will make API calls to Concur and create the expense report. Once it is done, the browser will open again, and using the session cookie extracted previously, the browser will automatically log in to Concur, and show the user that the expense report is created.

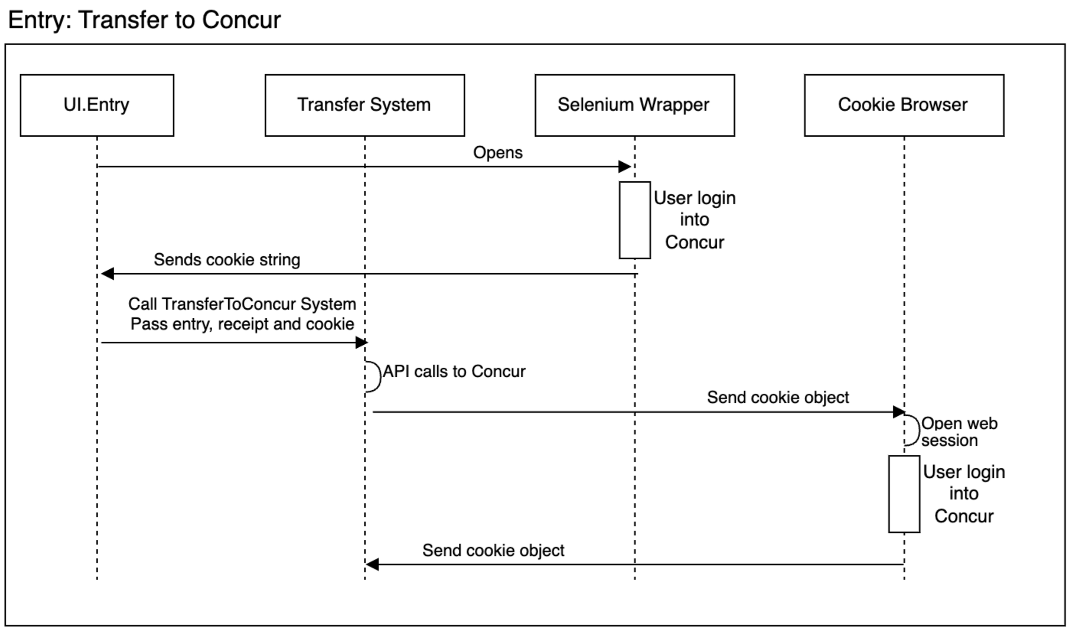


Fig. 24: Entry (Transfer to Concur) Sequence Diagram

<u>User actions in “AddRecord” page</u>

User auto-populates record details with OCR
When the user clicks on the “OCR scan” button, the system will send the image path to the OCR system, and the OCR will return the receipt details and populate the fields in the page with the relevant details.

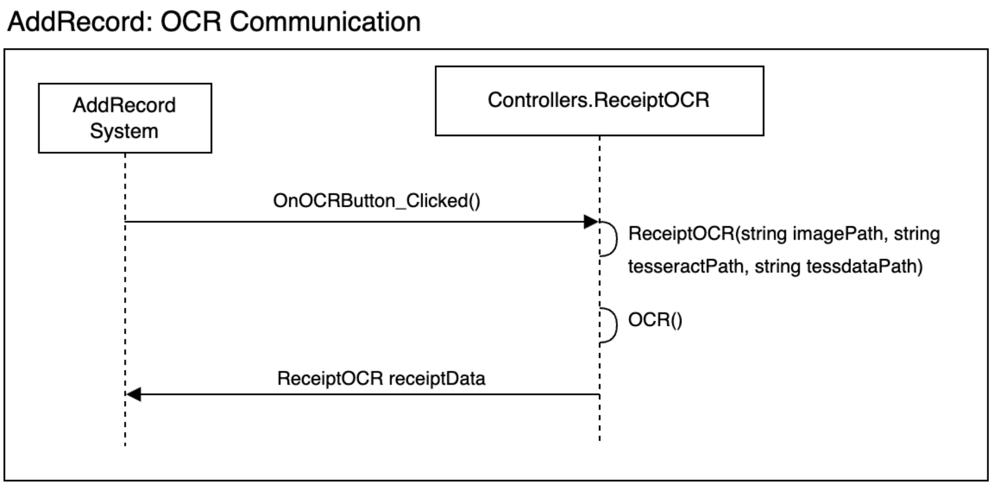

AddRecord: OCR Communication
AddRecord System
Controllers.ReceiptOCR
OnOCRButton_Clicked()
ReceiptOCR(string imagePath, string tesseractPath, string tessdataPath)
OCR()
ReceiptOCR receiptData

Fig. 25: Add Record(OCR) Sequence Diagram

User adds record to the entry
When the user is done validating the details of the record, they will click on the “Add record” button. The database will then use the details to build and return the receipt object, where it will be added to the entry. The user will then be redirected back to the “Entry” page, with the newly added record present in the record list.

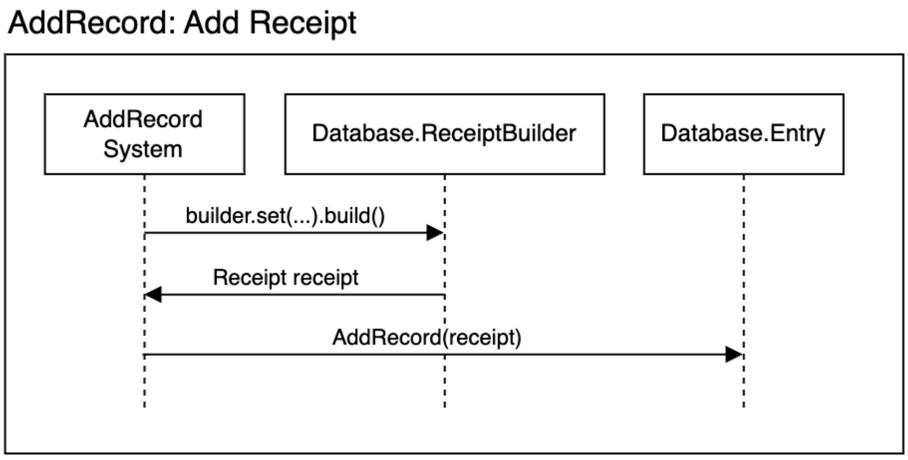


Fig. 26: Add Record(Add Receipt) Sequence Diagram

Implementation Challenges

MAUI Framework Difficulty

The team has made the decision to utilise the .NET Multi-platform App UI (MAUI) framework, leading to numerous challenges throughout the development process. The choice of MAUI was motivated by its status as a modern .NET framework capable of deployment on Windows, Mac, iOS, and Android, and it was officially released less than a year ago. However, due to its novelty, the framework brought forth several issues. These ranged from a dearth of proper documentation and limited usage, resulting in a lack of open-source examples, to a multitude of unfixed bugs, significantly extending the development timeline.

- Difficulty in setting up build environment in Visual Studio
- Difficulty with developing client applications for cross platform apps (i.e file directories, certain abstraction of services)
- Difficulty in setting up the UI testing due to there being no official way to UI test .NET MAUI apps and since it is not a web-based application, Selenium cannot be used
- Difficulty in setting up the UI for different platforms, as certain UI behaves and appear differently in different platforms

API Integration

The entire premise of the app was to simplify the process of submitting claims on SAP Concur, necessitating interaction with the web server. Unfortunately, official API access to SAP Concur was not available, which would have greatly simplified the development process. As a result, HTTP requests were mimicked to interact with SAP Concur, requiring extensive research into the packets sent to the platform. Difficulties were encountered in understanding how to query SAP Concur's GraphQL service, formulate the GraphQL queries according to their schema, and extract user variables to execute subsequent queries.

After figuring out how to interface with SAP Concur, the problem of authentication with SUTD EASE had to be addressed to login and execute queries. Initially, attempts were made to use MAUI's abstraction of a built-in browser to extract the necessary cookies, but extraction problems arose due to the cross-platform implementation. Subsequently, efforts were made to fully emulate the HTTP calls to authenticate the user into SAP Concur. However, challenges were faced in completing the SAML authentication with the Identity Service Provider (Google/Okta Verify for SUTD EASE) due to an inability to retrieve the required asymmetric key pairs for the transaction. As a workaround, an implementation using Selenium was decided upon, providing the required functionalities to fully emulate and extract the user authentication with SUTD EASE.

Database Design

In pursuit of the project's objective, the team aimed to achieve modularity and scalability for the product, intending to make it more adaptive to various systems implementing the same technology. This involved the utilisation of an OCR scan for pictures and the submission of different types of forms, such as timesheets and receipts.

To attain these goals, C# was chosen as the programming language by the team due to its strong typing, which would reduce errors and speed up development. Furthermore, the object-orientation of C# was believed to facilitate the desired modularity and scalability.

Database Program or Native File Directory?

During the Database design phase, the regular user journey of a student submitting a claim on SAP Concur was analysed by the team. The challenge of deciding between using a Database Program or the native file directory was encountered. This decision involved weighing the pros and cons of each option. While a Database Program would have ensured a robust and reliable database, the team considered an essential user requirement - the need for a convenient way to extract information like receipt images. Consequently, the decision was made to implement the "Database" using the native file directory to enhance the user experience, eliminating the need to open the application solely for extracting information.

However, this approach introduced a new set of challenges, ranging from the design of the database structure to the creation of the necessary database objects and methods. The team faced numerous obstacles during the database implementation, primarily stemming from methods such as reading from the database and modifying existing entries.

Database Design

Another challenge in the design phase was understanding what information needed to be stored in the database. This process involved meticulous work, with each team member responsible for their subsystem needing to comprehend their specific requirements. Despite being a somewhat arduous task, it was necessary to grasp the overall scale of the database. Requests from each subsystem were carefully examined by the team, and the data was collated, labelled, and classified to form the database objects. Ensuring proper labelling and classification proved challenging, requiring a comprehensive understanding of the entire system.

As seen from Fig. 01, the Database subsystem constitutes the central element responsible for ensuring the reliable transfer of data to various other subsystems. As a result, it stands as one of the more challenging subsystems to design.

At the heart of the Database System lies a single instance that serves as the root for the entire Database System. This can be found in the Database class, which is implemented as a Singleton. An instance of the Database is created whenever the application is launched.

As previously mentioned, the Database works with the native file directory. It must discern between two main file types, namely folders (representing subdirectories) and normal files (representing entries in the Database). Each entry is synonymous with a claim request in SAP Concur.

The next challenge arises from the fact that each entry can be of a different type. For example, an entry could represent a project claim, timesheet, fifth-row claim, etc. To handle this, the Entry Class employs composition. An inner MetaData class is used to classify the type of Entry instance required for SAP Concur. The MetaData contains the data necessary to create that specific claim or request in SAP Concur.

As there are multiple types of MetaData instances, the MetaData class is transformed into an interface, allowing different MetaData classes to implement it. For this project, the team chose to focus on Student Project Claims and, consequently, implemented only the StudentProjectClaimMetaData class.

Regarding the Entry instance, it was also identified that each Entry would contain multiple records, equivalent to the receipts or timesheets of each claim or request made in SAP Concur. Similarly, composition is applied to the Record instance to handle the various types of records that exist.

Design Patterns

The subsequent challenge involved the implementation of the Entry instance. The Entry instance necessitates a considerable amount of data, making it complicated and error-prone to implement using class constructors. Therefore, any subsystems attempting to implement an Entry instance without a builder class would encounter difficulty. To address this issue, the team delved into design patterns and explored the possible use of the Factory Design Pattern. However, this design pattern proved inadequate due to unimplemented methods in the parent class. After thorough analysis, the team opted for the Builder Design Pattern as a simpler and more elegant solution to this challenge. The Builder Design Pattern avoids the use of constructors, thus enhancing the usability of the Database System.

Subsequently, the team sought ways to make the Database adaptable to different potential functions in the future. This is when the Settings class was conceived. The Settings class contains the necessary information for the Database, such as the root directory of the application. Different future functionalities can be implemented as subtypes of this Settings class. For this project, the Concur Setting is implemented.

The final challenge in implementing the Database system relates to the need for an Adaptor class. This problem arose during the implementation when the Database required the conversion of a casted child back to its actual type. The primary function of this Adaptor class is to use the in-built object.GetType() method to revert the casted child back to its actual instance.

Lastly, to complete the Database System, the relevant methods were implemented for each class, with a strong focus on adhering to the Single Responsibility Principle. The overall implementation presented its fair share of challenges, primarily centred around creating, modifying, and deleting files from the Database.

The designing of the Database proved to be a complex task, necessitating the use of Design Patterns and careful considerations to ensure scalability and ease of use.

Testing

Testing was designed and implemented and segregated based on the team's individual system responsibilities. Unit Testing was performed using XUnit, implementing both white and black box testing which can be seen in the tables below. At the same time, a UI test was performed using Mac Automation Scripting on the product. The Specification Testings are performed to build confidence while Code Based Testings are performed to help detect faults. After the Unit Testing, an Implementation Test was performed. These tests help ensure that interactions between systems work as intended. Each Implementation Test contains a list of Unit Test Cases that should pass to ensure that the tested interaction would work as intended. Finally, the System Test Plan was performed and made use of the Use Case diagrams to ensure that user possible actions would be satisfied by the system. This included manually checking the inputs and outputs of the System.

UI Test Plan

Each Unit Test ensures that functions or methods perform to its specifications.

ID	Class	Assert Test	Stakeholder	User Action
1	MainPage	Sorting of names should work for both options	End User	Observe the sorting of files/folders
2	MainPage	Icons should be displayed correctly	End User	Check if icons are correct for folders and files
3	MainPage	Time should be displayed accurately	End User	Observe the displayed time
4	MainPage	Double-tap should function as expected	End User	Perform a double-tap on a file/folder

5	MainPage	Buttons should work correctly	End User	Click on New folder, new entry, rename, delete, sort, back
6	MainPage	Folder should be created in the root directory	End User	Check if a new folder is created in the root directory
7	EntryPage	Entry name change should be successful	End User	Change the entry name and verify the changes
8	EntryPage	Entry metadata update should be validated	End User	Update the entry metadata and validate the changes
9	EntryPage	Buttons should work correctly	End User	Click on save entry metadata, transfer to Concur, delete entry, new record, edit record, delete record
10	EntryPage	Incorrect data types should be handled	End User	Enter incorrect data types in the fields and check the behavior
11	EntryPage	Table should be updated correctly	End User	Check if the table is updated with the correct data
12	EntryPage	Existing entry data should load correctly	End User	Pass an existing entry file and check if the data loads properly
13	EntryPage	EntryName label should update with new name	End User	Enter a new entry name and confirm the update
14	EntryPage	User should be prompted to upload an image	End User	Click on AddRecordButton and verify the prompt
15	EntryPage	Image file navigation should work correctly	End User	Select a valid image file and check if it navigates to the record page

16	EntryPage	ConcurAPI should be initialized successfully	End User	Execute the "Concur_Clicked" method and check the initialization
17	RecordPage	Buttons should work correctly	End User	Test all buttons and their functionality
18	RecordPage	"Upload" button should only accept jpg or png files	End User	Attempt to upload different file types and verify behaviour
19	RecordPage	OCR Scan should work as expected	End User	Test the OCR Scan feature and check the results
20	RecordPage	Valid values should be entered in each input field	End User	Enter valid data types in each input field
21	RecordPage	Editing incorrect scans should work properly	End User	Edit incorrect scans and check if they are updated correctly
22	RecordPage	"Save details" button should go back to EntryPage	End User	Click on "save details" button and verify navigation
23	RecordPage	Entry should be saved in the database with receipt	End User	Check if the entry and receipt are saved in the database
24	RecordPage	Checkbox should function as expected	End User	Test the checkbox functionality
25	RecordPage	Image should be displayed correctly on the left-hand side	End User	Check if the image is displayed correctly on the left-hand side

Unit Test Plan

Utilities

ID	Class	Method	Assert Test	Stakeholder	User Action
1.1	Utilities	CheckNull	ArgumentNullException is thrown for variables with null value.	Systemwide	System wishes to check if current variable is null.
1.2	Utilities	CheckNull	Nothing is thrown.	Systemwide	System passes a variable that is not null.
1.3	Utilities	IsNumericType	Return True for valid numeric types.	Systemwide	System passes any of the following types: Byte SByte UInt Int Decimal Double Single
1.4	Utilities	IsNumericType	Return False for invalid numeric types.	Systemwide	System passes a non-numeric variable.
1.5	Utilities	IsNumericType	Return False if null valued variable passed.	Systemwide	System passes a null valued variable to the test.
1.6	Utilities	CheckIfNegative	ArgumentNullException is thrown for negative values.	Systemwide	System passes negative values.
1.7	Utilities	CheckIfNegative	Nothing is thrown.	Systemwide	System passes a positive value or zero.

1.8	Utilities	CheckIfEmptyString	ArgumentException thrown for empty string passed.	Systemwide	System passes an empty string (to check).
1.9	Utilities	CheckIfEmptyString	ArgumentException thrown for null string passed.	Systemwide	System passes an null string (to check).
1.10	Utilities	CheckIfEmptyString	Nothing is thrown.	Systemwide	System passes a non-empty string (to check).
1.11	Utilities	CheckDateTimeAheadOfNow	ArgumentException thrown if DateTime instance passed is pass the current DateTime (Present Time).	Systemwide	System passes a DateTime instance that is ahead of the present time (to check).
1.12	Utilities	CheckDateTimeAheadOfNow	Nothing is thrown.	Systemwide	System passes a DateTime instance that is before the present time (to check).
1.13	Utilities	CheckDateTimeAheadOfNow	Nothing is thrown.	Systemwide	System passes a DateTime instance that is earlier/same as compared to the present time (to check).
1.14	Utilities	CheckLastModifiedAheadOfCreation	ArgumentException thrown if LastModifiedDate ahead of CreationDate	Systemwide	System passes a LastModifiedDate that is later than CreationDate (to check).
1.15	Utilities	CheckLastModifiedAheadOfCreation	Nothing is thrown.	Systemwide	System passes a LastModifiedDate that is before than CreationDate (to check).

1.16	Utilities	CheckLastModifiedAheadOfCreation	Nothing is thrown.	Systemwide	System passes a LastModifiedDate that is same as CreationDate (to check).
1.17	Utilities	ConstEntryMetaDataPath	Return a string appended with “\EntryMetaData.json”	Systemwide	System asks to construct the path for Entry metadata (passes a path).
1.18	Utilities	ConstEntryMetaDataPath	ArgumentException thrown for empty string passed.	Systemwide	System ask to construct path but passes an empty string as argument.
1.19	Utilities	ConstReceiptsFdrPath	Return a string appended with “\Receipts.fdr”	Systemwide	System asks to construct the path for Receipts folder (passes a path).
1.20	Utilities	ConstReceiptsFdrPath	ArgumentException thrown for empty string passed.	Systemwide	System ask to construct path but passes an empty string as argument.
1.21	Utilities	ConstReceiptMetaDataPath	Return a string appended with “\Receipts.fdr\ReceiptJSON.fdr”	Systemwide	System asks to construct the path for Receipts MetaData folder (passes a path).
1.22	Utilities	ConstReceiptMetaDataPath	ArgumentException thrown for empty string passed.	Systemwide	System ask to construct path but passes an empty string as argument.
1.23	Utilities	CheckIfValidName	ArgumentException thrown for empty string passed.	Systemwide	System passes an empty string as argument.
1.24	Utilities	CheckIfValidName	ArgumentException thrown for null string passed.	Systemwide	System passes an null string as argument.

1.25	Utilities	CheckIfValidName	ArgumentException thrown for illegal characters passed.	Systemwide	System passes an illegal characters as argument.
1.26	Utilities	CheckIfValidName	Nothing is thrown.	Systemwide	System passes an acceptable string as argument

Database

Receipt

ID	Class	Method	Assert Test	Stakeholder	User Action
2.1	Receipt / ReceiptBuilder	Receipt: Constructor ReceiptBuilder: Set Functions Build()	Receipt was built and that all attributes of Receipt are as expected	Add Record System	Add Record System creates a new receipt/record entry
2.2	ReceiptBuilder	ReceiptBuilder: SetConversionRate SetReqAmount SetCurrency	Calculated converted amount is within expected tolerance (+/-0.005)	Add Record System	Add Record System requires a converted foreign currency amount to SGD.
2.3	ReceiptBuilder	ReceiptBuilder: SetReqAmount	ArgumentException thrown for negative values	Add Record System	Add Record System tries to set a negative value for ReqAmount.
2.4	ReceiptBuilder	ReceiptBuilder SetConversionRate	ArgumentException thrown for negative values	Add Record System	Add Record System tries to set a negative value for ConversionRate.
2.5	ReceiptBuilder	ReceiptBuilder Build	ArgumentNullException thrown for missing conversion rate.	Add Record System	Add Record System tries to build Receipt but a conversion rate was not assigned for foreign currency.

2.6	Receipt	Receipt Constructor	ArgumentNullException thrown for trying to build a Receipt with empty compulsory values.	Add Record System	Add Record System tries to create Receipt but is missing some compulsory fields. Default attributes set by ReceiptBuilder: SupplierName Comment IsBillable IsPersonalExpense PaymentType CityOfPurchase Currency ReceiptStatus ReqAmount CurrencyAmountSGD ConversionRate

2.7	Receipt	Receipt Set/Get reqAmount	Requested Amount is changed for the receipt	Add Record System	Add Record System editing reqAmount attribute of Receipt.
2.8	Receipt	Receipt Set reqAmount	ArgumentException thrown for trying to assign a negative value to reqAmount.	Add Record System	Add Record System editing reqAmount attribute of Receipt but sends a negative value.
2.9	Receipt	Receipt Set/Get conversionRate	Conversion Rate is changed for the receipt	Add Record System	Add Record System editing conversionRate attribute of Receipt.
2.10	Receipt	Receipt Set conversionRate	ArgumentException thrown for trying to assign a negative value to conversionRate.	Add Record System	Add Record System editing conversionRate attribute of Receipt but sends a negative value.
2.11	Receipt	Receipt Set conversionRate Get currencyAmountSG D	Conversion Rate is changed for the receipt and calculate the new currencyAmount in SGD	Add Record System	Add Record System wish to edit the conversionRate attribute of Receipt, and retrieve the currencyAmount in SGD of this receipt.
2.12	Receipt	Receipt Set/Get currencyAmountSG D	Currency Amount in SGD is changed for the receipt	Add Record System	Add Record System editing currencyAmountSGD attribute of Receipt.

2.13	Receipt	Receipt Set currencyAmountSGD	ArgumentException thrown for trying to assign a negative value to currencyAmountSGD.	Add Record System	Add Record System editing currencyAmountSGD attribute of Receipt but sends a negative value.
2.14	Receipt / ReceiptBuilder	Receipt: Constructor ReceiptBuilder: Set Functions Build()	Receipt was built and that all attributes of Receipt are as expected (Fuzzed input attributes)	Add Record System	Add Record System creates a new receipt/record entry

ReceiptOCR

ID	Class	Method	Assert Test	Stakeholder	User Action
3.1	ReceiptOCR	ReceiptOCR Constructor	ReceiptOCR instance constructed with attributes set. Does not throw an error when some attributes are null.	OCR System	OCR System creates the ReceiptOCR instance with any combination of attributes. (This instance can have empty values)
3.2	ReceiptOCR	RefineOCR	ReceiptOCR updates fields correctly. Does not throw an error when text is not in the correct format.	OCR System	OCR System attempts to update the reqAmonut and receiptNumber fields based on x,y coordinates of mouse click.

MetaData

ID	Class	Method	Assert Test	Stakeholder	User Action
4.1	StudentProjectClaimMetaData/ StudentProjectClaimMDBuilder	StudentProjectClaimMetaData Constructor StudentProjectClaimMDBuilder Set Functions Build()	StudentProjectClaimMetaData was built and all attributes of Metadata are as expected.	Entry System	Entry System creates the metadata information required to construct the Entry Instance. (Student Project Claim Entry)
4.2	StudentProjectClaimMDBuilder	StudentProjectClaimMetaData Constructor StudentProjectClaimMDBuilder Set Functions Build()	ArgumentNullException thrown if any of the attributes of the StudentProjectClaimMetaData was not set.	Entry System	Entry System tries to create an Entry's MetaData but receives an error instead as there is missing attributes.
4.3	StudentProjectClaimMDBuilder	StudentProjectClaimMDBuilder SetClaimDate	ArgumentException thrown if the ClaimDate to be assigned is ahead of present time.	Entry System	Entry System tries to assign a ClaimDate that is ahead of current time (Logically does not make sense) to the metadata builder.

4.4	StudentProjectClaimMetaData	StudentProjectClaimMetaData SetEntryBudget	ArgumentException thrown if the entry budget passed is negative.	Entry System	Entry System wishes to set the entry budget for the metadata but passed a negative value.
4.5	StudentProjectClaimMetaData	StudentProjectClaimMetaData Setter Function	Parameters of existing metadata can be set	Entry System	Entry System wishes to edit the attributes of existing metadata.
4.6	StudentProjectClaimMetaData	StudentProjectClaimMetaData Setter for EntryBudget	ArgumentException thrown if the EntryBudget to be assigned is a negative value.	Entry System	Entry System wishes to edit the attributes of existing MetaData but passed a negative value.
4.7	StudentProjectClaimMetaData	StudentProjectClaimMetaData Setter for ClaimDate	ArgumentException thrown if the ClaimDate to be assigned is ahead of present time.	Entry System	Entry System tries to assign a ClaimDate that is ahead of current time to the existing metadata.
4.8	StudentProjectClaimMetaData/ StudentProjectClaimMDBuilder	StudentProjectClaimMetaData Constructor StudentProjectClaimMDBuilder Set Functions Build()	StudentProjectClaimMetaData was built and all attributes of Metadata are as expected. (With Fuzzer input for attributes)	Entry System	Entry System creates the metadata information required to construct the Entry Instance. (Student Project Claim Entry)

Entry

ID	Class	Method	Assert Test	Stakeholder	User Action
5.1	Entry/EntryBuilder	Entry Constructor EntryBuilder Set Functions Build()	Entry was built and all attributes of Entry are as expected	Entry System	Entry System creates the Entry instance. (Student Project Claim Entry)
5.2	Entry/EntryBuilder	Entry Constructor EntryBuilder Set Functions Build()	ArgumentException thrown if any of the attributes of the Entry was not set.	Entry System	Entry System tries to create an Entry instance but receives an error instead as there is missing attributes.
5.3	EntryBuilder	EntryBuilder SetFilePath	ArgumentNullException thrown if attempting to SetFilePath when FileName has not been initialised.	Entry System	Entry System tries to set the file path before setting the file name when using the EntryBuilder.
5.4	EntryBuilder	Entry Constructor EntryBuilder Set Functions	IOException thrown if attempting to build an Entry that already exists	Entry System	Entry System tries to create an Entry instance but receives an error instead as the file name exists in the file path.

		Build()			
5.5	EntryBuilder	EntryBuilder SetFileName	IOException thrown if attempting to SetFileName when FileName is null	Entry System	Entry System tries to set the file name as null.
5.6	EntryBuilder	EntryBuilder SetFileName SetFilePath	IOException thrown if attempting to SetFilePath when FilePath is null	Entry System	Entry System tries to set the file path as null.
5.7	EntryBuilder	EntryBuilder SetFileName SetFilePath	IOException thrown if attempting to SetFilePath when FilePath is invalid	Entry System	Entry System tries to set the file path that doesnt exist
5.8	EntryBuilder	EntryBuilder SetCreationDate	ArgumentException thrown if attempting to set a creation date that is pass the present date/time.	Entry System	Entry System tries to set the CreationDate of the EntryBuilder with a date that is in the future.
5.9	EntryBuilder	EntryBuilder SetLastModified Date	ArgumentException thrown if attempting to set a last modified date that is pass the present date/time.	Entry System	Entry System tries to set the LastModifiedDate of the EntryBuilder with a date that is in the future.

5.10	Entry	Entry Constructor SetFolder Getter Folder	Folder variable of Entry has value of false.	Entry System/ Database System	Entry/Database System wants to check if file (Entry in this case) is a folder.
5.11	Entry	Entry AssignRecordID	Returns a unique RecordID for the receipt/record. (Checks which ID is used by getting the names of the files in \`\Receipts.fdr\ReceiptJSON.fdr\` folder).	Entry System/ Add Record System	Entry/Add Record System request for a new UNIQUE RecordID based on the current RecordIDs assigned for a particular Entry.
5.12	Entry	Entry AddRecord	Adds a record to the list of Records of Entry. Copies the receipt image from the imgpath to the Entry's \`\Receipts.fdr\` folder (names the receipt image its RecordID). Creates the metadata of the receipt and stores it as a JSON file to Entry's \`\Receipts.fdr\ReceiptJSON.fdr\` folder.	Entry System/ Add Record System	Entry System adds a Record/Receipt to the current Entry.

5.13	Entry	Entry DelRecord	Record is deleted based on having the same instance of the Record. Associated receipt/record images files in `\Receipts.fdr` folder are deleted. Associated receipt/record metadata is `\Receipts.fdr\ReceiptJSON.fdr` folder is deleted.	Entry System	Entry System wishes to delete a receipt/record from a particular entry using a known Record instance.
5.14	Entry	Entry DelRecord	ArgumentException thrown when attempting to delete a receipt/record based on having an instance of the Record when it does not exist in Records list.	Entry System	Entry System tries to delete a receipt/record from a particular entry using a known Record instance. HOWEVER, the record instance does not exist in the Records list of the entry.
5.15	Entry	Entry DelRecordByID	Record is deleted based on its RecordID and the ID passed. Associated receipt/record images files in `\Receipts.fdr` folder are deleted. Associated receipt/record metadata is `\Receipts.fdr\ReceiptJSON.fdr` folder is deleted.	Entry System	Entry System wishes to delete a receipt/record from a particular entry using a known RecordID.

5.16	Entry	Entry DelRecordByID	ArgumentException thrown when attempting to delete a receipt/record based on a RecordID of a Record when it does not exist in Records list.	Entry System	Entry System tries to delete a receipt/record from a particular entry using a known RecordID. HOWEVER, the record associated with the passed RecordID does not exist in the Records list of the entry.
5.17	Entry	Entry GetRecords	Returns a list of Records	Entry System	Entry System tries to retrieve the list of Records associated with a particular Entry Instance.
5.18	Entry	Entry GetRecord	Returns a record with the specified RecordID value.	Entry System	Entry System wants to retrieve a particular Record instance based on the a VALID RecordID (Exist in Records List).
5.19	Entry	Entry GetRecord	Returns a null object if specified RecordID value does not exist in Records List of a particular Entry Instance.	Entry System	Entry System wants to retrieve a particular Record instance based on the a RecordID that DOES NOT exist in the Records List of a particular entry.
5.20	Entry/EntryBuilder	Entry Constructor EntryBuilder Set Functions Build()	Entry was built and all attributes of Entry are as expected (Fuzzed attributes)	Entry System	Entry System creates the Entry instance. (Student Project Claim Entry)

Folder

ID	Class	Method	Assert Test	Stakeholder	User Action
6.1	Folder/FolderBuilder	Folder Constructor FolderBuilder Set Functions Build()	Folder was built and all attributes of Folder are as expected	File Management System	File Management System creates the Folder instance.
6.2	Folder/FolderBuilder	Folder Constructor FolderBuilder Set Functions Build()	ArgumentNullException thrown if any of the attributes of the Folder was not set.	File Management System	File Management System tries to create an Folder instance but receives an error instead as there is missing attributes.
6.3	FolderBuilder	FolderBuilder SetFilePath	ArgumentNullException thrown if attempting to SetFilePath when FileName has not been initialised.	File Management System	File Management System tries to set the file path before setting the file name when using the FolderBuilder.
6.4	FolderBuilder	FolderBuilder SetCreationDate	ArgumentException thrown if attempting to set a creation date that is pass the present date/time.	File Management System	File Management System tries to set the CreationDate of the FolderBuilder with a date that is in the future.
6.5	FolderBuilder	FolderBuilder SetLastModified	ArgumentException thrown if attempting to set a last modified	File Managem	File Management System tries to set the LastModifiedDate of the FolderBuilder

		Date	date that is pass the present date/time.	ent System	with a date that is in the future.
6.6	Folder	Folder Constructor SetFolder Getter Folder	Folder variable of Folder has value of true.	File Managem ent System/ Database System	File Management System/Database System wants to check if folder (Folder in this case) is a folder.
6.7	Folder	Folder SelectedAction StepIntoFolder	Sets the current working directory of the Database Instance of this System to the folder's FilePath (Update working path to navigate Database in the actual OS Directory)	File Managem ent System/ Database System	File Management/Database System wants to step into the folder.
6.8	Folder/Fol derBuilder	Folder Constructor FolderBuilder Set Functions Build()	Folder was built and all attributes of Folder are as expected (With fuzzing of input attribute)	File Managem ent System	File Management System creates the Folder instance.

FileDB

ID	Class	Method	Assert Test	Stakeholder	User Action
7.1	FileDB	FileDB Setter CreationDate	ArgumentException thrown if date to be assigned is in the future.	File Management System/ Entry System/ Database System	File Management/Entry/Database system tries to set the creation date of the file but it is in the future.
7.2	FileDB	FileDB Setter LastModifiedDate	ArgumentException thrown if date to be assigned is in the future.	File Management System/ Entry System/ Database System	File Management/Entry/Database system tries to set the last modified date of the file but it is in the future.
7.3	FileDB	FileDB UpdateModifiedDate	Sets the LastModifiedDate attribute of the file to the present datetime (DateTime.Now)	File Management System/ Entry System/ Database System	File Management/Entry/Database system tries to update the last modified date of the file.

Cookie

ID	Class	Method	Assert Test	Stakeholder	User Action
8.1	Cookie/CookieBuilder	Cookie Constructor CookieBuilder Set Functions Build()	Cookie was built and all attributes of Cookie are as expected	Transfer System	Transfer System creates the Cookie instance.
8.2	Cookie/CookieBuilder	Cookie Constructor CookieBuilder Set Functions Build()	ArgumentNullException thrown if any of the attributes of the Cookie was not set.	Transfer System	Transfer System tries to create a Cookie instance but receives an error instead as there are missing attributes.

CookieStorage

ID	Class	Method	Assert Test	Stakeholder	User Action
9.1	CookieStorage	CookieStorage Constructor Setter/Getter CookieStoragePath	CookieStorage sets a path to the CookieStorage.	Database System/ File Management System	Database/File Management System wishes to create a CookieStorage for the System. The path to the CookieStorage folder is also set.
9.2	CookieStorage	CookieStorage StoreCookie GetCookiePath	CookieStorage folder is created based on the CookieStoragePath. A Cookie Instance is stored as a JSON file in this folder as `<CookieStoragePath>\cookie.json	Database System/ File Management System	Database/File Management System tries to store a cookie instance into the Cookie Storage of the System.
9.3	CookieStorage	CookieStorage RetrieveCookie GetCookiePath	Returns a Cookie instance stored in CookieStorage.	Database System/ File Management System	Database/File Management System tries to retrieve the stored cookie instance in the Cookie Storage of the System.
9.4	CookieStorage	CookieStorage RetrieveCookie GetCookiePath	Return null, CookieStorage folder does not exist, when trying to retrieve stored cookie instance.	Database System/ File Management System	Database/File Management System tries to retrieve the stored cookie instance in the Cookie Storage of the System. HOWEVER, Cookie Storage folder does not exist. (i.e. Yet to be

					created)
9.5	CookieStorage	CookieStorage RetrieveCookie GetCookiePath	Return null, Cookie is expired (ExpiryDate < CurrentDate)	Database System/ File Management System	Database/File Management System tries to retrieve the stored cookie instance in the Cookie Storage of the System, HOWEVER Cookie has expired
9.6	CookieStorage	CookieStorage RetrieveCookie GetCookiePath	Return null, Cookie file does not exist, when trying to retrieve stored cookie instance. (For test: Will need to call StoreCookie method which should create both CookieStorage folder and Cookie JSON file. Then we will have to delete the Cookie file manually)	Database System/ File Management System	Database/File Management System tries to retrieve the stored cookie instance in the Cookie Storage of the System. HOWEVER, Cookie JSON file does not exist. (i.e. Yet to be created)
9.7	CookieStorage	CookieStorage ClearCookies	Deletes the cookie.json file in <CookieStoragePath>\cookie.json	Database System/ File Management System	Database/File Management System tries to clear all stored cookie instance in the Cookie Storage of the System.

Concur

ID	Class	Method	Assert Test	Stakeholder	User Action
10.1	Concur	Concur Constructor SetSettingsPath	Creates an instance of Concur and sets the appropriate SettingsPath based on OS. Windows: "%userprofile%\\documents\\Concur Solutionz\\settings.json" Mac: "$HOME\\Library\\ConcurSolutionz\\settings.json"	Database System/ File Management System	Database/File Management System tries to create an instance of Concur, the settings instance of the System.
10.2	Concur	Concur SetRootDirectory	RootDirectory of System is set to the argument passed, a JSON file is created in the appropriate <SettingsPath>\settings.json containing information of the path to the RootDirectory.	Database System/ File Management System	Database/File Management System wishes to set the root directory of the System.
10.3	Concur	Concur GetRootDirectory	Returns the rootDirectory of the System if settings JSON file exists.	Database System/ File Management System	Database/File Management System wishes to get the root directory of the System.

10.4	Concur	Concur GetRootDirectory	Returns null if JSON file for rootDirectory does not exist. (UI opens User Interface prompt)	Database System/ File Management System	Database/File Management System wishes to get the root directory of the System, but root directory was not set.
10.5	Concur	Concur Setter/Getter CookieStorage	Sets CookieStoragePath to be <RootDirectory>\CookieStorage.	Database System/ File Management System	Database/File Management System wishes to set the path to the CookieStorage for the System.

Database

ID	Class	Method	Assert Test	Stakeholder	User Action
11.1	Database	Database Instance	Creates a single common instance and returns that Database instance. (Singleton)	Database System	Database wishes to initiate the System.
11.2	Database	Database SetSetting GetSetting	Sets an instance of Settings and returns an instance of Settings.	Database System	Database wishes to set the settings for this System.
11.3	Database	Database Setwd Getwd	Sets working directory and returns working directory.	Database System	Database wishes to set and retrieve the working directory of this System.

11.4	Database	Database CreateFile FileCreator CreateFile CreateEntry PopulateReceiptFolder CopyFile	Creates an Entry type file containing a folder with FileName. With `\EntryMetaData.json” → containing the metadata of the Entry With `\Receipts.fdr` → Containing any images copied over via the imgPath attribute of the Records instance With `\Receipts.fdr\ReceiptJSON.fdr`	Database System/File Management System	Database/File Management System wishes to create an Entry File Type in the System.
11.5	Database	Database CreateFile FileCreator CreateFile CreateFolder	Creates a Folder type file containing a folder with FileName.	Database System/File Management System	Database/File Management System wishes to create a Folder File Type in the System.
11.6	Database	Database DeleteFile	File with path of argument passed is removed from the OS File Directory.	Database System/File Management System	Database/File Management System wishes to delete a File Type in the System using the path.

11.7	Database	Database GetFilePathsFromWD	Returns a list of file paths (string) with respect to the current working directory set.	Database System/File Management System	Database/File Management System wishes to retrieve the files in the current working directory (Contains both Entry and folders).
11.8	Database	Database GetFilePathsFromWD	Returns a list of Folder paths (string) with respect to the current working directory set.	Database System/File Management System	Database/File Management System wishes to retrieve the files in the current working directory (Contains Folder only).
11.9	Database	Database GetFilePathsFromWD	Returns a list of Entry file paths (string) with respect to the current working directory set.	Database System/File Management System	Database/File Management System wishes to retrieve the files in the current working directory (Contains Entry only).
11.10	Database	Database FileSelectByFileName	Changes the working directory to the folder's path name.	Database System/File Management System	Database/File Management System wishes to step into a folder type file using the File's name.
11.11	Database	Database FileSelectByFileName	Opens the Entry System.	Database System/File Management System/Entry System	File Management System wishes to open an Entry FIle type using the File's name.
11.12	Database	Database FileSelectByFileName	ThrowsException if filename not present in filesystem	Database System/File Management	File Management System wishes to open a nonexistent file using File's name.

				System	
11.13	Database	Database FileSelectByFilePath	Changes the working directory to the folder's path name.	Database System/File Management System	Database/File Management System wishes to step into a folder type file using the File's path.
11.14	Database	Database FileSelectByFilePath	Opens the Entry System.	Database System/File Management System/Entry System	File Management System wishes to open an Entry FIle type using the File's path.
11.15	Database	Database FileSelectByFilePath	ThrowsException if filepath not present in filesystem	Database System/File Management System	File Management System wishes to open a nonexistent file using File's path.
11.16	Database	Database FileGoBack	Drops the last \... of the working directory.	Database System/File Management System/Entry System	Database/File Management System wishes to go back by one file in System (Parent Directory).
11.17	Database	Database FileGoBack	Returns an unchanged working directory if it is the same as the root Directory.	Database System/File Management System/Entry System	Database/File Management/Entry System wishes to go back by one file in System (Parent Directory) but is already at the root directory.

11.18	Database	Database GetFileDetailFromName	Returns a tuple of <Metadata, List<Record>> using the fileName of the file.	Database System/File Management System/Entry System	Database/File Management System/Entry System wishes to obtain the metadata and the list of records tagged to the entry name indicated.
11.19	Database	Database RenameEntry	Renames Metadata file to new Entry Name	Database System/File Management System/Entry System	Database/File Management System/Entry System wishes to rename Entry name in the metadata

FileCreator

ID	Class	Method	Assert Test	Stakeholder	User Action
12.1	FileCreator	FileCreator: CreateFile CreateFolder	Folder is created in specified filepath	File Management System	File Management System tries to create a new Folder
12.2	FileCreator	FileCreator: CreateFile CreateEntry	Entry is created in specified filepath	File Management System	File Management System tries to create a new Entry
12.3	FileCreator	FileCreator: CreateFile	IOException is thrown for existing file	File Management System	File Management System tries to create an Entry/ Folder that exists
12.4	FileCreator	FileCreator: CopyFile	File is copied correctly to the destination path	File Management System	File Management System tries to copy a file that doesn't exist
12.5	FileCreator	FileCreator: CopyFile	File is copied correctly to the destination path, replacing the existing file	File Management System	File Management System tries to copy a file that already exists
12.6	FileCreator	FileCreator: CopyFile	DirectoryNotFoundException is thrown for incorrect destination path	File Management System	File Management System tries to copy a file to destination path that does not exist
12.7	FileCreator	FileCreator: PopulateReceip	Receipts and Metadata are correctly populated	File Management	File Management System tries to populate the receipts and metadata

		tFolder		System	of an entry when the entry folder have no populated receipts
12.8	FileCreator	FileCreator: PopulateReceiptFolder	Receipts and Metadata are overwritten with the new ones	File Management System	File Management System tries to populate the receipts and metadata of an entry when the entry folder already has populated receipts
12.9	FileCreator	FileCreator: PopulateReceiptFolder	DirectoryNotFoundException is thrown for incorrect destination path	File Management System	File Management System tries to populate receipts and metadata of an entry with incorrect path

Settings

ID	Class	Method	Assert Test	Stakeholder	User Action
13.1	Settings	Settings Constructor SetSettingsPath	Creates an instance of Settings and sets the appropriate SettingsPath based on OS. Windows: "%userprofile%\\documents\\ConcurSolutionz\\settings.json" Mac: "$HOME\\Library\\ConcurSolutionz\\settings.json"	Database System/ File Management System	Database/File Management System tries to create an instance of Concur, the settings instance of the System.
13.2	Settings	Settings SetRootDirectory	RootDirectory of System is set to the argument passed, a JSON file is created in the appropriate <SettingsPath>\settings.json containing information of the path to the RootDirectory.	Database System/ File Management System	Database/File Management System wishes to set the root directory of the System.
13.3	Settings	Settings GetRootDirectory	Returns the rootDirectory of the System if settings JSON file exists.	Database System/ File Management System	Database/File Management System wishes to get the root directory of the System.
13.4	Settings	Settings GetRootDirectory	Returns null if JSON file for rootDirectory does not exist. (UI opens User Interface prompt)	Database System/ File Management System	Database/File Management System wishes to get the root directory of the System, but root directory was not set.

ConcurAPI

ID	Class	Method	Assert Test	Stakeholder	User Action

14.1	ConcurAPI	Initialize()
		CreateClaim()
		GetReportKey()
		CreateExpense()
			Check if API can be called with a sample request and entry created	EntryPage	User clicks on the “submit to concur” button on EntryPage

14.2	ConcurAPI	CreateClaim()	Assert Exception Give a date past the current date	EntryPage	Sequential flow from 12.1
14.3	ConcurAPI	CreateClaim()	Assert Exception Give non-existent policy ID	EntryPage	Sequential flow from 12.1

14.4	ConcurAPI	CreateClaim()
		CreateExpense()
			Assert Exception
			Input fuzzy inputs into string entries
				EntryPage	Sequential flow from 12.1

14.5	ConcurAPI	CreateExpense()	Catch Exception Give a date past the current date	EntryPage	Sequential flow from 12.1

14.6	ConcurAPI	CreateClaim()
		CreateExpense()
			Assert Exception Input fuzzy inputs into string entries	EntryPage	Sequential flow from 12.1

EntryPage

ID	Class	Method	Assert Test	Stakeholder	User Action
15.1	EntryPage	EditRecord_Clicked()	Verify that the Record Page is opened	EntryPage	Click the Edit Record button

FileManagementUI

ID	Class	Method	Assert Test	Stakeholder	User Action
16.1	MainPage	OnBackClicked	Verify navigation to previous page, uses FileGoBack() method from Database	User - FileManagementUI	Click the back button
16.2	MainPage	OnDeleteClicked	Verify file deletion, uses DeleteFile() method from Database	User - FileManagementUI	Select a file and click the delete button
16.3	MainPage	OnFileDoubleTapped	Verify that the files inside the folder have been displayed or the file has been opened	User - FileManagementUI	Double click on a file
16.4	MainPage	OnFileTapped	Verify that the file/folder has been selected	User - FileManagementUI	Single click on a file
16.5	MainPage	OnNewEntryClicked	Verify that the Create Entry page is opened. Catch Exception thrown if failed.	User - FileManagementUI	Click the new entry button
16.6	MainPage	OnNewFolderClicked	Verify creation of a new folder. User is alerted if creation failed	User - FileManagementUI	Click the new folder button

16.7	MainPage	OnSortClicked	Verify that the files have been sorted according to the chosen sorting method	User - FileManagementUI	Click the sort button and select a sorting option
16.8	MainPage	RefreshPage	Verify that the files have been loaded from the database	User - FileManagementUI	Trigger the refresh action
16.9	MainPage	SortFiles	Verify sorting of files according to chosen sorting type	Developer - FileManagementUI	Call the SortFiles method with different sorting options

Unit Test Plan (Code Based Testing)

Code Based Testing was implemented to ensure Modified Condition/Decision Coverage. Only the Utilities and File Creator system was tested as these systems were the main systems used by all subsystems. Below shows the Control Flow Graphs all involved methods to be tested. This test was done using Xunit.

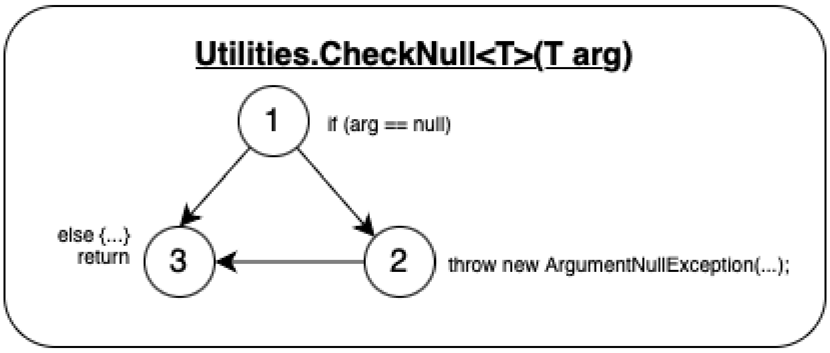

Utilities.CheckNull<T>(T arg)
1
if (arg == null)
else {...}
return
3
2
throw new ArgumentNullException(...);

Fig. 27: Utilities CheckNull Control Flow Graph

ID	Input	Expected
1	• arg = <string> null	Throw new ArgumentNullException(...)
1	• arg = <string> "Hi!"	pass

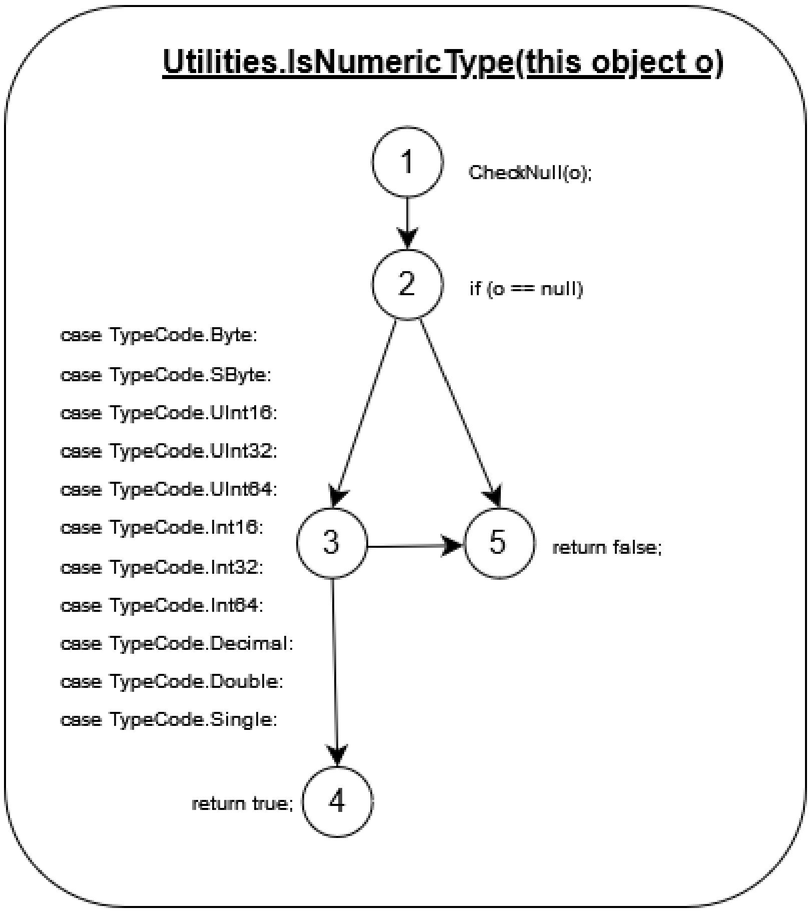


Fig. 28: Utilities isNumericType Control Flow Graph

ID	Input	Expected
1	• Value = <int> 324	true
2	• value = <string> "324.43" • Value = <decimal> 324.43	true
3	• value = <double> 324.43	true
4	• Value = <string> "HiEveryone!"	false
5	• Value = <bool> true	false
6	• Value = <double> null • Value = <int> null • Value = <decimal> null	false false false

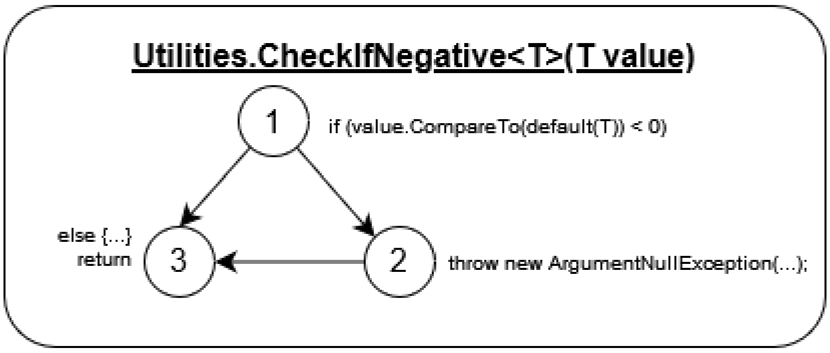

Utilities.CheckIfNegative<T>(T value)
1
if (value.CompareTo(default(T)) < 0)
2
throw new ArgumentNullException(...);
3
else {...}
return

Fig. 29: Utilities CheckIfNegative Control Flow Graph

ID	Input	Expected
1	• value = <decimal> -1231231.23m	Throw new ArgumentException(...)
2	• value = <decimal> 1231231.23m	pass

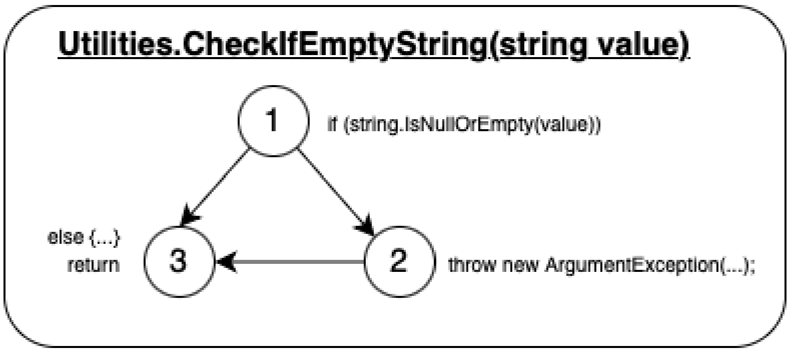

Utilities.CheckIfEmptyString(string value)
1
if (string.IsNullOrEmpty(value))
2
throw new ArgumentException(...);
3
else {...}
return

Fig. 30: Utilities CheckIfEmptyString Control Flow Graph

ID	Input	Expected
1	• value = <string> “”	Throw new ArgumentException(...)
2	• value = <string> null	Throw new ArgumentException(...)
3	• value = <string>”HIAndWelcome!”	pass

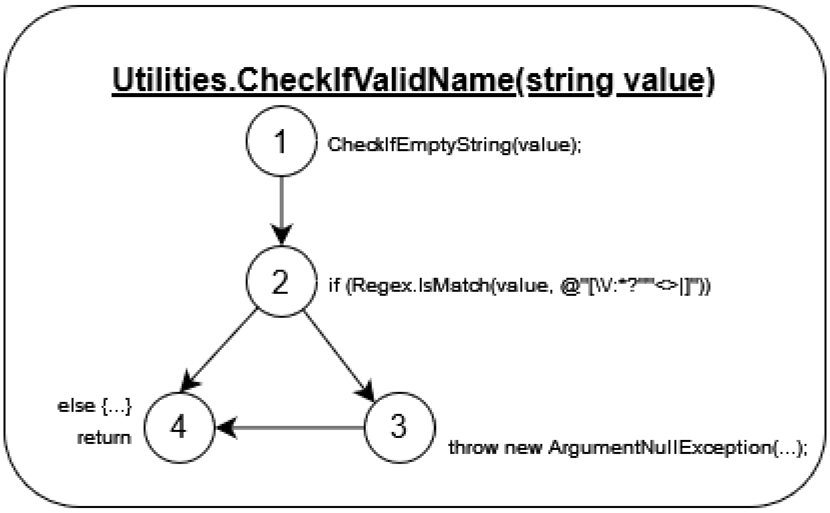

Utilities.CheckIfValidName(string value)
1
CheckIfEmptyString(value);
2
if (Regex.IsMatch(value, @"[\/:*?""<>|]"))
else {...}
return
4
3
throw new ArgumentNullException(...);

Fig. 31: Utilities CheckIfValidName Control Flow Graph

ID	Input	Expected
1	• value = <string> “”	Throw new ArgumentException(...)
2	• value = <string> null	Throw new ArgumentException(...)
3	• value = <string> "Capstone 2023/"	Throw new ArgumentException(...)
4	• value = <string> "Capstone\ 2023"	Throw new ArgumentException(...)
5	• value = <string> "Capstone: 2023"	Throw new ArgumentException(...)
6	• value = <string> "Capstone* 2023"	Throw new ArgumentException(...)
7	• value = <string> "Capstone? 2023"	Throw new ArgumentException(...)
8	• value = <string> "Capstone" 2023"	Throw new ArgumentException(...)
9	• value = <string> "Capstone< 2023"	Throw new ArgumentException(...)
10	• value = <string> "Capstone> 2023"	Throw new ArgumentException(...)
11	• value = <string> "Capstone| 2023"	Throw new ArgumentException(...)
12	• value = <string>”HiHongJingTheOneAndOnly”	pass

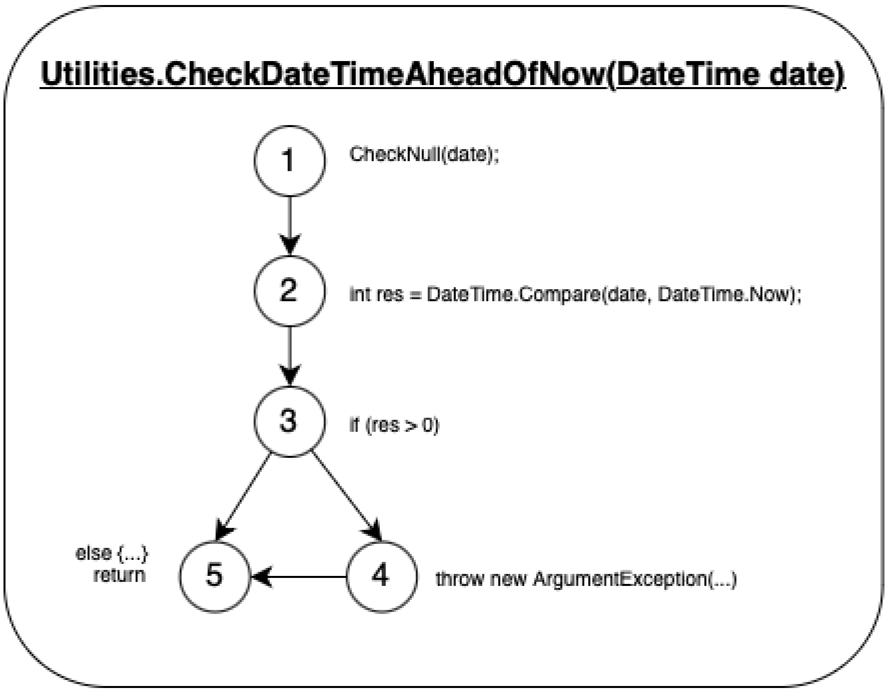

Utilities.CheckDateTimeAheadOfNow(DateTime date)
1
CheckNull(date);
2
int res = DateTime.Compare(date, DateTime.Now);
3
if (res > 0)
else {...}
return
5
4
throw new ArgumentException(...)

Fig. 32: Utilities CheckDateTimeAheadOfNow Control Flow Graph

ID	Input	Expected
1	• date = <DateTime> (DateTime.Now + 1 year)	Throw new ArgumentException(...)
2	• date = <DateTime> (DateTime.Now - 1 year)	pass
3	• Date = <DateTime> (DateTime.Now)	pass

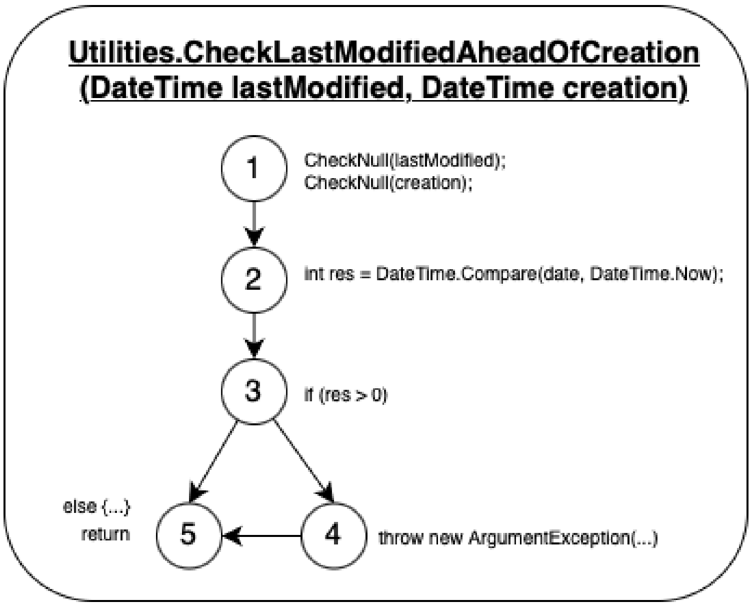

Utilities.CheckLastModifiedAheadOfCreation
(DateTime lastModified, DateTime creation)
1
CheckNull(lastModified);
CheckNull(creation);
2
int res = DateTime.Compare(date, DateTime.Now);
3
if (res > 0)
else {...}
return
5
4
throw new ArgumentException(...)

Fig. 33: Utilities CheckLastModifiedAheadOfCreation Control Flow Graph

ID	Input	Expected
1	• lastModified = <DateTime> (23 Jan 2023) • creation = <DateTime> (23 Dec 2022)	Throw new ArgumentException(...)
2	• lastModified = <DateTime> (23 Jan 2023) • creation = <DateTime> (24 Jan 2023)	pass
3	• lastModified = <DateTime> (23 Jan 2023) • creation = <DateTime> (23 Jan 2023)	pass

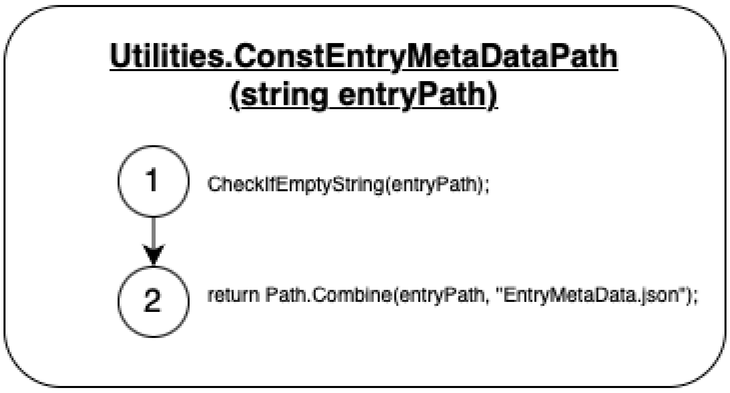

Utilities.ConstEntryMetaDataPath
(string entryPath)
1
CheckIfEmptyString(entryPath);
2
return Path.Combine(entryPath, "EntryMetaData.json");

Fig. 34: Utilities ConstEntryMetaDataPath Control Flow Graph

ID	Input	Expected
1	• entryPath = <String> “Capstone 2023”	“Capstone 2023/EntryMetaData.json”
2	• entryPath = <String> “”	Throw new ArgumentException(...)

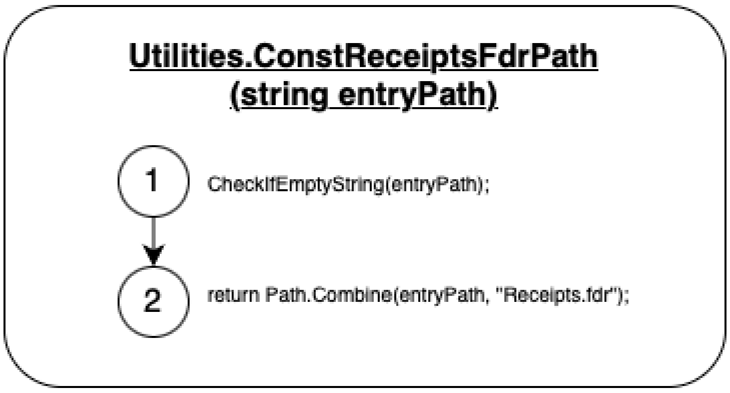

Utilities.ConstReceiptsFdrPath
(string entryPath)
1
CheckIfEmptyString(entryPath);
2
return Path.Combine(entryPath, "Receipts.fdr");

Fig. 35: Utilities ConstReceiptsFdrPath Control Flow Graph

ID	Input	Expected
1	• entryPath = <String> "Capstone 2023"	"Capstone 2023/Receipts.fdr"
2	• entryPath = <String> ""	Throw new ArgumentException(...)

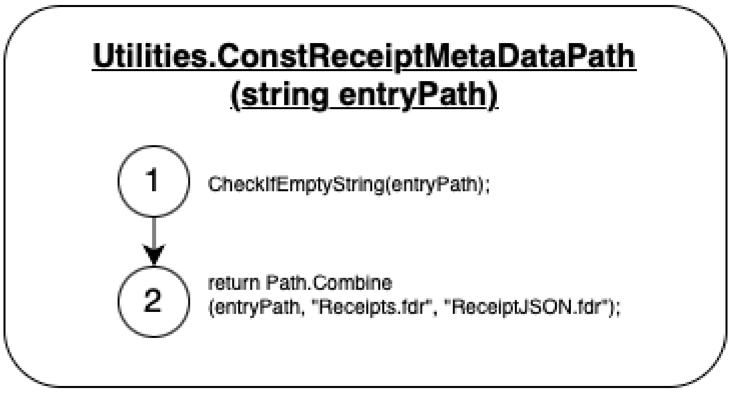

Utilities.ConstReceiptMetaDataPath
(string entryPath)
1
CheckIfEmptyString(entryPath);
2
return Path.Combine
(entryPath, "Receipts.fdr", "ReceiptJSON.fdr");

Fig. 36: Utilities ConstReceiptMetaDataPath Control Flow Graph

ID	Input	Expected
1	• entryPath = <String> "Capstone 2023"	"Capstone 2023/Receipts.fdr/ReceiptJSON.fdr"
2	• entryPath = <String> ""	Throw new ArgumentException(...)

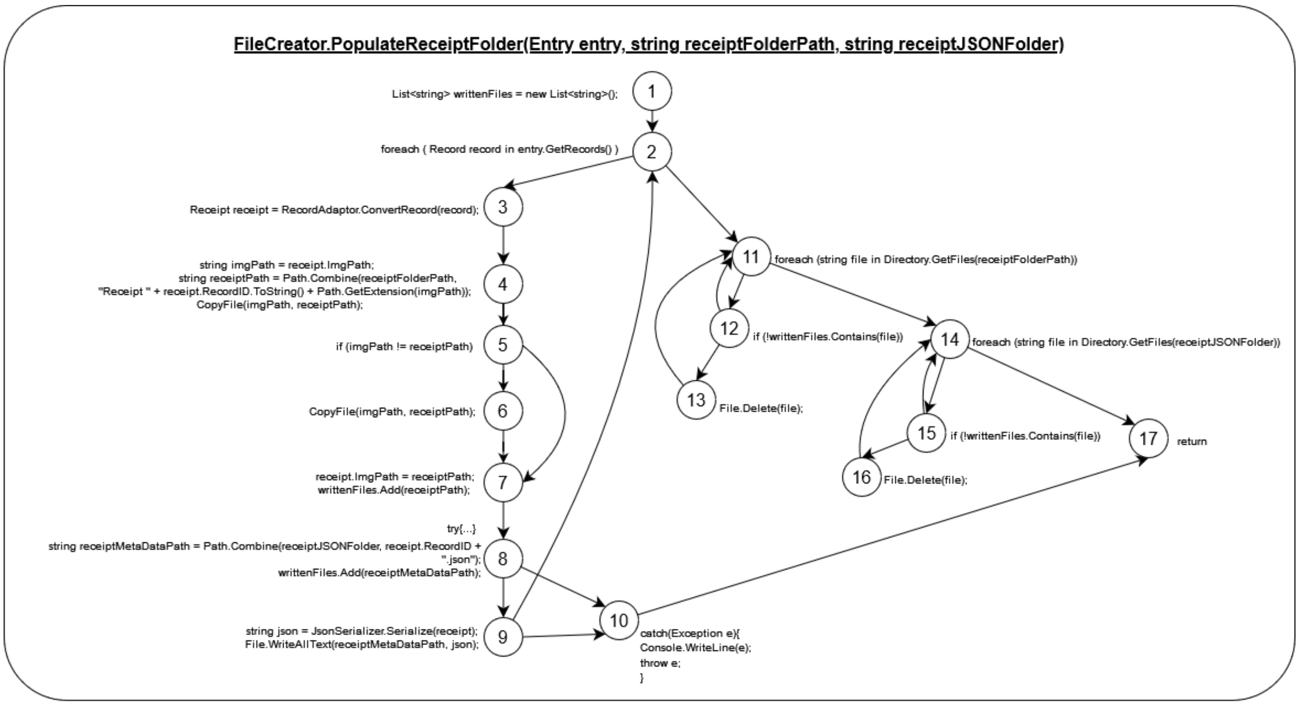

FileCreator.PopulateReceiptFolder(Entry entry, string receiptFolderPath, string receiptJSONFolder)
1
List<string> writtenFiles = new List<string>();
2
foreach (Record record in entry.GetRecords())
3
Receipt receipt = RecordAdaptor.ConvertRecord(record);
4
string imgPath = receipt.ImgPath;
string receiptPath = Path.Combine(receiptFolderPath,
"Receipt " + receipt.RecordID.ToString() + Path.GetExtension(imgPath));
CopyFile(imgPath, receiptPath);
5
if (imgPath != receiptPath)
6
CopyFile(imgPath, receiptPath);
7
receipt.ImgPath = receiptPath;
writtenFiles.Add(receiptPath);
8
try{...}
string receiptMetaDataPath = Path.Combine(receiptJSONFolder, receipt.RecordID + ".json");
writtenFiles.Add(receiptMetaDataPath);
9
string json = JsonSerializer.Serialize(receipt);
File.WriteAllText(receiptMetaDataPath, json);
10
catch(Exception e){
Console.WriteLine(e);
throw e;
}
11
foreach (string file in Directory.GetFiles(receiptFolderPath))
12
if (!writtenFiles.Contains(file))
13
File.Delete(file);
14
foreach (string file in Directory.GetFiles(receiptJSONFolder))
15
if (!writtenFiles.Contains(file))
16
File.Delete(file);
17
return

Fig. 37: FileCreator PopulateReceiptFolder Control Flow Graph

ID	**Input**	**Expected**
1	• Entry = <Entry> {records = <Receipt>[rec1, rec2, rec3]} • receiptFolderPath = "Capstone 2023/Receipts.fdr" • receiptJSONFolder = "Capstone 2023/Receipts.fdr/ReceiptJSON.fdr" • No existing receipts in "Capstone 2023/Receipts.fdr" • No existing receipt metadata in "Capstone 2023/Receipts.fdr"	rec1, rec2, rec3's metadata are stored in "Capstone 2023/Receipts.fdr/ReceiptJSON.fdr" rec1, rec2, rec3's receipt image are stored in "Capstone 2023/Receipts.fdr"
2	• Entry = <Entry> {records = <Receipt>[rec1' *(Same receipt name as rec1)*, rec2' *(Same receipt name as rec2)*, rec4]} • receiptFolderPath = "Capstone 2023/Receipts.fdr" • receiptJSONFolder = "Capstone 2023/Receipts.fdr/ReceiptJSON.fdr" • <Receipt>[rec1, rec2, rec3] exist in "Capstone 2023/Receipts.fdr" • <Receipt>[rec1, rec2, rec3] exist in "Capstone 2023/Receipts.fdr"	rec1', rec2', rec3, rec4's metadata are stored in "Capstone 2023/Receipts.fdr/ReceiptJSON.fdr" rec1', rec2', rec3, rec4's receipt image are stored in "Capstone 2023/Receipts.fdr"
3	• Entry = <Entry> {records = <Receipt>[rec1' *(Same receipt name as rec1)*, rec2' *(Same receipt name as rec2)*, rec4]} • receiptFolderPath = "Capstone 2023/Receipts.fdr" • receiptJSONFolder = "Capstone 2023/Receipts.fdr///\\/ReceiptJSON.fdr" • <Receipt>[rec1, rec2, rec3] exist in "Capstone 2023/Receipts.fdr" • <Receipt>[rec1, rec2, rec3] exist in "Capstone 2023/Receipts.fdr"	throw DirectoryNotFoundException(...)

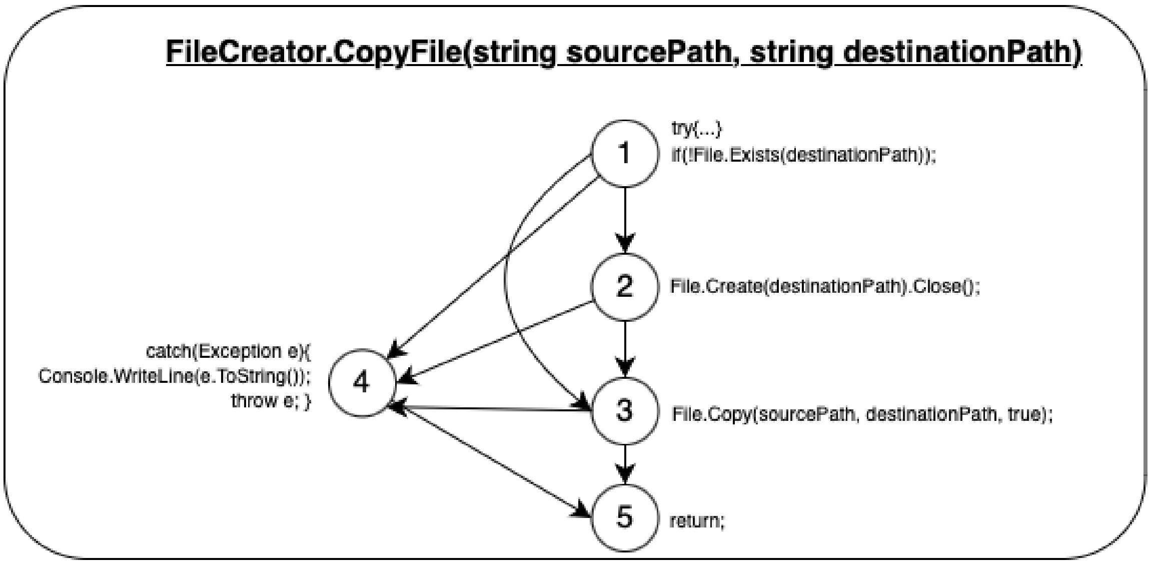
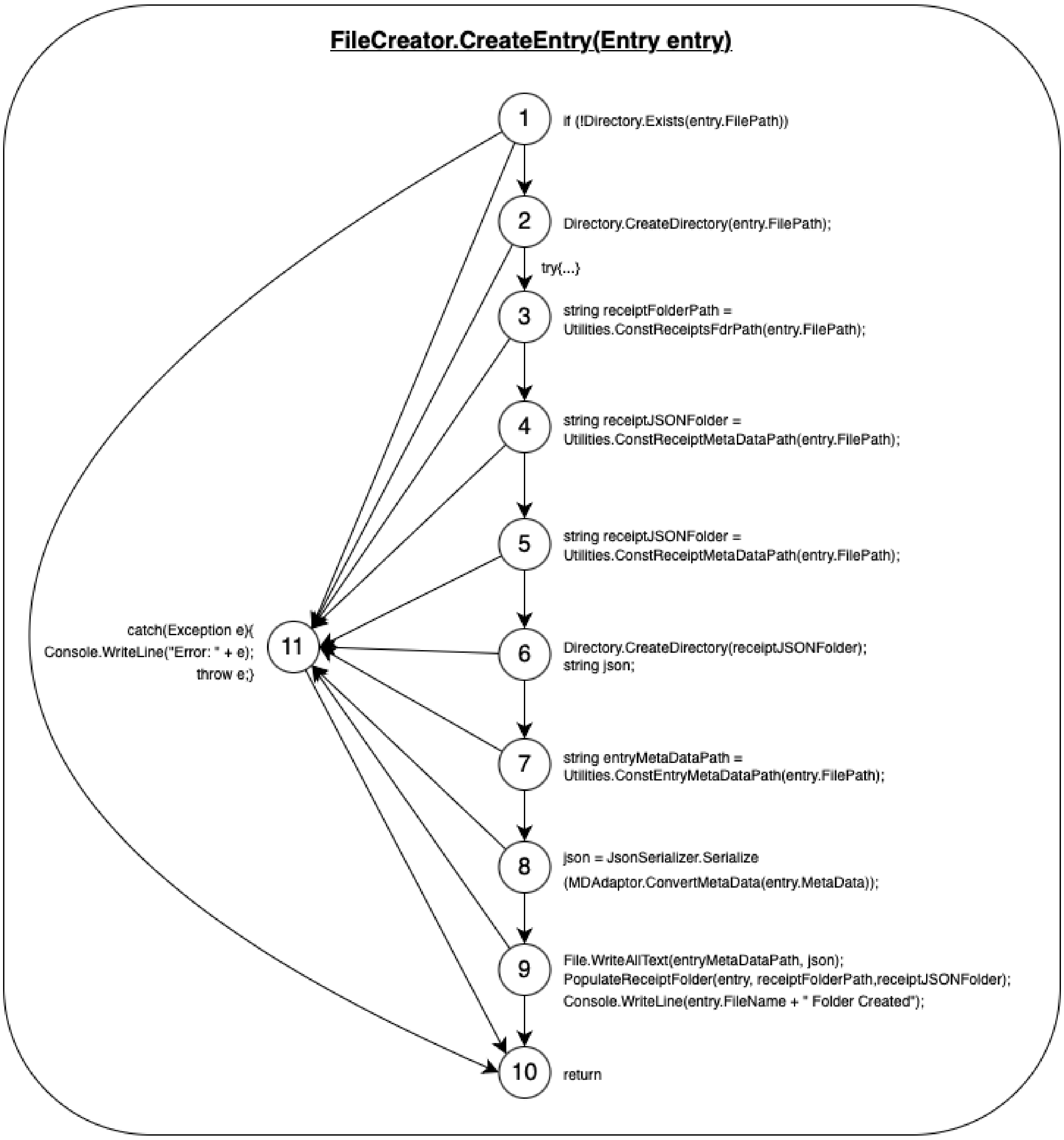

FileCreator.CopyFile(string sourcePath, string destinationPath)
1
try{...}
if(!File.Exists(destinationPath));
2
File.Create(destinationPath).Close();
3
File.Copy(sourcePath, destinationPath, true);
4
catch(Exception e){
Console.WriteLine(e.ToString());
throw e; }
5
return;

Fig. 38: FileCreator CopyFile Control Flow Graph

ID	Input	Expected
1	• sourcePath = <Path1 To File1> • destinationPath = <Path2> ○ <Path2> does not exist	<Path2> is created File1 is copied to <Path2>
2	• sourcePath = <Path1 To File2> • destinationPath = <Path2> ○ <Path2> exist	File2 is copied to <Path2>
3	• sourcePath = <Path1 To File2> • destinationPath = “\/\/\/\/\\\/”	throw DirectoryNotFoundException(...)



```
1  if (!Directory.Exists(entry.FilePath))
2  Directory.CreateDirectory(entry.FilePath);
3  string receiptFolderPath =
   Utilities.ConstReceiptsFdrPath(entry.FilePath);
4  string receiptJSONFolder =
   Utilities.ConstReceiptMetaDataPath(entry.FilePath);
5  string receiptJSONFolder =
   Utilities.ConstReceiptMetaDataPath(entry.FilePath);
6  Directory.CreateDirectory(receiptJSONFolder);
   string json;
7  string entryMetaDataPath =
   Utilities.ConstEntryMetaDataPath(entry.FilePath);
8  json = JsonSerializer.Serialize
   (MDAdaptor.ConvertMetaData(entry.MetaData));
9  File.WriteAllText(entryMetaDataPath, json);
   PopulateReceiptFolder(entry, receiptFolderPath,receiptJSONFolder);
   Console.WriteLine(entry.FileName + " Folder Created");
10 return
11 catch(Exception e){
   Console.WriteLine("Error: " + e);
   throw e;}
```

Fig. 39: FileCreator CreateEntry Control Flow Graph

ID	Input	Expected
1	• entry = <Entry> file	Entry's Receipt images are stored in receipt folder Entry's Receipt metadata are stored in receipt metadata folder Entry's metadata is stored in the entry's folder
2	• entry = <Entry> entry { entry.FileName = "test\/\/\/\\.entry" entry.FilePath = "...test/\/\/\/\\.entry"	Throw Exception(...);

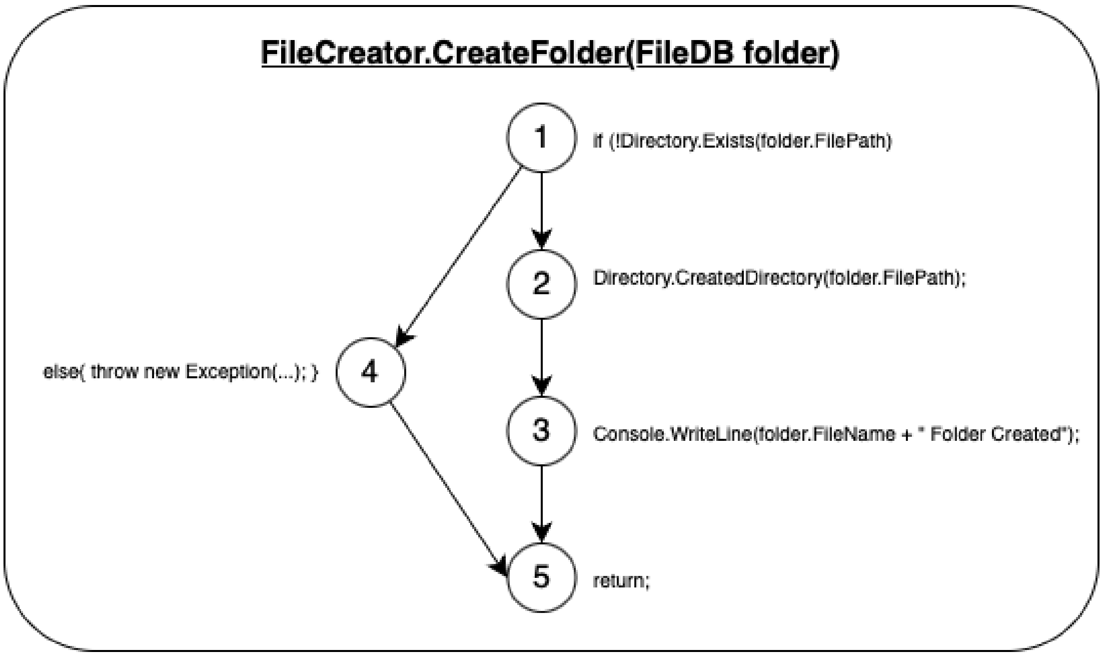

FileCreator.CreateFolder(FileDB folder)
1
if (!Directory.Exists(folder.FilePath)
2
Directory.CreatedDirectory(folder.FilePath);
3
Console.WriteLine(folder.FileName + " Folder Created");
4
else{ throw new Exception(...); }
5
return;

Fig. 40: FileCreatorCreateFolder Control Flow Graph

ID	Input	Expected
1	• folder = <Folder> file { file.FilePath }	Folder object is created at file.FilePath
2	• folder = <Folder> file { file.FilePath *(FilePath exist in file management system)* }	Throw Exception(...);

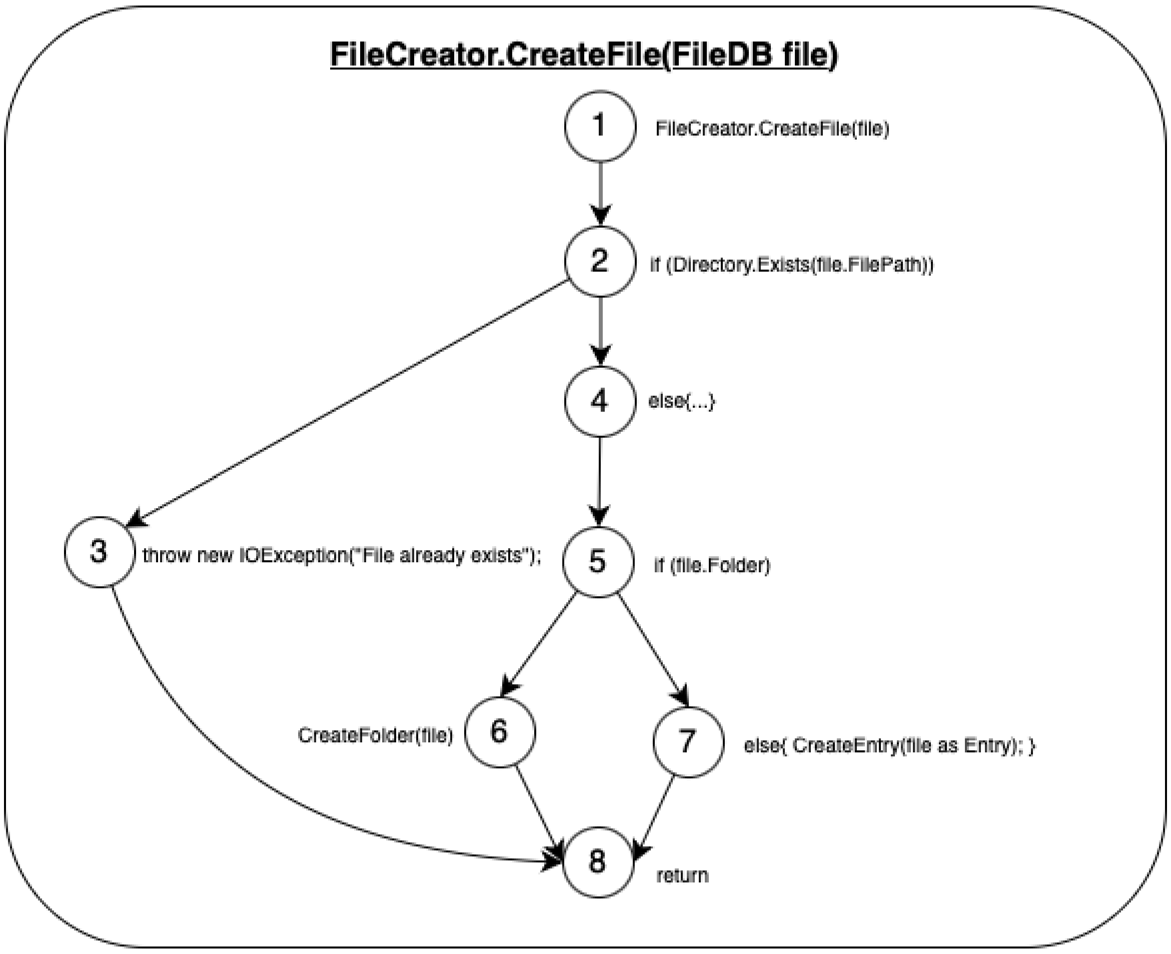

FileCreator.CreateFile(FileDB file)
1
FileCreator.CreateFile(file)
2
if (Directory.Exists(file.FilePath))
4
else{...}
3
throw new IOException("File already exists");
5
if (file.Folder)
CreateFolder(file)
6
7
else{ CreateEntry(file as Entry); }
8
return

Fig. 41: FileCreator CreateFile Control Flow Graph

ID	Input	Expected
1	• file = <Entry> entry{ entry.FilePath = <Path1>} • <Path1> does not exist	Entry instance is created at <Path1>
2	• file = <Folder> folder { folder.FilePath = <Path1>} • <Path1> does not exist	Folder instance is created at <Path1>
3	• file = <Folder> folder { folder.FilePath = <Path1>} • <Path1> Exist!	Throw IOException(...)

Integration Test Plan

These test plans were implemented to ensure that system actions that require interaction between two or more systems perform as intended. They involve the Unit Test Case IDs that indicate which Unit Test Cases are involved in this interaction and should succeed for a satisfactory output for the tested System action.

Action	System Affected	Methods Invoked	Unit Test Case ID
Pre-Test	1. Database.Utilities	Utilities Methods	1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 1.10, 1.11, 1.12, 1.13, 1.14, 1.15, 1.16, 1.17, 1.18, 1.19, 1.20, 1.21, 1.22, 1.23, 1.24, 1.25

App Initialisation	1. Database.Database
	2. Database.ConcurSettings
		ConcurSettings.ConcurSettings()
		ConcurSettings.GetRootDirectory()
		Database.SetSetting()
		Database.GetSetting()
		ConcurSettings.SetRootDirectory()
		Database.setwd()
			10.1, 10.2, 10.3, 10.4,
			11.1, 11.2, 11.3

Creating Concur Settings	1. Database.ConcurSettings
	2. Database.CookieStorage
	3. Database.Database
		ConcurSettings.ConcurSettings()
		ConcurSettings.SetRootDirectory()
		ConcurSettings.CookieStorage.Set()
		CookieStorage.CookieStorage()
		Database.SetSetting()
			9.1,
			10.1, 10.2, 10.5, 11.1, 11.2

Populating File Management	1. Database.Database	Database.GetFileNamesFromWD()	11.1, 11.7, 11.8, 11.9

Navigating (Selecting) File Management	1. Database.Database	Database.FileSelectByFileName(string fileName)
		Database.FileSelectByFilePath(string filePath)
		Database.GetFileNamesFromWD()
			11.1, 11.7, 11.8, 11.9, 11.13, 11.14, 11.15, 11.16

Navigating (Going Back) File Management	1. Database.Database	Database.FileGoBack()
		Database.GetFileNamesFromWD()
			11.1, 11.7, 11.8, 11.9, 11.17, 11.18

Add Folder File Management	1. Database.Database
	2. Database.FolderBuilder
		FolderBuilder.set()
		FolderBuilder.build()
		Database.CreateFile(Folder Object)
			6.1, 6.2, 6.3, 6.4, 6.5, 6.8
			11.1, 11.4, 11.5

Delete File from File Management	1. Database.Database	Database.DeleteFileByFilePath()
		Database.GetFileNamesFromWD()
			11.1, 11.6, 11.7, 11.8, 11.9

Create New Entry	1. Database.Database
	2. Database.MDBuilder
	3. Database.EntryBuilder
	4. Database.Entry
		ReceiptBuilder.set()
		MDBuilder.set()
		MDBuilder.build()
		EntryBuilder.set()
		EntryBuilder.build()
		Entry.AssignRecordID
		Database.CreateFile()
			4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7,
			5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.8, 5.17
			11.1, 11.4, 11.5

Opening Existing Entry from File Management	1. Database.Database
	2. Database.MDAdaptor
		Database.getFileDetailFromName()
		MDAdaptor.ConvertMetaData()
			2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.14
			4.8,
			11.1, 11.19,

Add Record to Existing Entry	1. Database.MDBuilder
	2. Database.Receipt
	3. Database.Entry
		MDBuilder.set()
		MDBuilder.build()
		ReceiptBuilder.set()
		ReceiptBuilder.build()
		EntryBuilder.set()
		EntryBuilder.build()
		--- 	
		Entry.AddRecord()
			2.1, 2.2, 2.3, 2.4, 2.5, 2.14
			4.8,
			5.9,
			11.1, 11.19,

Modify Metadata of Existing Entry	1. Database.MDBuilder
	2. Database.Receipt
	3. Database.Entry
		MDBuilder.set()
		MDBuilder.build()
		ReceiptBuilder.set()
		ReceiptBuilder.build()
		EntryBuilder.set()
		EntryBuilder.build()
		--- 	
		Entry.Metadata.setter()
			2.1, 2.2, 2.3, 2.4, 2.5, 2.14
			4.4, 4.5, 4.6, 4.7, 4.8,
			11.1, 11.19,

Delete Record from Existing Entry	1. Database.Receipt
	2. Database.Entry
		MDAdaptor.ConvertMetaData()
		ReceiptBuilder.set()
		ReceiptBuilder.build()
		--- 	
		Entry.DelRecord()
			2.1, 2.2, 2.3, 2.4, 2.5, 2.14
			4.8,
			5.10, 5.11, 5.12, 5.13,
			11.1, 11.19,

Transfer to Concur	1. ConcurAPI
	2. Selenium Wrapper
	3. Cookie Browser
		ConcurAPI.Initialize()
		ConcurAPI.CreateClaim()
		ConcurAPI.GetReportKey()
		ConcurAPI.CreateExpense()
		ConcurAPI.GetAllExpenses()
		ConcurAPI.UploadImage()
		ConcurAPI.LinkImageToRequest()
			13.1, 13.2, 13.3, 13.4, 13.5, 13.6, 13.7, 13.8

Creating Cookie	1. Database.Cookie
	2. Database.Cookie.CookieBuilder
	3. Database.CookieStorage
		CookieBuilder.set()
		CookieBuilder.build()
		CookieStorage.storeCookie()
			8.1,
			9.2

UI Test Plan

Below shows a list of actions that a user would perform and the intended output of the UI. This test aims to ensure that the UI is able to execute what was requested by the user. This test was done using AppleScript Automator.

Class	Assert Test	Stakeholder	User Action
MainPage	Sorting of names should work for both options	End User	Observe the sorting of files/folders
MainPage	Icons should be displayed correctly	End User	Check if icons are correct for folders and files
MainPage	Time should be displayed accurately	End User	Observe the displayed time
MainPage	Double-tap should function as expected	End User	Perform a double-tap on a file/folder
MainPage	Buttons should work correctly	End User	Click on New folder, new entry, rename, delete, sort, back
MainPage	Folder should be created in the root directory	End User	Check if a new folder is created in the root directory
EntryPage	Entry name change should be successful	End User	Change the entry name and verify the changes
EntryPage	Entry metadata update should be validated	End User	Update the entry metadata and validate the changes
EntryPage	Buttons should work correctly	End User	Click on save entry metadata, transfer to Concur, delete entry, new record, edit record, delete record
EntryPage	Incorrect data types should be handled	End User	Enter incorrect data types in the fields and check the behavior
EntryPage	Table should be updated correctly	End User	Check if the table is updated with the correct data
EntryPage	Existing entry data should load correctly	End User	Pass an existing entry file and check if the data loads properly
EntryPage	EntryName label	End User	Enter a new entry name and

	should update with new name		confirm the update
EntryPage	User should be prompted to upload an image	End User	Click on AddRecordButton and verify the prompt
EntryPage	Image file navigation should work correctly	End User	Select a valid image file and check if it navigates to the record page
EntryPage	ConcurAPI should be initialized successfully	End User	Execute the "Concur_Clicked" method and check the initialization
RecordPage	Buttons should work correctly	End User	Test all buttons and their functionality
RecordPage	"Upload" button should only accept jpg or png files	End User	Attempt to upload different file types and verify behaviour
RecordPage	OCR Scan should work as expected	End User	Test the OCR Scan feature and check the results
RecordPage	Valid values should be entered in each input field	End User	Enter valid data types in each input field
RecordPage	Editing incorrect scans should work properly	End User	Edit incorrect scans and check if they are updated correctly
RecordPage	"Save details" button should go back to EntryPage	End User	Click on "save details" button and verify navigation
RecordPage	Entry should be saved in the database with receipt	End User	Check if the entry and receipt are saved in the database
RecordPage	Checkbox should function as expected	End User	Test the checkbox functionality
RecordPage	Image should be displayed correctly on the left-hand side	End User	Check if the image is displayed correctly on the left-hand side

System Test Plan

This list of test plans ensure the top-level specification/requirements of the system. It makes use of the Use Case Diagrams and indicates the sequential inputs and outputs of the system.

File management system

TEST ID	File_Management_Use_Case_1
Name	View File Directory
Objective	Allows users to view the contents of the file directory
Pre-conditions	The user has access to the file management system The system has been configured with root directory
Post-conditions	Success: The user can see the list of folders and entries in the current directory. Failure: The user is unable to view the list of folders and entries in the current directory due to a system error or lack of access permissions.
Actors	Primary: - User Secondary: - Database
Trigger	User selects the "View File Directory" option / opens the application.
Input	- User selects the "View File Directory" option / opens the application.
Output	- File Directory is displayed with appropriate folders and entries.

TEST ID	File_Management_Use_Case_2
Name	Create Folder
Objective	Allows users to create a new folder
Pre-conditions	Allows users to create a new folder
Post-conditions	Success: A new folder is created in the current directory. Failure: The creation of the folder fails, and no new folder is added to the current directory.

TEST ID	File_Management_Use_Case_2
Actors	Primary: - User Secondary: - Database
Trigger	User selects the "Create Folder" option
Input	- User selects the “Create Folder” option
Output	- A new window is displayed prompting user to name the new folder
Input	- User enters folder name and confirms
Output	- Database creates a new folder with the defined name in the native file directory. - The File Directory is updated with the new folder

TEST ID	File_Management_Use_Case_3
Name	Rename Folder
Objective	Allows users to rename an existing folder
Pre-conditions	- The user has access to the file management system - A folder exists
Post-conditions	Success: The selected folder is renamed. Failure: The renaming of the folder fails, and the selected folder retains its original name.
Actors	Primary: - User Secondary: - Database
Trigger	User selects the "Rename" option for a specific folder
Input	- User selects a folder in the File Directory
Output	- The selected folder is highlighted in the File Directory
Input	- User Selects “Rename”
Output	- A prompt appears for user to key in new folder name
Input	- User enters new folder name and confirms
Output	- The selected folder is renamed and updated in the native file directory as well as the application’s File Directory

TEST ID	File_Management_Use_Case_4
Name	Delete Folder
Objective	Allows users to delete an existing folder
Pre-conditions	The user has access to the file management system and a folder exists
Post-conditions	The selected folder is deleted
Actors	Primary: - User Secondary: - Database
Trigger	User selects the "Delete" option for a specific folder
Input	- User selects a folder in the File Directory
Output	- The selected folder is highlighted in the File Directory
Input	- User selects the "Delete" button
Output	- System prompts user to confirm decision
Input	- User selects the "Confirm" button
Output	- Folder is deleted from the native file directory - Application's File Directory is updated (without the deleted folder) and displayed

TEST ID	File_Management_Use_Case_5
Name	Sort Current Directory
Objective	Allows users to sort the folders and entries in the current directory
Pre-conditions	The user has access to the file management system and folders/entries exist in the current directory
Post-conditions	The folders and entries in the current directory are sorted according to the selected criteria
Actors	Primary: - User Secondary: - Database

TEST ID	File_Management_Use_Case_5
Trigger	User selects the "Sort" option
Input	- User selects the “Sort” Option
Output	- System prompts user for Sort option (Alphabetic or Creation Date)
Input	- User selects one of the sort options
Output	- Application’s File Directory either sorts the files alphabetically or sorts by creation date

TEST ID	File_Management_Use_Case_6
Name	Select Folder
Objective	Allows users to select a specific folder
Pre-conditions	The user has access to the file management system and folders exist
Post-conditions	The selected folder becomes the current directory
Actors	Primary: - User Secondary: - Database
Trigger	User selects a folder from the list of folders in the current directory
Input	- User double clicks a folder
Output	- Application File Directory displays the contents of the folder’s directory

TEST ID	File_Management_Use_Case_7
Name	Create Entry
Objective	Allows users to create a new entry within the selected folder
Pre-conditions	The user has access to the file management system and a folder is selected
Post-conditions	A new entry is created within the selected folder.

TEST ID	File_Management_Use_Case_7
Actors	<u>Primary:</u> - User <u>Secondary:</u> - Database
Trigger	User selects the "Create Entry" option.
Input	- User selects the “Create Entry” option
Output	- A new window is displayed prompting user to name the new entry
Input	- User enters entry name and confirms
Output	- Database creates a new entry with the defined name in the native file directory. - The File Directory is updated with the new entry

TEST ID	File_Management_Use_Case_8
Name	Select Entry
Objective	Allows users to select a specific entry within the selected folder
Pre-conditions	The user has access to the file management system, a folder is selected, and entries exist within the folder
Post-conditions	The selected entry becomes the active entry.
Actors	<u>Primary:</u> - User <u>Secondary:</u> - Database
Trigger	User selects an entry from the list of entries within the selected folder
Input	- User double clicks on an Entry
Output	- The Entry Page is loaded with the selected contents of the selected Entry.

TEST ID	File_Management_Use_Case_9
Name	Rename Entry
Objective	Allows users to rename an existing entry

TEST ID	File_Management_Use_Case_9
Pre-conditions	The user has access to the file management system, a folder is selected, and an entry exists
Post-conditions	The selected entry is renamed
Actors	Primary: - User Secondary: - Database
Trigger	User selects the "Rename" option for a specific Entry
Input	- User selects an Entry in the File Directory
Output	- The selected Entry is highlighted in the Application’s File Directory
Input	- User Selects “Rename”
Output	- A prompt appears for user to key in new entry name
Input	- User enters new entry name and confirms
Output	- The selected entry is renamed and updated in the native file directory as well as the application’s File Directory

TEST ID	File_Management_Use_Case_10
Name	Rename Entry
Objective	Allows users to rename an existing entry
Pre-conditions	The user has access to the file management system, a folder is selected, and an entry exists
Post-conditions	The selected entry is renamed
Actors	Primary: - User Secondary: - Database
Trigger	User selects the "Rename" option for a specific Entry
Input	- User selects an Entry in the File Directory
Output	- The selected Entry is highlighted in the Application’s File Directory

TEST ID	File_Management_Use_Case_10
Input	- User Selects “Rename”
Output	- A prompt appears for user to key in new entry name
Input	- User enters new entry name and confirms
Output	- The selected entry is renamed and updated in the native file directory as well as the application’s File Directory

Entry System

TEST ID	View Entry Use Case
Name	View Entry
Objective	Allow users to see the details of the entry (Receipts of the entry, budget of the entry)
Pre-conditions	- Users can access the File Management system (File Directory) - There are entries in the Database
Post-conditions	Success - The details of the entry is displayed to the user Failure - Details of entry failed to be retrieved
Actors	Primary - User Secondary - Database
Trigger	- User clicks into an entry in the file directory - User made a change to the Entry Information
Input	- User double clicks on a selected Entry in the Application’s file directory
Output	- System extracts the metadata of the selected entry and populates the Entry’s metadata - The Entry Page is populated with the its metadata

TEST ID	Fetch Records Use Case
Name	Fetch Records of Entry
Objective	Fetches the relevant Entry Records from the Database
Pre-condition	- Entry requested exist in Database

TEST ID	Fetch Records Use Case
s	- User has made a request to the Database for an Entry
Post-conditions	Success - The requested Entry record information is passed to the user interface of the Entry System Failure - The requested Entry record information failed to transfer to the user interface of the Entry System
Actors	Primary - Database Secondary - User
Trigger	- The user/Entry System request for a particular Entry's record Information from the database
Input	- User double clicks on a selected Entry in the Application's file directory
Output	- System extracts the metadata of the receipts associated with the selected entry - The Entry Page is populated with the list of receipts based on its metadata

TEST ID	Select Records/Receipts Use Case
Name	Select Records/Receipts
Objective	Allow users to select their desired records/receipts, and perform various actions on it.
Pre-conditions	- User has selected to view an Entry - Information on the requested Entry has been displayed on the Entry System interface
Post-conditions	Success - The record/receipt is shown to be highlighted Failure - The record/receipt is not highlighted
Actors	Primary - User
Trigger	- User clicks on the record/receipt in the records/receipts table
Input	- User selects an existing record/receipt in the list of receipts
Output	- Selected receipt is highlighted
Input	- User selects "Edit Record"

TEST ID	Select Records/Receipts Use Case
Output	- The Record Page is populated with the information of the selected record
Input	- User does changes to some of the fields in the record and selects “Save Details”
Output	- The updated information of the record is saved and reflected back into its metadata - The Entry Page is loaded and displays the updated information of the selected receipt

TEST ID	Transfer to Concur
Name	Select Transfer to Concur
Objective	Allow user to transfer the entry information with its relevant records to the Concur system
Pre-condition s	- User has selected to view an Entry - Information on the requested Entry has been displayed on the Entry System interface
Post-conditio ns	Success - A concur claim and a list of receipt entries would be populated in the system. Failure - API calls return with failure/lack of information
Actors	Primary - User Secondary - EntryPage - SeleniumWrapper
Trigger	- User clicks on the “Transfer to Concur” button near the top of the entry page.
Input	- User selects the “Transfer to Concur” button
Output	- System loads Transfer to Concur system to handle this function. - The Concur Login Page is opened on a browser.

TEST ID	Rename Current Entry Use Case
Name	Rename Current Entry
Objective	Allow user to rename an Entry

TEST ID	Rename Current Entry Use Case
Pre-conditions	- User has selected to view an Entry - Information on the requested Entry has been displayed on the Entry System interface
Post-conditions	Success - The Entry's name is changed Failure - The Entry's name remains unchanged
Actors	Primary - User Secondary - Database
Trigger	- User clicks on the Rename button near the top of the Entry System interface
Input	- User selects on the "Rename" Icon/Button
Output	- System will prompt user for new name for the Entry
Input	- User keys in a new Entry name and confirms
Output	- System updates the Entry's name by updating its metadata - System then displays the new Entry name in the existing Entry page.

TEST ID	Set Current Entry Budget Use Case
Name	Set Current Entry Budget
Objective	Allow user to set the budget for the current Entry
Pre-conditions	- User has selected to view an Entry - Information on the requested Entry has been displayed on the Entry System interface
Post-conditions	Success - The Entry's budget is set Failure - The Entry's budget is not set
Actors	Primary - User Secondary - Database
Trigger	- User selects the Set Budget button in the Entry System interface
Input	- User keys in the budget of the current Entry in the Entry

TEST ID	Set Current Entry Budget Use Case
	Budget input box
Output	- The budget would be reflected on the UI
Input	- User selects “Update Entry MetaData”
Output	- The system informs the user of the saved metadata and updates the Entry’s metadata in the native File Directory - The Entry Page reflects the change in Entry Budget

TEST ID	Add Records/Receipts Use Case
Name	Add Records/Receipt
Objective	Allow users to add records/receipts to the Entry
Pre-conditions	- User has selected to view an Entry - Information on the requested Entry has been displayed on the Entry System interface
Post-conditions	Success - Opens the Add Record user interface Failure - Fails to open the Add Record user interface
Actors	Primary - User Secondary - Database
Trigger	- User clicks on the New Record button near the top of the table of records/receipts
Input	- User selects “New Record” Button
Output	- System prompts user for a receipt image
Input	- User selects an appropriate receipt image
Output	- System loads the Add Record page with the selected receipt image - Entry System loads the Add Record system

Add Record System

TEST ID	Add_record_use_case_1
Name	Upload Receipt
Objective	To add a new expense receipt to the system for claim processing

TEST ID	Add_record_use_case_1
Pre-conditions	- User has a softcopy of the receipt saved on their computers - Receipts are in the format of 'jpg', 'jpeg' or 'png'
Post-conditions	Success The receipt is digitised and relevant data is extracted and saved in the system Failure Retry again
Actors	Primary Actor User Secondary Actor OCR technology
Trigger	User selects "New Record" from the Entry page
Input	User selects an appropriate receipt image
Output	Uploaded receipt is displayed in the Add Record page

TEST ID	Add_record_use_case_2
Name	Display scanned information of receipt
Objective	To use the extracted data to automatically populate the finance claim form in SAP Concur
Pre-conditions	- OCR technology has extracted data from a receipt
Post-conditions	Success SAP Concur claim forms fields are automatically populated with the extracted data Failure Manually key in those fields
Actors	Primary Actor User Secondary Actor - OCR System - SAP Concur System
Trigger	User selects "OCR Scan" from the Add Record page
Input	OCR system extracts the data from the uploaded receipt
Output	Appropriate fields on the receipts are extracted and digitised

TEST ID	Add_record_use_case_3
Name	Enter receipt information manually
Objective	Allow the user to manually input the receipt information
Pre-conditions	User should have uploaded receipt or seen the scanned information of receipt
Post-conditions	Success Receipt information was successfully entered Failure Receipt information was not entered successfully
Actors	Primary Actor User Secondary Actor Website
Trigger	User clicks on “Edit incorrect scans”
Input	User manually enter the necessary fields required for the Concur claim form
Output	User clicks on “Save details” and the extracted data is saved

TEST ID	Add_record_use_case_4
Name	Select Textbox
Objective	Allow users to select a specific region of the scanned receipt image to populate a text field.
Pre-conditions	User has already scanned a receipt and the OCR data has been generated
Post-conditions	Success Text from the selected region of the receipt image replaces the previous field data if parse success Failure Previous field data remains the same
Actors	Primary Actor User Secondary Actor OCR system
Trigger	User clicks on a Textbox area in the receipt (After selecting “OCR” button)

TEST ID	Add_record_use_case_4
Input	User selects a region of the uploaded receipt image
Output	The OCR system retrieves text data from that region

TEST ID	Add_record_use_case_5
Name	Request OCR scan on receipt
Objective	A request is sent to OCR component to scan and analyse the receipt
Pre-conditions	User has uploaded receipt
Post-conditions	Success Upon successful request, receipt is scanned using OCR Failure Upon unsuccessful request, user is asked retry or retake photo or reupload receipt
Actors	Primary Actor User Secondary Actor OCR technology
Trigger	User clicks on “OCR Scan” button
Input	Receipt is scanned using OCR
Output	Appropriate fields on the receipt are extracted and digitised

Transfer System

TEST ID	Transfer_system_use_case_1
Name	Browser opens for user to login
Objective	Extract session cookie token from user login
Pre-conditions	- Entry with receipts loaded - User is on EntryPage
Post-conditions	<u>Success</u> Browser closes <u>Failure</u> Error message pops up showing the reason for failure
Actors	<u>Primary Actor</u> User <u>Secondary Actor</u> SAP Concur Selenium Browser Entry Page ConcurAPI
Trigger	User clicks on “Transfer to API” page on EntryPage
Input	- NIL
Output	- User Cookie

TEST ID	Transfer_system_use_case_2
Name	Create Request/Claim on Concur
Objective	Create a claim on concur using the claim’s metadata
Pre-conditions	- User is logged into Concur - Claim’s metadata is loaded
Post-conditions	<u>Success</u> A claim is created on Concur <u>Failure</u> Error message pops up showing that claim’s metadata information failed to load from database
Actors	<u>Primary Actor</u> User <u>Secondary Actor</u> Database
Trigger	User successfully logs into Concur

TEST ID	Transfer_system_use_case_2
Input	- User cookie
Output	- Claim ID and claim key

TEST ID	Transfer_system_use_case_3
Name	Create an entry/receipt linked to claim
Objective	User submits the entry to SAP Concur based on the receipt
Pre-conditions	- User is logged into Concur - Receipts are selected
Post-conditions	Success Entry created on Concur Failure Error message pops up showing the reason for failure
Actors	Primary Actor User Secondary Actor SAP Concur API (GraphQL)
Trigger	Claim created on Concur
Input	- Claim ID and claim key
Output	- Expense ID

TEST ID	Transfer_system_use_case_4
Name	Upload receipt image to concur
Objective	Upload the image linked to the entry on Concur
Pre-conditions	- User is logged into Concur - Entry is created
Post-conditions	Success Image uploaded onto Concur Failure Error message pops up showing the reason for failure
Actors	Primary Actor User

TEST ID	Transfer_system_use_case_4
	<u>Secondary Actor</u> SAP Concur API (GraphQL)
Trigger	Followed up after creating the expense
Input	- Claim ID, claim key and receipt image
Output	- Image ID

TEST ID	Transfer_system_use_case_6
Name	Submit Entry
Objective	User submits the claim to SAP Concur based on the receipt
Pre-conditions	- User is logged into Concur - Receipts are selected
Post-conditions	<u>Success</u> Success message pops up, closes submit page <u>Failure</u> Error message pops up showing the reason for failure
Actors	<u>Primary Actor</u> User <u>Secondary Actor</u> SAP Concur API (GraphQL)
Trigger	User clicks on submit button on Concur
Input	- Image ID and expense ID
Output	- NIL

Lessons Learnt

Design Research
MAUI was chosen as a framework in the beginning due to its feature of the app being multi-platform, allowing on the spot photo capture of receipts from android devices running the application. However, this framework was chosen with too little research done into the amount of online documentation available. This has been a challenge for the team when it came to the Integration period of the product as there were many undocumented bugs. Through this project, the team learnt a valuable lesson to put emphasis on researching during the design phase to prevent future complications.

Software Design Documentation
The project highlighted the importance of Software Design Documentation. This included the important Unified Modelling Language (UML) such as the Use Case Diagrams, Class Diagrams and Sequence Diagrams. These diagrams were pivotal in ensuring that design requirements and specifications are clearly communicated to all team members. The importance of these diagrams was accentuated when the team had some miscommunications during our development phases. Through these miscommunications, the team realised the importance of clear and detailed UMLs and proceeded to reevaluate the UMLs. Subsequently, design communication amongst team members improved significantly.

Coding Practices & Clean Code
Writing clean code is crucial in group projects, as it plays an important role in preventing confusion while coding practices ensure understanding and standardisation. Due to the scale of this project, there were times when multiple members were working on the same system. Clear function naming as well as the inclusion of docstrings helps with communicating function's specifications. This also helped in maintenance of code in the future and debugging when the time came. As such, the team learnt that clean code is fundamental coding practice to ensure maintainability of the product as well as communication within the team.

Agile
The team chose the Agile framework as the software development process. Taking into account Agile principles, the team decided to use the Scrum Methodology. Projects were split into sprints, each with its own deliverables. This helps the team focus on the task at hand and allows for inter-team communication. At the same time, it gives the team an objective to work towards and keeps the team on schedule. The team also conducted weekly huddles to keep everyone well informed of the progress of the team. This helped minimise the amount of miscommunication within the team.

Throughout the project, the team did face difficulties implementing the Agile Methodology due to it being relatively new to the team. However, regular clarifications of doubts helped to mitigate issues with regards applying this new methodology.

Scope Creep
The team was ambitious. The initial idea of how the final product should be in the early stages of the project was rather simple. But as the weeks went by, the vision of the final product started scaling up. The team was experiencing scope creep from wanting to add more features into our final product. This meant that the team had to be flexible for the envisioned product to be successful. However, due to unforeseen time delays as well as workload and other commitments, the team had to reevaluate their deliverables. Sticking true to the Agile Methodology, the team had regular meetings and decided to cut down on the non-essential features of the product (eg. taking a photo of the receipt) and to push for a stable and working Minimum Viable Product. The team also had to understand their limits and decide if a quality-of-life feature should be implemented based on their capabilities.

Deliverables

Github Repository: https://github.com/kwokkeith/ConcurSolutionz.git
Demonstration Video: https://youtu.be/O2sew6b3bDg