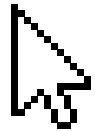
Software engineering Spring 2022:

Clipboard Manager – Clippie



Contributors > Tina Chen Nawaf Otaishan Lisa Moon Dany Sigha

Contents of this presentation

We will take 22 minutes to:

1. Deliver a technical presentation (10 minutes)
    a. A high level walkthrough of our code
    b. Demonstrate and explain with tests of the relevant code sections
2. Lessons learned from our development
    a. Redesign/ refactoring
    b. Problems encountered and solved
    c. Limitations of frameworks used and work around for bugs encountered
3. Demo
    a. Walkthrough of our program and how it functions
    b. Features implemented and limitations
4. A discussion of the process
    a. Our initial process
    b. Our successes and our shortcomings

The clipboard manager

The problem/ the opportunity

Clipboard managers as we know them have the following shortcomings:

- One copy can be stored at a time
- No proper visual interface for the content of the clipboard

Our mission

- Improve the workflow of users
- Keep record of potentially important and sensitive information
- Provide a hassle-free way to swiftly store information

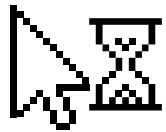
Technical presentation

Clippie is:

- A software application with a graphical user interface component developed using Python's PyQt5 framework
- Target devices include laptops and desktops
- The project will consist of two main parts
  - The visual interface
  - The sqlite3 database

The main features include:

- copy
- paste
- delete
- search
- set "shelf-life"
- group
- protect



01

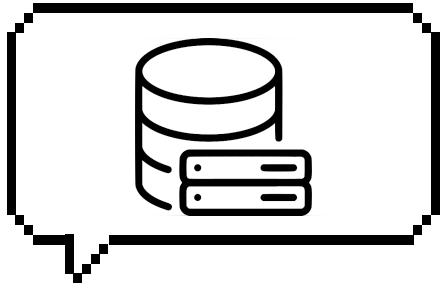
Collecting and storing data

01

Retrieving the clipboard content

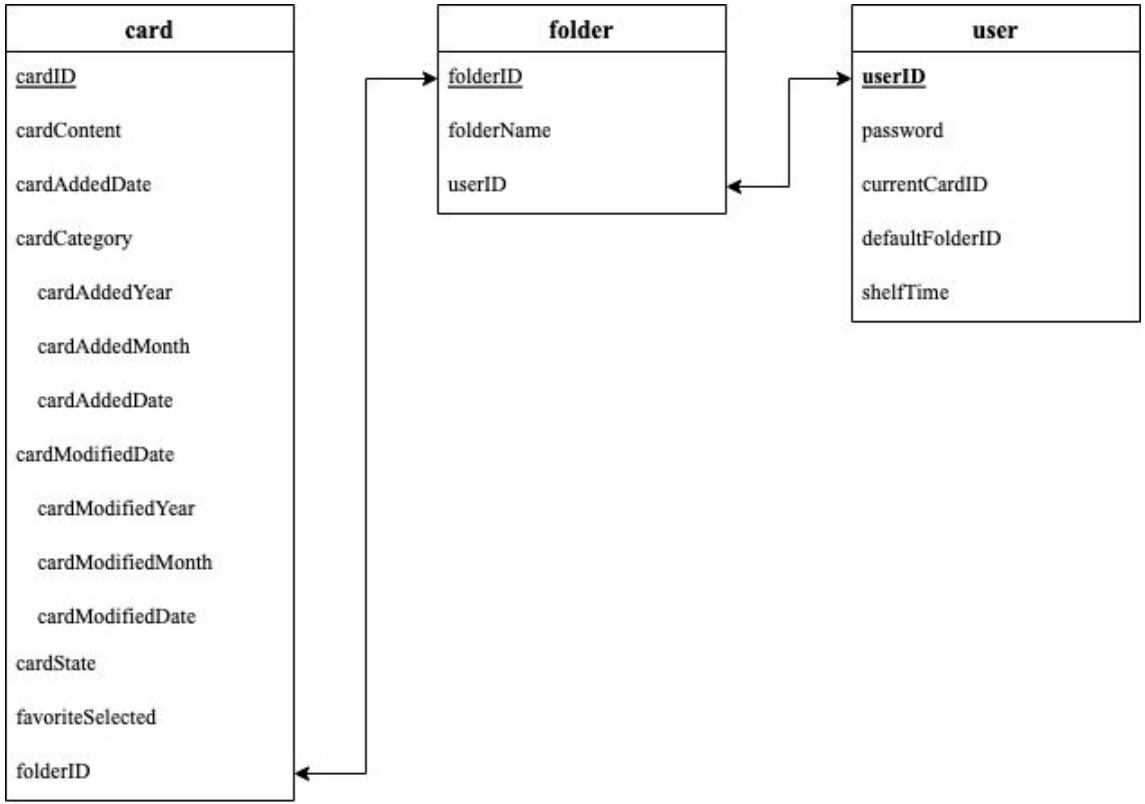
```
class ClipboardManager:

    def __init__(self, card_renderer, dao):  # main,
        self._pb = NSPasteboard.generalPasteboard()
        self._currentCount = NSPasteboard.generalPasteboard().changeCount()
        self._timer = QTimer()  # set up your QTimer
        self._cardRenderer = card_renderer
        self._dao = dao

    def updateUI(self):
        """Check if clipboard item count changed and update interface accordingly by
creating
        new widget"""

        if self._currentCount != self._pb.changeCount():
            data_type = self._pb.types()  # only used to check data type
            card_id = uuid.uuid4().hex

            if NSStringPboardType in data_type:
                pbstring = self._pb.stringForType_(NSStringPboardType)

                if validators.url(pbstring):
                    category = "URL"
                else:
                    category = "Text"

                self._dao.storeCard(card_id, pbstring, category, 0, 0)

                card_generator.CardObject( self._cardRenderer.parent,
                                           self._cardRenderer._position,self._dao,
                                           self._cardRenderer).addToInterface(card_id,
                                          pbstring, category,
                                           datetime.datetime.now(),
                                          datetime.datetime.now(),
                                           1, 0, 0)
```

```
      elif NSTIFFPboardType in data_type:
                    pbimage = self._pb.dataForType_(NSTIFFPboardType)
                     image = Image.open(io.BytesIO(pbimage))

                    filepath = "img_copy/" + str(uuid.uuid4()) + ".png"
                    image.save(filepath, quality=95)
                    category = "Image"
                    self._dao.storeCard(card_id, filepath, category, 0, 0)

                    card_generator.CardObject(self._cardRenderer.parent,

                                              self._cardRenderer._position,
                                              self._dao,
                                              self._cardRenderer).addToInterface(card_id,
                                              filepath, category,
                                              datetime.datetime.now(),
                                              datetime.datetime.now(), 1, 0, 0)

      self._currentCount = self._pb.changeCount()

def manage_clip(self):
   self._currentCount = self._pb.changeCount()
   self._timer.timeout.connect(lambda: self.updateUI()) #connect it to your update function
   self._timer.start(1000)   # set it to timeout in 1 second
```



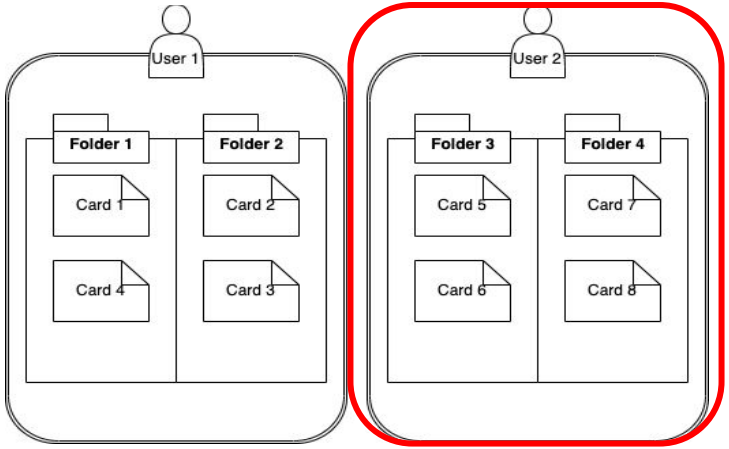
02
Database & Main Functions

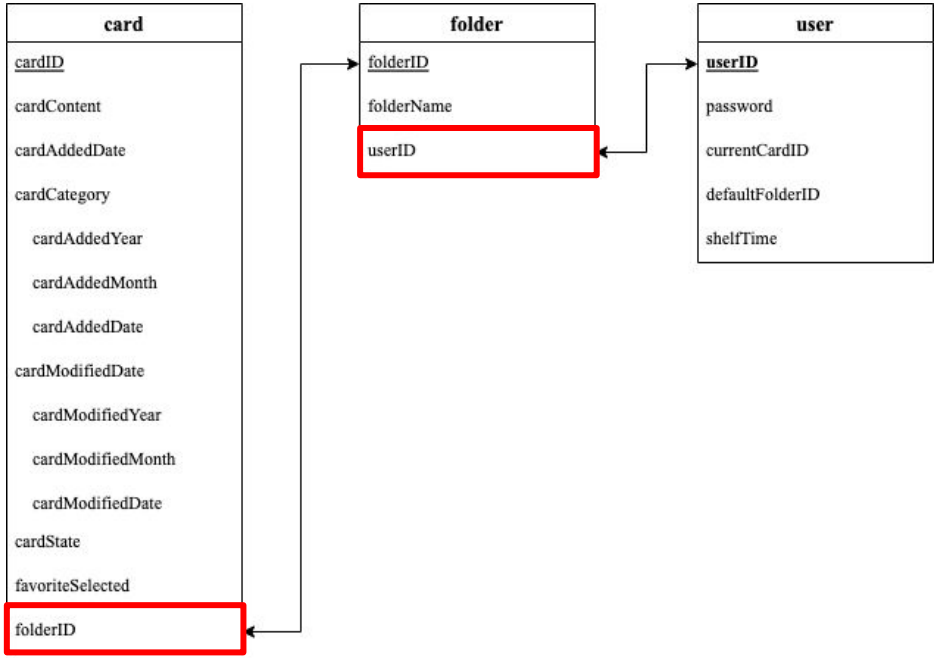
Database Entity



Database Entity

Example Data:





Main Functions

- Adding new data
  - addCard()
  - createFolder()
  - addUser()
- Deleting the data
  - deleteCard()
  - deleteFolder()
- Showing cards
  - searchCard()
  - showCardDataType()
  - showAllFavorite()
- Other Essential Functions
  - pasteCard()
  - automaticDelete_shelftime()
  - favoriteCard()
  - changeStateCard(): Hide and show

AddCard()

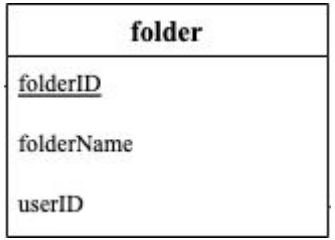
- Explanation:
  - For adding the card data to the database
- Parameters:
  - userID
  - cardID
  - content: card content
  - category: card category
  - sourceApplication: source application (ex. Chrome, Wechat)
  - dataType: (ex. Text, Image, URL)
- Code:

```
def addCard(userID, cardID, content, category, sourceApplication, dataType): # to be
implemented even more later -> copy card
    # find the user's default folder
    cursor.execute('SELECT defaultFolderID FROM user WHERE userID == ' + str(userID) + ";")
    defaultFolderID_loc = cursor.fetchall()[0][0]

    # insert the card data into the database
    cursor.execute('INSERT INTO card(cardID, cardContent, cardCategory, sourceApplication,
    datatype, cardAddedDate, cardModifiedDate, cardState, favoriteSelected, folderID)
    VALUES(?, ?, ?, ?, ?, datetime("now", "localtime"), datetime("now", "localtime"), 1, 0,
    ?)', (cardID, content, category, sourceApplication, dataType, defaultFolderID_loc))
    cursor.execute('UPDATE user SET currentCardID = ' + str(cardID) + ' WHERE userID == ' +
    str(userID) + ';')
```

card
cardID
cardContent
cardAddedDate
cardCategory
cardAddedYear
cardAddedMonth
cardAddedDate
cardModifiedDate
cardModifiedYear
cardModifiedMonth
cardModifiedDate
cardState
favoriteSelected
folderID

createFolder()

- Explanation:
  - For adding the folder data to the database
- Parameters:
  - userID
  - folderName
- Code:

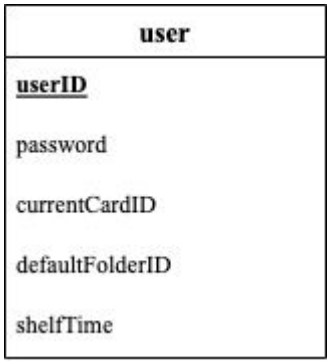



```
# create folder
def createFolder(userID, folderName):
    cursor.execute("INSERT INTO folder(folderName, userID) VALUES(?, ?)", (folderName, userID))
```

addUser()

- Explanation:
  - For adding the user data to the database.
- Parameters:
  - password
- Code:

```
def addUser(password):
    # this sets the currentCardID (the most recent cardID) and the defaultFolderID as -1 temporarily.
    # the currentCardID will be changed if the user adds the first card
    # the defaultFolderID will be changed soon within this function
    cursor.execute('INSERT INTO user(password, currentCardID, defaultFolderID, shelfTime) VALUES(?, -
    1, -1, 24)', (password))

    # get the most new ID of the user.
    cursor.execute('SELECT userID from user')
    new_userID = cursor.fetchall()[-1][0]

    # because the default folder is automatically generated when the new user registers for the
    program create the default folder for the user
    createFolder(new_userID, "Default")

    # set the data of the default folder.
    cursor.execute('SELECT folderID from folder WHERE userID == ' + str(new_userID) + ';')
    new_user_defaultFolderID = cursor.fetchall()[0][0]

    # update the default folder ID for this user.
    cursor.execute('UPDATE user SET defaultFolderID = ' + str(new_user_defaultFolderID) + ' WHERE
    userID == ' + str(new_userID) + ';')
```



deleteCard()

- Explanation:
  - For deleting the card data from the database.
- Parameters:
  - cardID
- Code:

```
# delete
def deleteCard(cardID):
    cursor.execute('DELETE FROM card WHERE cardID == ' + str(cardID) + ';')
```

deleteFolder()

- Explanation:
  - Deletes folder from the database.
- Parameters:
  - folderID
- Code:

```
# delete folder
def deleteFolder(folderID):
    cursor.execute('DELETE FROM folder WHERE folderID == ' + str(folderID) + ';')
```

searchCard()

- Explanation:
  - For showing the content that the user is searching for.
- Parameters:
  - userID
  - searchType: the mode that the user selects to search in.
    - “byKeyword”: search by specific keyword
      - (ex. “Computer science”)
    - “byDate”: search by date parameter
      - (ex. 2013-02-21~2015-04-21)
  - searchContent: the section that is inside the search box.
- Return Value:
  - List of tuples that contains all the data

searchCard()

```
def searchCard(userID, searchType, searchContent):
    # for keyword search please enter the search content as the user enters
    # for date search please keep the format of start_date~end_date
        # ex) '2013-02-21~2015-04-21' when start_date == 2013-02-21 AND end_date == 2015-04-21
    if (searchType == "byKeyword"):
        cursor.execute('SELECT ' + cardAttributes + ' FROM card, folder WHERE folder.userID == ' +
        str(userID) + ' AND cardContent LIKE "%' + str(searchContent) +'%" ORDER BY cardModifiedDate
        DESC;')
        pass
    elif (searchType == 'byDate'):
        startDate = searchContent.split('~')[0]
        endDate = searchContent.split('~')[1]
        cursor.execute('SELECT ' + cardAttributes + ' FROM card, folder WHERE folder.userID == ' +
        str(userID) + ' AND folder.folderID == card.folderID AND cardAddedDate >= datetime("' +
        str(startDate) +'") AND cardAddedDate <= datetime("' + str(endDate) +'");')

    return cursor.fetchall()
```

showCardDataType()

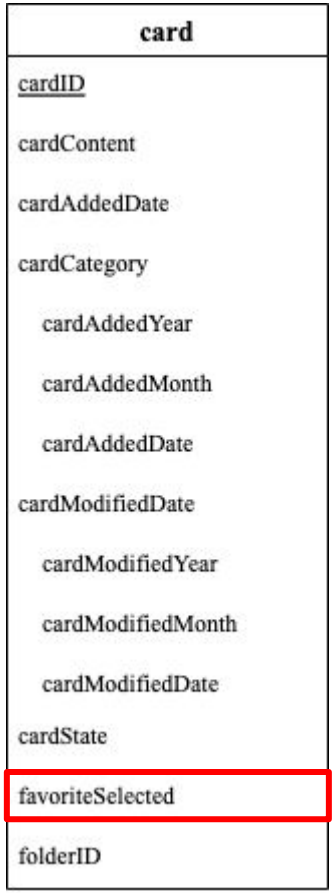
- Explanation:
  - To show all the cards which has specific data type
- Parameters:
  - userID
  - datatype: data type like Text, Image, URL
- Return Value:
  - List of tuples that contains all the data
- Code:

```
def showCardDataType(userID, datatype):
    cursor.execute('SELECT ' + cardAttributes + ' FROM folder, card WHERE userID == ' + str(userID) +
    ' AND folder.folderID == card.folderID AND card.datatype == "' + str(datatype).capitalize() +
    '";')
    return cursor.fetchall()
```

showAllFavorite()

- Explanation:
  - To show all the cards which user pressed favorite/star(to represent favorite option) button.
- Parameters:
  - userID
- Return Value:
  - List of tuples of all the card information that have been favorited by the user
- Code:

```
def showAllFavorite(userID):
    cursor.execute('SELECT ' + cardAttributes + ' FROM folder, card WHERE userID == ' + str(userID) +
    ' AND folder.folderID == card.folderID AND favoriteSelected == 1;')

    return cursor.fetchall()
```



pasteCard()

- Explanation:
  - For pasting option
- Parameters:
  - cardID
- Return Value
  - cardContent (String)
- Code:

```
# paste function to be implemented later
def pasteCard(cardID):
    cursor.execute('SELECT cardContent FROM card WHERE cardID == ' + str(cardID) + ';')
    result = cursor.fetchall()
    return result[0][0]
```

automaticDelete_shelftime()

- Explanation:
  - For deleting the card from the database, but based on the shelf time.
  - Runs everyday as the date changes.
- Parameters:
  - userID
- Code:

```
def automaticDelete_shelftime(userID):
    # this must be called once in a while by the main system for the system to check and delete

    # fetch the shelftime of the user
    cursor.execute('SELECT userID, shelfTime FROM user WHERE userID == ' + str(userID))
    shelftime_months = cursor.fetchall()[0][1]

    # removes all the cards from the database that is older than the shelftime.
    cursor.execute('SELECT cardAddedDate FROM card WHERE cardAddedDate <= datetime(\'now\',
    \'localtime\', \'-' + str(shelftime_months) +' months\');')
```

favoriteCard()

- Explanation:
  - For changing the favorited state of the card
- Parameters:
  - cardID
- Code:

```
# favorite function
def favoriteCard(cardID):
    cursor.execute('SELECT cardID, favoriteSelected FROM card WHERE cardID == ' + str(cardID) + ';')
    if (cursor.fetchall()[0][1]):   # if the card is favorited, set to 0 (not favorited)
        cursor.execute('UPDATE card SET favoriteSelected = 0 WHERE cardID == ' + str(cardID) + ';')
    else:                           # if the card is not favorited, set to 1 (favorited)
        cursor.execute('UPDATE card SET favoriteSelected = 1 WHERE cardID == ' + str(cardID) + ';')
```

changeStateCard()

- Explanation:
  - For changing the showing option of the card (hidden/not hidden)
- Parameters:
  - cardID
- Code:

```
# hide card
def changeStateCard(cardID):
    cursor.execute('SELECT cardID, cardState FROM card WHERE cardID == ' + str(cardID) + ';')
    if (cursor.fetchall()[0][1]):   # if the card is not hidden, set to 0 (hidden)
        cursor.execute('UPDATE card SET cardState = 0 WHERE cardID == ' + str(cardID) + ';')
    else:                           # if the card is hidden, set to 1 (not hidden)
        cursor.execute('UPDATE card SET cardState = 1 WHERE cardID == ' + str(cardID) + ';')
```

Our data access object

```
class password_decorator:
    def __init__(self, function):
        self.function = function
        # self.dao = DataAccessor()
    def password_is_valid(self, pwd):
        h = db.get_password()
        return bcrypt.checkpw(pwd.encode('utf-8'), h)
    def __call__(self, *args, **kwargs):
        pwd = args[0]
        # if self.password_is_valid(pwd):
        if self.password_is_valid(pwd):
            self.function(*args, **kwargs)
            result = True
        else:
            result = False
        return result

@password_decorator
def change_password(oldpwd, newpwd):
    salt = bcrypt.gensalt()
    hashed_pwd =
bcrypt.hashpw(newpwd.encode('utf-8'), salt)
    db.change_password(hashed_pwd)
```

```
class DataAccessor:
    def __init__(self):
        pass
# decryption takes place here
    # UI grabs the decryption key from
user and pass to data access object
    def storeCard(self, card_id, content,
dataType, hideCard, favoriteCard):
        db.addCard(1, card_id, content,
dataType, hideCard, favoriteCard)
        # card.Card(content, dataType,
db.getLastCardID()+1)
    def deleteCard(self, id):
        db.deleteCard(id)
        # db.deleteCard()
    # def updateCard(self):
    #     pass
    def readCard(self, id):
        return db.pasteCard(id)
```

Our data access object

```
def send_email(self):
    # conn = sqlite3.connect('ClipboardManager_DB.db')
    # cursor = conn.cursor()
    # records = cursor.execute("""SELECT email FROM user LIMIT 1""")
    # email = ''
    # for row in records:
    #     email = row[0]
    email = self.get_email()
    port = 465  # For SSL
    smtp_server = "smtp.gmail.com"
    sender_email = "yourclipboardmanager@gmail.com"
    os = platform.system()
    receiver_email = email
    # if os == 'Darwin':
    #     password = "ecqibpmoeknjxwbm"
    # elif os == 'Windows':
    #     password = "qourwmshfkltqnnc"
    password = "ecqibpmoeknjxwbm"
    temp = ''.join(random.choices(string.ascii_lowercase + string.digits, k=10))
    self.set_password(temp)
    message = ("""\
      Subject: Your temporary password.
      This is your temporary password: {}. Please use this password to sign in and
      remember to change password to your own.""".format(temp))
    context = ssl.create_default_context()
    with smtplib.SMTP_SSL(smtp_server, port, context=context) as server:
        server.login(sender_email, password)
        server.sendmail(sender_email, receiver_email, message)
```

The card class

```
import datetime
import uuid


class Card:

    def __init__(self, id, cardContent, cardCategory, addedDate, modifiedDate, cardFolder, hideCard=False, favoriteCard =False):
        # "make a UUID based on the host ID and current time" .hex removes dashes and
        # turns it into a string
        self.cardID = id
        self.cardContent = cardContent
        self.cardCategory = cardCategory
        self.addedDate = addedDate
        self.modifiedDate = modifiedDate
        self.cardFolder = cardFolder
        if hideCard == 0:
            self.hideCard = False
        elif hideCard == 1:
            self.hideCard = True
        else:
            self.hideCard = hideCard

        if favoriteCard == 0:
            self.favoriteCard = False
        elif favoriteCard == 1:
            self.favoriteCard = True
        else:
            self.favoriteCard = favoriteCard
```

02

Our interfaces

01

The main window

The classes involved

Name	Functionality
`class UI(QMainWindow)`	Serves as the main window for our application. All clipboard content is displayed here.
`class CardRenderer`	Keeps track of the position of the cards in the grid layout on the main window. Contains the method that first adds cards to the interface when the program starts.
`class DataAccessor`	Acts as the intermediary between the interface and the database.
`class ClipboardManager`	Keeps track of the changes in the clipboard every second.
`class CardObject`	Creates the card labels on the interface one at a time whenever it is requested to do so.
`class Card`	Creates instances of a card object that stores the info of cards in the database on the fly.
`clipBoardManager_DB.py` (not a class)	Allows our program to store data beyond a single iteration.

```
class UI(QMainWindow):

    ALL_STATE = False
    FAVORITE_STATE = False
    TEXT_STATE = False
    IMAGE_STATE = False
    URL_STATE = False


    def __init__(self):
        super(UI, self).__init__()

        # load the UI
        uic.loadUi("new_main_window.ui", self)

        self.gridLayout2 = QGridLayout(self.scrollAreaWidgetContents_2)
        self.gridLayout2.setSpacing(0)
        self.gridLayout2.setObjectName("gridLayout_2")
        self.gridLayout2.setContentsMargins(0, 0, 0, 0)
        self.gridLayout2.setHorizontalSpacing(0)
        self.gridLayout2.setVerticalSpacing(0)

        self.dao = dao.DataAccessor()
        self.cardMaker = card_generator.CardRenderer(self.gridLayout2, self.dao)

        self.menu = self.findChild(QGridLayout, "gridLayout")

        self.settings_button = self.findChild(QPushButton, "settings_button")
        self.settings_button.clicked.connect(self.openWindow)
        self.left_menu_frame = self.findChild(QFrame, "left_menu_frame")
        self.side_bar_button = self.findChild(QPushButton, "side_bar_button")
        self.side_bar_button.clicked.connect(self.hideMenu)
        self.all_cards = self.findChild(QPushButton, "all_cards")
        self.all_cards.clicked.connect(self.press_all)
        self.favorite_cards = self.findChild(QPushButton, "favorite_cards")
        self.favorite_cards.clicked.connect(self.favorite_it)
        self.text_cards = self.findChild(QPushButton, "text_cards")
        self.text_cards.clicked.connect(self.press_text)
        self.image_cards = self.findChild(QPushButton, "image_cards")
        self.image_cards.clicked.connect(self.press_image)
```

```
def setState(self):
    for i in range(self.menu.count()):
        widget = self.menu.itemAt(i).widget()
        if widget.objectName() == "all_cards" and self.ALL_STATE:
            self.setfocus(widget)
        elif widget.objectName() == "favorite_cards" and self.FAVORITE_STATE:
            self.setfocus(widget)
        elif widget.objectName() == "text_cards" and self.TEXT_STATE:
            self.setfocus(widget)
        elif widget.objectName() == "image_cards" and self.IMAGE_STATE:
            self.setfocus(widget)
        elif widget.objectName() == "link_cards" and self.URL_STATE:
            self.setfocus(widget)
        elif widget.objectName() != "label_4":
            widget.setStyleSheet("QPushButton:hover{\n"
                                 "background-color: rgba(255, 227, 251, 59)\n;"
                                 "}"
                                  u"QPushButton{\n"
                                         "border: none\n"
                                         "})"
def setfocus(self, button):
    button.setStyleSheet("QPushButton{\n"
                         "  background-color: rgba(255, 227, 251, 59)\n;"
                         "}"
                         u"QPushButton{\n"
                         "border: solid\n"
                         "border-color: black\n"
                          "border-width: 1px\n"
                         "}"
                        )
def resetGrid(self):
    self.cardMaker._position = [0, 0, 1, 1]
    layout = self.gridLayout2
    for i in reversed(range(layout.count())):
        widgetToRemove = layout.itemAt(i).widget()
        # remove it from the layout list
        layout.removeWidget(widgetToRemove)
        # remove it from the gui
        widgetToRemove.setParent(None)
        widgetToRemove.deleteLater()
```

```
if __name__ == "__main__":
    app = QApplication(sys.argv)
    # MainWindow = QMainWindow()
    ui = UI()
    # ui.setupUi(MainWindow)
    ui.cardMaker.initializeUI(ui.dao.getAllCards())
    gc = grabClip.ClipboardManager(ui.cardMaker, ui.dao)
    gc.manage_clip()
    # MainWindow.show() - we do not show the window yet

    widget = QStackedWidget()
    window1 = loginPage.login(ui.dao, widget, ui)  # page 1
    window2 = loginPage.newUser(ui.dao, widget, ui)
    widget.addWidget(window1)
    widget.addWidget(window2)
    # widget.setFixedHeight(803)
    # widget.setFixedWidth(789)
    widget.setFixedHeight(493)
    widget.setFixedWidth(370)
    qr = widget.frameGeometry()
    cp = QDesktopWidget().availableGeometry().center()
    qr.moveCenter(cp)
    widget.move(qr.topLeft())

    qr1 = ui.frameGeometry()
    cp1 = QDesktopWidget().availableGeometry().center()
    qr1.moveCenter(cp)
    ui.move(qr1.topLeft())

    widget.show()

    sys.exit(app.exec_())
```

02

The Settings Page

```python
class mainSetting(QDialog):

    def __init__(self, parent, data_access_object, widget):
        super(mainSetting, self).__init__()

        # load ui file
        uic.loadUi("settings1.ui", self)

        # performance
        self._parent = parent
        self.widget = widget
        self.dao = data_access_object
        self.password.clicked.connect(self.gotoPasswordPage)
        self.pushButton_3.clicked.connect(self.gotoShelftime)
        self.pushButton_2.clicked.connect(self.goBack)#
        self.reset.clicked.connect(self.gotoResetPage)

        self.show()

    # goes to the password pages
    def gotoPasswordPage(self):

        if self.dao.get_password_state():
            self.widget.setCurrentIndex(self.widget.currentIndex() + 3)
        else:
            self.widget.setCurrentIndex(self.widget.currentIndex() + 1)

    def gotoShelftime(self):
        self.widget.setCurrentIndex(self.widget.currentIndex() + 2)

    def gotoResetPage(self):
        self.widget.setCurrentIndex(self.widget.currentIndex() + 4)

    def goBack(self):
        self.widget.close()
        self._parent.show()
```

Function:

- Determines the functionality of the widgets (buttons, input boxes, labels)
- Calls the other windows related to other setting features that user may choose to use

```
class setPassword(QDialog):

    def __init__(self, data_access_object, widget):
        super(setPassword, self).__init__()

        uic.loadUi("settings2.ui", self)

        self.dao = data_access_object
        self.widget = widget
        self.pwd1.setEchoMode(QtWidgets.QLineEdit.Password)
        self.pwd2.setEchoMode(QtWidgets.QLineEdit.Password)
        self.pushButton_2.clicked.connect(self.goBack)
        self.pushButton.clicked.connect(self.set_password)

    def goBack(self):
        self.widget.setCurrentIndex(self.widget.currentIndex() - 1)

    def set_password(self):

        pwd1 = self.pwd1.text()
        pwd2 = self.pwd2.text()
        if (pwd1 == pwd2):
            self.dao.set_password(pwd1)
            self.label_4.setText("Password successfully saved")
        else:
            self.label_4.setText("The passwords do not match. Please try again.")
        self.pwd1.clear()
        self.pwd2.clear()
```

Function:

- In charge of loading the interface for the user to set their first password and all the widgets needed to make it happen

```python
class currentPasswordPage(QDialog):
    def __init__(self, data_access_object, widget):
        super(currentPasswordPage, self).__init__()

        uic.loadUi("settings5.ui", self)
        self.dao = data_access_object
        self.widget = widget
        self.pushButton_2.clicked.connect(self.goBack)
        self.pushButton_3.clicked.connect(self.sendEmail)
        self.pushButton.clicked.connect(self.setPassword)
        self.pushButton_4.clicked.connect(self.turnOffPasswordPage)
        self.oldpwd.setEchoMode(QtWidgets.QLineEdit.Password)
        self.pwd1.setEchoMode(QtWidgets.QLineEdit.Password)
        self.pwd2.setEchoMode(QtWidgets.QLineEdit.Password)

    def goBack(self):
        self.widget.setCurrentIndex(self.widget.currentIndex() - 3)

    def setPassword(self):
        oldpwd = self.oldpwd.text()
        pwd1 = self.pwd1.text()
        pwd2 = self.pwd2.text()
        if (pwd1 == pwd2 and self.dao.change_password(oldpwd, pwd1)):
            self.label_4.setText("Password successfully saved")

        elif (pwd1 == pwd2 and not self.dao.change_password(oldpwd, pwd1)):
            self.label_4.setText("The current password entered is incorrect. Please try again.")

        elif (pwd1 != pwd2 and self.dao.change_password(oldpwd, pwd1)):
            self.label_4.setText("The new passwords do not match. Please try again.")

        else:
            self.label_4.setText("One or more input has been incorrect. Please try again.")

        self.oldpwd.clear()
        self.pwd1.clear()
        self.pwd2.clear()
```

Function:

- In charge of loading window for changing the current password on file. Allows user to receive temporary password if forgotten, change password, and disable password.

```
class disablePasswordPage(QDialog):

    def __init__(self, data_access_object, widget):
        super(disablePasswordPage, self).__init__()
        uic.loadUi("disablePwd.ui", self)

        self.widget = widget
        self.dao = data_access_object

        self.enter_2.clicked.connect(self.turnOffPassword)
        self.pushButton_2.clicked.connect(self.goBack)
        self.lineEdit_3.setEchoMode(QtWidgets.QLineEdit.Password)

    def turnOffPassword(self):
        pwd = self.lineEdit_3.text()
        if self.dao.password_is_valid(pwd) == True:
            self.dao.set_password_state(0)
            self.widget.setCurrentIndex(self.widget.currentIndex() - 5)
        else:
            self.label_3.setText("The current password entered is incorrect. Please try again.")
            self.lineEdit_3.clear()



    def goBack(self):
        self.widget.setCurrentIndex(self.widget.currentIndex() - 2)
```

Function:

- Loads interface for disabling password

```
class shelftime(QDialog):
    def __init__(self, data_access_object, widget):
        super(shelftime, self).__init__()

        uic.loadUi("settings3.ui", self)

        self.dao = data_access_object
        self.widget = widget
        self.pushButton_3.clicked.connect(self.goBack)
        self.pushButton.clicked.connect(self.changeShelftime)

    def goBack(self):
        self.widget.setCurrentIndex(self.widget.currentIndex() - 2)

    def changeShelftime(self):
        self.menu = self.findChild(QComboBox, "menu")
        if (self.menu.currentIndex() == 0):
            self.dao.changeShelftime(1)

        elif (self.menu.currentIndex() == 1):
            self.dao.changeShelftime(2)

        elif (self.menu.currentIndex() == 2):
            self.dao.changeShelftime(3)

        elif (self.menu.currentIndex() == 3):
            self.dao.changeShelftime(6)
        else:
            self.dao.changeShelftime(12)

        self.label_3.setText("Shelftime successfully updated.")
```

Function:

- Load interface for user to change shelftime of cards on file

```python
class resetApplication(QDialog):
    def __init__(self, data_access_object, widget, parent):
        super(resetApplication, self).__init__()

        uic.loadUi("settings4.ui", self)

        self.dao = data_access_object
        self.widget = widget
        self.parent = parent
        self.pushButton_2.clicked.connect(self.goBack)
        self.pushButton.clicked.connect(self.resetDB)


    def goBack(self):
        self.widget.setCurrentIndex(self.widget.currentIndex() - 4)

    def resetDB(self):
        self.dao.resetdb()
        self.widget.close()
        self.parent.close()
```

Function:

- Loads the interface to reset the app
- Allows user to reset the whole application and erases all data

03

The Login Page

```
class login(QDialog):
    def __init__(self, data_access_object, widget, MainWindow):
        super(login, self).__init__()

        #load ui file
        uic.loadUi("welcomescreen.ui", self)
        self.MainWindow = MainWindow
        self.widget = widget
        self.dao = data_access_object
        self.login.clicked.connect(self.gotoNextPage)
        self.show()

    def gotoNextPage(self):
        if self.dao.get_user_status() != 0:
            if self.dao.get_password_state() == True:
                self.widget.setCurrentIndex(self.widget.currentIndex() + 2)
            else:
                self.widget.close()
                self.MainWindow.show()
        else:
            self.widget.setCurrentIndex(self.widget.currentIndex() + 1)
```

Function:

- Loads interface that determines whether there's a new user or not in order to load different interfaces under certain circumstances

```
class login2(QDialog):
    def __init__(self, data_access_object, widget, MainWindow):
        super(login2, self).__init__()

        #load ui file
        uic.loadUi("welcomescreen2.ui", self)
        self.dao = data_access_object
        self.flag = False
        self.widget = widget
        self.MainWindow = MainWindow
        self.lineEdit.setEchoMode(QtWidgets.QLineEdit.Password)
        #performance

        self.login.clicked.connect(self.gotoNextPage)
        self.show()

    #goes to the password pages
    def gotoNextPage(self):
        self.lineEdit.setEchoMode(QtWidgets.QLineEdit.Password)
        pwd = self.lineEdit.text()
        if self.dao.password_is_valid(pwd):
            self.widget.close()
            self.MainWindow.show()
        else:
            self.label_3.show()
            self.label_3.setText("Incorrect password. Please try again.")
            self.lineEdit.clear()

    def sendEmail(self):
        self.dao.send_email()
        self.label_3.show()
        self.label_3.setText("A temporary password was sent to the email on file. Please check you
```

Function:

- Loads interface for logging into system under the case that user sets a password

```
class newUser(QDialog):

    def __init__(self, data_access_object, widget, MainWindow):
        super(newUser, self).__init__()

        #load ui file
        uic.loadUi("new_user.ui", self)

        #performance
        self.dao = data_access_object
        self.widget = widget
        self.enter.clicked.connect(self.gotoMainPage)
        self.MainWindow = MainWindow
        self.show()

    # goes to the password pages
    def gotoMainPage(self):
        email1 = self.lineEdit_3.text()
        email2 = self.lineEdit_4.text()
        if (email1 == email2):
            self.dao.create_user(email1)
            self.widget.close()
            self.MainWindow.show()

            # MainWindow = QMainWindow()
            # ui = new_main_window.UI()
            # ui.setupUi(MainWindow)
            # cardMaker = card_generator.CardRenderer(ui.gridLayout2, self.dao)
            # cardMaker.initializeUI(self.dao.getAllCards())
            # gc = grabClip.ClipboardManager(cardMaker, self.dao)
            # gc.manage_clip()
            # self.reject()
            # MainWindow.setVisible(1)
            # widget.setCurrentIndex(widget.currentIndex() + 1)

        else:
            self.label_3.setText("Emails do not match, please try again.")
            self.lineEdit_3.clear()
            self.lineEdit_4.clear()
```

Function:

- Loads interface for entering user info for new users

03

Reflections

Issues Encountered

Problem

- Access the clipboard content (images, text and URLs)
- Create visual cards on the screen and binding them to card objects
- The Qt grid layout structure does not match the requirements of our dynamic card storage system
- Need to filter content displayed by category
- The DB requires card IDs to perform functions in card table but when a new card is stored does not have access to the the AUTOINCREMENT cardID of the DB

Solution

- PyObjective C framework, validators and PILLOW packages
- Use a cardObject class that creates the label and stores the DB card table content on the fly
- We re-initialize the grid by querying the database when a card is removed or added to the system.
- Keep track of the program’s state with class variables in the main window class and a private variable in the cardRender class
- Generate and store our proprietary ID with the UUID4 class

Issues Encountered

Problem

- Working remotely on different time zones
  - New York GMT-4
  - Korea GMT+9
  - Saudi Arabia GMT+3
- We weren't experienced with Github
- We were not familiar with pyqt5
- Covid impacted one of our team member

Solution

- We had to delegate task and hold meetings in unfavorable times for some members
- Over time we were able to optimize how we handle our version control system.
- We had to take some time to learn before during the project
- We familiarize ourselves with sqlite to make minor adjustments to our database during the final stretch.

Issues Encountered

Problem

- We had trouble determining which window we were going to make the parent window from which the other windows popped up from. We were stuck for a while and didn't know how to generate the main window after the user logs in

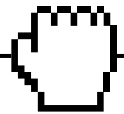
Solution

- We made the main app interface our parent window from which the other windows will appear from. The login page is called in the main function of the main window class and the main window is passed into the classes of the other sub windows in order to control the state of the main window accordingly

Issues Encountered

Problem

- Google was not allowing our app to have access to gmail
- I was having trouble sending emails because only gmails (even our school even didn't work) could receive the emails sent
- My windows were not maintaining the layout when the window dimensions changed



Solution

- I changed the settings of my google account to allow third party access through designated app passwords
- I changed the port I used and it worked! The port also operate using SSL, which makes the email sending more secured
- There are widgets called frames and layouts in Qt Designer that allowed me to keep the layout of widgets no matter how the window changes

Process

What worked:
Delegating tasks
Pair programming

What didn’t:

Weekly communication

Week 6						
Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
	What to prepare before: Individual progress			What to prepare before: Research and ideas		
	To During Weekly LA Meeting: Ask questions			To During Team meeting: Discuss task and next steps		
	To do after: Prepare for team meeting			To do after: Dany Preparing for technical presentation Tina: Work on search function Nawaf: Copy implementation and sql 3 for python		

Week summary:
We have divided the tasks to work on getting done as soon as possible. (Ideally before Tuesday)

Top issues and concerns:
Getting the project up and running and start implementing features

Help needed or Pair up needed: