Tzeyee Koh

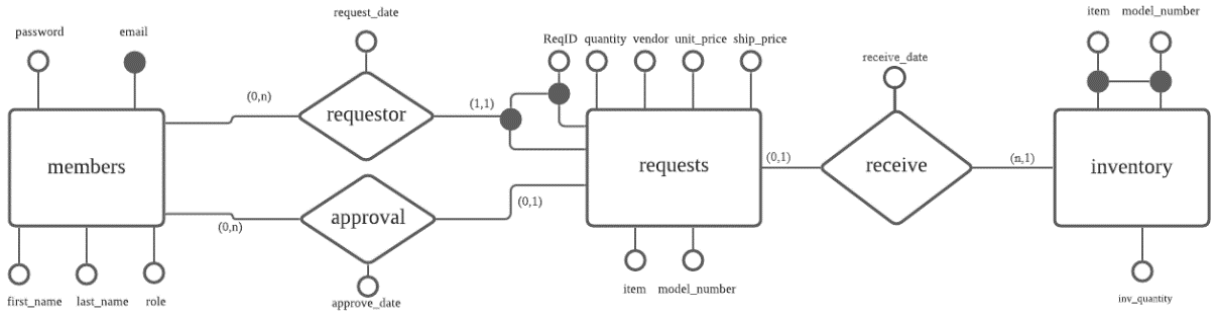
Procurement Requests System

The system facilitates the request and approval procurements (e.x. tools, electronics, raw materials) for a robotics club. The system has both club staff and club member profiles. A club member submits a request to purchase an item. A staff member evaluates the procurement request and gives approval for the student to make the purchase. The system allows users to track the approval status of the requests, view the types of items being procured, and check whether the item has been received. Furthermore, the website will have pages for users to view the total number of items received from orders and track the club's expenses. Once an item has been shipped and received, it will become part of the club's inventory, where inventory are identified by the item and model number. Administrators can create, modify and delete entries.

Real-world Application

In my robotics club, members procure items for the fabrication of their robots. In order to receive reimbursement or use club funds for purchasing these items, members have to seek approval from staff. Originally, this was done through email. However, due to the growing number of procurement requests, a system needs to be put in place where both staff and students can submit, view, and track the status of the procurement requests. This can help track our club's procurement expenses and streamline the process of submitting and receiving approval to purchase items.

ER Diagram



Schema

```
CREATE TABLE members(
    first_name VARCHAR(64) NOT NULL,
    last_name VARCHAR(64) NOT NULL,
    email VARCHAR(64) NOT NULL PRIMARY KEY,
    role VARCHAR(16) NOT NULL,
    password VARCHAR(64) NOT NULL
);

CREATE TABLE requests(
    ReqID NUMERIC NOT NULL PRIMARY KEY,
    request_date DATE NOT NULL,
    item VARCHAR(512) NOT NULL,
    model_number VARCHAR(16) NOT NULL,
    quantity NUMERIC NOT NULL,
    vendor VARCHAR(128) NOT NULL,
    unit_price NUMERIC NOT NULL,
    ship_price NUMERIC NOT NULL,

    receive_date DATE, -- If NULL, not received/ if NOT NULL, part of inventory

    requestor VARCHAR(64) REFERENCES members(email) NOT NULL,
    CONSTRAINT CHK_request CHECK (unit_price>0 AND ship_price>0 AND quantity>0)
);

CREATE TABLE inventory(
    item VARCHAR(512) NOT NULL,
    model_number VARCHAR(64) NOT NULL,
    inv_quantity NUMERIC NOT NULL,
    PRIMARY KEY (item, model_number)
);

CREATE TABLE approval(
    staff_approver VARCHAR(64) NOT NULL,
    approve_date DATE NOT NULL,
    ReqID  NUMERIC NOT NULL,
    PRIMARY KEY(staff_approver, ReqID),
    FOREIGN KEY (staff_approver) REFERENCES members(email),
    FOREIGN KEY (ReqID) REFERENCES requests(ReqID)
);
```

Tables & Functional Dependencies

$R_{overall}$ = {first_name, last_name, email, role, password,
ReqID, request_date, item, model_number, quantity, vendor, unit_price, ship_price,
receive_date, requestor,
inv_quantity,
staff_approver, approve_date }

$\sum_{overall}$ = {{ReqID} → {request_date, item, model_number, quantity, vendor, unit_price, ship_price, receive_date, requestor}
{email} → {first_name, last_name, role, password}
{item, model_number, receive_date} → {inv_quantity}
{staff_approver, ReqID } → {approve_date}
}

$R_{members}$ = {first_name, last_name, email, role, password}
$\sum_{members}$ = {{email} → {first_name, last_name, role, password}}
Table in BCNF. {email} is a superkey and primary key of the table.

$R_{requests}$ = {ReqID, request_date, item, model_number, quantity, vendor, unit_price, ship_price, receive_date, requestor}
$\sum_{requests}$ = {{ReqID} → {request_date, item, model_number, quantity, vendor, unit_price, ship_price, receive_date, requestor}}
Table in BCNF. {ReqID} is a superkey and primary key of the table

$R_{inventory}$ = {item, model_number, inv_quantity}
$\sum_{inventory}$ = {{item, model_number} → {inv_quantity}}
Table in BCNF. {item, model_number} is a superkey

$R_{approval}$= {staff_approver, approve_date, ReqID}
$\sum_{approval}$ = {{ staff_approver, ReqID } → {approve_date}}
Table in BCNF. { staff_approver, ReqID } is a superkey

Entries in all tables can be uniquely identified by their primary keys.

Originally, I decided to create an inventory table instead of using the “CREATE VIEW” query as I wanted to display the inventory quantities as a bar chart on the inventory page. However, when implementing this, I found that the chart displayed was hard to read and not user friendly. As such, I reverted to displaying the table of items and quantities instead.

Features

Login Sessions: Users can log into the system with their email and password. System will label the user's session as Student or Staff depending on their role. Only users that are logged in can access the features of the database. Anonymous users can view the data, but cannot make changes. Logging out ('closeportal') will end the user's session and set the session to default session to an anonymous user.

```
def login(request):
    ##Retreive Input
    email = request.GET.get('email')
    password = request.GET.get('password')

    try:
        query ="""SELECT m.role FROM members m
                WHERE email = '%s' AND password = '%s' """ % (email, password)
        c = connection.cursor()
        c.execute(query)

        role = c.fetchall()
        user = role[0][0]
    except IndexError:
        return render(request, 'AppPR/error.html')

    if user == 'Staff':
        request.session['usertype'] = user
        return HttpResponseRedirect('/AppPR')
    elif user == 'Student':
        request.session['usertype'] = user
        return HttpResponseRedirect('/AppPR')
```

```
def index(request):
    try:
        if request.session['usertype'] == 'Staff':
            return render(request,'AppPR/staffportal.html')
        elif request.session['usertype'] == 'Student':
            return render(request,'AppPR/studentportal.html')
        else:
            request.session['usertype'] = 'Anon'
            return render(request, 'AppPR/index.html')
    except KeyError:
        request.session['usertype'] = 'Anon'
        return render(request, 'AppPR/index.html')

def closeportal(request):
    request.session['usertype'] = 'Anon'
    return HttpResponseRedirect('/AppPR')
```

Submit requests and Receive items: Students and staff can raise procurement requests and receive items. A unique ReqID is generated for each new request by incrementing the previous ReqID by +1. This is to ensure that all ReqIDs are unique and requests can be identified. Members can receive an item by entering its unique ReqID. This will update the request status to received and automatically insert the current date. These are the main features available to all members.

```
def newReq(request):

    ##Retrieve new ReqID
    newReqID = "SELECT MAX(ReqID)+1 As newReqID FROM requests"
    # The connection object.
    r = connection.cursor()
    # Execute query by connection object
    r.execute(newReqID)

    newReqID = r.fetchall()


    ##Retrieve input
    ReqID = newReqID[0][0]
    req_date = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    item = request.GET.get('item')
    model_number = request.GET.get('model_number')
    quantity = request.GET.get('quantity')
    vendor = request.GET.get('vendor')
    unit_price = request.GET.get('unit_price')
    ship_price = request.GET.get('ship_price')
    requestor = request.GET.get('requestor')
    ## Query
    query ="""INSERT INTO requests (ReqID, request_date, item, model_number, quantity, vendor, unit_price, ship_price, receive_date, requestor)
            VALUES (%s,'%s','%s','%s',%s,'%s',%s,%s,NULL,'%s')""" % (ReqID, req_date, item, model_number, quantity, vendor, unit_price, ship_price, requestor)
    # The connection object.
    c = connection.cursor()
    # Execute query by connection object
    c.execute(query)

    return requests(request)
```

```
def recItem(request):
    ##Retreive Input
    ReqID = request.GET.get('ReqID')
    currdate = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    query ="""UPDATE requests
            SET receive_date = '%s'
            WHERE ReqID = %s
            AND ReqID IN(
                SELECT a.ReqID FROM approval a
            )""" % (currdate, ReqID)
    c = connection.cursor()
    c.execute(query)

    #Print Updated

    return requests(request)
```

View Requests: Under “Requests”, users can view all raised requests and the status of the request. Furthermore, the user can filter the requests by its status (see view.py for queries) and search requests by the name of the requestor. A view is created in order to display the name of the requestor and the status of the request. This makes the table more readable since requestors are naturally identified by name, not email, and the status of a request references the approval table. Overall, by creating a view, the results page is able to display a more readable table that is easy to understand.

```
def requests(request):
    ## Query
    query ="""DROP VIEW IF EXISTS view_requests;
            CREATE VIEW view_requests AS
            SELECT r.ReqID, m.first_name ||' '||m.last_name AS name, r.request_date, r.item, r.model_number, r.quantity, r.vendor, r.unit_price, r.ship_price, (CASE
                                                                                                                                                                WHEN r.receive_date IS NULL
                                                                                                                                                                AND r.ReqID NOT IN (
                                                                                                                                                                SELECT a.ReqID FROM approval a)
                                                                                                                                                                THEN CAST('Pending Approval'AS VARCHAR(64))
                                                                                                                                                                WHEN r.receive_date IS NULL THEN CAST('Pending Delivery' AS VARCHAR(64))
                                                                                                                                                                ELSE CAST('Received: ' || r.receive_date AS VARCHAR(64))
                                                                                                                                                                END) AS status
            FROM requests r, members m
            WHERE r.requestor = m.email
            ORDER BY request_date DESC;
            SELECT * FROM view_requests;"""
    # The connection object.
    c = connection.cursor()
    # Execute query by connection object
    c.execute(query)
    # Fetch all the rows. fetchall() returns a list of tuples.
    requests = c.fetchall()
    requests_dict = {'requests': requests}
    return render(request,'AppPR/requests.html',requests_dict)
```

Add Members: This is a feature only available to staff. It allows staff to add new members to the database, giving them access to features. This utilizes a simple “INSERT INTO” SQL query. In order to maintain the consistency of the data in the members table, only “Student” and “Staff” roles are available when entering data in the HTML form.

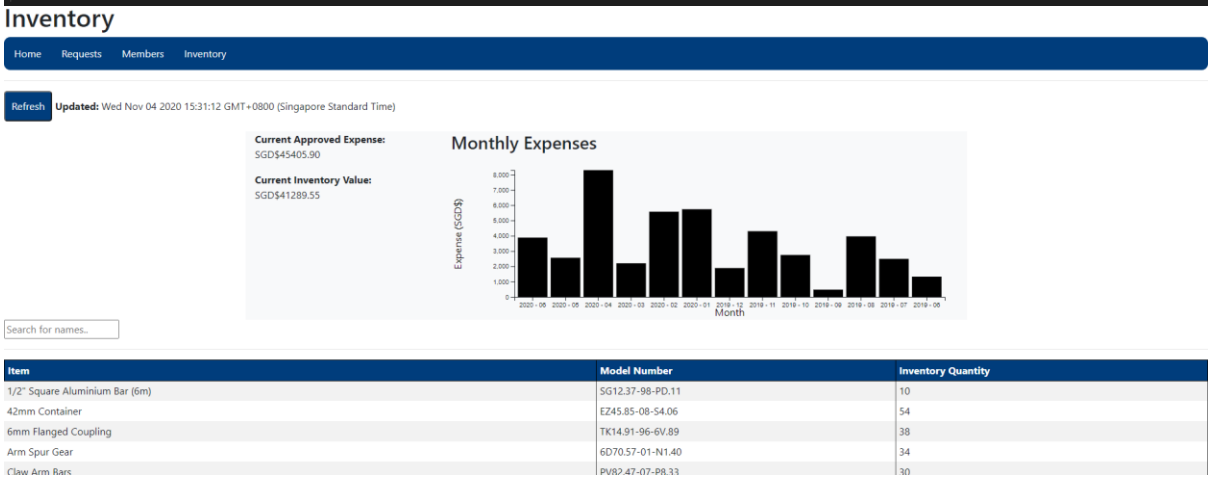
Approve Requests: This features is for staff to approve any member’s request. It utilizes an “INSERT INTO” statement combined with a “SELECT” clause in order to check that the approver for the request is different from the requestor and verify the password of the approver. This is to prevent staff from approving their own requests.

```
def aprReq(request):
    ##Retreive Input
    ReqID = request.GET.get('ReqID')
    currdate = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    staff_approver = request.GET.get('staff_approver')
    staff_password = request.GET.get('staff_password')


    query ="""INSERT INTO approval(staff_approver, approve_date, ReqID)
            SELECT m.email,'%s', r.ReqID
            FROM members m, requests r
            WHERE m.role = 'Staff' AND m.email = '%s' AND r.ReqID = %s AND r.requestor <> m.email AND m.password = '%s';""" % (currdate, staff_approver, ReqID, staff_password)
    c = connection.cursor()
    c.execute(query)

    #Print Updated
    return requests(request)
```

Inventory: This page allows users to view the current stock of items received. As there are common items requested by members, the inventory page sums up the quantity of all the similiar items requested and received from all requests. In order to ensure the inventory quantity is always up to date, the table is updated every time the user enters the page. Furthermore, a timestamp on the page shows the user the last time the page was updated and a refresh button allows the user to update the table manually.
In addition, users can view the current approved club expenses, track the monthly spending, and view the current inventory value. The approved club expenses and monthly spending are the total amount spent on purchasing and shipping items from requests that have been approved by staff. The current inventory value is the value of all items received and in the inventory.

```
def inventory(request):
    ## Query
    query ="""DELETE FROM inventory;
            INSERT INTO inventory(item, model_number, inv_quantity)
            SELECT item, model_number, SUM(quantity) AS inv_quantity FROM requests
            WHERE receive_date IS NOT NULL
            GROUP BY (item, model_number);
            SELECT * FROM inventory
            ORDER BY item;"""
    # The connection object.
    c = connection.cursor()
    # Execute query by connection object
    c.execute(query)
    # Fetch all the rows. fetchall() returns a list of tuples.
    inv = c.fetchall()

    expense_query = """SELECT SUM(r.ship_price)+SUM(r.quantity * r.unit_price) AS expense
                    FROM requests r
                    WHERE EXISTS(
                    SELECT a.ReqID FROM approval a
                    WHERE a.ReqID = r.ReqID);"""
    e = connection.cursor()
    e.execute(expense_query)
    exp = e.fetchall()

    inv_value_query = """SELECT SUM(r.quantity * r.unit_price) AS expense
                        FROM requests r
                        WHERE receive_date IS NOT NULL;"""
    iv = connection.cursor()
    iv.execute(inv_value_query)
    inv_value = iv.fetchall()

    ##Run Exp Chart
    expChart(request)

    return render(request, 'AppPR/inventory.html', {'records': inv, 'expense': exp, 'inv_value': inv_value})
```

```
def expChart(request):
    ## Query
    query ="""SELECT to_char(a.approve_date, 'YYYY - MM'), SUM((r.unit_price * r.quantity) + r.ship_price) AS total_cost FROM approval a, requests r
            WHERE a.ReqID = r.ReqID
            GROUP BY to_char(a.approve_date, 'YYYY - MM')
            ORDER BY to_char(a.approve_date, 'YYYY - MM') DESC"""
    # The connection object.
    c = connection.cursor()
    # Execute query by connection object
    c.execute(query)
    # Fetch all the rows. fetchall() returns a list of tuples.
    expc = c.fetchall()
    return JsonResponse(expc, safe=False)
```



Accessibility
http://it2002-23-i.comp.nus.edu.sg:8080/Project/AppPR/
local: http://127.0.0.1:8000/AppPR/

Video Demonstration
https://youtu.be/DvGnaM_4dsM