Project 3 Report

Contributor	Contribution
Olivia Fishbough (27338647)	Part 2: code, video, readme
Itzel Medina (29751993)	Part 1: code, video, readme
Analicia Sosa (29523167)	Part 3: code, video, readme Bonus option

Project Video: https://youtu.be/qMkSM3wgeP4

Phase 1

1. Installing docker desktop should automatically install docker-compose as well which is what the project relies on to run. To check to make sure docker compose is successfully installed, run the following command in a command prompt window:

```
C:\Users\olivi>docker-compose version
Docker Compose version v2.31.0-desktop.2
```

2. Create a project directory and navigate to it

```
PS C:\Users\mmedi\OneDrive\Documents\GitHub\CECS327-Project-3\p2p-node> mkdir p2p-node
```

```
PS C:\Users\mmedi\OneDrive\Documents\GitHub\CECS327-Project-3\p2p-node> cd p2p-node
```

3. Create app.py. This is the file that will contain the two flask endpoints to handle file uploads and downloads.

app.py

```python

# Create the Flask app instance
app = Flask(__name__)

# Make sure the storage folder exists to save uploaded files
os.makedirs('./storage', exist_ok=True)


# Route to handle file uploads
@app.route('/upload', methods=['POST'])
def upload_file():
    # Get the uploaded file from the request
    file = request.files['file']

    # Save the file to the local storage folder
    file.save(f"./storage/{file.filename}")

    # Return a JSON response confirming the upload
    return jsonify({
        "status": "uploaded",
        "filename": file.filename
    })


# Route to handle file downloads
@app.route('/download/<filename>', methods=['GET'])
def download_file(filename):
    # Send the requested file from the storage folder
    return send_from_directory('./storage', filename)


# Run the app on all network interfaces, port 5000
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)

```

4. Create the dockerfile

```
FROM python:3.10-slim

WORKDIR /app

COPY app.py .

RUN pip install flask

# Create storage directory
RUN mkdir -p /app/storage

EXPOSE 5000

CMD ["python", "app.py"]

```

5. Create the file we will be uploading and name it mydoc.txt

```
This is a test file for our project
```

6. Build the docker image

```
PS C:\Users\mmedi\OneDrive\Documents\GitHub\CECS327-Project-3\p2p-node> docker build -t p2p-node .
```

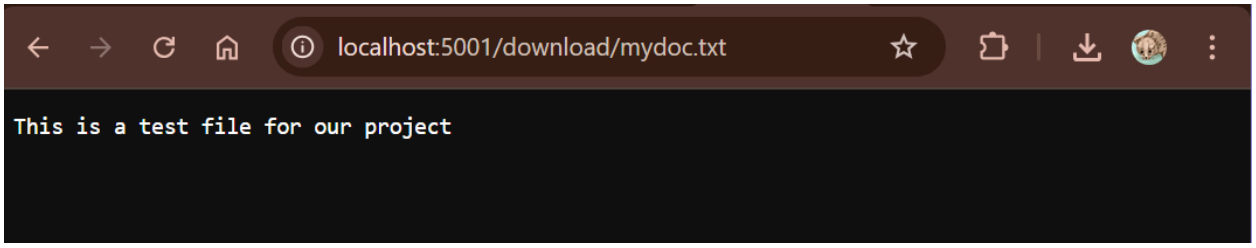
7. Mount a volume in Docker so the files persist:

```
PS C:\Users\mmedi\OneDrive\Documents\GitHub\CECS327-Project-3\p2p-node> docker run -d -p 5001:5000 -v
"${PWD}/storage:/app/storage" --name node1 p2p-node
```

8. Run the following command to upload the file

```
PS C:\Users\mmedi\OneDrive\Documents\GitHub\CECS327-Project-3\p2p-node> curl.exe -F "file=@mydoc.txt"
http://localhost:5001/upload
```

9. Open your browser and go to http://localhost:5001/download/mydoc.txt



Phase 2

10. Add the following code to our app file to add flask endpoints that will allow us to store information directly in our docker container. The function store key function looks for a json type parameter to store a key : value pair of information and stores the information in the key storage. The get_value function takes the parameter of a key and it returns the information connected to it from the saved storage.

```python
# Set up flask endpoint to read values
@app.route('/kv', methods=['POST'])
def store_key_value():
    data = request.get_json()
    key = data.get('key')
    value = data.get('value')

    if key and value:
        key_storage[key] = value
        return jsonify({"status": "success", "key": key, "value": value})
    else:
        return jsonify({"status": "error", "message": "Invalid input"}), 400

# Recieve key value being stored
@app.route('/kv/<key>', methods=['GET'])
def get_value(key):
    value = key_storage.get(key)
    if value:
        return jsonify({"key": key, "value": value})
    else:
        return jsonify({"error": "Key not found"}), 404
```

11. To store the information use the following command:

```
PS C:\Users\olivi\OneDrive\Desktop\School\CECS 327\CECS327-Project-3\p2p-node> Invoke-RestMethod -Uri http://localhost:5001/kv -Met
hod POST -Headers @{ "Content-Type" = "application/json" } -Body '{"key":"color","value":"blue"}'

key   status  value
---   ------  -----
color success blue
```

This is a bit different from the command given to us in the assignment instructions; this is due to using windows command prompt which doesn't directly support curl commands.

12. To retrieve the information to check that the information correctly saved, enter the following command:

```
PS C:\Users\olivi\OneDrive\Desktop\School\CECS 327\CECS327-Project-3\p2p-node> curl.exe http://localhost:5001/kv/color
{"key":"color","value":"blue"}
```

And it should list all the saved keys and their respective information that they store ont he local network.

Phase 3

1. The initial attempt at solving this portion of the assignment was to hardcode a peers list in the environment variables of each node object.

2. Create a docker-compose file to construct many different nodes at once

```
▷ Run Service
node2:
  build: .
  container_name: node2
  ports:
    - "5002:5000"
  environment:
    - ROLE=peer
  networks:
    p2pnet:
  depends_on:
    - bootstrap

▷ Run Service
node3:
  build: .
  container_name: node3
  ports:
    - "5003:5000"
  environment:
    - ROLE=peer
  networks:
    p2pnet:
  depends_on:
    - bootstrap

▷ Run Service
```

3. Update the kv get and set methods to query peers if it can't find the requested data and implement hashing

```python
@app.route('/kv/<key>', methods=['GET'])
def get_value(key):
    responsible_node = hash_key_to_node(key)

    if responsible_node != own_url:
        try:
            res = requests.get(f"{responsible_node}/kv/{key}")
            return res.json(), res.status_code
        except Exception as e:
            return jsonify({"error": f"Forwarding failed: {str(e)}"}), 500

    value = key_storage.get(key)
    if value:
        return jsonify({"key": key, "value": value})
    else:
        return jsonify({"error": "Key not found"}), 404

```

```python
# Helper function to find responsible node
def hash_key_to_node(key):
    if not peers:
        return own_url  # Fallback for single node

    hash_val = hashlib.sha1(key.encode()).hexdigest()
    sorted_peers = sorted(peers | {own_url}, key=lambda p: hashlib.sha1(p.encode()).hexdigest())
    for peer in sorted_peers:
        if hashlib.sha1(key.encode()).hexdigest() <= hashlib.sha1(peer.encode()).hexdigest():
            return peer
    return sorted_peers[0]

```

4. The node were able to successfully store key/value pairs and fetch them from other nodes.

Option 1

1. Implement pings

```python
# Health check endpoint
@app.route('/ping', methods=['GET'])
def ping():
    sender = request.args.get("sender")
    if sender and sender != own_url and sender not in peers:
        peers.add(sender)
        print(f"Added peer from ping: {sender}")
        # Notify sender back
        try:
            requests.post(f"{sender}/peers", json={"peers": [own_url]})
        except Exception as e:
            print(f"Failed to notify sender {sender}: {e}")

    return jsonify({"status": "alive", "node": own_url})

```

2. In order to make the system able to automatically add and remove new and inactive peers, we needed to rework the hardcoded peers list. We settled for having the nodes register themselves with the bootstrap node when they start up. They retrieve the peers list from the bootstrap node and regularly ping those nodes. When the bootstrap node goes down, they're still able to ping one another and monitor the active nodes.
3. Add in logic to monitor peer activity

p2p-node > app.py

```python
def monitor_peers():
    while True:
        dead_peers = []
        for peer in list(peers):
            if peer == own_url:
                continue
            try:
                res = requests.get(f"{peer}/ping", params={"sender": own_url}, timeout=3)
                if res.status_code != 200:
                    raise Exception(f"Non-200 status: {res.status_code}")
                try:
                    data = res.json()
                except ValueError:
                    raise Exception("Invalid JSON in ping response")

                if data.get("status") != "alive":
                    raise Exception("Ping status not alive")
            except Exception as e:
                print(f"[{own_url}] Removing unresponsive peer: {peer} ({e})")
                dead_peers.append(peer)
        for dead in dead_peers:
            peers.discard(dead)
            for peer in list(peers):
                try:
                    requests.post(f"{peer}/peers", json={"remove": [dead]})
                except Exception as e:
                    print(f"[{own_url}] Failed to notify {peer} about dead peer {dead}: {e}")

```