```
# Three Tier App Hello World

## Bussines Logic Definition

#
    #
    inc: N
    ->
    add() ->

    #
    get_value() <-
    N <-


## Tasks

### 1. REST endpoint + Bussines Logic

#### Bussines Logic Implementation
  + Programming Languague: Python
  + File Name: logic.py
  + Implement the Bussines Logic in a class.
  + The value will be a record (k-v pair) on a Redis
database:
      key: 'the_value', value: unsigned_int
  + Requirements:
   - thread-safe

#### REST endpoint for the Bussines Logic
  + Programming Languague: Python
  + File Name: main_endpoint.py
  + POST /add/<inc> will call add()
  + GET /value/> will call get_counter()

#### Docker
   + Write a Dockerfile to create a docker container
   where the endpoint and the logic must be installed

### 2. Front End

#### Bussines Logic Proxy
```

+ Programming Language: JavaScript
+ Write a class with methods of the bussines logic.
+ Class name: Bussines_Logic_Proxy
+ Each method will send a REST request to the corresponding bussines method.

User Interface (UI)
+ Write a nice html page that allows the user call the bussines logic mehtods.
+ Programming language: HTML, CSS, Javascript
+ File Name: user_interface.html
+ The UI must call the apropriate method in the Bussiness Logic Proxy

HTML server
+ Programming Languague: Python
+ File Name: main_server.py
+ Write a html server from which the UI can be downloaded
+ Port Number: 9999

Docker
+ Write a Dockerfile to create a docker container from where the UI can be downloaded through main_server.py

3. Database Container
+ The database used by the Bussines Logic will be Redis
+ If necessary, create a DockerFile to configure a docker container

4. Orchestration
+ Write a docker-compose.yml to launch the three containers:
- bussines logic endpoint
- user interface
- database

Below is a complete, **ready-to-use** implementation for the Three-Tier “Hello World” App:

Business Logic → REST API → Front End → Redis DB → Docker → Docker Compose

You can copy-paste these into your project folders.

Directory structure recommendation:

cpp Copy code

```
three-tier-app/
 ├── logic/
 │    ├── logic.py
 │    ├── main_endpoint.py
 │    └── Dockerfile
 ├── frontend/
 │    ├── main_server.py
 │    ├── user_interface.html
 │    └── Dockerfile
 ├── redis/
 │    └── Dockerfile (optional)
 └── docker-compose.yml
```

---

1. BUSINESS LOGIC + REST ENDPOINT

logic.py

Thread-safe Redis counter with lock.

(Using redis-py + threading.Lock)

python Copy code

```python
# logic.py
import redis
import threading

class BusinessLogic:
    def __init__(self, host="redis", port=6379):
        self.redis = redis.Redis(host=host, port=port, decode_responses=Tru
        self.lock = threading.Lock()
        self.key = "the_value"

        # Initialize value if missing
        with self.lock:
            if not self.redis.exists(self.key):
```

```
            if not self.redis.exists(self.key):
                self.redis.set(self.key, 0)

    def add(self, inc: int):
        with self.lock:
            current = int(self.redis.get(self.key))
            new_val = current + inc
            self.redis.set(self.key, new_val)
            return new_val

    def get_value(self):
        with self.lock:
            return int(self.redis.get(self.key))
```

---

main_endpoint.py

Simple REST API with Flask.

python Copy code

```python
# main_endpoint.py
from flask import Flask, jsonify
from logic import BusinessLogic

app = Flask(__name__)
logic = BusinessLogic()

@app.post("/add/<int:inc>")
def add(inc):
    new_val = logic.add(inc)
    return jsonify({"new_value": new_val})

@app.get("/value")
def value():
    v = logic.get_value()
    return jsonify({"value": v})

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8000)
```

Dockerfile (Business Logic container)

dockerfile Copy code

```
# logic/Dockerfile
FROM python:3.11

WORKDIR /app

COPY logic.py .
COPY main_endpoint.py .

RUN pip install flask redis

EXPOSE 8000

CMD ["python", "main_endpoint.py"]
```

---

2. FRONT END

Bussines_Logic_Proxy.js

(Embedded inside HTML, but providing separately for clarity)

javascript Copy code

```
class Bussines_Logic_Proxy {
    constructor(base_url) {
        this.base_url = base_url;
    }

    async add(inc) {
        const res = await fetch(`${this.base_url}/add/${inc}`, { method: "
        return await res.json();
    }

    async get_value() {
        const res = await fetch(`${this.base_url}/value`);
        return await res.json();
    }
}
```

user_interface.html

html Copy code

```
<!DOCTYPE html>
<html>
<head>
    <title>Three Tier App</title>
    <style>
        body { font-family: Arial; margin: 40px; background: #f3f3f3; }
        .container { background: white; padding: 20px; border-radius: 6px;
        button { padding: 10px; margin-top: 10px; width: 100%; }
        input { width: 100%; padding: 10px; margin-top: 10px; }
        h2 { text-align: center; }
    </style>
</head>
<body>

<div class="container">
    <h2>Three Tier App</h2>

    <input id="inc" type="number" placeholder="Increment value">
    <button onclick="callAdd()">Add</button>

    <button onclick="callGet()">Get Current Value</button>

    <h3 id="result"></h3>
</div>

<script>
class Bussines_Logic_Proxy {
    constructor(base_url) { this.base_url = base_url; }

    async add(inc) {
        const res = await fetch(`${this.base_url}/add/${inc}`, { method: "
        return await res.json();
    }
    async get_value() {
        const res = await fetch(`${this.base_url}/value`);
        return await res.json();
    }
```

```
}

const proxy = new Bussines_Logic_Proxy("http://logic:8000");

async function callAdd() {
    const inc = document.getElementById("inc").value;
    const res = await proxy.add(inc);
    document.getElementById("result").innerText = "New value: " + res.new_v
}

async function callGet() {
    const res = await proxy.get_value();
    document.getElementById("result").innerText = "Current value: " + res.v
}
</script>

</body>
</html>
```

---

main_server.py

Simple file server.

python Copy code

```
# main_server.py
from flask import Flask, send_from_directory

app = Flask(__name__)

@app.get("/")
def serve_ui():
    return send_from_directory(".", "user_interface.html")

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=9999)
```

Dockerfile (Frontend container)

dockerfile Copy code

```
# frontend/Dockerfile
FROM python:3.11

WORKDIR /app

COPY main_server.py .
COPY user_interface.html .

RUN pip install flask

EXPOSE 9999

CMD ["python", "main_server.py"]
```

---

3. DATABASE CONTAINER (Redis)

You can use the official Redis image, but if required:

redis/Dockerfile (optional)

dockerfile Copy code

```
FROM redis:7
```

If unnecessary, omit this folder and use `image: redis:7` in docker-compose.

---

4. DOCKER COMPOSE

docker-compose.yml

yaml Copy code

```
version: "3.9"

services:

  redis:
    image: redis:7
    container_name: redis_db
    ports:
```

```
    ports:
      - "6379:6379"

  logic:
    build: ./logic
    container_name: business_logic
    ports:
      - "8000:8000"
    depends_on:
      - redis

  frontend:
    build: ./frontend
    container_name: ui_server
    ports:
      - "9999:9999"
    depends_on:
      - logic
```

RUNNING EVERYTHING

From root folder:

bash Copy code

```
docker-compose up --build
```

Then:

UI:

http://localhost:9999

REST API:

http://localhost:8000/value
POST → http://localhost:8000/add/5

Redis stores:

cpp Copy code

```
key: the_value
```

```
value: <unsigned int>
```

If you'd like, I can also provide:

✅ Kubernetes version
✅ Unit tests
✅ README.md
✅ Logging middleware
✅ TypeScript front-end version

Just tell me!