# **IEEE Project Report Template**

### **Title Page**

* **Project Title:** 471 Programming Assignment
* **Authors:** Tiffany Tran, Michael Rivera, Tauheed Janjua
* **Affiliation:** California State University, Fullerton
* **Course:** Computer Networks
* **Programming Language:** Python
* **Date:** 11/29/2024

### **Abstract**

This project implements a File Transfer Protocol (FTP) simulation using Python. It facilitates client-server communication over a TCP connection, allowing file uploads, downloads, and directory listing. The implementation highlights Python's role in networking and showcases the use of socket programming to simulate real-world protocols.

### **Keywords**

Python, Computer Networks, Socket Programming, FTP Simulation, Client-Server Model

### **1. Introduction**

1.1 **Background**

* The project focuses on the client-server model using Python's socket library to simulate FTP functionality. Networking is crucial for data exchange, and this project demonstrates fundamental aspects of network programming.

1.2 **Objective**

* To build a Python-based system that supports file transfers between a client and a server.

1.3 **Scope**

* The system supports commands like uploading, downloading, listing server files, and disconnecting. Advanced features like encryption or multi-client support are not included.

### **2. Related Work**

* Python's socket library is central to this project, providing APIs for network connections. Comparisons are made with traditional FTP clients and servers, focusing on basic functionalities.)

### **3. System Design and Architecture**

3.1 **System Overview**

* The project includes a server that manages file storage and a client that interacts via predefined commands.

3.2 **Architecture Diagram**

* A diagram illustrating client-server interaction will be included.

3.3 **Modules**

* Client module: Handles user commands and connects to the server.
* Server module: Manages incoming connections and executes commands.

### **4. Implementation**

4.1 **Technologies Used**

* Socket programming via Python's socket library
* Python 3.124.2 **Key Features**
* Highlight the main features of your project.
  + Commands: GET (download), PUT (upload), LS (list files), and QUIT (disconnect).
  + Server runs first and listens on a specific port for client connections.

4.3 **Code Overview**

* Key snippets include socket initialization, data transfer, and command handling

4.4 **Algorithms**

* Basic file I/O operations, command parsing, and network message handling.

### **5. Results and Discussion**

5.1 **Testing Scenarios**

* File upload/download between client and server.
* Handling invalid commands and client disconnection

5.2 **Performance Metrics**

* Basic latency and successful file transmission tests.

5.3 **Challenges and Resolutions**

* Network delays and handling file overwrites were challenging but resolved by command validation.

5.4 **Discussion**

* The solution is functional for small-scale file transfers but lacks scalability and encryption.

### **6. Conclusion and Future Work**

6.1 **Conclusion**

* The project demonstrates foundational FTP operations and highlights Python's flexibility in network programming.

6.2 **Future Work**

* Potential improvements include adding SSL for secure transmission, supporting concurrent clients, and expanding file management features.

**8. Appendices**

* Please see GitHub/Zip File

### **Example Sections for Python Networking:**

#### **Key Implementation Example:**

python

Copy code

# Simple Python socket server example

import socket

# Create a socket object

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# Bind to a local address and port

server\_socket.bind(('127.0.0.1', 8080))

# Start listening for connections

server\_socket.listen(5)

print("Server is listening...")

while True:

client\_socket, addr = server\_socket.accept()

print(f"Connection from {addr}")

data = client\_socket.recv(1024).decode()

print(f"Received: {data}")

client\_socket.send("Acknowledged".encode())

client\_socket.close()