Advanced Firewall Application Documentation

1. Components

1.1 Singleton Pattern: `Settings` Class

The `Settings` class ensures that only one instance of the class exists. This instance holds the default policy for the firewall.

```
python
Copy code
class Settings:
    _instance = None

    def __new__(cls):
        if cls._instance is None:
            cls._instance = super(Settings, cls).__new__(cls)
            cls._instance.default_policy = "allow"
        return cls._instance

    def get_default_policy(self):
        return self.default_policy
```

1.2 Strategy Pattern: `FirewallStrategy` Abstract Base Class

The `FirewallStrategy` abstract base class defines the interface for all filter strategies.

```
python
Copy code
from abc import ABC, abstractmethod

class FirewallStrategy(ABC):
    @abstractmethod
    def filter(self, packet: dict) -> bool:
        pass
```

1.3 Concrete Strategies

1.3.1 `IPFilter` Class

Filters packets based on IP addresses using whitelists and blacklists.

```
python
Copy code
class IPFilter(FirewallStrategy):
    def __init__(self):
        self.whitelist = set()
        self.blacklist = set()

    def add_to_whitelist(self, ip):
        self.whitelist.add(ip)

    def add_to_blacklist(self, ip):
        self.blacklist.add(ip)

    def filter(self, packet: dict) -> bool:
        ip = packet.get("ip")
        if ip in self.blacklist:
            return False
        if self.whitelist and ip not in self.whitelist:
            return False
        return True
```

1.3.2 `PortFilter` Class

Filters packets based on allowed ports.

```
python
Copy code
class PortFilter(FirewallStrategy):
    def __init__(self):
        self.allowed_ports = set()

    def allow_port(self, port):
        self.allowed_ports.add(port)

    def filter(self, packet: dict) -> bool:
        port = packet.get("port")
        if self.allowed_ports and port not in self.allowed_ports:
            return False
        return True
```

1.3.3 `ProtocolFilter` Class

Filters packets based on allowed protocols.

```
python
Copy code
class ProtocolFilter(FirewallStrategy):
    def __init__(self):
        self.allowed_protocols = set()

    def allow_protocol(self, protocol):
        self.allowed_protocols.add(protocol)

    def filter(self, packet: dict) -> bool:
        protocol = packet.get("protocol")
        if self.allowed_protocols and protocol not in
self.allowed_protocols:
            return False
        return True
```

1.4 Factory Pattern: `StrategyFactory` Class

The `StrategyFactory` class provides a method to instantiate different filter strategies.

```
python
```

```
Copy code
class StrategyFactory:
    @staticmethod
    def get_strategy(strategy_type: str):
        if strategy_type == "ip_filter":
            return IPFilter()
        elif strategy_type == "port_filter":
            return PortFilter()
        elif strategy_type == "protocol_filter":
            return ProtocolFilter()
        else:
            raise ValueError(f"Unknown strategy type: {strategy_type}")
```

2. Main Script

The main script sets up the firewall filters and processes packet inputs from the user.

```
python
Copy code
def main():
    import re

    def is_valid_ip(ip):
        pattern = re.compile(r'^(\d{1,3}\.){3}\d{1,3}$')
        return pattern.match(ip) is not None

    settings = Settings()
    default_policy = settings.get_default_policy()

    ip_filter = StrategyFactory.get_strategy("ip_filter")
    port_filter = StrategyFactory.get_strategy("port_filter")
    protocol_filter = StrategyFactory.get_strategy("protocol_filter")

    ip_filter.add_to_whitelist("192.168.1.1")
    ip_filter.add_to_blacklist("10.0.0.1")

    port_filter.allow_port(80)
    port_filter.allow_port(443)

    protocol_filter.allow_protocol("TCP")
    protocol_filter.allow_protocol("UDP")

    allowed_protocols = {"TCP", "UDP"}

    print("Advanced Firewall is running...")
    print("Enter packet details in the format 'ip, port, protocol' (e.g.,
'192.168.1.1, 80, TCP')")
    while True:
        packet_input = input("Enter packet details (or type 'exit' to
quit): ")
        if packet_input.lower() == 'exit':
            break
        try:
            errors = []
            parts = packet_input.split(', ')
            if len(parts) != 3:
                errors.append("Invalid packet format. Please enter the
details in the format 'ip, port, protocol' (e.g., '192.168.1.1, 80,
TCP').")
```

```
            else:
                ip, port, protocol = parts

                if not is_valid_ip(ip):
                    errors.append("Invalid IP format. Please enter a valid
IP address (e.g., 192.168.1.1).")

                try:
                    port = int(port)
                except ValueError:
                    errors.append("Invalid port number. Please enter a
numeric port value.")

                if protocol not in allowed_protocols:
                    errors.append(f"Invalid protocol. Please enter a valid
protocol (e.g., {', '.join(allowed_protocols)}).")

            if errors:
                for error in errors:
                    print(error)
                continue

            packet = {
                "ip": ip,
                "port": port,
                "protocol": protocol
            }

            if ip_filter.filter(packet) and port_filter.filter(packet) and
protocol_filter.filter(packet):
                print("Packet allowed")
            else:
                print("Packet denied")
        except Exception as e:
            print("An unexpected error occurred:", e)

if __name__ == "__main__":
    main()
```

3. Web Interface

3.1 HTML and CSS

The web interface allows users to input packet details and see if the packet is allowed or denied.

```
html
Copy code
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Firewall Application</title>
    <style>
        @keyframes backgroundAnimation {
            0% { background-color: #71b7e6; }
            25% { background-color: #9b59b6; }
            50% { background-color: #e67e22; }
```

```css
    75% { background-color: #2ecc71; }
    100% { background-color: #71b7e6; }
}

@keyframes lineAnimation {
    0% { transform: translateY(-100%); }
    100% { transform: translateY(100%); }
}

body {
    font-family: Arial, sans-serif;
    animation: backgroundAnimation 10s infinite;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    margin: 0;
    position: relative;
    overflow: hidden;
}

.lines {
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    pointer-events: none;
}

.line {
    position: absolute;
    width: 1px;
    height: 100%;
    background-color: rgba(255, 255, 255, 0.2);
    animation: lineAnimation 5s linear infinite;
}

.line:nth-child(1) { left: 10%; }
.line:nth-child(2) { left: 20%; }
.line:nth-child(3) { left: 30%; }
.line:nth-child(4) { left: 40%; }
.line:nth-child(5) { left: 50%; }
.line:nth-child(6) { left: 60%; }
.line:nth-child(7) { left: 70%; }
.line:nth-child(8) { left: 80%; }
.line:nth-child(9) { left: 90%; }

.container {
    background-color: #fff;
    border-radius: 12px;
    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
    padding: 30px;
    max-width: 400px;
    width: 100%;
    text-align: center;
    border: 2px solid #3498db;
    z-index: 1;
    position: relative;
}
```

```css
h1 {
    color: #3498db;
    margin-bottom: 20px;
}

.welcome-message {
    color: #555;
    border: 2px solid #3498db;
    padding: 10px;
    border-radius: 8px;
    margin-bottom: 30px;
    font-size: 18px;
}

.form-group {
    margin-bottom: 15px;
    text-align: left;
}

label {
    display: block;
    font-weight: bold;
    margin-bottom: 5px;
    color: #555;
}

input {
    width: calc(100% - 20px);
    padding: 10px;
    border: 2px solid #ccc;
    border-radius: 6px;
    transition: border-color 0.3s;
    font-size: 14px;
}

input:focus {
    border-color: #3498db;
    outline: none;
}

button {
    width: 100%;
    padding: 12px;
    background-color: #3498db;
    border: none;
    border-radius: 6px;
    color: white;
    font-size: 16px;
    cursor: pointer;
    transition: background-color 0.3s;
}

button:hover {
    background-color: #2980b9;
}

.result {
    margin-top: 20px;
    font-weight: bold;
    font-size: 18px;
    padding: 10px;
```

```
            border-radius: 6px;
        }

        .result.allowed {
            color: #27ae60;
            background-color: #eafaf1;
        }

        .result.denied {
            color: #c0392b;
            background-color: #fdecea;
        }

        .watermark {
            position: absolute;
            bottom: 10px;
            right: 10px;
            font-size: 12px;
            color: rgba(0, 0, 0, 0.5);
        }
    </style>
</head>
<body>
    <div class="lines">
        <div class="line"></div>
        <div class="line"></div>
        <div class="line"></div>
        <div class="line"></div>
        <div class="line"></div>
        <div class="line"></div>
        <div class="line"></div>
        <div class="line"></div>
        <div class="line"></div>
    </div>
    <div class="container">
        <h1>Firewall Application</h1>
        <p class="welcome-message">Welcome to Firewall Application</p>
        <form id="firewallForm">
            <div class="form-group">
                <label for="ip">IP Address:</label>
                <input type="text" id="ip" name="ip" required>
            </div>

            <div class="form-group">
                <label for="port">Port:</label>
                <input type="number" id="port" name="port" required>
            </div>

            <div class="form-group">
                <label for="protocol">Protocol:</label>
                <input type="text" id="protocol" name="protocol" required>
            </div>

            <button type="submit">Check Packet</button>
        </form>
        <div class="result" id="result"></div>
        <div class="watermark">Made By: Alwin, Sijo, Avinash, Asish</div>
    </div>

    <script>
        class IPFilter {
            constructor() {
                this.whitelist = new Set();
                this.blacklist = new Set();
            }

            addToWhitelist(ip) {
                this.whitelist.add(ip);
            }

            addToBlacklist(ip) {
                this.blacklist.add(ip);
            }

            filter(packet) {
                const ip = packet.ip;
                if (this.blacklist.has(ip)) {
                    return false;
                }
                if (this.whitelist.size > 0 && !this.whitelist.has(ip)) {
                    return false;
                }
                return true;
            }
        }

        class PortFilter {
            constructor() {
                this.allowedPorts = new Set();
            }

            allowPort(port) {
                this.allowedPorts.add(port);
            }

            filter(packet) {
                const port = packet.port;
                if (this.allowedPorts.size > 0 &&
!this.allowedPorts.has(port)) {
                    return false;
                }
                return true;
            }
        }

        class ProtocolFilter {
            constructor() {
                this.allowedProtocols = new Set();
            }

            allowProtocol(protocol) {
                this.allowedProtocols.add(protocol);
            }

            filter(packet) {
                const protocol = packet.protocol;
                if (this.allowedProtocols.size > 0 &&
!this.allowedProtocols.has(protocol)) {
                    return false;
                }
                return true;
            }
        }

        const ipFilter = new IPFilter();
        ipFilter.addToWhitelist("192.168.1.1");
        ipFilter.addToBlacklist("10.0.0.1");

        const portFilter = new PortFilter();
        portFilter.allowPort(80);
        portFilter.allowPort(443);

        const protocolFilter = new ProtocolFilter();
        protocolFilter.allowProtocol("TCP");
        protocolFilter.allowProtocol("UDP");

        setTimeout(function() {
            const welcomeMessage = document.querySelector(".welcome-
message");
            if (welcomeMessage) {
                welcomeMessage.style.display = "none";
            }
        }, 6000);

        document.getElementById("firewallForm").addEventListener("submit",
function(event) {
            event.preventDefault();

            const ip = document.getElementById("ip").value;
            const port = parseInt(document.getElementById("port").value);
            const protocol = document.getElementById("protocol").value;

            const packet = {
                ip: ip,
                port: port,
                protocol: protocol
            };

            const resultElement = document.getElementById("result");
            if (ipFilter.filter(packet) && portFilter.filter(packet) &&
protocolFilter.filter(packet)) {
                resultElement.textContent = "Packet allowed";
                resultElement.className = "result allowed";
            } else {
                resultElement.textContent = "Packet denied";
                resultElement.className = "result denied";
            }
        });
    </script>
</body>
</html>
```

4. Usage

1. **Run the Python Script:**

   Execute the Python script to start the firewall. The script will prompt you to enter packet details in the format 'ip, port, protocol'.

2. **Web Interface**

   Open the HTML file in a web browser. Enter the IP address, port, and protocol of the packet to check if it is allowed or denied by the firewall.

Conclusion

The Advanced Firewall Application provides a robust and flexible framework for filtering network packets based on customizable criteria. By using design patterns, the application ensures maintainability and ease of extension for future requirements.