Copilot

Certainly! Let’s break down the steps for parsing the configuration file for your web server project. I’ll provide additional clarity and resources for each step:

1. **Understanding the Configuration File Format**:
    - **Purpose**: Before diving into parsing, thoroughly understand the format of the configuration file. In your case, you can take inspiration from the NGINX configuration file’s ‘server’ part.
    - **Resources**:
        - Look at NGINX configuration examples to understand how they structure their server blocks.
        - Familiarize yourself with the concepts of server blocks, ports, and hosts.
2. **Identify Key Configuration Parameters**:
    - **Purpose**: Determine the essential parameters you need to extract from the configuration file. These include:
        - Port and host for each ‘server’
        - Server names (optional)
        - Default error pages
        - Client body size limit
        - Route configurations
    - **Resources**:
        - Refer to your project requirements and NGINX documentation for specifics related to each parameter.
3. **Designing Data Structures**:
    - **Purpose**: Create data structures (e.g., structs or classes) in your code to represent the parsed configuration settings.
    - **Resources**:
        - Depending on your programming language (C++ in this case), design suitable data structures to organize the parsed data.
4. **Parsing Logic**:
    - **Purpose**: Implement the logic to read the configuration file and populate your data structures with the parsed settings.
    - **Resources**:
        - Use file I/O operations to read the configuration file.
        - Parse the file contents according to the defined format (e.g., custom format or JSON).
5. **Error Handling**:
    - **Purpose**: Handle cases such as missing or malformed configuration files gracefully.
    - **Resources**:
        - Implement robust error handling mechanisms.
        - Provide meaningful error messages to guide users when issues occur.
6. **Testing**:
    - **Purpose**: Thoroughly test your configuration file parsing functionality.
    - **Resources**:

        - Create test scenarios with both valid and invalid configuration files.
        - Verify that your parsing logic correctly extracts all necessary settings.

7. **Documentation**:

    - **Purpose**: Document your code and provide instructions for users on how to format the configuration file.
    - **Resources**:
        - Add comments within your code to explain the parsing logic.
        - Write clear documentation for users, including examples of correctly formatted configuration files.

Remember that these steps lay the groundwork for your project. As you progress, you can build upon this foundation and add more features to your web server. Now, let's address the bonus features:

1. **Support Cookies and Session Management**:

    - Implement cookie handling and session management. Cookies allow you to maintain state between client requests.
    - Resources: Look into libraries or frameworks that handle cookies and sessions in your chosen programming language.

2. **Handle Multiple CGI**:

    - Extend your server to support multiple Common Gateway Interface (CGI) scripts. CGI allows interaction between web servers and external programs.
    - Resources: Understand how CGI works and how to execute external scripts from your server.

Remember to compare your implementation with NGINX behavior, especially regarding features like `server_name`. And don't forget to test thoroughly and ensure your server is resilient! Good luck with your project, and feel free to ask if you need further assistance! 😊