Analyze Spring Boot Projects

To analyze a Spring Boot project (which may consist of multiple modules) and extract all APIs defined with `@GetMapping` or `@PostMapping` annotations, we need to traverse the project directory, identify controller classes, extract the API paths, and print them in a specified format. Below is a complete Python script that accomplishes this task, inspired by the structure of the provided code but tailored to the new requirements. The output will list each controller with its associated GET and POST paths in the format "XXController: get path post path".

Approach

1. **Traverse the Directory**: Recursively find all `.java` files in the project directory and its subdirectories.
2. **Identify Controllers**: Check each Java file for `@Controller` or `@RestController` annotations to confirm it's a controller class.
3. **Extract Class-Level Paths**: Look for `@RequestMapping` annotations at the class level to determine base paths.
4. **Extract Method-Level APIs**: Find methods annotated with `@GetMapping` or `@PostMapping`, extract their paths, and combine them with base paths if present.
5. **Organize and Print**: Collect GET and POST paths per controller and print them in the specified format.

Python Script

```
import os
import sys
import re
from collections import defaultdict

def find_java_files(root_dir):
    """
    Recursively find all .java files in the given directory and its subdirectories.

    Args:
        root_dir (str): The root directory to start the search from.

    Yields:
        str: The full path to each .java file.
    """
    for dirpath, dirnames, filenames in os.walk(root_dir):
        for filename in filenames:
            if filename.endswith('.java'):
```

```
                yield os.path.join(dirpath, filename)

def extract_paths(line, annotation_type):
    """
    Extract path values from a Spring annotation (@GetMapping, @PostMapping, or @RequestMapping).

    Args:
        line (str): The line containing the annotation.
        annotation_type (str): The type of annotation ('GetMapping', 'PostMapping', or 'RequestMapping').

    Returns:
        list: A list of path strings extracted from the annotation.
    """
    if annotation_type in ['GetMapping', 'PostMapping']:
        match = re.search(rf'@{annotation_type}\((.*)\)', line)
        if match:
            content = match.group(1)
            # Extract all string literals within quotes
            paths = re.findall(r'"([^"]*)"', content)
            return paths
        return []
    elif annotation_type == 'RequestMapping':
        match = re.search(r'@RequestMapping\((.*)\)', line)
        if match:
            content = match.group(1)
            # Look for 'value' or 'path' attribute
            value_match = re.search(r'(value|path)\s*=\s*({[^}]*}|"[^"]*")', content)
            if value_match:
                value = value_match.group(2)
                if value.startswith('{'):
                    paths = re.findall(r'"([^"]*)"', value)
                else:
                    paths = [value.strip('"')]
                return paths
            # If no 'value' or 'path', assume direct path specification
            paths = re.findall(r'"([^"]*)"', content)
            return paths
        return []

if __name__ == '__main__':
```

```
# Parse command-line arguments
if len(sys.argv) != 2:
    print("Usage: python script.py <root_directory>")
    sys.exit(1)

root_dir = sys.argv[1]
if not os.path.isdir(root_dir):
    print(f"[ERROR] The specified path is not a directory: {root_dir}")
    sys.exit(1)

print(f"[INFO] Starting analysis of directory: {root_dir}")

# Initialize a dictionary to store controller mappings
controllers = defaultdict(lambda: {'GET': [], 'POST': []})
total_files = 0
error_files = 0

# Process all Java files
for java_file in find_java_files(root_dir):
    try:
        with open(java_file, 'r', encoding='utf-8') as f:
            lines = f.readlines()

        # Check if the file is a controller
        if any('@Controller' in line or '@RestController' in line for line in lines):
            controller_name = os.path.basename(java_file).replace('.java', '')

            # Find the class declaration line to separate class-level and method-level annotations
            class_line_index = None
            for i, line in enumerate(lines):
                if re.search(r'public\s+(class|abstract\s+class|interface)\s+\w+', line):
                    class_line_index = i
                    break
            if class_line_index is None:
                continue

            # Extract class-level @RequestMapping as base paths
            base_paths = []
            for line in lines[:class_line_index]:
                if re.search(r'\s*@RequestMapping', line):
```

```
                    base_paths = extract_paths(line, 'RequestMapping')
                    break
            if not base_paths:
                base_paths = ['']

            # Extract method-level @GetMapping and @PostMapping
            get_paths = []
            post_paths = []
            for line in lines[class_line_index:]:
                if re.search(r'\s*@GetMapping', line):
                    paths = extract_paths(line, 'GetMapping')
                    for base in base_paths:
                        for path in paths:
                            full_path = base + path
                            get_paths.append(full_path)
                elif re.search(r'\s*@PostMapping', line):
                    paths = extract_paths(line, 'PostMapping')
                    for base in base_paths:
                        for path in paths:
                            full_path = base + path
                            post_paths.append(full_path)

            # Store unique paths
            get_paths = sorted(list(set(get_paths)))
            post_paths = sorted(list(set(post_paths)))

            if get_paths or post_paths:
                controllers[controller_name]['GET'] = get_paths
                controllers[controller_name]['POST'] = post_paths

        total_files += 1
    except Exception as e:
        print(f"[ERROR] Could not read file {java_file}: {e}")
        error_files += 1

# Print summary
print(f"[INFO] Total Java files attempted: {total_files + error_files}")
print(f"[INFO] Successfully processed: {total_files}")
print(f"[INFO] Files with errors: {error_files}")
print(f"[INFO] Total controllers found: {len(controllers)}")
```

```
    # Print results in the specified format
    for controller, mappings in sorted(controllers.items()):
        print(f"{controller}:")
        for path in mappings['GET']:
            print(f"get {path}")
        for path in mappings['POST']:
            print(f"post {path}")
```

Explanation

- **Imports**: We use `os` for directory traversal, `sys` for command-line arguments, `re` for regular expressions, and `defaultdict` to organize controller data.
- `find_java_files`: This function recursively yields all `.java` files in the project directory, handling multiple modules naturally via `os.walk`.
- `extract_paths`: This function parses `@GetMapping`, `@PostMapping`, or `@RequestMapping` annotations to extract path values. It handles:
  - Single paths (e.g., `@GetMapping("/path")`).
  - Multiple paths (e.g., `@GetMapping({"/path1", "/path2"})`).
  - Named attributes (e.g., `@RequestMapping(value = "/path")`).
- **Main Logic**:
  - **Command-Line Handling**: Takes a root directory as input, similar to the provided script.
  - **Controller Detection**: Checks for `@Controller` or `@RestController` in the file content.
  - **Base Paths**: Extracts class-level `@RequestMapping` paths before the class declaration.
  - **API Extraction**: Processes lines after the class declaration for `@GetMapping` and `@PostMapping`, combining paths with base paths.
  - **Output**: Prints each controller with its GET and POST paths, one per line, prefixed with "get"or "post".
- **Error Handling**: Catches exceptions during file reading and reports them, mirroring the robustness of the provided code.

Sample Output

For a Spring Boot project with a file `UserController.java`:

```
@RestController
@RequestMapping("/users")
public class UserController {
    @GetMapping("/all")
```

```
    public ResponseEntity<?> getAllUsers() { ... }

    @PostMapping("/create")
    public ResponseEntity<?> createUser() { ... }
}
```

Running `python script.py /path/to/project` might produce:

```
[INFO] Starting analysis of directory: /path/to/project
[INFO] Total Java files attempted: 10
[INFO] Successfully processed: 10
[INFO] Files with errors: 0
[INFO] Total controllers found: 1
UserController:
get /users/all
post /users/create
```

Notes

- **Simplifications**: The script assumes annotations are on single lines and paths are string literals without complex expressions. For production use, a Java parser might be more robust.
- **GET and POST Only**: As requested, it focuses on `@GetMapping` and `@PostMapping`, ignoring other methods like PUT or DELETE (though the thinking trace considered `@RequestMapping` with methods).
- **Path Concatenation**: Combines class-level and method-level paths directly (e.g., `/users` + `/all` = `/users/all`). Spring normalizes multiple slashes, so this is sufficient for display purposes.

This script meets the query's requirements while maintaining a structure similar to the provided code, ensuring familiarity and reliability.