Clean Code Standards: Hike Review

Flask (Python):

- Detailed Comments:
    - Developer-defined functions should have brief descriptions to share functionality intentions
- Naming Convention:
    - Give functions, variables, and files meaningful names
    - Functions:
        - Lower camel case unless functions from outside libraries are being used
    - Variables:
        - Lower camel case, start the variable name with a lowercase letter (exampleVariable)
        - Variables referencing outside library or field database should use snake cake (example_field)
    - Classes:
        - Upper camel case, start the variable name with an uppercase letter (ExampleClass)
    - Files:
        - Snake lowercase
- SQL Executions
    - keywords should be separated by a new line
- Unit test names should use the following format:
    - test<FunctionName>
    - Ex: testgetHikeData
- Avoid mutable global state
- Operators should be spaced out
- Code Structure:
    - No one liners
    - Blank line between definitions, classes, and code blocks
    - No white spaces at the end of each line
    - No line should be longer than 130 characters

UI (React Native):

- Detailed Comments:
    - Developer-defined functions should have brief descriptions to share functionality intentions
- Naming Convention:
    - Functions:
        - snake_case for all functions
        - Examples: “func getCurrentLocation()” or “func analyze()”
    - Variables:
        - Constants should be all capital letters and defined at the top of source code (EXAMPLE_CONSTANT)

        - Intermediate variables should be camelCase (int exampleVariable)
    - Files:
        - PascalCase for all files
        - Examples: “index.tsx” or “new_file.tsx”
- Code Structure:
    - Interfacing for repetitive code structures:
        - I.e. Hike
            - interface Hike {
            - trail_id: number;
            - trail_name: string;
            - location: string;
            - difficulty: string;
            - distance: number;
            - description: string;
            - created_at: string;
            - }
- File Structure:
    - Enabling tab capabilities:
        - To use tabs in an effective and easy to read/manage way we use a root layout.tsx file and a separate “(tabs)” folder for each tab
        - Root layout file initializes tab functionality and enables switching between tabs and displaying the respective files
    - Components:
        - Use a components folder to componentize large features