Project Name: AL

Coding Guidelines
=================

1. File and Directory Structure

- Keep a well-organized directory structure.
- Basic file structure:

```
    |-- al-bluering-frontend
        |-- public
        |-- src
            |-- api
                |-- ajax.js
                |-- index.js
            |-- assets
                |-- img
                |-- font
            |-- components
                |-- OneComponent
                    |-- OneComponent.jsx
                    |-- OneComponent.less
            |-- config
            |-- page
                |-- OnePage
                    |-- SubPage
                        |-- SubPage.jsx
                    |-- OnePage.jsx
                |-- Constructure.jsx
            |-- utils
        |-- App.js
        |-- index.js
        |-- setupProxy.js
    |-- al-bluering-backend
        |-- routes
        |-- controllers
        |-- config
        |-- models
        |-- services
        |-- server.js
    ------------------------
```

Frontend:

public: static pages

api: store file contributing interactions with backend, including packaged axios functions.

assets: static materials such as image, font or video
components: reusable web components which includes base code and style sheet
config: config file for changing pages or components
page: page which directed by url and contains several components. Including a Constructure.jsx which describe the constructure of whole website.
utils: some tools to help with development
App.js & index.js: entry of frontend program. DO NOT MODIFY IT.
setupProxy.js: proxy config file. Could connect to multiple backend.

Backend:
routes: direct request to controllers according to request url
controllers: handle inputs and return outcome to frontend
services: handle actual services of each request
config: config file if necessary (including database)
models: schemas of database
server.js: entry of backend server

2. HTML

- Use valid and semantic HTML5.
- Use indentation of 2 spaces.
- Avoid inline styles; use external LESS for styling.
- Avoid insert JavaScript code inside html file.
- Comment complex sections and important elements.

3. LESS

- Use external .less files for styles instead of .css.
- Use lower camel case names (e.g., .menuItem).
- Group related rules together.
- Comment code when using hacks or workarounds.
- Do not use fix size (e.g., 1 px). Use related size (e.g., 1 rem).
- Only **components** has style sheet (.less). If other part (including **page**) needs to modify style, use props to send params to **components**

4. JavaScript

- Use ES6+ syntax when possible.
- Use meaningful variable and function names (camelCase).
- Keep functions and methods focused on a single task.
- Comment complex or non-obvious code sections.
- Use modular approach when building larger applications.
- Routes will do nothing but direct req and res to related controller.
- Controller will **NOT** do actual service but processing inputs & outputs and sending back to

frontend.
- Services will handle the actual business including event handling, module using or database query, but **NOT** communicate with frontend.
- Use json object to communicate between Service Layer and Controller Layer, the basic structure is

    Result = {status: Integer, data: Object, msg: String}
    status: identify the result should be successful or unsuccessful
    data: successful result content
    msg: unsuccessful error message if necessary
    this could be more or less if necessary

- Use async/await to process request between different layer.

5. Version Control (Git)

- Use version control for all code files.
- Make **frequent**, **meaningful** commits.
- Write clear commit messages explaining changes.
- Always pull before pushing to avoid conflicts.

6. React
- Use react functional component instead of react class component.
- Use ant design as Only external component library.
- Comment function if necessary.
- Use hooks when handling life cycle.

7. Performance

- Minimize HTTP requests by combining CSS and JavaScript files.
- Optimize images and other assets.
- Use lazy loading for non-critical resources.
- Consider performance implications when writing code.
- Only **components** and **pages** should use *UpperCamelCase* to name file and folder; **other file/function/variable** should use *lowerCamelCase* to name.
- Only **components** contains html label. **Pages** will only use **components** instead of creating html label inside **page**.
- Do not use any magic numbers. Create constants if necessary.
- Use upper letter and underline to name constants. (e.g., ONE_CONSTANT)

8. Code Review

- Conduct regular code reviews among team members.
- Provide constructive feedback and suggestions.
- Ensure adherence to these coding guidelines.

9. Documentation

- Maintain clear and up-to-date project documentation.
- Document code structure, dependencies, and usage instructions.

By following these coding guidelines, we aim to create consistent, maintainable, and high-quality code across our project. Let's work together to make AL an awesome project!

Last Updated: August 16, 2023