ChatGPT

ERP Client Agent Guidelines

These guidelines apply to contributions in the **front-end** project of the ERP system (`erp-client`), which is generated using JHipster. They complement the unified standards while focusing on client-side responsibilities.

Purpose

Provide future developers and testers with enough context to understand why a client-side change was made and how the resulting user interface should behave. Client documentation also comes in two complementary forms:

1. **Technical analysis and man-pages** — Case-study style write-ups of UI workflows and design decisions. These documents explain the user interface flows, the components and services involved, and any trade-offs you considered. They live in `man_pages/` at the root of the repository and in `erp-client/man_pages/`. Within each `man_pages` directory, organise files into subfolders by topic or module (e.g. `report-submission/` or `lease-liability/`).
2. **User stories and manuals** — Narrative descriptions of what a user wants to accomplish and how the user interface supports that goal. They live in `user-stories/` at the root and in `erp-client/user-stories/`, also subdivided by topic. The user manual in `user-pages/` should be updated to reflect each front-end feature you modify or add.

For every significant front-end feature, include both forms of documentation where appropriate. Focus on the user experience: avoid low-level implementation details and instead describe pages, buttons, inputs and expected outcomes.

Commit Messages

Use a short imperative summary (maximum 72 characters) followed by an optional body. The summary should describe what the change does (e.g. **"Improve report submission form validation"**). If the change is complex, the body should explain the rationale and reference any documentation you added.

Documentation Tasks

When you implement or modify a client-side feature, perform the following steps:

1. **Technical write-up** — Draft a detailed analysis of the affected UI workflows. Explain the business requirement, the key components or services involved, and why the change was necessary. Save this Markdown document in:
2. `man_pages/` at the root of the repository, and
3. `erp-client/man_pages/` in a topic-specific subfolder.
4. **User stories** — Describe the scenario from an end-user's perspective. Outline the persona, the pages and interactions, and the expected outcome. Save these in `user-stories/` at the root and in `erp-client/user-stories/` under the appropriate topic subfolder.
5. **User manual** — Update or create pages in `user-pages/` that explain how to use the feature. Ensure there is a section for every front-end feature you have modified or added.

6. **Integration with the back-end** — Coordinate with server developers when adding new API endpoints or modifying existing ones. Document any assumptions or requirements in the technical write-up and user stories.
7. **Tests** — Where feasible, write unit and integration tests for your components. Focus on verifying the user flows described in your user stories.
8. **Sensitive files** — Update the top-level `.gitignore` to exclude any configuration, credentials or generated artefacts introduced by your change.

Project Structure and Custom Code

Follow the standard JHipster client architecture and naming conventions. Avoid modifying generated service classes directly; instead, implement custom logic in classes suffixed with `Extension` (e.g. `ReportSubmissionFormServiceExtension`) within the appropriate subpackages (such as `services`, `components`, etc.). If you need to alter a generated resource or component, create a copy with a `Prod` suffix and inject your extension. This ensures your enhancements remain separate from the generated code while still being used at runtime.

Keep your code changes scoped to the client module and document any impacts on other modules. For example, if a new UI requires a back-end API, coordinate with server developers and document the interaction in the relevant user stories.

Quality Assurance

Create unit tests and run them locally before creating a pull request. Ensure that formatting and linting checks pass. Integration tests should cover the main workflows described in your user stories. If there are limitations that prevent full test coverage, document these limitations alongside the user stories so reviewers understand what has been verified manually.

By following these guidelines, the ERP client project will maintain high-quality documentation and efficient development practices.
