ChatGPT

ERP System (Server) Agent Guidelines

These guidelines apply to contributions in the **back-end** project of the ERP system (`erp-system`), which is generated using JHipster. They mirror the unified standards but focus on server-side responsibilities.

Purpose

Provide future developers with enough context to understand why a server-side change was made and how the resulting feature should behave. Server documentation comes in two complementary forms:

1. **Technical analysis and man-pages** — Case-study style write-ups of workflows and design decisions. These documents explain the business purpose, the services and resources involved, and any trade-offs you considered. They live in `man_pages/` at the root of the repository and in `erp-system/man_pages/`. Within each `man_pages` directory, organise files into subfolders by topic or module (for example, `report-submission/` or `lease-liability/`).
2. **User stories and manuals** — Narrative descriptions of what a user wants to accomplish and how the system supports that goal. They live in `user-stories/` at the root and in `erp-system/user-stories/`, similarly subdivided by topic. The user manual in `user-pages/` should be updated to describe each server-side feature you modify or add.

For every significant back-end feature, especially those related to the report-submission service, include both forms of documentation when appropriate. Be thorough but realistic: focus on explaining the *why* and *how* rather than duplicating trivial code behaviour.

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. **“Add audit log to report submission”**) rather than how it was done. 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 server-side feature, perform the following steps:

1. **Technical write-up** — Draft a detailed analysis of the affected workflows. Explain the business requirement, the key services or components involved, and why the change was necessary. Save this Markdown document in:
2. `man_pages/` at the root of the repository, and
3. `erp-system/man_pages/` in a topic-specific subfolder (create one if it doesn’t exist).
4. **User stories** (if applicable) — Describe the scenario from an end-user’s perspective. Outline the role, the steps through the UI, and the expected outcome. Save these in `user-stories/` at the root and in `erp-system/user-stories/` under the appropriate topic subfolder.
5. **User manual** — Update or create pages in `user-pages/` that explain how to use the feature in practice. Ensure there is a section for every feature you have modified or added.

6. **Entity definitions** — If you add, remove or rename fields on a JHipster entity, update the corresponding `/.jhipster/*.json` file so future code generation is aware of the change.
7. **Integrated tests** — Where feasible, write integration or end-to-end tests that follow your user stories. These tests should cover the full workflow from the front-end call (if any) through to the back-end services involved in report submission. This helps ensure that the documented behaviour is verified by the system.
8. **Queries** — If a workflow involves custom SQL, place a PostgreSQL script in `erp-system/queries/` showing the relevant query. These scripts are for review and manual testing; they will not be executed automatically.
9. **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 server layout and naming conventions. Avoid modifying generated service classes directly; instead, implement custom logic in classes suffixed with `Extension` (e.g. `ReportSubmissionServiceExtension`) within the appropriate subpackages (`service`, `repository`, `web.rest`, etc.) under the `io.github.erp.erp` namespace. To apply your enhancements in production without altering the generated resource, create a copy of the generated resource class and append a `Prod` suffix to its name. This `Prod` resource should inject your `Extension` class and serve as the entry point used at runtime. This approach keeps your custom logic separate from the generated code while still being invoked in production.

Keep your code changes scoped to the server module and document any impacts on other modules. For example, a new API endpoint should be accompanied by the corresponding UI changes in the client, and both should be described 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 and verify that the system behaves as documented. If there are limitations that prevent full test coverage (for example, complex UI interactions), document these limitations alongside the user stories so reviewers understand what has been verified manually.

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