ChatGPT

ERP Deployment Project Agent Guidelines

These guidelines apply to the **deployment project** of the ERP system, which encompasses packaging, configuration, and infrastructure for deploying both the server and client applications. They extend the unified standards to cover deployment-specific concerns.

Purpose

Ensure that deployment-related changes are well-documented and reproducible. Provide enough context so that future maintainers understand why an infrastructure change was made and how it affects the deployment process. Deployment documentation also comes in two complementary forms:

1. **Technical analysis and man-pages** — Detailed write-ups of deployment workflows and configuration decisions. These documents might cover topics such as containerisation, CI/CD pipelines, environment variables, scaling strategies or infrastructure as code. They live in `man_pages/` at the root of the repository and in a `deployment/man_pages/` folder. Within each `man_pages` directory, organise files into subfolders by topic (for example, `docker/`, `kubernetes/` or `ci-cd/`).
2. **User stories and manuals** — High-level narratives that describe what an operator or administrator needs to achieve (e.g. "As a DevOps engineer, I want to deploy a new version to staging with zero downtime"). These live in `user-stories/` at the root and in `deployment/user-stories/`, subdivided by theme. The user manual in `user-pages/` should be updated or created to provide step-by-step instructions for deployment tasks.

For every significant deployment change, include both types of documentation where appropriate. Focus on clarity and reproducibility.

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 Docker Compose configuration for development"**). 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 deployment-related feature, perform the following steps:

1. **Technical write-up** — Draft a detailed analysis of the deployment workflow or configuration affected. Explain the purpose, the tools or services involved (e.g. Docker, Kubernetes, GitHub Actions), and why the change was necessary. Save this Markdown document in:
2. `man_pages/` at the root of the repository, and
3. `deployment/man_pages/` in a topic-specific subfolder.
4. **User stories** — Describe the scenario from an operator's perspective. Outline the role, the steps they perform and the expected outcome. Save these in `user-stories/` at the root and in `deployment/user-stories/` under the appropriate topic subfolder.

5. **User manual** — Update or create pages in `user-pages/` that explain how to perform deployment tasks (e.g. running the application locally, deploying to staging, promoting to production). Ensure there is a section for every deployment feature you have modified or added.
6. **Configuration and scripts** — Keep deployment scripts and configuration files (e.g. Dockerfiles, Helm charts, CI/CD workflows) in their respective directories. Document any sensitive values and ensure they are not committed to the repository.
7. **Tests** — Where feasible, write tests for the deployment pipeline (for example, pipeline scripts or infrastructure tests). If automated tests are not practical, document manual verification steps.
8. **Sensitive files** — Update the top-level `.gitignore` to exclude credentials, secrets or generated artefacts introduced by your change.

Project Structure and Custom Code

Follow best practices for deployment configuration and avoid modifying generated server or client code here. Keep configuration and infrastructure code modular and reusable. If you need to customise generated deployment artifacts, create separate files or scripts (with clear naming) rather than editing the originals directly. Document why the customisation is necessary and how it should be maintained.

Coordinate with both the server and client teams when deployment changes impact application behaviour. Describe any required environment variables, service endpoints or scaling considerations in the relevant technical write-ups and user stories.

Quality Assurance

Test your deployment scripts and infrastructure changes in a controlled environment (e.g. a staging environment) before merging them. Ensure that formatting and linting checks pass for configuration files where applicable (e.g. YAML or JSON). If automated testing is not possible, document the manual steps taken to verify that the deployment works as intended.

By following these guidelines, the ERP deployment project will remain maintainable, well-documented and reliable.
