Lista de Restrições

Restrição	Razão (lógica)
O software deve ser desenvolvido em Java.	Popularidade no mercado, robustez, e suporte a um amplo ecossistema de bibliotecas e frameworks, além de ser amplamente utilizado em aplicações corporativas e suporte à diversas plataformas.
O framework de desenvolvimento Java Spring Boot deve ser utilizado.	A facilidade de configuração inicial e tempo de desenvolvimento reduzido com configurações pré-estabelecidas tornam este framework o mais eficaz para esse contexto.
A biblioteca Java Swing deve ser utilizada para desenvolver a interface gráfica do sistema.	Rápida e fácil de implementar, é parte do Java Development Kit (JDK) e não requer dependências externas.
O sistema deve utilizar MySQL como o banco de dados relacional.	Código aberto e amplamente utilizado, com boa performance e suporte a escalabilidade, além de ser bem suportado pela comunidade Java.
Deve ser compatível com servidores que possuam Java Runtime Environment (JRE) versão 8 ou superior.	Para garantir a compatibilidade do sistema com versões amplamente adotadas do JRE, evitando problemas de execução em ambientes de produção comuns.
As operações CRUD devem ser realizadas com um tempo de resposta máximo de 2 segundos.	A fim de manter uma experiência de usuário aceitável e responsiva mesmo que em situações de carga moderada.
O sistema deve implementar autenticação e autorização de usuários usando o framework Spring Security.	Para garantir que apenas usuários autenticados possam acessar ou modificar os dados, protegendo o sistema contra acessos não autorizados e atendendo aos requisitos básicos de segurança.
O software deve ser compatível com sistemas Windows, Linux e MacOS.	Java é uma linguagem multiplataforma, portanto o sistema deve ser capaz de rodar em diferentes ambientes operacionais, garantindo flexibilidade no uso e no deploy.

Todas as bibliotecas e frameworks utilizados devem ter licenciamento compatível com uso comercial (exemplo: MIT, Apache).	Para garantir que todas as dependências do sistema estejam em conformidade com as políticas de licenciamento.
O sistema deve ser capaz de suportar um crescimento de até 10.000 registros por entidade sem perda significativa de performance.	Para assegurar que o sistema possa escalar conforme o crescimento dos dados sem perder a usabilidade.
A interface do usuário deve ser simples e intuitiva.	Facilitar o uso para usuários finais, garantindo que mesmo usuários com pouca ou nenhuma experiência técnica sejam capazes de operar o sistema com facilidade.
O sistema deve estar em conformidade com a Lei Geral de Proteção de Dados (LGPD) - a Lei federal n. 13.853 de 2019, de 8 de julho.	Garantir que o sistema atenda às exigências legais relacionadas ao tratamento de dados pessoais, evitando penalidades legais e assegurando a privacidade dos usuários.
O desenvolvimento deve seguir (quando possível) práticas de Clean Code e padrões de projeto onde aplicável.	Assegurar que o código seja limpo, legível e fácil de manter, além de permitir futuras expansões e melhorias.
Todas as funcionalidades devem ter cobertura de testes unitários e de integração.	Garantir a confiabilidade do sistema, minimizando o risco de bugs e problemas em produção, e facilitando a manutenção do sistema a longo prazo.
O projeto deve seguir o modelo de ramificação Git Flow.	Modelo de ramificação popular que organiza o desenvolvimento em branches como *main*, *develop*, *feature*, *release*, e *hotfix*, facilitando o controle de versões e a integração contínua. Ele garante que o código em produção (branch *main*) seja sempre estável e que novas funcionalidades sejam desenvolvidas e testadas de forma isolada antes da integração.
As mensagens de commit devem ser claras e seguir o padrão Conventional Commits.	Mensagens de commit padronizadas ajudam na rastreabilidade das mudanças, facilitando a revisão de código e o entendimento do histórico do projeto.

Todo código deve passar por revisão via Pull Request (PR) antes de receber um merge para a *main*.	Revisões de código garantem a qualidade do código, ajudam a identificar bugs e mantêm a consistência com as práticas de desenvolvimento definidas.
O projeto deve ser configurado com integração contínua (CI) utilizando o GitHub Actions para rodar testes automatizados a a cada commit.	Garante que o código não quebre funcionalidades existentes e que a qualidade seja mantida ao longo do desenvolvimento.
O repositório GitHub deve ser versionado seguindo o Semantic Versioning (SemVer).	Define uma convenção para atribuir números de versão (ex: 1.0.0), onde mudanças de API incompatíveis, funcionalidades adicionadas e correções de bugs são claramente indicadas pela mudança de números de versão.
Toda mudança significativa no código ou arquitetura deve ser documentada em um arquivo CHANGELOG.md e na wiki do repositório seguindo o modelo Keep a Changelog.	Ajuda na comunicação com outros desenvolvedores e usuários finais, além de facilitar a manutenção do sistema a longo prazo.
Os nomes das branches devem seguir um padrão específico, como feature/, bugfix/, release/, etc.	Identificação rápida do propósito de cada branch, melhorando a comunicação entre os desenvolvedores e a organização do repositório.
O merge para a branch main deve ser realizado apenas após passar por um processo de revisão e aprovação de, pelo menos, dois desenvolvedores, e todos os testes de CI devem ser aprovados.	Garante que o código que vai para produção seja de alta qualidade e sem problemas conhecidos, evitando interrupções ou bugs em produção.