

# Controle de Estoque de Livros Didáticos

Senai

Limeira - SP

- Diogo Scherrer
- Gabriel Furtunato
- Gabriel Xavier

# 1. Visão Geral do Sistema

## 1.1 Contexto

O SENAI envia remessas de livros didáticos para as unidades. Os livros ficam no almoxarifado e são retirados pelos instrutores para distribuição em sala de aula. Atualmente existe falha no controle das saídas, causando ruptura de estoque e atrasos no aprendizado.

## 1.2 Objetivo

Desenvolver o **Back-End** do **SenaiStock**, uma **API RESTful** em **Laravel + MySQL**, para manter o **saldo de estoque sempre atualizado**, registrando:

- **Entradas** (abastecimento/chegada de livros)
- **Saídas** (baixa manual/retiradas para turmas)
- **Alertas de baixo estoque**

## 1.3 Escopo

- Definir requisitos funcionais e não funcionais
  - Definir regras de negócio
  - Definir atores/perfis e permissões
  - Definir endpoints esperados (alto nível)
  - Definir critérios de aceitação
- 

# 2. Stakeholders e Atores

## 2.1 Stakeholders

- **Almoxarifado (SENAI)**: precisa do saldo confiável e rapidez na operação.
- **Coordenação**: precisa acompanhar e garantir reposição.
- **Instrutores**: dependem do material disponível
- **TI / Desenvolvimento**: mantém e implanta o sistema.

## 2.2 Atores (usuários do sistema)

- **Almoxarife** (operacional): cadastra livros, registra entradas e saídas.
  - **Coordenador** (gestão): consulta relatórios, monitora baixo estoque e pode ajustar/cadastrar.
  - **Sistema/API**: valida regras e controla permissões.
- 

# 3. Premissas, Restrições e Dependências

## 3.1 Premissas

- Todo livro controlado possui **Título, ISBN e Matéria**.
- Entradas e saídas devem gerar atualização do saldo de forma consistente.
- Todo usuário que altera estoque deve estar autenticado.

## 3.2 Restrições

- Back-End em **Laravel (PHP)**
- Banco relacional **MySQL**
- Modelagem via **Eloquent ORM**
- **API RESTful** retornando **JSON**
- Padrões **PSR** e **Clean Code**
- Versionamento e publicação no **GitHub**
- Testes via **Insomnia/Postman**

## 3.3 Dependências

- Ambiente de execução PHP / Laravel e MySQL
  - Ferramenta de testes (Insomnia / Postman)
  - Repositório GitHub
-

## 4. Requisitos Funcionais (RF)

### RF-01 — Autenticação (Login)

**Descrição:** O sistema deve permitir autenticação para garantir acesso apenas a funcionários autorizados.

**Entrada:** email / usuário e senha.

**Saída:** token de autenticação e dados básicos do usuário.

**Prioridade:** Alta.

**Critérios de aceitação:**

- Deve ser possível realizar login com credenciais válidas.
  - Deve retornar erro e status adequado quando inválidas.
  - Rotas protegidas devem bloquear acesso sem token.
- 

### RF-02 — Controle de Perfil e Permissões

**Descrição:** O sistema deve controlar o que cada perfil pode fazer.

**Perfis:** Almoxarife, Coordenador.

**Prioridade:** Alta.

**Regras mínimas:**

- Almoxarife: pode cadastrar livro, registrar entrada e saída.
- Coordenador: pode consultar, monitorar baixo estoque e também registrar movimentações.

**Critérios de aceitação:**

- Um usuário sem permissão não consegue executar rotas restritas.
-

## **RF-03 — Cadastro de Livros (Catálogo)**

**Descrição:** O sistema deve permitir cadastrar livros no catálogo.

**Campos:** Título, ISBN, Matéria.

**Prioridade:** Alta.

**Critérios de aceitação:**

- Não permitir ISBN duplicado.
  - Validar campos obrigatórios.
  - Retornar o livro cadastrado em JSON.
- 

## **RF-04 — Listagem de Livros**

**Descrição:** O sistema deve listar livros cadastrados com seus dados e saldo atual.

**Prioridade:** Alta.

**Critérios de aceitação:**

- Deve retornar lista em JSON.
  - Deve permitir paginação (recomendado).
  - Deve permitir filtro por título/ISBN/matéria (recomendado).
- 

## **RF-05 — Entrada de Estoque (Abastecimento)**

**Descrição:** Registrar chegada de livros e somar ao saldo atual do livro.

**Entrada:** livro\_id (ou ISBN) e quantidade.

**Saída:** saldo atualizado e registro da movimentação.

**Prioridade:** Alta.

**Critérios de aceitação:**

- Quantidade deve ser > 0.
  - Deve registrar “movimentação do tipo ENTRADA”.
  - O saldo deve aumentar corretamente.
-

## RF-06 — Saída de Estoque (Baixa Manual)

**Descrição:** Registrar retirada de livros e subtrair do saldo atual.

**Entrada:** livro\_id (ou ISBN), quantidade e observação (ex.: turma/justificativa).

**Saída:** saldo atualizado e registro da movimentação.

**Prioridade:** Altíssima (principal do sistema).

**Critérios de aceitação:**

- Quantidade deve ser > 0.
  - Deve registrar “movimentação do tipo SAÍDA”.
  - Deve exigir observação (recomendado).
  - Não permitir saída se **quantidade > saldo disponível** (ver RN-01).
  - Retornar erro claro “Estoque insuficiente” com status adequado.
- 

## RF-07 — Consulta de Saldo por Livro

**Descrição:** Permitir consultar o saldo atual de um livro específico.

**Prioridade:** Média.

**Critérios de aceitação:**

- Retornar dados do livro + saldo atual.
- 

## RF-08 — Monitoramento de Baixo Estoque

**Descrição:** Listar livros com estoque abaixo de um nível mínimo configurado.

**Entrada:** parâmetro mínimo (opcional) ou valor padrão (ex.: 10).

**Saída:** lista de livros abaixo do mínimo.

**Prioridade:** Alta.

**Critérios de aceitação:**

- Se não informar mínimo, usar padrão.
  - Retornar lista em JSON ordenada por menor saldo.
-

## **RF-09 — Histórico de Movimentações**

**Descrição:** Permitir consultar entradas e saídas registradas.

**Filtros:** por livro, por período, por tipo (entrada/saída), por usuário.

**Prioridade:** Média/Alta.

**Critérios de aceitação:**

- Retornar lista paginada.
  - Deve mostrar: tipo, quantidade, data/hora, usuário, observação, livro.
- 

## **RF-10 — Logout**

**Descrição:** Encerrar sessão/token.

**Prioridade:** Média.

**Critérios de aceitação:**

- Após logout, token não deve acessar rotas protegidas.
-

## **5. Regras de Negócio (RN)**

### **RN-01 — Estoque não pode ficar negativo**

Se uma saída solicitar quantidade maior que o saldo atual, a operação deve ser **bloqueada**.

### **RN-02 — Quantidade deve ser positiva**

Não permitir entrada/saída com quantidade 0 ou negativa.

### **RN-03 — ISBN deve ser único**

Não permitir cadastro de dois livros com o mesmo ISBN.

### **RN-04 — Movimentação deve registrar autor e data**

Toda entrada/saída deve gravar **usuário autenticado** e **timestamp**.

### **RN-05 — Operações críticas devem ser atômicas**

Entrada/saída devem ser executadas em **transação**, garantindo consistência.

### **RN-06 — Nível mínimo de estoque**

O sistema deve usar um nível mínimo padrão (ex.: 10) ou permitir parâmetro na consulta.

---

# **6. Requisitos Não Funcionais (RNF)**

## **RNF-01 — Padrão de API**

- API RESTful
- Respostas em JSON
- Uso correto de status HTTP (200, 201, 400, 401, 403, 404, 409, 422)

## **RNF-02 — Segurança**

- Rotas de alteração de estoque protegidas por autenticação
- Senhas armazenadas com hash seguro
- Validação de entrada (evitar SQL injection; padrão Laravel já ajuda)

## **RNF-03 — Qualidade de Código**

- Padrões PSR
- Clean Code
- Separação de responsabilidades (Controllers, Services, Requests)

## **RNF-04 — Persistência e Integridade**

- MySQL com chaves e restrições
- Índices para ISBN e buscas frequentes

## **RNF-05 — Performance mínima**

- Listagens com paginação
- Consultas filtradas e indexadas

## **RNF-06 — Versionamento e rastreabilidade**

- Git + GitHub
- Commits descritivos
- Tags/releases (opcional)

## **RNF-07 — Testabilidade**

- Rotas testáveis via Insomnia/Postman
  - Evidências de testes (prints/coleção exportada)
-

## **7. Casos de Uso**

### **UC-01 — Realizar Login**

Ator: Almoxarife/Coordenador

Fluxo: informa credenciais → sistema valida → retorna token.

### **UC-02 — Cadastrar Livro**

Ator: Almoxarife/Coordenador

Fluxo: informa título/ISBN/matéria → valida → salva → retorna livro.

### **UC-03 — Registrar Entrada**

Ator: Almoxarife/Coordenador

Fluxo: escolhe livro + quantidade → valida → soma saldo → registra movimentação.

### **UC-04 — Registrar Saída**

Ator: Almoxarife/Coordenador

Fluxo: escolhe livro + quantidade + observação → valida → verifica saldo → subtrai → registra movimentação.

### **UC-05 — Consultar Baixo Estoque**

Ator: Almoxarife/Coordenador

Fluxo: solicita lista → sistema filtra por mínimo → retorna.

---

## 8. Requisitos de Interface (API) — visão de endpoints

- POST /auth/login
- POST /auth/logout
- POST /books
- GET /books
- GET /books/{id}
- POST /stock/entries
- POST /stock/exits
- GET /stock/low?min=10
- GET /stock/movements?book\_id=&type=&from=&to=&user\_id=

# **10. Metodologia de Desenvolvimento**

## **10.1 Metodologia Ágil Utilizada**

O projeto será desenvolvido utilizando **Scrum**, com divisão em Sprints previamente definidas pelo o docente.

O desenvolvimento será incremental, permitindo que a cada Sprint sejam realizadas melhorias, correções e validações, garantindo evolução contínua do sistema.

## **10.2 Organização das Sprints**

O projeto está dividido em 6 Sprints, contemplando:

- Levantamento e refinamento de requisitos
- Modelagem (UML e Banco de Dados)
- Implementação da API
- Testes e validações
- Implantação
- Apresentação final

## 10.3 Ferramenta de Gestão

Para organização das tarefas e acompanhamento do progresso, foi utilizada a ferramenta **Trello**, onde foram criados quadros, listas e cartões correspondentes às Sprints e atividades do projeto.

Através do **GitHub** foi realizado o upload dos arquivos do projeto, como os códigos e as documentações

Toda a prototipação do projeto foi desenvolvida e construído usando o **Figma**

Link do Trello:

<https://trello.com/invite/b/699f268b07de738132c049e8/ATTI29d0a4c4c35580343d8fadd22690eaa70C116F6A/controle-de-estoque-de-livros-didaticos-metodologia-scrum>

Link do GitHub:

[https://github.com/Furtunato/ControledeEstoque\\_LivrosDidaticos](https://github.com/Furtunato/ControledeEstoque_LivrosDidaticos)

Link do Figma:

<https://www.figma.com/design/9p1XpNNuL7FRum50qBvw4D/Sem-t%C3%ADtulo?node-id=0-1&p=f>