Claro! Aqui está um **modelo de descrição dos principais endpoints de uma API para controle de suprimentos hospitalares**, com foco em clareza e organização. Essa estrutura pode ser usada em um **documento técnico**, **manual de API**, **Swagger** ou em sua documentação conforme a ABNT (se estiver em um TCC ou artigo).

**🔗 Principais Endpoints da API – Sistema de Suprimentos Hospitalares**

A seguir, são descritos os principais endpoints da API RESTful desenvolvida para gerenciar suprimentos hospitalares. A API utiliza o padrão **HTTP** com respostas em **JSON**.

**1. GET /api/suprimentos**

**Descrição:** Retorna a lista de todos os suprimentos cadastrados no sistema.

* **Parâmetros:** Nenhum
* **Resposta:**

[

{

"id": 1,

"nome": "Dipirona Sódica",

"miligramagem": "500mg",

"tipo": "Medicamento injetável",

"quantidade": 1200,

"valor\_unitario": 0.45

},

...

]

**2. GET /api/suprimentos/{id}**

**Descrição:** Retorna os dados de um suprimento específico pelo ID.

* **Parâmetros de rota:** id (inteiro) – ID do suprimento
* **Resposta:**

{

"id": 1,

"nome": "Dipirona Sódica",

"miligramagem": "500mg",

"tipo": "Medicamento injetável",

"quantidade": 1200,

"valor\_unitario": 0.45

}

**3. POST /api/suprimentos**

**Descrição:** Cadastra um novo suprimento no estoque.

* **Corpo da requisição:**

{

"nome": "Paracetamol",

"miligramagem": "750mg",

"tipo": "Comprimido",

"quantidade": 5000,

"valor\_unitario": 0.30

}

* **Resposta (201 Created):**

{

"message": "Suprimento cadastrado com sucesso.",

"id": 2

}

**4. PUT /api/suprimentos/{id}**

**Descrição:** Atualiza os dados de um suprimento existente.

* **Parâmetros de rota:** id
* **Corpo da requisição:**

{

"quantidade": 6000,

"valor\_unitario": 0.32

}

* **Resposta:**

{

"message": "Suprimento atualizado com sucesso."

}

**5. DELETE /api/suprimentos/{id}**

**Descrição:** Remove um suprimento do sistema.

* **Parâmetros de rota:** id
* **Resposta:**

{

"message": "Suprimento removido com sucesso."

}

**6. GET /api/relatorios/estoque-baixo**

**Descrição:** Lista os suprimentos com estoque abaixo do mínimo definido.

* **Resposta:**

[

{

"id": 7,

"nome": "Lidocaína",

"quantidade": 50,

"estoque\_minimo": 100

},

...

]

Se você quiser, posso gerar isso como um **documento técnico (PDF/Word)** ou criar uma **documentação Swagger/OpenAPI**. Também posso incluir autenticação, paginação e exemplos de erros (como 404, 400, 500), se desejar. Deseja adicionar esses detalhes?

Claro! A seguir está um **exemplo completo e bem estruturado** de como descrever a seção de **Modelagem e Arquitetura** de um projeto de sistema (por exemplo, para um controle de suprimentos hospitalares), incluindo os pontos solicitados: **diagrama de classes**, **DER**, **endpoints da API** e **tecnologias de persistência**. Este modelo está alinhado ao padrão de escrita técnica e pode ser adaptado para **TCCs**, **monografias**, ou documentação de software conforme as normas da **ABNT** (NBR 14724:2023).

**3. Modelagem e Arquitetura**

Esta seção apresenta a estrutura lógica e técnica do sistema desenvolvido, detalhando os principais componentes que o compõem, sua arquitetura de software, os modelos de dados e os recursos expostos por meio da API.

**3.1 Arquitetura do Sistema**

O sistema foi desenvolvido utilizando a arquitetura **RESTful em três camadas (Three-Tier)**, composta por:

* **Camada de Apresentação:** Interface Web para o usuário (painel administrativo em React).
* **Camada de Aplicação:** API desenvolvida em Node.js com Express.
* **Camada de Persistência:** Banco de dados relacional PostgreSQL.

Essa separação facilita a manutenção, escalabilidade e reutilização de código.

**3.2 Diagrama de Classes**

O **diagrama de classes** representa a estrutura de dados da aplicação e suas relações. A seguir, um exemplo simplificado (caso deseje a imagem, posso gerar uma para você):

+------------------+

| Suprimento |

+------------------+

| - id: int |

| - nome: string |

| - miligrama: str |

| - tipo: string |

| - quantidade: int|

| - valorUnit: float|

+------------------+

+------------------+

| Categoria |

+------------------+

| - id: int |

| - nome: string |

+------------------+

Relacionamento: Suprimento \* --- 1 Categoria

**3.3 Diagrama Entidade-Relacionamento (DER)**

O DER representa graficamente o modelo relacional do banco de dados.

**Entidades principais:**

* **Suprimento(id, nome, miligramagem, tipo, quantidade, valor\_unitario, categoria\_id)**
* **Categoria(id, nome)**

**Relacionamento:**

* Um suprimento pertence a uma categoria (1:N)

*(Se desejar, posso gerar um DER real em imagem para você com notações padrão)*

**3.4 Endpoints da API**

A API expõe os seguintes principais endpoints:

| **Método** | **Rota** | **Descrição** |
| --- | --- | --- |
| GET | /api/suprimentos | Lista todos os suprimentos |
| GET | /api/suprimentos/{id} | Busca suprimento por ID |
| POST | /api/suprimentos | Cadastra novo suprimento |
| PUT | /api/suprimentos/{id} | Atualiza dados de um suprimento |
| DELETE | /api/suprimentos/{id} | Remove suprimento do sistema |
| GET | /api/relatorios/estoque-baixo | Lista suprimentos com estoque baixo |

A API utiliza respostas no padrão **JSON** e segue boas práticas de versionamento e padronização de URIs.

**3.5 Tecnologias de Persistência**

Para a persistência dos dados, foi utilizado o **PostgreSQL**, banco de dados relacional que oferece robustez, segurança e suporte a consultas complexas. A comunicação com o banco foi feita através do ORM **Prisma** (ou Sequelize, TypeORM — conforme o seu projeto), permitindo abstração da linguagem SQL e maior produtividade no desenvolvimento.

**Tabela de Tecnologias Utilizadas**

| **Camada** | **Tecnologia** |
| --- | --- |
| Backend/API | Node.js + Express |
| Banco de Dados | PostgreSQL |
| ORM | Prisma |
| Frontend (opcional) | React + Axios |
| Autenticação (extra) | JWT ou OAuth 2.0 |

**3.6 Considerações**

A modelagem foi pensada para garantir integridade dos dados, rastreabilidade e escalabilidade. Além disso, os endpoints da API foram projetados seguindo as convenções REST para facilitar a integração com outras aplicações ou sistemas hospitalares existentes.

Se desejar, posso:

* **Gerar imagens reais** do **Diagrama de Classes** e **DER**
* **Exportar a documentação como PDF**
* **Incluir exemplos de JSON de entrada/saída para os endpoints**
* **Adaptar para um caso real (ex: Hospital X, Farmácia Y, etc.)**

Quer que eu gere as imagens ou exporte para você?

Claro! Abaixo estão exemplos de **JSON de entrada e saída** para os principais endpoints da API de um sistema de controle de suprimentos hospitalares. Esses exemplos ajudam na documentação técnica e são muito úteis para desenvolvedores que vão consumir a API.

**🔄 Exemplos de Requisições e Respostas (JSON)**

**📥 1. POST /api/suprimentos**

**Descrição:** Cadastra um novo suprimento.

**🔸 Requisição (JSON):**

{

"nome": "Dipirona Sódica",

"miligramagem": "500mg",

"tipo": "Medicamento injetável",

"quantidade": 1200,

"valor\_unitario": 0.45,

"categoria\_id": 1

}

**✅ Resposta (201 Created):**

{

"message": "Suprimento cadastrado com sucesso.",

"id": 5

}

**📤 2. GET /api/suprimentos**

**Descrição:** Retorna todos os suprimentos cadastrados.

**✅ Resposta:**

[

{

"id": 1,

"nome": "Dipirona Sódica",

"miligramagem": "500mg",

"tipo": "Medicamento injetável",

"quantidade": 1200,

"valor\_unitario": 0.45,

"categoria": {

"id": 1,

"nome": "Analgésicos"

}

},

{

"id": 2,

"nome": "Cloreto de Sódio 0,9%",

"miligramagem": null,

"tipo": "Solução intravenosa",

"quantidade": 2000,

"valor\_unitario": 2.20,

"categoria": {

"id": 2,

"nome": "Soluções"

}

}

]

**🔍 3. GET /api/suprimentos/{id}**

**Descrição:** Retorna um suprimento específico.

**✅ Resposta:**

{

"id": 1,

"nome": "Dipirona Sódica",

"miligramagem": "500mg",

"tipo": "Medicamento injetável",

"quantidade": 1200,

"valor\_unitario": 0.45,

"categoria": {

"id": 1,

"nome": "Analgésicos"

}

}

**✏️ 4. PUT /api/suprimentos/{id}**

**Descrição:** Atualiza um suprimento existente.

**🔸 Requisição:**

{

"quantidade": 1000,

"valor\_unitario": 0.50

}

**✅ Resposta:**

{

"message": "Suprimento atualizado com sucesso."

}

**❌ 5. DELETE /api/suprimentos/{id}**

**Descrição:** Exclui um suprimento.

**✅ Resposta:**

{

"message": "Suprimento removido com sucesso."

}

**⚠️ 6. GET /api/relatorios/estoque-baixo**

**Descrição:** Lista suprimentos com estoque abaixo do mínimo definido.

**✅ Resposta:**

[

{

"id": 3,

"nome": "Lidocaína",

"quantidade": 50,

"estoque\_minimo": 100

},

{

"id": 6,

"nome": "Atadura de Gaze",

"quantidade": 80,

"estoque\_minimo": 150

}

]

Se você quiser, posso gerar a documentação no estilo **Swagger/OpenAPI**, em **Markdown**, ou exportar como **PDF/Word** com toda a estrutura para seu projeto.

Deseja isso?

Para aplicar **endpoints em Java** utilizando **Spark Java** (microframework web) e **Gson** (para conversão JSON), você pode seguir os passos abaixo. Spark facilita a criação de APIs RESTful de forma rápida e simples, enquanto Gson cuida da serialização e desserialização dos objetos Java para JSON.

**✅ 1. Dependências (pom.xml)**

Se estiver usando **Maven**, adicione ao seu pom.xml:

<dependencies>

<!-- Spark Java -->

<dependency>

<groupId>com.sparkjava</groupId>

<artifactId>spark-core</artifactId>

<version>2.9.4</version>

</dependency>

<!-- Gson -->

<dependency>

<groupId>com.google.code.gson</groupId>

<artifactId>gson</artifactId>

<version>2.10.1</version>

</dependency>

</dependencies>

**🛠️ 2. Modelo Java (Classe Suprimento)**

public class Suprimento {

public int id;

public String nome;

public String miligramagem;

public String tipo;

public int quantidade;

public double valor\_unitario;

// Construtor, getters e setters (opcional para Gson)

}

**🌐 3. Endpoints com Spark e Gson**

import static spark.Spark.\*;

import com.google.gson.Gson;

import java.util.\*;

public class App {

static List<Suprimento> suprimentos = new ArrayList<>();

static Gson gson = new Gson();

public static void main(String[] args) {

// GET /api/suprimentos

get("/api/suprimentos", (req, res) -> {

res.type("application/json");

return gson.toJson(suprimentos);

});

// GET /api/suprimentos/:id

get("/api/suprimentos/:id", (req, res) -> {

int id = Integer.parseInt(req.params(":id"));

Suprimento s = suprimentos.stream()

.filter(x -> x.id == id)

.findFirst()

.orElse(null);

if (s == null) {

res.status(404);

return gson.toJson(Map.of("message", "Suprimento não encontrado"));

}

res.type("application/json");

return gson.toJson(s);

});

// POST /api/suprimentos

post("/api/suprimentos", (req, res) -> {

Suprimento novo = gson.fromJson(req.body(), Suprimento.class);

novo.id = suprimentos.size() + 1;

suprimentos.add(novo);

res.status(201);

return gson.toJson(Map.of("message", "Cadastrado com sucesso", "id", novo.id));

});

// PUT /api/suprimentos/:id

put("/api/suprimentos/:id", (req, res) -> {

int id = Integer.parseInt(req.params(":id"));

Suprimento atualizado = gson.fromJson(req.body(), Suprimento.class);

for (Suprimento s : suprimentos) {

if (s.id == id) {

s.quantidade = atualizado.quantidade;

s.valor\_unitario = atualizado.valor\_unitario;

return gson.toJson(Map.of("message", "Atualizado com sucesso"));

}

}

res.status(404);

return gson.toJson(Map.of("message", "Não encontrado"));

});

// DELETE /api/suprimentos/:id

delete("/api/suprimentos/:id", (req, res) -> {

int id = Integer.parseInt(req.params(":id"));

boolean removed = suprimentos.removeIf(s -> s.id == id);

if (removed) {

return gson.toJson(Map.of("message", "Removido com sucesso"));

} else {

res.status(404);

return gson.toJson(Map.of("message", "Não encontrado"));

}

});

}

}

**📦 Executando**

1. Compile e execute o projeto com Maven:
2. mvn compile exec:java -Dexec.mainClass="App"
3. Acesse via http://localhost:4567/api/suprimentos

**💡 Dica**

Se quiser persistência real, pode integrar com:

* **SQLite ou PostgreSQL (via JDBC)**
* ORMs como **JPA** ou **Hibernate**

Se quiser, posso te ajudar a:

* Salvar os dados em banco de dados real
* Gerar um CRUD completo com HTML (front-end)
* Ou criar uma versão com autenticação básica

Deseja avançar para algum desses passos?

Perfeito! Vamos agora integrar o projeto Spark Java + Gson com um **banco de dados real** (como **PostgreSQL** ou **SQLite**) para **persistência dos dados** de suprimentos.

**🛠️ Resumo do que vamos fazer:**

1. Usar **PostgreSQL** (mas posso adaptar para SQLite se preferir)
2. Criar a tabela suprimentos
3. Integrar Spark Java via **JDBC**
4. Modificar os endpoints para usar o banco de dados

**✅ 1. Dependências no pom.xml**

Adicione o driver JDBC do PostgreSQL:

<dependency>

<groupId>org.postgresql</groupId>

<artifactId>postgresql</artifactId>

<version>42.7.3</version>

</dependency>

**🗃️ 2. Banco de Dados: PostgreSQL**

**Crie o banco e a tabela:**

CREATE DATABASE hospital\_db;

\c hospital\_db

CREATE TABLE suprimentos (

id SERIAL PRIMARY KEY,

nome TEXT NOT NULL,

miligramagem TEXT,

tipo TEXT,

quantidade INTEGER,

valor\_unitario NUMERIC

);

**🧩 3. Classe utilitária de conexão (JDBC)**

import java.sql.\*;

public class Database {

private static final String URL = "jdbc:postgresql://localhost:5432/hospital\_db";

private static final String USER = "seu\_usuario";

private static final String PASSWORD = "sua\_senha";

public static Connection connect() throws SQLException {

return DriverManager.getConnection(URL, USER, PASSWORD);

}

}

**🚀 4. Exemplo de Endpoint com JDBC (GET e POST)**

import static spark.Spark.\*;

import com.google.gson.Gson;

import java.sql.\*;

import java.util.\*;

public class App {

static Gson gson = new Gson();

public static void main(String[] args) {

// GET /api/suprimentos

get("/api/suprimentos", (req, res) -> {

List<Suprimento> suprimentos = new ArrayList<>();

try (Connection conn = Database.connect()) {

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery("SELECT \* FROM suprimentos");

while (rs.next()) {

Suprimento s = new Suprimento();

s.id = rs.getInt("id");

s.nome = rs.getString("nome");

s.miligramagem = rs.getString("miligramagem");

s.tipo = rs.getString("tipo");

s.quantidade = rs.getInt("quantidade");

s.valor\_unitario = rs.getDouble("valor\_unitario");

suprimentos.add(s);

}

} catch (SQLException e) {

res.status(500);

return gson.toJson(Map.of("error", e.getMessage()));

}

res.type("application/json");

return gson.toJson(suprimentos);

});

// POST /api/suprimentos

post("/api/suprimentos", (req, res) -> {

Suprimento s = gson.fromJson(req.body(), Suprimento.class);

String sql = "INSERT INTO suprimentos (nome, miligramagem, tipo, quantidade, valor\_unitario) VALUES (?, ?, ?, ?, ?)";

try (Connection conn = Database.connect();

PreparedStatement pstmt = conn.prepareStatement(sql, Statement.RETURN\_GENERATED\_KEYS)) {

pstmt.setString(1, s.nome);

pstmt.setString(2, s.miligramagem);

pstmt.setString(3, s.tipo);

pstmt.setInt(4, s.quantidade);

pstmt.setDouble(5, s.valor\_unitario);

pstmt.executeUpdate();

ResultSet keys = pstmt.getGeneratedKeys();

if (keys.next()) {

int id = keys.getInt(1);

res.status(201);

return gson.toJson(Map.of("message", "Cadastrado com sucesso", "id", id));

} else {

res.status(500);

return gson.toJson(Map.of("error", "Falha ao obter ID"));

}

} catch (SQLException e) {

res.status(500);

return gson.toJson(Map.of("error", e.getMessage()));

}

});

// Adicione outros endpoints (PUT, DELETE, etc.) da mesma forma

}

}

**✅ 5. Classe Suprimento**

public class Suprimento {

public int id;

public String nome;

public String miligramagem;

public String tipo;

public int quantidade;

public double valor\_unitario;

}

**🧪 Testando**

1. Compile:

mvn compile

1. Execute:

mvn exec:java -Dexec.mainClass="App"

1. Teste com Postman, curl ou seu frontend.

**❓Quer que eu converta isso para usar SQLite local ao invés de PostgreSQL?**

Ou preferiria que eu gerasse **todas as rotas (GET, POST, PUT, DELETE)** com JDBC prontos?

Posso também criar o **projeto completo em ZIP** se quiser baixar direto.

Ótima pergunta! Para **integrar seu backend Spark Java com um frontend em Java Swing (JFrames)**, você pode usar **requisições HTTP (GET, POST, etc.)** diretamente do cliente Swing para consumir a API.

Isso permite que sua **interface gráfica Swing funcione como um cliente REST** — ou seja, ela envia e recebe dados da sua API como se fosse um navegador ou aplicativo web.

**✅ Fluxo geral da integração**

1. API rodando com Spark Java + Gson + Banco de Dados
2. Interface Swing com JFrames e JButtons
3. Swing usa HttpURLConnection ou HttpClient para se comunicar com a API
4. A resposta (JSON) é convertida com Gson e mostrada nos componentes da interface (como JTable, JTextField, etc.)

**📦 Exemplo completo: Tela Swing que lista suprimentos da API**

**🔹 1. Classe Swing (SuprimentoFrame.java)**

import javax.swing.\*;

import java.awt.\*;

import java.io.InputStreamReader;

import java.net.HttpURLConnection;

import java.net.URL;

import com.google.gson.reflect.TypeToken;

import com.google.gson.Gson;

import java.util.List;

public class SuprimentoFrame extends JFrame {

private JTextArea areaTexto;

public SuprimentoFrame() {

setTitle("Suprimentos Hospitalares");

setSize(500, 400);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

setLayout(new BorderLayout());

JButton btnCarregar = new JButton("Carregar Suprimentos");

areaTexto = new JTextArea();

add(btnCarregar, BorderLayout.NORTH);

add(new JScrollPane(areaTexto), BorderLayout.CENTER);

btnCarregar.addActionListener(e -> carregarSuprimentos());

}

private void carregarSuprimentos() {

try {

URL url = new URL("http://localhost:4567/api/suprimentos");

HttpURLConnection conn = (HttpURLConnection) url.openConnection();

conn.setRequestMethod("GET");

InputStreamReader reader = new InputStreamReader(conn.getInputStream());

Gson gson = new Gson();

List<Suprimento> lista = gson.fromJson(reader, new TypeToken<List<Suprimento>>() {}.getType());

areaTexto.setText(""); // limpa

for (Suprimento s : lista) {

areaTexto.append(

String.format("ID: %d\nNome: %s\nTipo: %s\nQtd: %d\nValor: R$ %.2f\n\n",

s.id, s.nome, s.tipo, s.quantidade, s.valor\_unitario)

);

}

} catch (Exception e) {

areaTexto.setText("Erro ao carregar dados: " + e.getMessage());

}

}

public static void main(String[] args) {

SwingUtilities.invokeLater(() -> {

new SuprimentoFrame().setVisible(true);

});

}

}

**🔸 2. Classe Suprimento.java (mesmo modelo usado no backend)**

public class Suprimento {

public int id;

public String nome;

public String miligramagem;

public String tipo;

public int quantidade;

public double valor\_unitario;

}

**💡 E se eu quiser cadastrar ou atualizar?**

Você pode adicionar **campos de entrada (JTextField)** e, ao clicar em "Salvar", usar um POST com JSON assim:

HttpURLConnection conn = (HttpURLConnection) new URL("http://localhost:4567/api/suprimentos").openConnection();

conn.setRequestMethod("POST");

conn.setDoOutput(true);

conn.setRequestProperty("Content-Type", "application/json");

Suprimento novo = new Suprimento();

novo.nome = "Paracetamol";

// ...

String json = new Gson().toJson(novo);

OutputStream os = conn.getOutputStream();

os.write(json.getBytes());

os.flush();

**🧩 Tecnologias usadas**

| **Camada** | **Tecnologia** |
| --- | --- |
| Backend REST | Spark Java + Gson + JDBC |
| Banco de Dados | PostgreSQL ou SQLite |
| Frontend Desktop | Java Swing (JFrame) |

**🎁 Posso te ajudar com:**

* Cadastro e edição com formulário (JTextFields + POST/PUT)
* Integração com JTable ao invés de JTextArea
* Projeto completo com backend + Swing (ZIP ou GitHub)
* Uso com SQLite local (mais fácil sem servidor de banco externo)

Quer que eu monte um exemplo de **cadastro (POST)** com formulário Swing?

Perfeito! Abaixo está um **exemplo funcional completo de uma tela Swing em Java** que permite **cadastrar suprimentos** no seu sistema via **POST na API Spark Java**. Ela envia os dados como JSON usando **Gson** e HttpURLConnection.

**✅ 1. Estrutura do formulário Swing para cadastro**

**📄 Classe: CadastroSuprimentoFrame.java**

import javax.swing.\*;

import java.awt.\*;

import java.io.OutputStream;

import java.net.HttpURLConnection;

import java.net.URL;

import com.google.gson.Gson;

public class CadastroSuprimentoFrame extends JFrame {

private JTextField txtNome, txtMiligramagem, txtTipo, txtQuantidade, txtValor;

private JButton btnSalvar;

private JTextArea txtStatus;

public CadastroSuprimentoFrame() {

setTitle("Cadastro de Suprimento");

setSize(400, 350);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

setLayout(new GridLayout(7, 2));

add(new JLabel("Nome:"));

txtNome = new JTextField();

add(txtNome);

add(new JLabel("Miligramagem:"));

txtMiligramagem = new JTextField();

add(txtMiligramagem);

add(new JLabel("Tipo:"));

txtTipo = new JTextField();

add(txtTipo);

add(new JLabel("Quantidade:"));

txtQuantidade = new JTextField();

add(txtQuantidade);

add(new JLabel("Valor Unitário:"));

txtValor = new JTextField();

add(txtValor);

btnSalvar = new JButton("Salvar");

txtStatus = new JTextArea();

txtStatus.setEditable(false);

JScrollPane scroll = new JScrollPane(txtStatus);

add(btnSalvar);

add(new JLabel()); // espaço

add(new JLabel("Status:"));

add(scroll);

btnSalvar.addActionListener(e -> enviarCadastro());

}

private void enviarCadastro() {

try {

Suprimento s = new Suprimento();

s.nome = txtNome.getText();

s.miligramagem = txtMiligramagem.getText();

s.tipo = txtTipo.getText();

s.quantidade = Integer.parseInt(txtQuantidade.getText());

s.valor\_unitario = Double.parseDouble(txtValor.getText());

Gson gson = new Gson();

String json = gson.toJson(s);

URL url = new URL("http://localhost:4567/api/suprimentos");

HttpURLConnection conn = (HttpURLConnection) url.openConnection();

conn.setRequestMethod("POST");

conn.setDoOutput(true);

conn.setRequestProperty("Content-Type", "application/json");

try (OutputStream os = conn.getOutputStream()) {

os.write(json.getBytes());

os.flush();

}

int status = conn.getResponseCode();

if (status == 201) {

txtStatus.setText("✅ Suprimento cadastrado com sucesso!");

limparCampos();

} else {

txtStatus.setText("❌ Erro: código " + status);

}

} catch (Exception e) {

txtStatus.setText("❌ Erro: " + e.getMessage());

}

}

private void limparCampos() {

txtNome.setText("");

txtMiligramagem.setText("");

txtTipo.setText("");

txtQuantidade.setText("");

txtValor.setText("");

}

public static void main(String[] args) {

SwingUtilities.invokeLater(() -> new CadastroSuprimentoFrame().setVisible(true));

}

}

**📄 Classe modelo: Suprimento.java**

public class Suprimento {

public int id;

public String nome;

public String miligramagem;

public String tipo;

public int quantidade;

public double valor\_unitario;

}

**🧪 Teste agora:**

1. Garanta que sua **API Spark esteja rodando** com o endpoint:
   * POST /api/suprimentos
   * Recebe JSON com os campos nome, miligramagem, tipo, quantidade, valor\_unitario
2. Execute o CadastroSuprimentoFrame
3. Preencha o formulário e clique em "Salvar"

**🔄 Próximos passos possíveis:**

* ✅ Adicionar validação nos campos (ex: número positivo)
* ✅ Adicionar listagem com JTable
* ✅ Usar PUT para atualizar registros existentes
* ✅ Integrar com um banco real (PostgreSQL ou SQLite)

Quer que eu crie a **tela de edição** com JTable + botão "Editar"?

Perfeito! Vamos criar uma **interface Swing com JTable** para:

1. **Listar todos os suprimentos**
2. **Selecionar um item para edição**
3. **Editar e salvar via PUT para a API**

**✅ Requisitos**

* API funcionando em http://localhost:4567/api/suprimentos
* Endpoints:
  + GET /api/suprimentos → lista suprimentos
  + PUT /api/suprimentos/{id} → atualiza suprimento

**📄 1. Classe TelaSuprimentos.java (listagem + edição)**

java

CopiarEditar

import javax.swing.\*;

import javax.swing.table.DefaultTableModel;

import java.awt.\*;

import java.io.\*;

import java.net.\*;

import java.util.List;

import com.google.gson.reflect.TypeToken;

import com.google.gson.\*;

public class TelaSuprimentos extends JFrame {

private JTable tabela;

private DefaultTableModel modelo;

private JTextField txtNome, txtMiligramagem, txtTipo, txtQuantidade, txtValor;

private JButton btnAtualizar;

private Gson gson = new Gson();

private List<Suprimento> lista;

public TelaSuprimentos() {

setTitle("Gerenciar Suprimentos");

setSize(800, 400);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

setLayout(new BorderLayout());

modelo = new DefaultTableModel(new String[]{"ID", "Nome", "Tipo", "Qtd", "Valor"}, 0);

tabela = new JTable(modelo);

add(new JScrollPane(tabela), BorderLayout.CENTER);

JPanel form = new JPanel(new GridLayout(6, 2));

txtNome = new JTextField(); txtMiligramagem = new JTextField();

txtTipo = new JTextField(); txtQuantidade = new JTextField(); txtValor = new JTextField();

btnAtualizar = new JButton("Salvar Alterações");

form.add(new JLabel("Nome:")); form.add(txtNome);

form.add(new JLabel("Miligramagem:")); form.add(txtMiligramagem);

form.add(new JLabel("Tipo:")); form.add(txtTipo);

form.add(new JLabel("Quantidade:")); form.add(txtQuantidade);

form.add(new JLabel("Valor Unitário:")); form.add(txtValor);

form.add(btnAtualizar);

add(form, BorderLayout.SOUTH);

carregarSuprimentos();

tabela.getSelectionModel().addListSelectionListener(e -> {

int i = tabela.getSelectedRow();

if (i >= 0) {

Suprimento s = lista.get(i);

txtNome.setText(s.nome);

txtMiligramagem.setText(s.miligramagem);

txtTipo.setText(s.tipo);

txtQuantidade.setText(String.valueOf(s.quantidade));

txtValor.setText(String.valueOf(s.valor\_unitario));

}

});

btnAtualizar.addActionListener(e -> atualizarSuprimento());

}

private void carregarSuprimentos() {

try {

URL url = new URL("http://localhost:4567/api/suprimentos");

HttpURLConnection conn = (HttpURLConnection) url.openConnection();

conn.setRequestMethod("GET");

lista = gson.fromJson(new InputStreamReader(conn.getInputStream()),

new TypeToken<List<Suprimento>>() {}.getType());

modelo.setRowCount(0);

for (Suprimento s : lista) {

modelo.addRow(new Object[]{s.id, s.nome, s.tipo, s.quantidade, s.valor\_unitario});

}

} catch (Exception e) {

JOptionPane.showMessageDialog(this, "Erro ao carregar: " + e.getMessage());

}

}

private void atualizarSuprimento() {

int row = tabela.getSelectedRow();

if (row < 0) {

JOptionPane.showMessageDialog(this, "Selecione um suprimento para atualizar.");

return;

}

Suprimento s = lista.get(row);

s.nome = txtNome.getText();

s.miligramagem = txtMiligramagem.getText();

s.tipo = txtTipo.getText();

s.quantidade = Integer.parseInt(txtQuantidade.getText());

s.valor\_unitario = Double.parseDouble(txtValor.getText());

try {

URL url = new URL("http://localhost:4567/api/suprimentos/" + s.id);

HttpURLConnection conn = (HttpURLConnection) url.openConnection();

conn.setRequestMethod("PUT");

conn.setRequestProperty("Content-Type", "application/json");

conn.setDoOutput(true);

String json = gson.toJson(s);

try (OutputStream os = conn.getOutputStream()) {

os.write(json.getBytes());

}

if (conn.getResponseCode() == 200) {

JOptionPane.showMessageDialog(this, "Atualizado com sucesso!");

carregarSuprimentos();

} else {

JOptionPane.showMessageDialog(this, "Erro ao atualizar: código " + conn.getResponseCode());

}

} catch (Exception ex) {

JOptionPane.showMessageDialog(this, "Erro: " + ex.getMessage());

}

}

public static void main(String[] args) {

SwingUtilities.invokeLater(() -> new TelaSuprimentos().setVisible(true));

}

}

**📄 2. Modelo Suprimento.java (reutilizado)**

java

CopiarEditar

public class Suprimento {

public int id;

public String nome;

public String miligramagem;

public String tipo;

public int quantidade;

public double valor\_unitario;

}

**🧪 Testando:**

1. Execute sua API Spark
2. Certifique-se de ter ao menos um suprimento cadastrado
3. Execute TelaSuprimentos
4. Clique na linha de um suprimento
5. Edite os dados e clique em “Salvar Alterações”

**✅ Próximos passos possíveis**

* Adicionar botão "Excluir" (via DELETE)
* Adicionar validação de campos (ex: try/catch para números)
* Usar JFormattedTextField ou JSpinner para números

Perfeito! Vamos adicionar a função de **exclusão de suprimentos** com um botão “Remover Suprimento” usando o método **DELETE da API**.

Abaixo está a versão atualizada da tela TelaSuprimentos.java com o botão de exclusão:

**✅ Atualizações na tela: TelaSuprimentos.java**

**🔄 1. Adicionar botão de exclusão no layout**

No construtor, adicione:

private JButton btnRemover; // Adicione esta linha no topo com os outros campos

Substitua a parte onde está o btnAtualizar e o painel inferior por:

btnAtualizar = new JButton("Salvar Alterações");

btnRemover = new JButton("Remover Suprimento");

JPanel botoes = new JPanel();

botoes.add(btnAtualizar);

botoes.add(btnRemover);

JPanel form = new JPanel(new GridLayout(6, 2));

txtNome = new JTextField(); txtMiligramagem = new JTextField();

txtTipo = new JTextField(); txtQuantidade = new JTextField(); txtValor = new JTextField();

form.add(new JLabel("Nome:")); form.add(txtNome);

form.add(new JLabel("Miligramagem:")); form.add(txtMiligramagem);

form.add(new JLabel("Tipo:")); form.add(txtTipo);

form.add(new JLabel("Quantidade:")); form.add(txtQuantidade);

form.add(new JLabel("Valor Unitário:")); form.add(txtValor);

form.add(botoes); // adiciona painel com os dois botões

add(form, BorderLayout.SOUTH);

**🗑️ 2. Lógica para exclusão (btnRemover)**

Abaixo do btnAtualizar.addActionListener(...), adicione:

btnRemover.addActionListener(e -> removerSuprimento());

E implemente o método removerSuprimento():

private void removerSuprimento() {

int row = tabela.getSelectedRow();

if (row < 0) {

JOptionPane.showMessageDialog(this, "Selecione um suprimento para remover.");

return;

}

Suprimento s = lista.get(row);

int confirm = JOptionPane.showConfirmDialog(this,

"Deseja realmente remover o suprimento '" + s.nome + "'?",

"Confirmação", JOptionPane.YES\_NO\_OPTION);

if (confirm != JOptionPane.YES\_OPTION) return;

try {

URL url = new URL("http://localhost:4567/api/suprimentos/" + s.id);

HttpURLConnection conn = (HttpURLConnection) url.openConnection();

conn.setRequestMethod("DELETE");

int status = conn.getResponseCode();

if (status == 200) {

JOptionPane.showMessageDialog(this, "Removido com sucesso!");

carregarSuprimentos();

limparCampos();

} else {

JOptionPane.showMessageDialog(this, "Erro ao remover: código " + status);

}

} catch (Exception ex) {

JOptionPane.showMessageDialog(this, "Erro: " + ex.getMessage());

}

}

**✅ 3. Método auxiliar: limpar campos após excluir**

Adicione ao final da classe:

private void limparCampos() {

txtNome.setText("");

txtMiligramagem.setText("");

txtTipo.setText("");

txtQuantidade.setText("");

txtValor.setText("");

}

**🧪 Como testar**

1. Execute a API com suporte ao DELETE /api/suprimentos/{id}
2. Execute TelaSuprimentos
3. Selecione um suprimento
4. Clique em “Remover Suprimento”
5. Confirme a exclusão

**🎯 Recursos implementados**

✅ Listar suprimentos com GET  
✅ Editar suprimentos com PUT  
✅ Remover suprimentos com DELETE  
✅ Interface gráfica responsiva e simples

Se quiser, posso empacotar tudo isso num **projeto Maven completo (ZIP ou GitHub)** ou te ajudar a transformar esse projeto em um .jar executável.

Quer isso?