🔧 Documentação Técnica - Terasteel Bookstore

📋 Especificações Técnicas Detalhadas

Frontend - Arquitetura JavaScript

Estrutura de Dados Principal
```javascript
// Objeto Livro
{
  id: Number,
  nome: String,
  autor: String,
  categoria: String,
  preco: Number,
  descricao: String,
  estoque: Number,
  promo: Boolean,
  imagem: String
}

// Estado do Carrinho
cart = [
  {
    ...livro,
    quantity: Number
  }
]

// Filtros Ativos
currentFilter = {
  categoria: String, // "todos" | categoria específica
  promo: Boolean     // true para apenas promoções
}
```

Funções Principais (script.js)

Navegação e Seções
```javascript
showSection(sectionId)          // Alterna entre seções da SPA
```

Sistema de Filtros
```javascript
setCategoryFilter(categoria)    // Define filtro de categoria
setPromoFilter(onlyPromo)       // Define filtro de promoção
searchBooks(term)               // Busca textual
```

```
getFilteredBooks()            // Aplica todos os filtros
```

Renderização
```javascript
renderBooks(customList)       // Renderiza grid de livros
escapeHtml(text)              // Sanitização de HTML
```

Carrinho de Compras
```javascript
addToCart(bookId)             // Adiciona item ao carrinho
updateCartDisplay()           // Atualiza interface do carrinho
changeQuantity(bookId, change) // Altera quantidade
removeFromCart(bookId)        // Remove item
toggleCart()                  // Abre/fecha sidebar
checkout()                    // Redireciona para finalização
```

Backend - API RESTful

Configuração do Servidor (server.js)
```javascript
const express = require("express");
const app = express();
const PORT = 3000;

app.use(express.json());
app.use("/livros", livrosRoutes);

app.listen(PORT, () => {
  console.log(`Servidor rodando em http://localhost:${PORT}`);
});
```

Rotas da API (routes/livros.js)

Configuração do Banco
```javascript
const pool = new Pool({
  user: "postgres",
  host: "localhost",
  database: "livraria",
  password: "123456",
  port: 5432,
});
```

Endpoints Implementados

1. **GET /livros** - Listar todos
```javascript
router.get("/", async (req, res) => {
  const result = await pool.query("SELECT * FROM livros");
  res.json(result.rows);
});
```

2. **GET /livros/:id** - Buscar por ID
```javascript
router.get("/:id", async (req, res) => {
  const result = await pool.query("SELECT * FROM livros WHERE id = $1", [id]);
  res.json(result.rows[0]);
});
```

3. **POST /livros** - Cadastrar novo
```javascript
router.post("/", async (req, res) => {
  const { nome, preco, descricao, estoque } = req.body;
  const result = await pool.query(
    "INSERT INTO livros (nome, preco, descricao, estoque) VALUES ($1, $2, $3, $4)
RETURNING *",
    [nome, preco, descricao, estoque]
  );
});
```

4. **PUT /livros/:id** - Atualizar
```javascript
router.put("/:id", async (req, res) => {
  const result = await pool.query(
    "UPDATE livros SET nome = $1, preco = $2, descricao = $3, estoque = $4 WHERE id =
$5 RETURNING *",
    [nome, preco, descricao, estoque, id]
  );
});
```

5. **DELETE /livros/:id** - Remover
```javascript
router.delete("/:id", async (req, res) => {
  await pool.query("DELETE FROM livros WHERE id = $1", [id]);
  res.status(204).send();
});
```

🎨 Estrutura CSS

Variáveis CSS Principais
```css
:root {
  --primary-color: #5a007a;
  --secondary-color: #ffffff;
  --accent-color: #ff6b35;
  --text-color: #333333;
  --border-radius: 8px;
  --box-shadow: 0 2px 10px rgba(0,0,0,0.1);
}
```

Classes Principais
- `.container` - Layout centralizado
- `.header-content` - Cabeçalho flexível
- `.books-grid` - Grid responsivo de livros
- `.book-card` - Card individual do livro
- `.cart-sidebar` - Carrinho lateral
- `.btn-primary` / `.btn-secondary` - Botões estilizados
- `.section` / `.section.active` - Seções da SPA

Responsividade
```css
/* Mobile First */
@media (min-width: 768px) { /* Tablet */ }
@media (min-width: 1024px) { /* Desktop */ }
```

📊 Fluxo de Dados

1. Carregamento Inicial
```
index.html → script.js → renderBooks() → Exibe catálogo
```

2. Filtros
```
Usuário seleciona filtro → setCategoryFilter() → getFilteredBooks() → renderBooks()
```

3. Busca
```
Input de busca → searchBooks() → Filtra array → renderBooks()
```

4. Carrinho
```
Clique "Adicionar" → addToCart() → Atualiza array cart → updateCartDisplay()
```

5. Checkout
```
Finalizar compra → checkout() → Redireciona → checkout.html → Processa pedido
```

🔐 Validações Implementadas

Frontend
- **Formulários**: Required fields, tipos de input
- **Carrinho**: Validação de estoque antes de adicionar
- **Busca**: Sanitização de entrada
- **HTML**: Escape de caracteres especiais

Backend
- **Campos obrigatórios**: nome, preco, estoque
- **Tipos de dados**: Validação de números e strings
- **SQL Injection**: Prepared statements
- **Existência**: Verificação antes de UPDATE/DELETE

🚀 Performance

Otimizações Frontend
- **Lazy Loading**: Imagens carregadas conforme necessário
- **Event Delegation**: Eventos eficientes no DOM
- **Debounce**: Busca com delay para evitar requisições excessivas
- **Minificação**: CSS e JS otimizados

Otimizações Backend
- **Connection Pool**: Reutilização de conexões DB
- **Async/Await**: Operações não-bloqueantes
- **Error Handling**: Tratamento adequado de erros
- **CORS**: Configuração para requisições cross-origin

🧪 Testes Sugeridos

Testes Funcionais
1. **Navegação**: Todas as páginas carregam corretamente
2. **Filtros**: Cada categoria filtra adequadamente
3. **Busca**: Encontra livros por título/autor/categoria
4. **Carrinho**: Adicionar/remover/alterar quantidade
5. **Checkout**: Processo completo de finalização
6. **Formulários**: Validação e envio

Testes de API
```bash
# Listar livros
curl http://localhost:3000/livros

# Buscar livro específico
curl http://localhost:3000/livros/1

# Cadastrar livro
curl -X POST http://localhost:3000/livros \
  -H "Content-Type: application/json" \
  -d '{"nome":"Teste","preco":25.90,"descricao":"Teste","estoque":5}'
```

Testes de Responsividade
- **Mobile**: 320px - 767px
- **Tablet**: 768px - 1023px
- **Desktop**: 1024px+

📈 Métricas de Qualidade

Código
- **Modularidade**: Funções específicas e reutilizáveis
- **Legibilidade**: Nomes descritivos e comentários
- **Manutenibilidade**: Estrutura organizada
- **Escalabilidade**: Fácil adição de novas funcionalidades

UX/UI
- **Usabilidade**: Interface intuitiva
- **Acessibilidade**: Semântica HTML adequada
- **Performance**: Carregamento rápido
- **Responsividade**: Funciona em todos os dispositivos

🔧 Configurações de Desenvolvimento

Dependências Backend
```json
{
  "dependencies": {
    "cors": "^2.8.5",
    "express": "^4.18.2",
    "pg": "^8.11.3"
  },
  "devDependencies": {
    "nodemon": "^3.0.1"
  }
}
```

Scripts NPM
```json
{
  "scripts": {
    "start": "node server.js",
    "dev": "nodemon server.js"
  }
}
```

Estrutura de Desenvolvimento
```
Desenvolvimento Local:
├── Frontend: Servidor local (Live Server/http-server)
├── Backend: http://localhost:3000
├── Database: PostgreSQL local
└── Assets: Imagens servidas estaticamente
```

Documentação técnica atualizada em 2025