Guia de Configuração de IDE

Este guia explica como configurar qualquer IDE para trabalhar com o Motor Procedural Educacional.

📋 Arquivos de Configuração

O projeto inclui configurações para múltiplas IDEs:

Arquivo	IDE	Propósito
`project.json`	Todas (JSON Schema)	Configuração estruturada do projeto
`project.yaml`	Todas (YAML)	Configuração legível do projeto
`.vscode/settings.json`	VSCode	Configurações do editor
`.vscode/extensions.json`	VSCode	Extensões recomendadas
`.vscode/launch.json`	VSCode	Configurações de debugging
`.vscode/tasks.json`	VSCode	Tarefas automatizadas
`.editorconfig`	Todas	Configuração universal de formatação

🎯 VSCode (Recomendado)

1. Instalação Rápida

Bash

```
# Clonar repositório
git clone https://github.com/seu-usuario/motor-procedural-edu.git
cd motor-procedural-edu

# Abrir no VSCode
code .
```

2. Instalar Extensões

VSCode sugerirá automaticamente as extensões em `.vscode/extensions.json`. Clique em "Install All" ou instale manualmente:

```bash
code --install-extension dbaeumer.vscode-eslint
code --install-extension esbenp.prettier-vscode
code --install-extension ms-vscode.vscode-typescript-next
code --install-extension firsttris.vscode-jest-runner
code --install-extension eamodio.gitlens
```

3. Instalar Dependências

```bash
pnpm install
```

4. Usar Tarefas

Abra a paleta de comandos (`Ctrl+Shift+P`) e procure por "Tasks":

- **pnpm install** - Instalar dependências
- **pnpm check** - Verificar tipos TypeScript
- **pnpm test** - Rodar testes
- **pnpm test (watch)** - Rodar testes em tempo real
- **pnpm format** - Formatar código
- **Exemplo: Dungeon** - Executar exemplo
- **Validação: Integração** - Validar integração
- **Build Completo** - Verificar tipos + rodar testes

5. Debugging

Abra a aba "Run and Debug" (`Ctrl+Shift+D`) e selecione uma configuração:

- **Executar Exemplo Dungeon** - Rodar exemplo com breakpoints
- **Executar Validação de Integração** - Validar com breakpoints
- **Rodar Testes** - Rodar testes com debugging
- **Rodar Testes (Watch)** - Testes em tempo real
- **Verificar TypeScript** - Verificar tipos

- **Validação Completa** - Verificar tipos + testes

6. Configurações Automáticas

VSCode aplicará automaticamente:

- ✅ Formatação com Prettier ao salvar
- ✅ Linting com ESLint
- ✅ TypeScript strict mode
- ✅ Indentação 2 espaços
- ✅ Quebra de linha LF
- ✅ Régua em 100 caracteres

🧠 JetBrains (IntelliJ, WebStorm, etc.)

1. Abrir Projeto

```Bash
# Abrir com IntelliJ
idea .

# Ou com WebStorm
webstorm .
```

2. Configurar SDK

1. Vá para **File → Project Structure**
2. Selecione **Node.js 22+** como SDK
3. Clique em **OK**

3. Instalar Dependências

JetBrains detectará `package.json` e sugerirá instalar. Clique em "Install" ou:

```Bash
pnpm install
```

4. Executar Tarefas

1. Abra **Run → Edit Configurations**
2. Clique em **+** e selecione **npm**
3. Configure:
   - **Name**: pnpm check
   - **Command**: check
   - **Package**: motor-procedural-edu

Repita para `test`, `format`, etc.

5. Debugging

1. Clique com botão direito em `examples/example-dungeon.ts`
2. Selecione **Run 'example-dungeon.ts'** ou **Debug 'example-dungeon.ts'**

6. Configurações Automáticas

JetBrains aplicará automaticamente:

- ✅ Formatação com Prettier
- ✅ Linting com ESLint
- ✅ TypeScript strict mode
- ✅ Indentação 2 espaços

🚀 Cursor (AI-Powered)

1. Abrir Projeto

```bash
cursor .
```

2. Usar AI Context

Cursor lerá `project.json` automaticamente para contexto de IA. Você pode:

- Pedir para "gerar novo algoritmo de PCG"
- Pedir para "criar adaptador Unity"
- Pedir para "otimizar performance"

Cursor entenderá a arquitetura e sugerirá mudanças alinhadas.

3. Instalar Dependências

```
Bash

pnpm install
```

4. Usar Tarefas

Mesmas tarefas do VSCode (Cursor é baseado em VSCode).

5. Debugging

Mesmas configurações do VSCode.

📱 GitHub Codespaces

1. Abrir no Codespaces

1. Vá para o repositório no GitHub
2. Clique em **Code → Codespaces → Create codespace on main**

2. Instalar Dependências

```
Bash

pnpm install
```

3. Usar VSCode Online

Codespaces é VSCode online, então use as mesmas tarefas e debugging.

🔧 Neovim / Vim

1. Instalar LSP

```
Bash

# Com vim-plug
Plug 'neovim/nvim-lspconfig'
Plug 'williamboman/mason.nvim'
Plug 'williamboman/mason-lspconfig.nvim'
```

2. Configurar TypeScript

Lua

```
require("lspconfig").tsserver.setup({})
```

3. Instalar Dependências

Bash

```
pnpm install
```

4. Rodar Tarefas

Bash

```
pnpm check
pnpm test
pnpm tsx examples/example-dungeon.ts
```

📊 Compreendendo project.json

O arquivo `project.json` contém metadados que qualquer IDE pode ler:

JSON

```
{
  "ecosystem": {
    "roles": {
      "aprendiz": { "api": "src/edu/api/educationalApi.ts" },
      "engenheiro": { "api": "src/core/**" },
      "fundador": { "api": "src/edu/api/educationalApi.ts" }
    }
  },
  "architecture": {
    "layers": {
      "api": { "path": "src/edu/api", "exports": [...] },
      "core": { "path": "src/core", "modules": {...} }
    }
  },
  "evolutionGuidance": {
    "nextSteps": [
      { "title": "Interface Web", "effort": "medium", "impact": "high" }
```

```
      ]
    }
  }
```

IDEs podem usar isso para:

- **Auto-complete**: Sugerir funções baseado em role
- **Navegação**: Ir para módulo certo baseado em tarefa
- **Refatoração**: Entender dependências
- **Roadmap**: Mostrar próximos passos

🎓 Fluxo de Desenvolvimento

Para Aprendiz

1. Abrir VSCode
2. Executar tarefa "Exemplo: Dungeon"
3. Explorar código em `src/edu/api/`
4. Modificar parâmetros e re-executar

Para Engenheiro

1. Abrir VSCode
2. Executar tarefa "Build Completo"
3. Explorar código em `src/core/`
4. Modificar algoritmo
5. Executar tarefa "pnpm test (watch)"
6. Verificar mudanças em tempo real

Para Fundador

1. Abrir VSCode
2. Explorar `src/edu/api/` (API pública)
3. Verificar `project.json` para roadmap
4. Planejar próximas features

🐛 Troubleshooting

Erro: "Cannot find module 'typescript'"

```Bash
pnpm install
```

Erro: "pnpm: command not found"

```Bash
npm install -g pnpm
```

Erro: "Node.js version too old"

```Bash
# Instalar Node.js 22+
nvm install 22
nvm use 22
```

Erro: "ESLint not working"

1. Abra paleta de comandos (`Ctrl+Shift+P`)
2. Procure por "ESLint: Restart ESLint Server"

Erro: "Prettier not formatting"

1. Abra paleta de comandos (`Ctrl+Shift+P`)
2. Procure por "Format Document"
3. Selecione "Prettier - Code formatter"

📚 Referências

- VSCode Documentation
- JetBrains Documentation
- Cursor Documentation
- Neovim LSP
- project.json Schema
- project.yaml Schema

✨ Dicas

1. **Use tarefas**: Evita digitar comandos repetidamente
2. **Use debugging**: Entenda o fluxo de geração
3. **Use AI (Cursor)**: Peça sugestões baseado em arquitetura
4. **Leia project.json**: Entenda roadmap e próximos passos
5. **Siga as regras**: Prettier + ESLint garantem qualidade

Última atualização: 10 de Janeiro de 2026

Versão: 1.0.0