Resumo da Arquitetura do Projeto ZoyBlock

Antes: Estrutura Monolítica

Na versão antiga, toda a lógica da aplicação estava centralizada em um único arquivo app.py, incluindo:
- Rotas Flask
- Comunicação Serial com o Arduino
- Execução do código Python gerado pelo Blockly
- Interface gráfica com Tkinter
- Configuração de mDNS e servidor

Estrutura típica:
src/
static/
assets/
app.py (TUDO aqui dentro)

Problemas:
- Código acoplado e difícil de manter
- Baixa reutilização e testabilidade
- Pouca escalabilidade com novos dispositivos e serviços

Depois: Arquitetura Modularizada

A nova estrutura separa responsabilidades em camadas e módulos especializados:

app/
  routes/   -> Rotas HTTP com Flask
  services/   -> Lógica de negócio (Arduino, Wi-Fi, sensores)
  templates/   -> HTML e interface web
  static/   -> CSS, JS, imagens, Blockly
  __init__.py   -> Inicialização do app Flask
run.py   -> Ponto de entrada (Flask + Tkinter)

Benefícios:
- Código limpo e reutilizável
- Facilita testes e manutenção
- Suporte nativo a múltiplos dispositivos e expansões futuras

Resumo da Arquitetura do Projeto ZoyBlock

- Mantém compatibilidade com interface gráfica Tkinter

Funcionamento da Nova Arquitetura

- services/: armazena lógica e estado compartilhado (ex: conexão com Arduino)
- routes/: define as rotas HTTP que consomem os serviços
- run.py: inicia o Flask, configura o servidor e exibe interface Tkinter
- Blockly envia código Python para /executar, que é interpretado com exec()

Toda comunicação com o Arduino é centralizada em serial_service.py com as funções:
- conectar(porta)
- obter_dispositivo()
- listar_portas()

Como Criar Novas Rotas

1. Criar um novo arquivo em app/routes/, por exemplo sensor_routes.py:

```
from flask import Blueprint, jsonify
from app.services.sensor_service import ler_sensor

sensor_bp = Blueprint("sensor", __name__)
@sensor_bp.route("/sensor/linha")
def sensor_linha():
    return jsonify({"valor": ler_sensor()})
```

2. Registrar o Blueprint em app/__init__.py:

```
app.register_blueprint(sensor_bp)
```

Como Criar um Novo Serviço

1. Criar um arquivo em app/services/, por exemplo sensor_service.py:

```
def ler_sensor():
    return 123
```

2. Usar o serviço em qualquer rota:

```
from app.services.sensor_service import ler_sensor
```

Resumo da Arquitetura do Projeto ZoyBlock

Conclusão

A nova arquitetura melhora a organização, escalabilidade e manutenção do projeto.

Agora é possível adicionar novos robôs, sensores e firmwares de forma limpa e modular.