*#!/usr/bin/env python3*

"""

Production server for ArcSpatialDB with automatic reconnection

This server will automatically restart if it fails or crashes

"""

*import* time

*import* sys

*import* os

*import* signal

*import* logging

*from* datetime *import* datetime

*from* waitress *import* serve

*from* app *import* app

*# Set up logging*

logging.basicConfig(

*level*=logging.INFO,

*format*='%(asctime)s - %(levelname)s - %(message)s',

*handlers*=[

        logging.FileHandler('server.log'),

        logging.StreamHandler(sys.stdout)

    ]

)

logger = logging.getLogger(\_\_name\_\_)

class AutoReconnectServer:

    def \_\_init\_\_(*self*, *max\_retries*=5, *retry\_delay*=10):

*self*.max\_retries = *max\_retries*

*self*.retry\_delay = *retry\_delay*

*self*.retry\_count = 0

*self*.running = True

*# Get configuration*

*try*:

*from* config *import* FLASK\_HOST, FLASK\_PORT

*self*.host = FLASK\_HOST

*self*.port = FLASK\_PORT

*except* ImportError:

*# Fallback configuration*

*self*.host = "0.0.0.0"

*self*.port = 5000

*# Set up signal handlers for graceful shutdown*

        signal.signal(signal.SIGINT, *self*.signal\_handler)

        signal.signal(signal.SIGTERM, *self*.signal\_handler)

    def signal\_handler(*self*, *signum*, *frame*):

        """Handle shutdown signals gracefully"""

        logger.info(f"Received signal {*signum*}. Shutting down gracefully...")

*self*.running = False

        sys.exit(0)

    def start\_server(*self*):

        """Start the server with automatic reconnection"""

        logger.info("🚀 Starting ArcSpatialDB Production Server with Auto-Reconnect")

        logger.info(f"📍 Host: {*self*.host}")

        logger.info(f"🔌 Port: {*self*.port}")

        logger.info(f"🌐 URL: http://{*self*.host}:{*self*.port}")

        logger.info("=" \* 50)

*while* *self*.running and *self*.retry\_count < *self*.max\_retries:

*try*:

                logger.info(f"🔄 Attempt {*self*.retry\_count + 1}/{*self*.max\_retries}")

                logger.info("✅ Starting server...")

*# Start the production server*

                serve(app, *host*=*self*.host, *port*=*self*.port, *threads*=4)

*except* KeyboardInterrupt:

                logger.info("🛑 Server stopped by user")

*self*.running = False

*break*

*except* Exception *as* e:

*self*.retry\_count += 1

                logger.error(f"❌ Server failed: {e}")

*if* *self*.retry\_count < *self*.max\_retries:

                    logger.info(f"⏳ Waiting {*self*.retry\_delay} seconds before retry...")

                    time.sleep(*self*.retry\_delay)

*# Increase delay for next retry (exponential backoff)*

*self*.retry\_delay = min(*self*.retry\_delay \* 2, 300)  *# Max 5 minutes*

*else*:

                    logger.error(f"❌ Maximum retries ({*self*.max\_retries}) reached. Server will not restart.")

*break*

*if* not *self*.running:

            logger.info("✅ Server shutdown completed")

*else*:

            logger.error("❌ Server failed permanently")

def main():

    """Main function to start the auto-reconnect server"""

    server = AutoReconnectServer(*max\_retries*=5, *retry\_delay*=10)

    server.start\_server()

*if* \_\_name\_\_ == '\_\_main\_\_':

    main()