```
# main.py
# Main orchestrator for the Network Traffic Analyzer system

import argparse
import logging
import signal
import sys
import threading
import time
from pathlib import Path

from database import DatabaseManager
from packet_capture import PacketCapture
from api_server import NetworkAnalyzerAPI

class NetworkAnalyzerSystem:
    def __init__(self, config):
        """Initialize the complete network analyzer system"""
        self.config = config
        self.running = False
        self.threads = []

        # Setup logging
        self.setup_logging()
        self.logger = logging.getLogger(__name__)

        # Initialize components
        self.db = None
        self.capture = None
        self.api = None

        # Setup signal handlers
        signal.signal(signal.SIGINT, self.signal_handler)
        signal.signal(signal.SIGTERM, self.signal_handler)

    def setup_logging(self):
        """Configure logging for the system"""
        log_level = getattr(logging, self.config.log_level.upper())
        log_format = '%(asctime)s - %(name)s - %(levelname)s - %(message)s'

        # Configure root logger
        logging.basicConfig(
            level=log_level,
            format=log_format,
            handlers=[
                logging.StreamHandler(sys.stdout),
                logging.FileHandler('network_analyzer.log') if self.config.log_file else
logging.NullHandler()
```

```
            ]
        )

    def signal_handler(self, signum, frame):
        """Handle system signals for graceful shutdown"""
        self.logger.info(f"Received signal {signum}, shutting down...")
        self.stop()

    def start(self):
        """Start all system components"""
        self.logger.info("Starting Network Traffic Analyzer System")

        try:
            # Initialize database
            self.init_database()

            # Start packet capture if enabled
            if self.config.enable_capture:
                self.start_packet_capture()

            # Start API server if enabled
            if self.config.enable_api:
                self.start_api_server()

            # Start maintenance tasks
            self.start_maintenance()

            self.running = True
            self.logger.info("System started successfully")

            # Wait for shutdown signal
            self.wait_for_shutdown()

        except Exception as e:
            self.logger.error(f"Failed to start system: {e}")
            self.stop()
            return False

        return True

    def init_database(self):
        """Initialize database connection and schema"""
        self.logger.info("Initializing database connection")

        db_config = {
            'host': self.config.db_host,
            'port': self.config.db_port,
            'database': self.config.db_name,
            'user': self.config.db_user,
            'password': self.config.db_password,
            'min_conn': self.config.db_min_conn,
            'max_conn': self.config.db_max_conn
        }

        self.db = DatabaseManager(**db_config)
        self.logger.info("Database connection established")

    def start_packet_capture(self):
        """Start packet capture in a separate thread"""
        self.logger.info("Starting packet capture module")

        db_config = {
            'host': self.config.db_host,
            'port': self.config.db_port,
            'database': self.config.db_name,
            'user': self.config.db_user,
            'password': self.config.db_password
        }

        self.capture = PacketCapture(
            interface=self.config.interface,
            node_id=self.config.node_id,
            db_config=db_config
        )

        # Start capture in separate thread
        capture_thread = threading.Thread(
            target=self.capture.start_capture,
            args=(self.config.packet_filter,),
            daemon=True
        )
        capture_thread.start()
        self.threads.append(capture_thread)

        self.logger.info(f"Packet capture started on interface {self.config.interface}")

    def start_api_server(self):
        """Start API server in a separate thread"""
        self.logger.info("Starting API server")

        db_config = {
            'host': self.config.db_host,
            'port': self.config.db_port,
            'database': self.config.db_name,
            'user': self.config.db_user,
            'password': self.config.db_password
        }

        self.api = NetworkAnalyzerAPI(db_config)

        # Start API server in separate thread
        api_thread = threading.Thread(
            target=self.api.run,
            kwargs={
                'host': self.config.api_host,
                'port': self.config.api_port,
                'debug': self.config.api_debug
            },
            daemon=True
        )
        api_thread.start()
        self.threads.append(api_thread)

        self.logger.info(f"API server started on {self.config.api_host}:{self.config.api_port}")

    def start_maintenance(self):
        """Start maintenance tasks"""
        if self.config.enable_cleanup:
            self.logger.info("Starting maintenance tasks")

            maintenance_thread = threading.Thread(
                target=self.maintenance_worker,
                daemon=True
            )
            maintenance_thread.start()
            self.threads.append(maintenance_thread)

    def maintenance_worker(self):
        """Background maintenance tasks"""
        while self.running:
            try:
                # Sleep for cleanup interval
                time.sleep(self.config.cleanup_interval * 3600)  # Convert hours to seconds

                if self.db and self.running:
                    self.logger.info("Running database cleanup")
                    self.db.cleanup_old_data(self.config.data_retention_days)

            except Exception as e:
                self.logger.error(f"Maintenance error: {e}")

    def wait_for_shutdown(self):
        """Wait for shutdown signal"""
        try:
            while self.running:
                time.sleep(1)
        except KeyboardInterrupt:
            pass

    def stop(self):
        """Stop all system components"""
        self.logger.info("Stopping Network Traffic Analyzer System")
        self.running = False

        # Stop packet capture
        if self.capture:
            self.capture.stop_capture()

        # Close database connections
        if self.db:
            self.db.close_connections()

        # Wait for threads to finish (with timeout)
        for thread in self.threads:
            if thread.is_alive():
                thread.join(timeout=5)

        self.logger.info("System stopped")

    def get_status(self):
        """Get system status"""
        status = {
            'running': self.running,
            'components': {
                'database': self.db is not None,
                'capture': self.capture is not None and self.capture.running,
                'api': self.api is not None,
            },
            'threads': len([t for t in self.threads if t.is_alive()])
        }

        if self.capture:
            status['capture_stats'] = self.capture.get_stats()

        return status

class SystemConfig:
    """Configuration class for the system"""
    def __init__(self, args):
        # Database configuration
        self.db_host = args.db_host
        self.db_port = args.db_port
        self.db_name = args.db_name
        self.db_user = args.db_user
        self.db_password = args.db_password
        self.db_min_conn = args.db_min_conn
        self.db_max_conn = args.db_max_conn

        # Packet capture configuration
        self.enable_capture = args.enable_capture
        self.interface = args.interface
        self.node_id = args.node_id
        self.packet_filter = args.packet_filter

        # API server configuration
        self.enable_api = args.enable_api
        self.api_host = args.api_host
        self.api_port = args.api_port
        self.api_debug = args.api_debug

        # Maintenance configuration
        self.enable_cleanup = args.enable_cleanup
        self.cleanup_interval = args.cleanup_interval
        self.data_retention_days = args.data_retention_days

        # Logging configuration
        self.log_level = args.log_level
        self.log_file = args.log_file

def main():
    """Main function"""
    parser = argparse.ArgumentParser(
        description='Network Traffic Analyzer - Distributed Security Monitoring System',
        formatter_class=argparse.RawDescriptionHelpFormatter,
        epilog="""
Examples:
  # Start complete system (capture + API)
  python main.py --enable-capture --enable-api

  # Start only packet capture node
  python main.py --enable-capture --node-id capture-node-1

  # Start only API server
  python main.py --enable-api --api-host 0.0.0.0 --api-port 8080

  # Custom database and interface
  python main.py --enable-capture --interface eth0 --db-host 192.168.1.100
        """
    )
    # Database options
    db_group = parser.add_argument_group('Database Configuration')
    db_group.add_argument('--db-host', default='localhost', help='Database host')
    db_group.add_argument('--db-port', type=int, default=5432, help='Database port')
    db_group.add_argument('--db-name', default='network_analyzer', help='Database name')
    db_group.add_argument('--db-user', default='postgres', help='Database user')
    db_group.add_argument('--db-password', default='password', help='Database password')
    db_group.add_argument('--db-min-conn', type=int, default=1, help='Minimum database
connections')
    db_group.add_argument('--db-max-conn', type=int, default=20, help='Maximum database
connections')

    # Capture options
    capture_group = parser.add_argument_group('Packet Capture Configuration')
    capture_group.add_argument('--enable-capture', action='store_true',
                              help='Enable packet capture module')
    capture_group.add_argument('--interface', help='Network interface to monitor')
    capture_group.add_argument('--node-id', help='Unique identifier for this capture node')
    capture_group.add_argument('--packet-filter', default='',
                              help='BPF filter for packet capture')

    # API options
    api_group = parser.add_argument_group('API Server Configuration')
    api_group.add_argument('--enable-api', action='store_true',
                          help='Enable REST API server')
    api_group.add_argument('--api-host', default='0.0.0.0', help='API server host')
    api_group.add_argument('--api-port', type=int, default=5000, help='API server port')
    api_group.add_argument('--api-debug', action='store_true', help='Enable API debug
mode')

    # Maintenance options
    maint_group = parser.add_argument_group('Maintenance Configuration')
    maint_group.add_argument('--enable-cleanup', action='store_true',
                            help='Enable automatic database cleanup')
    maint_group.add_argument('--cleanup-interval', type=int, default=24,
                            help='Cleanup interval in hours')
    maint_group.add_argument('--data-retention-days', type=int, default=30,
                            help='Days to retain data')

    # Logging options
    log_group = parser.add_argument_group('Logging Configuration')
    log_group.add_argument('--log-level', default='INFO',
                          choices=['DEBUG', 'INFO', 'WARNING', 'ERROR'],
                          help='Logging level')
    log_group.add_argument('--log-file', action='store_true',
                          help='Enable logging to file')

    # Parse arguments
    args = parser.parse_args()

    # Validate arguments
    if not args.enable_capture and not args.enable_api:
        parser.error("At least one of --enable-capture or --enable-api must be specified")

    # Create configuration
    config = SystemConfig(args)

    # Create and start system
    system = NetworkAnalyzerSystem(config)

    try:
        success = system.start()
        sys.exit(0 if success else 1)
    except Exception as e:
        print(f"System error: {e}")
        sys.exit(1)

if __name__ == "__main__":
    main()
```