```
#!/usr/bin/env python3
"""
Family Hub - Non-Interactive Automated Setup Script
Handles complete installation without user prompts
"""

import os
import sys
import subprocess
import secrets
import platform
import shutil
from pathlib import Path

class Colors:
    HEADER = '\033[95m'
    BLUE = '\033[94m'
    CYAN = '\033[96m'
    GREEN = '\033[92m'
    YELLOW = '\033[93m'
    RED = '\033[91m'
    END = '\033[0m'
    BOLD = '\033[1m'

def print_header(text):
    print(f"\n{Colors.BOLD}{Colors.HEADER}{'='*60}{Colors.END}")
    print(f"{Colors.BOLD}{Colors.HEADER}{text.center(60)}{Colors.END}")
    print(f"{Colors.BOLD}{Colors.HEADER}{'='*60}{Colors.END}\n")

def print_step(step, text):
    print(f"{Colors.BOLD}{Colors.CYAN}[Step {step}]{Colors.END} {text}")

def print_success(text):
    print(f"{Colors.GREEN}✓ {text}{Colors.END}")

def print_error(text):
    print(f"{Colors.RED}✗ {text}{Colors.END}")

def print_warning(text):
    print(f"{Colors.YELLOW}⚠ {text}{Colors.END}")

def print_info(text):
    print(f"{Colors.BLUE}ℹ {text}{Colors.END}")

def run_command(cmd, check=True, capture=False):
    """Run a shell command"""
    try:
        if capture:
```

```
            result = subprocess.run(cmd, shell=True, check=check,
                                    capture_output=True, text=True)
            return result.stdout.strip()
        else:
            subprocess.run(cmd, shell=True, check=check)
        return True
    except subprocess.CalledProcessError as e:
        if check:
            print_error(f"Command failed: {cmd}")
            print_error(f"Error: {e}")
        return False

def check_python_version():
    """Check if Python version is 3.11+"""
    print_step(1, "Checking Python version...")
    version = sys.version_info
    if version.major == 3 and version.minor >= 11:
        print_success(f"Python {version.major}.{version.minor}.{version.micro} detected")
        return True
    else:
        print_error(f"Python 3.11+ required. Found {version.major}.{version.minor}.
{version.micro}")
        return False

def check_postgresql():
    """Check if PostgreSQL is installed"""
    print_step(2, "Checking PostgreSQL installation...")
    if shutil.which('psql'):
        version = run_command('psql --version', capture=True)
        print_success(f"PostgreSQL found: {version}")
        return True
    else:
        print_warning("PostgreSQL not found - will use default configuration")
        print_info("You can configure database later in .env file")
        return False

def create_virtual_environment():
    """Create Python virtual environment"""
    print_step(3, "Creating virtual environment...")

    if os.path.exists('venv'):
        print_warning("Virtual environment already exists - using existing")
        return True

    if run_command(f'{sys.executable} -m venv venv'):
        print_success("Virtual environment created")
        return True
    return False

def get_venv_python():
```

```
    """Get path to virtual environment Python"""
    system = platform.system()
    if system == 'Windows':
        return os.path.join('venv', 'Scripts', 'python.exe')
    else:
        return os.path.join('venv', 'bin', 'python')

def get_venv_pip():
    """Get path to virtual environment pip"""
    system = platform.system()
    if system == 'Windows':
        return os.path.join('venv', 'Scripts', 'pip.exe')
    else:
        return os.path.join('venv', 'bin', 'pip')

def install_dependencies():
    """Install Python dependencies"""
    print_step(4, "Installing dependencies...")

    if not os.path.exists('requirements.txt'):
        print_error("requirements.txt not found!")
        return False

    pip = get_venv_pip()

    # Upgrade pip
    print_info("Upgrading pip...")
    run_command(f'{pip} install --upgrade pip', check=False)

    # Install dependencies
    print_info("Installing packages (this may take a few minutes)...")
    if run_command(f'{pip} install -r requirements.txt'):
        print_success("Dependencies installed")
        return True
    return False

def generate_secret_key():
    """Generate a secure random secret key"""
    return secrets.token_hex(32)

def setup_environment_file(has_postgres=False):
    """Create .env file with configuration"""
    print_step(5, "Setting up environment variables...")

    if os.path.exists('.env'):
        print_warning(".env file already exists - keeping existing configuration")
        return True

    print_info("Creating .env file with default configuration...")
```

```
    # Generate secret key
    secret_key = generate_secret_key()
    print_success(f"Generated Flask secret key")

    # Use default database configuration
    db_user = "family_hub_user"
    db_pass = secrets.token_urlsafe(16)
    db_host = "localhost"
    db_port = "5432"
    db_name = "family_hub"

    database_url = f"postgresql://{db_user}:{db_pass}@{db_host}:{db_port}/{db_name}"

    # Create .env file
    env_content = f"""# Family Hub Configuration
# Generated by automated setup script

# Flask Configuration
FLASK_SECRET_KEY={secret_key}
FLASK_ENV=production
FLASK_DEBUG=False

# Database Configuration
DATABASE_URL={database_url}

# Domain
DOMAIN=http://localhost:5000

# OpenAI Configuration (Optional)
# OPENAI_API_KEY=your-key-here
# OPENAI_BASE_URL=https://api.openai.com/v1

# Email Configuration (Optional)
# SMTP_HOST=smtp.gmail.com
# SMTP_PORT=587
# SMTP_USER=your-email@gmail.com
# SMTP_PASSWORD=your-app-password
"""

    with open('.env', 'w') as f:
        f.write(env_content)

    print_success(".env file created with default configuration")
    print_info("Edit .env file to customize settings (database, AI, email, etc.)")

    return True

def create_directories():
    """Create necessary directories"""
    print_step(6, "Creating directories...")
```

```
    directories = ['logs', 'static/uploads', 'static/profile_pics']

    for directory in directories:
        Path(directory).mkdir(parents=True, exist_ok=True)
        print_info(f"Created: {directory}")

    print_success("Directories created")
    return True

def create_placeholder_files():
    """Create placeholder files if main.py doesn't exist"""
    print_step(7, "Checking application files...")

    if not os.path.exists('main.py'):
        print_warning("main.py not found - creating placeholder")
        placeholder_main = """#!/usr/bin/env python3
from app import app

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=False)
"""
        with open('main.py', 'w') as f:
            f.write(placeholder_main)
        print_info("Created placeholder main.py - replace with your actual application")

    if not os.path.exists('app.py'):
        print_warning("app.py not found - you'll need to add your application code")

    return True

def display_next_steps(has_postgres):
    """Display next steps after installation"""
    print_header("Installation Complete!")

    print(f"{Colors.BOLD}✅ What was set up:{Colors.END}\n")
    print(f"  ✓ Python virtual environment (venv/)")
    print(f"  ✓ All dependencies installed")
    print(f"  ✓ Configuration file (.env)")
    print(f"  ✓ Application directories")

    print(f"\n{Colors.BOLD}📝 Next Steps:{Colors.END}\n")

    print(f"{Colors.CYAN}1. Activate virtual environment:{Colors.END}")
    system = platform.system()
    if system == 'Windows':
        print(f"   {Colors.YELLOW}venv\\Scripts\\activate{Colors.END}")
    else:
        print(f"   {Colors.YELLOW}source venv/bin/activate{Colors.END}")
```

```
    print(f"\n{Colors.CYAN}2. Configure your application:{Colors.END}")
    print(f"   {Colors.YELLOW}nano .env{Colors.END}")
    print(f"   Update the following if needed:")
    if not has_postgres:
        print(f"   {Colors.RED}  • DATABASE_URL (PostgreSQL connection){Colors.END}")
    print(f"     • OPENAI_API_KEY (for AI features)")
    print(f"     • SMTP settings (for email)")
    print(f"     • DOMAIN (for production)")

    if not has_postgres:
        print(f"\n{Colors.CYAN}3. Set up PostgreSQL:{Colors.END}")
        print(f"   Install PostgreSQL: https://www.postgresql.org/download/")
        print(f"   Create database:")
        print(f"   {Colors.YELLOW}createdb family_hub{Colors.END}")
        print(f"   {Colors.YELLOW}psql family_hub < schema.sql{Colors.END}")

    print(f"\n{Colors.CYAN}4. Start the application:{Colors.END}")
    print(f"   {Colors.YELLOW}python main.py{Colors.END}")
    print(f"   Or with Gunicorn:")
    print(f"   {Colors.YELLOW}gunicorn --bind=0.0.0.0:5000 --reuse-port main:app{Colors.END}")

    print(f"\n{Colors.CYAN}5. Open in browser:{Colors.END}")
    print(f"   {Colors.YELLOW}http://localhost:5000{Colors.END}")

    print(f"\n{Colors.GREEN}For help, see the docs/ folder or README.md{Colors.END}\n")

def main():
    """Main setup function"""
    print_header("Family Hub - Automated Setup")

    print(f"{Colors.BOLD}Non-Interactive Setup Mode{Colors.END}")
    print("Using default configuration for all settings\n")

    print(f"{Colors.BOLD}This script will:{Colors.END}")
    print("  • Check system requirements")
    print("  • Create virtual environment")
    print("  • Install dependencies")
    print("  • Configure environment variables (with defaults)")
    print("  • Create application directories")
    print("  • Prepare for first run\n")

    print_info("Starting automated setup...\n")

    # Track if PostgreSQL is available
    has_postgres = False

    # Run setup steps
    try:
        # Step 1: Check Python version
        if not check_python_version():
```

```
            print_error("Setup failed: Python version too old")
            sys.exit(1)

        # Step 2: Check PostgreSQL (non-blocking)
        has_postgres = check_postgresql()

        # Step 3: Create virtual environment
        if not create_virtual_environment():
            print_error("Setup failed: Could not create virtual environment")
            sys.exit(1)

        # Step 4: Install dependencies
        if not install_dependencies():
            print_error("Setup failed: Could not install dependencies")
            sys.exit(1)

        # Step 5: Setup environment file
        if not setup_environment_file(has_postgres):
            print_error("Setup failed: Could not create .env file")
            sys.exit(1)

        # Step 6: Create directories
        if not create_directories():
            print_error("Setup failed: Could not create directories")
            sys.exit(1)

        # Step 7: Check/create placeholder files
        create_placeholder_files()

        # Display next steps
        display_next_steps(has_postgres)

    except Exception as e:
        print_error(f"Unexpected error during setup: {e}")
        import traceback
        traceback.print_exc()
        sys.exit(1)

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        print_error("\n\nSetup cancelled by user")
        sys.exit(1)
    except Exception as e:
        print_error(f"\n\nUnexpected error: {e}")
        import traceback
        traceback.print_exc()
        sys.exit(1)
```