

Crow + SQLite Appointment System

Installation & Setup Guide

This guide explains how to set up and run the **Crow + SQLite Appointment System** backend locally.

1. Prerequisites

Make sure the following are installed on your system:

- **Operating System:** Windows 10/11
- **C++ Compiler:** MinGW-w64 or MSVC (C++17 required)
- **CMake:** v3.15 or higher
- **Git** (optional, for version control)

Required Libraries (included or vendored)

- **Crow** (C++ micro web framework)
- **SQLite3**

If SQLite is not already available, download `sqlite3.c` and `sqlite3.h` and place them in your project or link against SQLite.

2. Project Structure

```
crow_backend/
|
├── controllers/      # Route handlers (appointment, doctor, patient,
│   cancellation)
├── models/           # Database models
├── config/           # Configuration (n8n, constants)
├── Crow/              # Crow framework
├── httplib/           # HTTP utilities (if used)
├── build/             # CMake build directory
├── main.cpp           # Application entry point
├── CMakeLists.txt     # Build configuration
├── actual.db          # SQLite database
└── README.md
```

3. Database Setup

The project uses **SQLite**.

Tables Used

- Patient
- Doctor
- Appointment
- Doctor_Schedule
- Category

Ensure `actual.db` exists in the root directory. If missing, create it and run the SQL schema provided in the project documentation.

4. Build Instructions (Windows)

Step 1: Open Terminal

Open **Command Prompt** or **PowerShell** in the project root:

```
C:\Users\Admin\crow_backend
```

Step 2: Create Build Directory

```
mkdir build  
cd build
```

Step 3: Run CMake

```
cmake ..
```

If using MinGW explicitly:

```
cmake .. -G "MinGW Makefiles"
```

Step 4: Compile

```
cmake --build .
```

After a successful build, an executable (e.g. `server.exe`) will be generated.

5. Running the Server

From the `build` directory:

```
server.exe
```

You should see:

```
Server running at http://localhost:8080
```

6. Available Endpoints (Examples)

- POST /book_appointment
- POST /cancel_appointment
- GET /categories
- GET /doctors?category_id=ID

These endpoints are consumed by the frontend forms.

7. Configuration Notes

n8n Integration

- Webhook URL is defined in:

```
config/n8n_config.h
```

- Ensure the webhook URL is correct before running the server.
-

8. Common Issues & Fixes

Database is Locked

- Make sure the database is not open in another tool (e.g., DB Browser)
- Always save changes before closing DB tools

C++17 Errors

- Ensure compiler supports **C++17**
- Update `CMakeLists.txt`:

```
set(CMAKE_CXX_STANDARD 17)
```

Endpoint Abuse (No Auth)

- Project is intentionally **public-read / restricted-write**
- Admin routes are not exposed

9. Next Steps

- Add input validation
 - Add rate limiting or API keys
 - Build frontend deployment
 - Prepare final submission demo
-
-

README Notes

This project is built as an **academic full-stack backend system** using **Crow (C++)** and **SQLite**, focusing on clean MVC structure, real database interactions, and external workflow integration (n8n).

- No authentication is implemented by design
 - Read-only endpoints are public
 - Write operations are limited to patient-side flows
-

Status: Ready for final submission & demo

Alhamdulillah 