User guide for running the *Secure Remote Control* application

NOTE: For the instructions on running your local database, please see the accompanying document **at the end**.

We **highly recommend** reading through **the entire** document **carefully** before running the application.

For running the application locally, first start web admin with instructions under segment Group 2.

After that, start the Communication layer app with steps under Group 3.

After that is done, you can start the Android app with instructions under Group 1.

Keep in mind that if you want to start locally, both web admin and communication layer need to use the same database source (local or remote).

Also, when running projects locally, Web admin and communication layer support localhost resolving, but android app requires full ip address when connecting to communication layer.

You can find the local ip address of the machine where you start the web admin and communication layer by using the **ifconfig/ipconfig** command in your terminal.

Group 1 - Android

1. Install Android Studio Flamingo

Go to the official Android Studio website (https://developer.android.com/studio) and download the version called "Flamingo".
Follow the installation steps for your operating system (Windows, macOS, or Linux).

2. Clone the Client Side Android App Repository

Go to the following link:
https://github.com/SI-SecureRemoteControl
Open Client-Side-Android-app
Open Android Studio
Click on "Get from VCS" (Version Control System)
Paste the repository link and click "Clone"
After cloning, right-click on the project folder in the Project view and choose "Git → Pull" to make sure you have the latest code

3. Sync the Gradle

Gradle is the tool that builds your project.

When the project opens, a bar will appear at the top saying "Sync Now" – click it
If not, go to the menu and click "File → Sync Project with Gradle Files"
Wait for it to finish downloading everything it needs

4. Connect Your Phone or Build an APK

You can either connect your Android phone with a USB cable (make sure Developer Mode and USB Debugging are turned on), or
Go to "Build → Build Bundle(s) / APK(s) → Build APK(s)" to create a file you can install manually on your phone.

5. Run the Application

If your phone is connected, click the green "Play" button in Android Studio.
If you created an APK, install it on your phone and open the app manually.

6. Enter Server Address in the App

When the app opens, you'll see a field asking for a server address
Type:

ws://your_computer_ip_address:8080

For example, if your computer's IP address is **192.168.1.5**, you would enter:
ws://192.168.1.5:8080

Group 2 - Web app

1. Clone the Web App Repository

Go to the following link:
https://github.com/SI-SecureRemoteControl/web-app

Open a terminal (or Git Bash), navigate to a folder where you want the project to be and run the following commands:

```
git clone https://github.com/SI-SecureRemoteControl/web-app.git
cd [project-folder]
git pull
```

2. Install Node.js

Download Node.js from https://nodejs.org and install it if you don't have it
Make sure it includes npm (Node Package Manager)

3. Add the .env Files

There are **two** `.env` files to create:

- A backend `.env` file (inside the `backend` folder)
- A frontend-specific `.env` file (inside the `frontend` folder)

Do the following:

For the **backend** .env, create a new .env file with no name, inside project (web-app folder).

In the file, paste the following:

PORT=9000
DB_URI='mongodb+srv://root:root@cluster.qciyr2x.mongodb.net/Cluster?retry Writes=true&w=majo rity&appName=Cluster'
DB_URI_LOCAL='mongodb://localhost:27017/'
USE_LOCAL_DB=true
SECRET_KEY='10429b28ab5cb5042e393b2f20d76bb1'

A secret key can be **anything** you want, this is just an **example**.

Do not be confused with two database URIs, the local URI is used by developers when testing and adding new features. If you want to use a local database you can run docker compose which will pull Mongo images and start up a Mongo instance in your local docker container.

Brief explanations for each field in the .env file for backend:

1. PORT: Specifies the port on which the server should listen.
2. DB_URI: This is the connection string for a cloud-hosted database, such as MongoDB.
3. DB_URI_LOCAL: Connection string for a **locally hosted NoSQL database instance** (e.g. MongoDB), such as one running in Docker or directly installed.
4. USE_LOCAL_DB: A flag that determines **which database URI** to use — the **local** one or the **cloud** one.
5. SECRET_KEY: Used to sign and verify JWT tokens for authentication

For the **frontend** .env, create a new .env file with no name, inside frontend folder of the repository

In the file, paste the following:

VITE_BASE_URL=http://localhost:9000
VITE_WS_URL=ws://localhost:9000
VITE_API_UPLOAD_URL=http://localhost:8080/api/upload
VITE_COMM_LAYER_API_URL=http://localhost:8080

Brief explanations for each field in .env:

1. **VITE_BASE_URL**: base URL for the backend API or server during local development
2. **VITE_WS_URL**: WebSocket URL of backend for real-time communication.
3. **VITE_API_UPLOAD_URL**: This is the full URL to the upload API endpoint, which points to the local instance of the communication layer.
4. **VITE_COMM_LAYER_API_URL**: This is the full URL to the communication layer

6. Start the Backend

Open a terminal or command prompt
Navigate to the backend folder (usually the root folder of the project)

Run the following commands:

npm install
npm start

7. Start the Frontend

Open a new terminal or command prompt
Navigate to the `frontend` folder inside the project

Run the following commands:

npm install
npm run dev

8. Access the App

Once both backend and frontend are running, open your browser and go to the address shown in the terminal

Group 3 - Communication Layer

The Secure Remote Control Gateway is a Node.js-based server that acts as a communication layer between IT admins and Android devices. It facilitates secure WebSocket-based real-time interactions, including device registration, WebRTC signaling, and remote control commands.

1. ***Ensure you have node.js installed on your machine:***

   node -v
   npm -v

2. ***Clone the secure-remote-control-gateway repository:***

   git clone
https://github.com/SI-SecureRemoteControl/secure-remote-control-gateway.git

   cd secure-remote-control-gateway
   git pull

3. ***Install dependencies:***

   npm install

4. *Create a .env file and add the following fields:*

```
PORT=8080
DB_URI='mongodb+srv://root:root@cluster.qciyr2x.mongodb.net/Cluster?retryWrites=true&w=majority&appName=Cluster'
# Lokalni MongoDB (Docker)
DB_URI_LOCAL='mongodb://localhost:27017/'
USE_LOCAL_DB=false
JWT_SECRET='23181ac6f726f8d199b4c6732de22cc8b1869102118b74eab4dd8fbb7d18fc492fee2016f3f483ce369a3c0bfa9228daa8d0926865d8505d2607de6253930596'
WEBSOCKET_URL='ws://localhost:9000/ws/control/comm'
SERVICE_URL='http://YOUR_IP_ADRESS:8080'
```

Field explanations:

1. **PORT=8080**
   Specifies the port number the server or application will listen on (8080 is a common default for web servers).

2. **DB_URI**
   Connection string for a **remote MongoDB Atlas** database. It includes credentials and cluster information.

3. **DB_URI_LOCAL**
   Connection string for a **local MongoDB instance** running on your machine.

4. **USE_LOCAL_DB=false**
   Boolean flag to switch between the remote DB (false) or local DB (true).

5. **JWT_SECRET**
   A long secret key used to **sign and verify JSON Web Tokens (JWTs)** for authentication and security.

6. **WEBSOCKET_URL**
   WebSocket server URL used for **real-time bidirectional communication** between clients and the server.

7. **SERVICE_URL**
   URL of a **remote service endpoint**

5. ***Start the server:***

npm run start

Running this will trigger database migrations in migrate.js file in the Communication layer project.

Those migrations **are not necessary** in the project, rather they are used for cleaner database handling and changes history in the database scheme.

Secure Remote Control:
Local database setup

For this project, MongoDB is used as the primary database, but any other NoSQL database can be used as well.

The database is deployed on MongoDB Atlas, while for local development and testing, a Docker image is used to run the database inside a container.

To manage the database structure, the tool "mongo-migrate" is used, which allows version control and migration management.

MongoDB was chosen due to its flexibility in handling unstructured data, and because it allows multiple services to access the database without conflicts.

Since the system uses two independent services that communicate with the database, it was important to avoid complications related to concurrent access and simultaneous migrations.

MongoDB with the mongo-migrate tool enables controlled migrations, preventing issues in multi-environment access.

INSTRUCTIONS

First, make sure Docker is installed on your computer:

https://www.docker.com/products/docker-desktop

You also need to install the Mongo shell. To install it, do the following:

For Windows:
Go to the following link:
https://www.mongodb.com/try/download/shell

Choose the **msi** option under packages
After installation, add **mongosh** to your PATH

For macOS:
Open the terminal and run the following command:
***brew install mongodb/brew/*mongosh**

For Linux:

Open the terminal and follow the official MongoDB documentation for your distribution.
For Ubuntu and Debian-based systems, you can run the following commands:

curl -fsSL https://pgp.mongodb.com/server-6.0.asc | sudo gpg -o /usr/share/keyrings/mongodb-server-6.0.gpg --dearmor

echo "deb [signed-by=/usr/share/keyrings/mongodb-server-6.0.gpg] https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/6.0 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-6.0.list

sudo apt update
sudo apt install -y mongodb-mongosh

This will install the MongoDB shell (`mongosh`).
After installation, you can run **mongosh** in the terminal to enter the Mongo shell.

Setup:

Clone the repository of the **communication layer**

Open a terminal (or Git Bash), navigate to a folder where you want the project to be and run the following commands:

> **git clone https://github.com/SI-SecureRemoteControl/remote-control-gateway**
> **cd [project-folder]**
> **git pull**

After cloning or pulling the repository and accessing the project code, you need the .env file we created:

PORT=8080
DB_URI='mongodb+srv://root:root@cluster.qciyr2x.mongodb.net/Cluster?retryWrites=true&w=majority&appName=Cluster'
DB_URI_LOCAL='mongodb://localhost:27017/'
USE_LOCAL_DB=true

These are the two URIs used to connect to the deployed and local databases, respectively.

Make sure that when **testing**, you are using your local database by setting **USE_LOCAL_DB=true**, so that you avoid **interfering** with the production database.

In the terminal, run the following command to install all required dependencies:

npm install

Then, to start the application:

npm start

In the project folder, there is a file called **docker-compose.yml**

Run the following command:

docker-compose up -d

This will start the Docker container.

Check if the container is running by typing the following:

docker ps

You should see a list of containers.

Look for the one named **local-mongo**

Then run the following command

docker exec -it local-mongo mongosh

This will open the interactive Mongo shell inside the locally running MongoDB container

Once inside, you will see the prompt
test>

From here, you can do all sorts of manipulations in the database - inserting, querying, updating, and more. For full reference, visit the **official MongoDB documentation**.

Some useful commands include:

To list all databases,
show dbs

To enter a database,
use <database_name> (our is SecureRemoteControl)

To view all collections within the selected database,
show collections