Screen Recording Report API – Technical Documentation

GitHub Repository: https://github.com/monish8978/screenrecoding_report.git

Version: 1.1

Author: Mohammad Monish

Technology Stack: Python, FastAPI, MongoDB, Uvicorn

---

1. Project Overview

The Screen Recording Report API is a RESTful API built with FastAPI for managing user and client screen recording reports. It allows inserting, fetching, validating, and updating reports. MongoDB is used as the primary database with separate collections for user and client data.

---

2. Repository Structure

```
screenrecoding_report/
│
├── app.py               # Main FastAPI application
├── logger.py            # Logging configuration
├── service_check.py     # Service monitoring & management
├── settings.py          # Configuration (MongoDB, logging, service)
└── README.md            # Project documentation
```

---

3. Configuration (settings.py)

MongoDB

- URI: mongodb+srv://mongodb:mongodb@cluster0.1nfoz.mongodb.net/
- Database: screenrecoding

- Collections:
    - user_report → Stores user reports
    - client_report → Stores client reports & validation

FastAPI Server

- Port: 9006
- Accessible at http://:9006/

Logging

- Logs stored at /var/log/czentrix/screenrecoding_report.log
- Includes API requests, responses, database status, and errors

Systemd Service

- Service File: screenrecoding-report
- Manage via `systemctl` commands

---

4. Logging (logger.py)

- Python logging module with TimedRotatingFileHandler
- Rotates daily with 7 days backup
- Console and file logging

---

5. Service Management (service_check.py)

- Checks if service is active using `systemctl is-active`
- Restarts service if inactive
- Multithreading via ThreadPoolExecutor
- Garbage collection after service management

---

6. API Endpoints (app.py)

6.1 User Report Endpoints

POST /user_report

- Insert a new user report
- Request Body: JSON
- Response:

```
{
  "status": 200,
  "message": "User report inserted successfully",
  "data": {"inserted_id": "650fae12f123456789abcdef"}
}
```

GET /user_report

- Fetch all user reports
- Response:

```
{
  "status": 200,
  "message": "Found 12 user reports",
  "data": [
    {"clientId": 101, "macAddress": "00:1B:44:11:3A:B7", "isValid": true},
    ...
```

```
  ]
}
```

6.2 Client Report Endpoints

POST /client_report

- Insert a new client report

GET /client_report

- Fetch all client reports

GET /check_report_exists

- Check if a report exists
- Query Parameters: clientId, macAddress
- Response:

```
{
  "status": 200,
  "message": "Record found in user collection",
  "data": {"exists": true, "collection": "user_collection", "isValid": true}
}
```

PUT /client_report/update_validity

- Update 'isValid' field
- Request Body:

```
{
  "clientId": 101,
  "macAddress": "00:1B:44:11:3A:B7",
  "isValid": true
}
```

- Response:


```
{
  "status": 200,
  "message": "Client report updated successfully",
  "data": {"updated_count": 1}
}
```


---

7. Helper Functions

create_response(status, message, data)

- Standardizes all API responses
- Returns JSON object with status, message, and data

---

8. Database Connection

- MongoDB initialized at app startup
- Timeout: 5000 ms
- Exceptions logged

---

9. Running the API

Local:

```
python app.py
uvicorn app:app --host 0.0.0.0 --port 9006
```

Systemd:

```
systemctl status screenrecoding-report
```

systemctl start screenrecoding-report
systemctl restart screenrecoding-report

---

10. Key Features

- RESTful API with FastAPI
- MongoDB for scalable storage
- Separate collections for user & client data
- Daily log rotation
- Service auto-monitor & restart
- Standardized API responses
- Full CRUD support for client reports

---

11. Future Enhancements

- Pagination for large datasets
- Authentication & Authorization
- Filtering & search on reports
- Analytics/dashboard integration
