Credential-Grafana

A comprehensive system for managing Grafana credentials, automated dashboard creation, and Slack notifications. This application provides a centralized platform for managing multiple Grafana instances, user credentials, and automated dashboard creation with Slack integration.

System Architecture

Core Components

1. Frontend (React.js)
2. Backend (Node.js/Express)
3. PostgreSQL Database
4. Grafana Server
5. Slack Integration

Credential Manager Flow

```
graph TD
    Frontend --> Backend
    Backend --> UserMgmt[User Management]
    Backend --> CredMgmt[Credential Management]
    Backend --> DBCreation[Database Creation]

    DBCreation --> TimingDB[-timing_report]
    DBCreation --> QORDB[-QOR]
    DBCreation --> DRCDB[-DRC]

    CredMgmt --> |add| GrafanaKey[Add Grafana API Key]
    GrafanaKey --> AddDS[Add Database to Datasource in Grafana]
    AddDS --> StoreUID[Datasource UID to Credentials]
    StoreUID --> CreateDash[Creating Grafana Dashboard]
    CreateDash --> |add| SaveDB[Saving Grafana Dashboard Link in DB]
    SaveDB --> DashboardDB[DashboardTiming DB]
    DashboardDB --> LinkRetrieval[Dashboard Link Retrieval]

    LinkRetrieval --> Screenshot[Screenshot and Store]
    Screenshot --> SlackPost[Post to Slack]

    subgraph SlackConfig
        SlackToken[Slack Bot Token]
        SlackChannel[Slack Channel ID]
    end

    SlackConfig --> SlackPost
```

Required Credentials

- DB_host
- DB_port
- DB_user
- DB_pass
- DB_name_timing_report
- DB_name_QOR
- DB_name_drc
- Grafana URL
- Grafana_uid_qor
- grafana_uid_timing
- Grafana_timing
- Grafana_api_key
- slack_bot_token
- Slack_channel_id

Setup Instructions

1. PostgreSQL Setup

```
# Install PostgreSQL
sudo apt-get update
sudo apt-get install postgresql postgresql-contrib

# Start PostgreSQL service
sudo systemctl start postgresql
sudo systemctl enable postgresql

# Create database and user
sudo -u postgres psql
postgres=# CREATE DATABASE master_db;
postgres=# CREATE USER your_user WITH PASSWORD 'your_password';
postgres=# GRANT ALL PRIVILEGES ON DATABASE master_db TO your_user;
```

2. Grafana Setup

1. **Install Grafana**:

   ```
   sudo apt-get install -y software-properties-common
   sudo add-apt-repository "deb https://packages.grafana.com/oss/deb stable
   main"
   wget -q -O - https://packages.grafana.com/gpg.key | sudo apt-key add -
   sudo apt-get update
   sudo apt-get install grafana
   ```

2. **Start Grafana**:

```
sudo systemctl start grafana-server
sudo systemctl enable grafana-server
```

3. **Create Grafana API Key**:
   - Log in to Grafana (default: http://localhost:3000, admin/admin)
   - Go to Configuration → API Keys
   - Click "Add API key"
   - Name: "CredentialManager"
   - Role: "Admin"
   - Click "Add"
   - **IMPORTANT**: Copy the API key immediately - it won't be shown again

3. Slack Setup

1. **Create Slack App**:
   - Go to Slack API
   - Click "Create New App"
   - Choose "From scratch"
   - Name your app and select workspace

2. **Configure Bot Token**:
   - Go to "OAuth & Permissions"
   - Add these bot token scopes:
     - `chat:write`
     - `files:write`
     - `channels:read`
     - `channels:join`
   - Click "Install to Workspace"
   - Copy the "Bot User OAuth Token" (starts with `xoxb-`)

3. **Create Slack Channel**:
   - Create a new channel in your Slack workspace
   - Invite your bot to the channel: `/invite @YourBotName`
   - Get Channel ID:
     - Right-click the channel
     - Copy link
     - The ID is the last part of the URL

4. Application Setup

1. **Clone Repository**:

```
git clone [repository-url]
cd credential-grafana
```

2. **Install Dependencies**:

```
# Backend
cd server_side
npm install

# Frontend
cd frontend
npm install
```

3. **Configure Environment**:

   Create `server_side/.env`:

```
# Server Configuration
PORT=8050
HOST=0.0.0.0
FRONTEND_URL=http://localhost:8051

# Database Configuration
MASTER_DB_HOST=localhost
MASTER_DB_PORT=5432
MASTER_DB_USER=your_user
MASTER_DB_PASS=your_password
MASTER_DB_NAME=master_db

# Grafana Configuration
GRAFANA_BASE_URL=http://localhost:3000
GRAFANA_API_KEY=your_grafana_api_key

# Slack Configuration
SLACK_BOT_TOKEN=xoxb-your-bot-token
SLACK_CHANNEL_ID=your-channel-id

# GIF Configuration
GIF_CAPTURE_DIR=captureGrafanaGIF
CAPTURE_WIDTH=1920
CAPTURE_HEIGHT=1080
CAPTURE_FRAMES=10
CAPTURE_FRAME_DELAY=500
CAPTURE_TIMEOUT=120000
```

   Create `frontend/.env`:

```
PORT=8051
REACT_APP_API_URL=http://localhost:8050
```

4. **Start Services**:

```
# Start Backend
cd server_side
npm start

# Start Frontend (in new terminal)
cd frontend
npm start
```

5. **Access Application**:

   - Frontend: http://localhost:8051
   - Backend API: http://localhost:8050

Usage

1. **Create User**:

   - Access frontend interface
   - Click "Add User"
   - Enter username
   - System will automatically:
     - Create user databases
     - Set up Grafana data sources
     - Generate API keys

2. **Add Credentials**:

   - Select user from list
   - Click "Add Credential"
   - Add required credentials (DB, Grafana, Slack)

3. **Monitor Dashboards**:

   - System automatically scans for new tables
   - Creates Grafana dashboards
   - Captures screenshots
   - Sends notifications to Slack

Complete Application Flow

1. User Management Flow

```
graph TD
    A[User Registration] --> B[Create User Record]
    B --> C[Generate API Key]
    C --> D[Set Default Credentials]
    D --> E[Create User Databases]
    E --> F[Setup Grafana Data Sources]
```

2. Dashboard Creation Pipeline

```
graph TD
    A[Periodic Scan] --> B[Check New Tables]
    B --> C[Create Dashboard]
    C --> D[Capture GIF]
    D --> E[Store in Database]
    E --> F[Send to Slack]

    subgraph "Periodic Scan"
        A --> G[Get Active Users]
        G --> H[Process Each User]
        H --> I[Check Each Database Type]
    end

    subgraph "Dashboard Creation"
        C --> J[Load Template]
        J --> K[Replace Placeholders]
        K --> L[Create in Grafana]
    end

    subgraph "GIF Capture"
        D --> M[Launch Puppeteer]
        M --> N[Navigate to Dashboard]
        N --> O[Capture Frames]
        O --> P[Create GIF]
    end
```

3. Database Structure

```
graph TD
    A[users] --> B[credentials]
    A --> C[dashboardtiming]
    A --> D[dashboardqor]
    A --> E[dashboarddrc]
```

Detailed Process Flow

1. User Creation and Setup

1. User registration through API
2. Creation of user record in master_db
3. Generation of API keys and default credentials
4. Creation of user-specific databases:

    - `{username}-timing_report`
    - `{username}-QOR`
    - `{username}-drc`
5. Setup of Grafana data sources

2. Dashboard Creation Process

1. **Periodic Scan** (Every 60 seconds):

```
const dashboardScanInterval = parseInt(process.env.DASHBOARD_SCAN_INTERVAL);
setInterval(async () => {
  const userIds = await getAllUserIds();
  for (const uid of userIds) {
    await processSingleUser(uid);
  }
}, dashboardScanInterval);
```

2. **Database Processing**:
    - Scan for new tables in user databases
    - Check table schema compatibility
    - Determine dashboard type (TIMING/QOR/DRC)
3. **Dashboard Creation**:
    - Load appropriate template based on type
    - Replace placeholders with actual values
    - Create dashboard via Grafana API
    - Store dashboard URL in database
4. **GIF Capture**:
    - Launch headless browser
    - Navigate to dashboard URL
    - Capture multiple frames
    - Create animated GIF
    - Store GIF path in database
5. **Slack Notification**:
    - Check for unprocessed dashboards
    - Upload GIF to Slack
    - Send formatted message with:
        - Dashboard link
        - Preview GIF
        - Creator information
    - Mark dashboard as processed

Database Schema

Master Database Tables

1. **users**

   ```
   CREATE TABLE public.users (
       id integer PRIMARY KEY,
       username varchar(255) UNIQUE NOT NULL,
       api_key_hash text,
       api_key_plain text,
       created_at timestamp DEFAULT now(),
       is_active boolean DEFAULT true
   );
   ```

2. **credentials**

   ```
   CREATE TABLE public.credentials (
       id integer PRIMARY KEY,
       user_id integer REFERENCES users(id),
       key_name varchar(255),
       key_value text,
       username varchar(255)
   );
   ```

3. **Dashboard Tables**

   ```
   CREATE TABLE public.dashboardtiming/qor/drc (
       id integer PRIMARY KEY,
       user_id integer,
       username text,
       table_name text,
       dashboard_url text,
       local_snapshot_url text,
       slack_sent_at timestamp with time zone
   );
   ```

API Endpoints

User Management

- `POST /api/v1/users` - Create new user
- `GET /api/v1/users` - List all users
- `GET /api/v1/users/:id` - Get user details
- `DELETE /api/v1/users/:id` - Delete user

Credential Management

- `GET /api/v1/users/:userId/credentials` - List credentials
- `POST /api/v1/users/:userId/credentials` - Add credential
- `PUT /api/v1/users/:userId/credentials/:keyName` - Update credential
- `DELETE /api/v1/users/:userId/credentials/:keyName` - Delete credential

Environment Configuration

Backend (.env)

```
# Server Configuration
PORT=8050
HOST=0.0.0.0
FRONTEND_URL=http://localhost:8051

# Database Configuration
MASTER_DB_HOST=localhost
MASTER_DB_PORT=5432
MASTER_DB_USER=postgres
MASTER_DB_PASS=your_password
MASTER_DB_NAME=master_db

# Grafana Configuration
GRAFANA_BASE_URL=http://localhost:3000
GRAFANA_USER=admin
GRAFANA_PASSWORD=your_password

# GIF Configuration
GIF_CAPTURE_DIR=captureGrafanaGIF
CAPTURE_WIDTH=1920
CAPTURE_HEIGHT=1080
CAPTURE_FRAMES=10
CAPTURE_FRAME_DELAY=500
CAPTURE_TIMEOUT=120000
```

Frontend (.env)

```
PORT=8051
REACT_APP_API_URL=http://localhost:8050
```

Installation

1. Clone repository

2. Install dependencies:

   ```
   # Backend
   cd server_side
   ```

```
npm install

# Frontend
cd frontend
npm install
```

3. Configure environment variables

4. Initialize database:

```
psql -U postgres < server_side/db/schema.sql
```

5. Start services:

```
# Backend
cd server_side
npm start

# Frontend
cd frontend
npm start
```

Error Handling

1. **Database Errors**
   - Connection pool management
   - Transaction rollback
   - Automatic retry mechanism
2. **Grafana Integration**
   - API error handling
   - Dashboard creation retry
   - Template validation
3. **Slack Integration**
   - File upload retry
   - Message formatting fallback
   - Error logging and monitoring

Monitoring

1. **Application Logs**

```
tail -f logs/app.log
```

2. **Database Monitoring**

```sql
SELECT * FROM pg_stat_activity;
```

3. **Process Monitoring**

```
pm2 status
pm2 logs
```

License

MIT

Contributing

Please read CONTRIBUTING.md for details on our code of conduct and the process for submitting pull requests.

Documentation

For detailed documentation about the system, including:

- System architecture
- Issues and solutions
- Setup instructions
- API endpoints
- Troubleshooting guide

Please see DOCUMENTATION.md

Quick Start

1. Clone the repository
2. Install dependencies:

```
npm install
```

3. Set up environment variables (see .env.example)

4. Initialize the database:

```
node server_side/db/init-db.js
```

5. Start the server:

```
node server_side/index.js
```

License

MIT