Credential Management System Documentation

Table of Contents

System Overview

The Credential Management System is a comprehensive solution that integrates:

- User and credential management
- Automated Grafana dashboard creation
- Database management for timing reports, QOR, and DRC data
- Slack notifications with dashboard previews
- GIF capture and sharing functionality

Architecture

Core Components

1. **Frontend (React.js)**
    - User management interface
    - Credential management
    - Dashboard monitoring
    - Real-time updates
2. **Backend (Node.js/Express)**
    - RESTful API endpoints
    - Service orchestration
    - Database management
    - Background processes
3. **PostgreSQL Databases**
    - Master database for user management
    - User-specific databases for:
        - Timing reports

        - QOR data
        - DRC data

4. **Grafana Integration**

    - Automated dashboard creation
    - Data source management
    - Template-based visualization
    - API key handling

5. **Slack Integration**

    - Real-time notifications
    - Dashboard preview sharing
    - GIF capture and upload
    - Interactive messages

System Flow

1. User Management Flow

```
// User creation and setup process
async function createUser(username) {
  // 1. Create user record
  const user = await createUserRecord(username);

  // 2. Generate API keys
  const apiKey = await generateApiKey();

  // 3. Create user databases
  await createUserDatabases(username);

  // 4. Setup Grafana datasources
  await setupGrafanaDatasources(username);

  // 5. Set default credentials
  await setDefaultCredentials(user.id);

  return user;
}
```

2. Dashboard Creation Pipeline

```
// Dashboard creation and notification process
async function processDashboards() {
  // 1. Scan for new tables
  const tables = await scanNewTables();

  // 2. Create dashboards
```

```
    const dashboards = await createDashboards(tables);

    // 3. Capture previews
    await captureGIFs(dashboards);

    // 4. Send notifications
    await sendSlackNotifications(dashboards);
  }
```

Database Schema

1. Users Table

```sql
CREATE TABLE public.users (
    id SERIAL PRIMARY KEY,
    username VARCHAR(255) UNIQUE NOT NULL,
    api_key_hash TEXT,
    api_key_plain TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    is_active BOOLEAN DEFAULT true
);
```

2. Credentials Table

```sql
CREATE TABLE public.credentials (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
    username VARCHAR(255),
    key_name VARCHAR(255) NOT NULL,
    key_value TEXT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(user_id, key_name)
);
```

3. Dashboard Tables

```sql
CREATE TABLE public.dashboardtiming (
    id SERIAL PRIMARY KEY,
    user_id INTEGER,
    username TEXT,
    table_name TEXT,
    dashboard_url TEXT,
    local_snapshot_url TEXT,
    source TEXT DEFAULT 'grafana',
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    app_user_id INTEGER,
    app_username VARCHAR(255),
    slack_sent_at TIMESTAMP WITH TIME ZONE,
    FOREIGN KEY (app_user_id) REFERENCES users(id)
);
```

Services

1. Slack Service

The Slack service handles all communication with Slack, including:

- Dashboard notifications
- GIF uploads
- Message formatting
- Error handling

Key features:

```
class SlackService {
  async processUnsentDashboards() {
    try {
      // Get users with Slack credentials
      const users = await this.getUsersWithSlackCredentials();

      // Process each user's dashboards
      for (const user of users) {
        await this.processUserDashboards(user);
      }
    } catch (error) {
      console.error('[SlackService] Error processing unsent dashboards:', error);
    }
  }

  async processUserDashboards(user) {
    try {
      // Process each dashboard type
      await this.processDashboardType('timing', user);
      await this.processDashboardType('qor', user);
      await this.processDashboardType('drc', user);
    } catch (error) {
      console.error(`[SlackService] Error processing dashboards for user
${user.username}:`, error);
    }
  }

  async uploadGifToSlack(botToken, filePath, channelId) {
    // Multi-step upload process
    // 1. Get upload URL from Slack
    const step1 = await axios.post(
      "https://slack.com/api/files.getUploadURLExternal",
      qs.stringify({
        filename: "dashboard.gif",
        length: stats.size
      }),
      {
        headers: {
          Authorization: `Bearer ${botToken}`,
          "Content-Type": "application/x-www-form-urlencoded"
        }
      }
    );

    // 2. Upload file to URL
    const form = new FormData();
    form.append("file", fs.createReadStream(normalizedPath));

    // 3. Complete upload
    const step3 = await axios.post(
      "https://slack.com/api/files.completeUploadExternal",
      {
        files: [{ id: file_id, title: "Dashboard GIF" }],
        channel_id: channelId
      }
    );
  }

  async postDashboardLink(botToken, channelId, dashboardUrl, tableName,
creatorName) {
    const message = {
      channel: channelId,
      blocks: [
        {
          type: "header",
          text: {
            type: "plain_text",
            text: `New Dashboard Created by ${creatorName}`
          }
        },
        {
          type: "section",
          text: {
            type: "mrkdwn",
            text: `*Dashboard Link:*\n${dashboardUrl}`
          }
        }
      ]
    };
    return await axios.post('https://slack.com/api/chat.postMessage', message, {
      headers: {
        'Authorization': `Bearer ${botToken}`,
        'Content-Type': 'application/json'
      }
    });
  }
}
```

2. Grafana Service

Handles all Grafana-related operations:

- Data source creation
- Dashboard management
- API key validation
- Template management

3. Database Service

Manages database operations:

- Connection pooling
- Transaction management
- Error handling
- Database creation/deletion

Error Handling

1. Database Errors

```javascript
// Connection pool management
const pool = new Pool({
  connectionTimeoutMillis: 5000,
  idleTimeoutMillis: 30000,
  max: 20
});

pool.on('error', (err) => {
  console.error('Unexpected error on idle client:', err);
});

// Automatic refresh mechanism
setInterval(refreshPool, refreshInterval);
```

2. API Error Handling

```javascript
// Global error handler
app.use((err, req, res, next) => {
  console.error('[ERROR]', err);
  res.status(500).json({
    error: 'Internal server error',
    message: err.message
  });
});
```

Environment Setup

Required Environment Variables

```
# 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_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
```

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
```

Troubleshooting Guide

Common Issues and Solutions

1. **Database Connection Issues**

```
# Check PostgreSQL status
systemctl status postgresql

# Verify connection
psql -h $DB_HOST -p $DB_PORT -U $DB_USER -d $DB_NAME

# Check active connections
SELECT * FROM pg_stat_activity;
```

2. **Grafana Integration Issues**

```
# Verify Grafana API access
curl -H "Authorization: Bearer $GRAFANA_API_KEY" $GRAFANA_URL/api/health

# Check data sources
curl -H "Authorization: Bearer $GRAFANA_API_KEY" $GRAFANA_URL/api/datasources
```

3. **Slack Integration Issues**

```
# Test bot token
curl -H "Authorization: Bearer $SLACK_BOT_TOKEN" https://slack.com/api/auth.test

# Check channel access
curl -H "Authorization: Bearer $SLACK_BOT_TOKEN"
https://slack.com/api/conversations.info?channel=$CHANNEL_ID
```

Monitoring

1. **Application Logs**

```
# View real-time logs
tail -f logs/app.log

# Check error logs
grep ERROR logs/app.log
```

2. **Performance Monitoring**

```
# Check system resources
top
df -h
free -m

# Monitor Node.js process
pm2 monit
```