API Model & Design - Telecom Mission Control SaaS

This document outlines the RESTful API model and design strategies used in the Telecom Mission Control SaaS platform, optimized for performance, scalability, and security.

# 1. Authentication APIs

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Endpoint | Description | Request/Response |
| POST | /api/auth/login | Authenticate user and return access/refresh tokens | Request: { username, password } Response: { tokens: { accessToken, refreshToken }, username, role } |
| POST | /api/auth/refresh | Refresh expired access token | Request: { refreshToken } Response: { accessToken } |

# 2. User Management APIs

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Endpoint | Description | Request/Response |
| GET | /api/users | Fetch all users (admin only) | Response: [ { username, role } ] |
| POST | /api/users | Create a new user | Request: { username, password, role } Response: { username, role } |

# 3. Tower APIs

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Endpoint | Description | Request/Response |
| GET | /api/towers | Fetch all towers | Response: [ { location, supportedCarriers, supportedDevices } ] |
| POST | /api/towers | Add a new tower | Request: { location, supportedCarriers, supportedDevices } |

# 4. Device APIs

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Endpoint | Description | Request/Response |
| GET | /api/devices | Get all devices | Response: [ { deviceId, os, carrier, userId, towerId } ] |
| POST | /api/devices | Add a new device | Request: { deviceId, os, carrier, userId, towerId } |

# 5. Policy APIs

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Endpoint | Description | Request/Response |
| GET | /api/policies | Get all policies | Response: [ { appName, role, allowedActions, deniedActions } ] |
| POST | /api/policies | Create new policy | Request: { appName, role, allowedActions, deniedActions } |
| PUT | /api/policies/:id | Update a policy | Request: same as POST |
| DELETE | /api/policies/:id | Delete a policy | No request body |

# Performance & Design Notes

- All APIs use async/await and optimized MongoDB queries (e.g., indexed fields)  
- Backend uses pagination (optional with limit/skip for large datasets)  
- Data validation is performed via express-validator or Mongoose schema checks  
- JWT tokens are used for stateless authentication  
- Backend supports CORS and secure headers (helmet)  
- Errors are returned with consistent structure: { message, statusCode }  
- Reusable response caching is supported in frontend for listing APIs