Authentication & Authorization Documentation

Supervision App Backend

Table of Contents

Overview

The Supervision App uses a **JWT-based authentication system** with **role-based authorization**. It supports two user roles: `admin` and `user` (field doctors), with different permission levels for each role.

Key Features:

- JWT Access & Refresh Tokens
- Role-based Access Control (RBAC)
- Password Hashing with bcryptjs
- Rate Limiting for Security
- Input Validation
- Secure Session Management
- Soft Delete for Users

Authentication Flow

1. User Registration Flow

1. Client sends POST request to `/api/auth/register` with user data
2. API validates input using express-validator
3. API hashes password using bcryptjs
4. API inserts user into database
5. API generates JWT access and refresh tokens
6. API returns tokens and user information to client

2. User Login Flow

1. Client sends POST request to `/api/auth/login` with credentials
2. API finds user by username or email in database
3. API verifies password using bcryptjs compare
4. API generates JWT access and refresh tokens
5. API updates user's last login timestamp
6. API returns tokens and user information to client

3. Token Refresh Flow

1. Client sends POST request to `/api/auth/refresh` with refresh token
2. API verifies refresh token signature and expiration
3. API validates user still exists and is active in database
4. API generates new access and refresh tokens
5. API returns new tokens to client

Authorization System

Role-Based Access Control (RBAC)

Role	Permissions	Description
admin	Full CRUD on all data, User management, System administration	System administrators and supervisors
user	CRUD on own forms (before sync), Read-only after sync	Field doctors and medical officers

Permission Matrix

Endpoint	Admin	User	Guest
`GET /health`	Yes	Yes	Yes
`POST /api/auth/login`	Yes	Yes	Yes
`POST /api/auth/register`	Yes	No	No
`GET /api/users`	Yes	No	No
`POST /api/users`	Yes	No	No
`PUT /api/users/:id`	Yes	No	No
`DELETE /api/users/:id`	Yes	No	No
`GET /api/auth/profile`	Yes	Yes	No
`PUT /api/auth/change-password`	Yes	Yes	No

API Endpoints

Authentication Endpoints

1. User Registration

```http
POST /api/auth/register
Content-Type: application/json

{
  "username": "string (3-50 chars, alphanumeric + underscore)",
  "email": "string (valid email)",
  "password": "string (min 8 chars, 1 uppercase, 1 lowercase, 1 number)",
  "fullName": "string (2-100 chars)",
  "role": "string (optional: 'admin' | 'user', default: 'user')"
}
```

Response (201):

```json
```

```
{
  "message": "User registered successfully",
  "user": {
    "id": 1,
    "username": "john_doe",
    "email": "john@example.com",
    "fullName": "John Doe",
    "role": "user",
    "createdAt": "2025-01-03T10:00:00.000Z"
  },
  "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

2. User Login

http

```
POST /api/auth/login
Content-Type: application/json

{
  "username": "string (username or email)",
  "password": "string"
}
```

Response (200):

json

```
{
  "message": "Login successful",
  "user": {
    "id": 1,
    "username": "admin",
    "email": "admin@supervision-app.com",
    "fullName": "System Administrator",
    "role": "admin"
  },
  "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

3. Refresh Token

```http
POST /api/auth/refresh
Content-Type: application/json

{
  "refreshToken": "string"
}
```

Response (200):

```json
{
  "message": "Token refreshed successfully",
  "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

4. Get User Profile

```http
GET /api/auth/profile
Authorization: Bearer <access_token>
```

Response (200):

```json
{
  "user": {
    "id": 1,
    "username": "admin",
    "email": "admin@supervision-app.com",
    "fullName": "System Administrator",
    "role": "admin",
    "createdAt": "2025-01-03T10:00:00.000Z",
    "updatedAt": "2025-01-03T10:00:00.000Z"
  }
}
```

5. Change Password

```http
PUT /api/auth/change-password
Authorization: Bearer <access_token>
Content-Type: application/json

{
  "currentPassword": "string",
  "newPassword": "string (min 8 chars, 1 uppercase, 1 lowercase, 1 number)",
  "confirmPassword": "string (must match newPassword)"
}
```

Response (200):

```json
{
  "message": "Password changed successfully"
}
```

6. Logout

```http
POST /api/auth/logout
Authorization: Bearer <access_token>
```

Response (200):

```json
{
  "message": "Logout successful"
}
```

7. Verify Token

```http
POST /api/auth/verify
Authorization: Bearer <access_token>
```

Response (200):

```json
{
  "message": "Token is valid",
  "user": {
    "id": 1,
    "username": "admin",
    "email": "admin@supervision-app.com",
    "fullName": "System Administrator",
    "role": "admin"
  }
}
```

User Management Endpoints (Admin Only)

1. Get All Users

```http
GET /api/users?page=1&limit=10&search=john&role=user&isActive=true
Authorization: Bearer <admin_access_token>
```

Query Parameters:

- `page`: Page number (default: 1)
- `limit`: Items per page (default: 10)
- `search`: Search in username, email, fullName
- `role`: Filter by role ('admin' | 'user')
- `isActive`: Filter by active status (true | false)

Response (200):

```json
```

```
{
  "users": [
    {
      "id": 2,
      "username": "dr_smith",
      "email": "smith@hospital.com",
      "fullName": "Dr. John Smith",
      "role": "user",
      "isActive": true,
      "createdAt": "2025-01-03T10:00:00.000Z",
      "updatedAt": "2025-01-03T10:00:00.000Z"
    }
  ],
  "pagination": {
    "currentPage": 1,
    "totalPages": 3,
    "totalUsers": 25,
    "hasNextPage": true,
    "hasPrevPage": false
  }
}
```


2. Get User by ID

```
http

GET /api/users/:id
Authorization: Bearer <admin_access_token>
```

3. Create User

```
http
```

```
POST /api/users
Authorization: Bearer <admin_access_token>
Content-Type: application/json

{
  "username": "dr_new",
  "email": "new@hospital.com",
  "password": "Doctor123!",
  "fullName": "Dr. New Doctor",
  "role": "user"
}
```

4. Update User

```
http

PUT /api/users/:id
Authorization: Bearer <admin_access_token>
Content-Type: application/json

{
  "username": "updated_username",
  "email": "updated@email.com",
  "fullName": "Updated Name",
  "role": "user",
  "isActive": true
}
```

5. Delete User (Soft Delete)

```
http

DELETE /api/users/:id
Authorization: Bearer <admin_access_token>
```

6. Restore User

```
http

PUT /api/users/:id/restore
Authorization: Bearer <admin_access_token>
```

7. Reset User Password

```http
PUT /api/users/:id/reset-password
Authorization: Bearer <admin_access_token>
Content-Type: application/json

{
  "newPassword": "NewPassword123!"
}
```

🔑 JWT Token Management

Token Structure

Access Token Payload:

```json
{
  "userId": 1,
  "username": "admin",
  "role": "admin",
  "iat": 1641024000,
  "exp": 1641110400
}
```

Token Configuration:

```javascript
// Environment Variables
JWT_SECRET=your_super_secret_jwt_key_here
JWT_REFRESH_SECRET=your_refresh_token_secret_key_here
JWT_EXPIRES_IN=24h
JWT_REFRESH_EXPIRES_IN=7d
```

Token Validation Process:

1. **Extract token** from Authorization header (`Bearer <token>`)
2. **Verify signature** using JWT_SECRET

3. **Check expiration** time
4. **Validate user** still exists and is active in database
5. **Attach user info** to request object

Token Generation:

```javascript
const generateTokens = (user) => {
  const payload = {
    userId: user.id,
    username: user.username,
    role: user.role
  };

  const accessToken = jwt.sign(payload, process.env.JWT_SECRET, {
    expiresIn: process.env.JWT_EXPIRES_IN || '24h'
  });

  const refreshToken = jwt.sign(payload, process.env.JWT_REFRESH_SECRET, {
    expiresIn: process.env.JWT_REFRESH_EXPIRES_IN || '7d'
  });

  return { accessToken, refreshToken };
};
```

User Roles & Permissions

Admin Role

Capabilities:

- Full CRUD operations on all data
- User account management
- System configuration
- Data export functionality
- View all supervision forms
- Monitor sync status
- Reset user passwords
- Soft delete/restore users

Use Cases:

- IT Administrators
- Health Ministry Officials
- Senior Medical Supervisors
- System Managers

User Role (Field Doctors)

Capabilities:

- CRUD operations on own forms (before sync)
- Read-only access after sync
- Change own password
- View own profile
- Export own data
- Cannot access other users' data
- Cannot manage users
- Cannot access admin endpoints

Use Cases:

- Field Doctors
- Medical Officers
- Health Post Staff
- Mobile Medical Teams

Security Features

1. Password Security

- **Hashing Algorithm:** bcryptjs with 12 salt rounds
- **Password Requirements:**
  - Minimum 8 characters
  - At least 1 uppercase letter
  - At least 1 lowercase letter
  - At least 1 number

- **Storage:** Only hashed passwords stored in database

2. Rate Limiting

```javascript
// Global Rate Limiting
windowMs: 15 * 60 * 1000, // 15 minutes
max: 100 requests per IP

// Auth Endpoint Rate Limiting
windowMs: 15 * 60 * 1000, // 15 minutes
max: 5 login attempts per IP
```

3. Security Headers (Helmet.js)

- Content Security Policy
- X-Frame-Options
- X-Content-Type-Options
- Referrer-Policy
- Strict-Transport-Security

4. CORS Configuration

```javascript
const corsOptions = {
  origin: process.env.ALLOWED_ORIGINS.split(','),
  credentials: true,
  optionsSuccessStatus: 200
};
```

5. Input Validation

- **express-validator** for all inputs
- SQL injection prevention
- XSS protection
- Data sanitization

6. Database Security

- **Parameterized queries** prevent SQL injection

- **Connection pooling** with limits
- **Transaction support** for data integrity
- **Soft deletes** preserve audit trail

Error Handling

Authentication Errors

401 Unauthorized

json

```
{
  "error": "Unauthorized",
  "message": "Access token is required"
}
```

json

```
{
  "error": "Unauthorized",
  "message": "Invalid token"
}
```

json

```
{
  "error": "Unauthorized",
  "message": "Token expired"
}
```

json

```
{
  "error": "Unauthorized",
  "message": "Invalid credentials"
}
```

403 Forbidden

json

```
{
  "error": "Forbidden",
  "message": "Admin access required"
}
```

```
json

{
  "error": "Forbidden",
  "message": "You can only modify your own resources"
}
```

400 Bad Request (Validation Errors)

```
json

{
  "error": "Validation Error",
  "message": "Invalid input data",
  "details": [
    {
      "field": "password",
      "message": "Password must be at least 8 characters long",
      "value": "123"
    }
  ]
}
```

409 Conflict

```
json

{
  "error": "Conflict",
  "message": "Username or email already exists"
}
```

429 Too Many Requests

```
json
```

```
{
  "error": "Too Many Requests",
  "message": "Too many authentication attempts, please try again later."
}
```

Database Schema

Users Table

```sql
CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  username VARCHAR(50) UNIQUE NOT NULL,
  email VARCHAR(100) UNIQUE NOT NULL,
  password_hash VARCHAR(255) NOT NULL,
  role VARCHAR(20) NOT NULL DEFAULT 'user',
  full_name VARCHAR(100) NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  is_active BOOLEAN DEFAULT true
);
```

Indexes for Performance

```sql
CREATE INDEX idx_users_username ON users(username);
CREATE INDEX idx_users_email ON users(email);
CREATE INDEX idx_users_role ON users(role);
```

Default Admin User

```sql
-- Created during migration
username: 'admin'
email: 'admin@supervision-app.com'
password: 'Admin123!' (hashed)
role: 'admin'
full_name: 'System Administrator'
```

Code Structure & Files

Project Directory Structure

```
supervision-app-backend/
├── config/
│   └── database.js          # Database configuration and connection
├── middleware/
│   ├── auth.js              # JWT authentication middleware
│   ├── errorHandler.js      # Global error handling
│   └── validation.js        # Input validation rules
├── routes/
│   ├── auth.js              # Authentication endpoints
│   └── users.js             # User management endpoints
├── scripts/
│   └── migrate.js           # Database migration script
├── node_modules/            # Dependencies
├── .env                     # Environment variables
├── .gitignore               # Git ignore rules
├── package.json             # Project dependencies
├── package-lock.json        # Dependency lock file
└── server.js                # Main server file
```

Authentication-Related Files

Primary Files

File	Purpose	Key Functions
`server.js`	Main server entry point	Express setup, middleware configuration, route mounting
`config/database.js`	Database connection	Connection pooling, query execution, transaction handling
`middleware/auth.js`	Authentication middleware	Token validation, user verification, role checking
`routes/auth.js`	Authentication routes	Login, register, token refresh, password change
`routes/users.js`	User management routes	CRUD operations for users (admin only)

Supporting Files

File	Purpose	Key Functions
`middleware/validation.js`	Input validation	Request validation rules, error formatting
`middleware/errorHandler.js`	Error handling	Global error processing, response formatting
`scripts/migrate.js`	Database setup	Table creation, default user setup
`.env`	Configuration	Environment variables, secrets

Key Code Locations

JWT Token Generation

File: `routes/auth.js` **Lines:** ~30-45

```javascript
const generateTokens = (user) => {
  const payload = { userId: user.id, username: user.username, role: user.role };
  const accessToken = jwt.sign(payload, process.env.JWT_SECRET, { expiresIn: '24h' });
  const refreshToken = jwt.sign(payload, process.env.JWT_REFRESH_SECRET, { expiresIn: '7d' });
  return { accessToken, refreshToken };
};
```

Authentication Middleware

File: `middleware/auth.js` **Lines:** ~5-55

```javascript
const authenticateToken = async (req, res, next) => {
  // Token extraction and validation logic
};
```

Database Connection

File: `config/database.js` **Lines:** ~5-25

```javascript
const pool = new Pool(dbConfig);
class Database {
  static async query(text, params = []) { /* ... */ }
}
```

Password Hashing

File: routes/auth.js **Lines:** ~80-90 (registration), ~120-130 (login verification)

```javascript
const hashedPassword = await bcrypt.hash(password, saltRounds);
const isPasswordValid = await bcrypt.compare(password, user.password_hash);
```

User Role Authorization

File: middleware/auth.js **Lines:** ~70-85

```javascript
const requireAdmin = (req, res, next) => {
  if (req.user.role !== 'admin') {
    return res.status(403).json({ error: 'Forbidden', message: 'Admin access required' });
  }
  next();
};
```

Rate Limiting Configuration

File: server.js **Lines:** ~20-35

```javascript
const limiter = rateLimit({
  windowMs: 15 * 60 * 1000,
  max: 100
});
```

Database Schema

File: scripts/migrate.js **Lines:** ~15-50 (users table), ~200-250 (default admin creation)

```javascript
await db.query(`CREATE TABLE IF NOT EXISTS users (...)`);
```

Environment Configuration

File: .env

```bash
# Database settings
DB_HOST=localhost
DB_PORT=5432
DB_NAME=supervision_app
DB_USER=postgres
DB_PASSWORD=admin123

# JWT settings
JWT_SECRET=your_jwt_secret
JWT_REFRESH_SECRET=your_refresh_secret
JWT_EXPIRES_IN=24h
JWT_REFRESH_EXPIRES_IN=7d
```

Dependency Configuration

File: `package.json` **Key authentication dependencies:**

```json
{
  "dependencies": {
    "bcryptjs": "^2.4.3",          // Password hashing
    "jsonwebtoken": "^9.0.2",      // JWT token handling
    "express-validator": "^7.0.1", // Input validation
    "helmet": "^7.0.0",            // Security headers
    "express-rate-limit": "^6.10.0" // Rate limiting
  }
}
```

Middleware Chain Order

File: `server.js` **Lines:** ~25-50

```javascript
```

```
// Security middleware
app.use(helmet());
app.use(limiter);
app.use(cors(corsOptions));
app.use(express.json());

// Routes
app.use('/api/auth', authRoutes);
app.use('/api/users', authenticateToken, userRoutes);

// Error handling
app.use(errorHandler);
```

Database Table Definitions

File: `scripts/migrate.js` **Key tables for authentication:**

- **Users table:** Lines ~15-35
- **Indexes:** Lines ~150-160
- **Default admin creation:** Lines ~200-250

Validation Rules

File: `middleware/validation.js` **Key validations:**

- **User registration:** Lines ~25-50
- **Login validation:** Lines ~55-65
- **Password change:** Lines ~100-120

Testing Guide

Test Environment Setup

```bash
# Start server
npm run dev

# Create database tables
npm run migrate create
```

Postman Testing Collection

1. Health Check

```
Method: GET
URL: http://127.0.0.1:3000/health
Headers: (none)
Body: (none)
```

2. Admin Login

```
Method: POST
URL: http://127.0.0.1:3000/api/auth/login
Headers:
  Content-Type: application/json
Body (raw JSON):
{
  "username": "admin",
  "password": "Admin123!"
}
```

3. Get Admin Profile

```
Method: GET
URL: http://127.0.0.1:3000/api/auth/profile
Headers:
  Authorization: Bearer YOUR_ACCESS_TOKEN
Body: (none)
```

4. Create User - Dr. Smith

```
Method: POST
URL: http://127.0.0.1:3000/api/users
Headers:
  Content-Type: application/json
  Authorization: Bearer YOUR_ACCESS_TOKEN
Body (raw JSON):
{
  "username": "dr_smith",
  "email": "smith@hospital.com",
  "password": "Doctor123!",
  "fullName": "Dr. John Smith",
  "role": "user"
}
```

5. Create User - Dr. Patel

```
Method: POST
URL: http://127.0.0.1:3000/api/users
Headers:
  Content-Type: application/json
  Authorization: Bearer YOUR_ACCESS_TOKEN
Body (raw JSON):
{
  "username": "dr_patel",
  "email": "patel@hospital.com",
  "password": "Doctor456!",
  "fullName": "Dr. Priya Patel",
  "role": "user"
}
```

6. Get All Users

```
Method: GET
URL: http://127.0.0.1:3000/api/users
Headers:
  Authorization: Bearer YOUR_ACCESS_TOKEN
Body: (none)
```

7. Get Users with Pagination

```
Method: GET
URL: http://127.0.0.1:3000/api/users?page=1&limit=5&search=smith
Headers:
  Authorization: Bearer YOUR_ACCESS_TOKEN
Body: (none)
```

8. Update User

```
Method: PUT
URL: http://127.0.0.1:3000/api/users/2
Headers:
  Content-Type: application/json
  Authorization: Bearer YOUR_ACCESS_TOKEN
Body (raw JSON):
{
  "fullName": "Dr. John Smith Jr.",
  "email": "johnsmith@hospital.com",
  "isActive": true
}
```

9. Reset User Password

```
Method: PUT
URL: http://127.0.0.1:3000/api/users/2/reset-password
Headers:
  Content-Type: application/json
  Authorization: Bearer YOUR_ACCESS_TOKEN
Body (raw JSON):
{
  "newPassword": "NewPassword123!"
}
```

10. Change Admin Password

```
Method: PUT
URL: http://127.0.0.1:3000/api/auth/change-password
Headers:
  Content-Type: application/json
  Authorization: Bearer YOUR_ACCESS_TOKEN
Body (raw JSON):
{
  "currentPassword": "Admin123!",
  "newPassword": "MySecurePassword123!",
  "confirmPassword": "MySecurePassword123!"
}
```

11. Test Regular User Login

```
Method: POST
URL: http://127.0.0.1:3000/api/auth/login
Headers:
  Content-Type: application/json
Body (raw JSON):
{
  "username": "dr_smith",
  "password": "Doctor123!"
}
```

12. Refresh Token

```
Method: POST
URL: http://127.0.0.1:3000/api/auth/refresh
Headers:
  Content-Type: application/json
Body (raw JSON):
{
  "refreshToken": "YOUR_REFRESH_TOKEN_FROM_LOGIN"
}
```

13. Verify Token

```
Method: POST
URL: http://127.0.0.1:3000/api/auth/verify
Headers:
  Authorization: Bearer YOUR_ACCESS_TOKEN
Body: (none)
```

14. Logout

```
Method: POST
URL: http://127.0.0.1:3000/api/auth/logout
Headers:
  Authorization: Bearer YOUR_ACCESS_TOKEN
Body: (none)
```

15. Delete User (Soft Delete)

```
Method: DELETE
URL: http://127.0.0.1:3000/api/users/3
Headers:
  Authorization: Bearer YOUR_ACCESS_TOKEN
Body: (none)
```

16. Restore Deleted User

```
Method: PUT
URL: http://127.0.0.1:3000/api/users/3/restore
Headers:
  Authorization: Bearer YOUR_ACCESS_TOKEN
Body: (none)
```

17. Test Permission Denied (User tries to access admin endpoint)

```
Method: GET
URL: http://127.0.0.1:3000/api/users
Headers:
  Authorization: Bearer USER_ACCESS_TOKEN_NOT_ADMIN
Body: (none)
```

Expected: 403 Forbidden

18. Test Invalid Token

```
Method: GET
URL: http://127.0.0.1:3000/api/auth/profile
Headers:
  Authorization: Bearer invalid_token_here
Body: (none)
```

Expected: 401 Unauthorized

Testing Order:

1. **Login as admin** (Test #2) → Copy `accessToken`
2. **Create users** (Tests #4, #5)
3. **List users** (Test #6) → Verify creation
4. **Update user** (Test #8)
5. **Login as regular user** (Test #11) → Copy user token
6. **Test permissions** (Test #17) → Should fail with 403
7. **Test admin functions** with admin token

Security Best Practices

1. Token Security

- Use strong, unique JWT secrets
- Set appropriate token expiration times
- Implement token refresh mechanism
- Validate tokens on every request
- Check user status in database

2. Password Security

- Enforce strong password policies
- Use bcryptjs with high salt rounds
- Never store plain text passwords
- Implement password change functionality
- Force password change on first login

3. API Security

- Rate limiting on authentication endpoints
- Input validation and sanitization
- CORS configuration
- Security headers with Helmet.js
- Parameterized database queries

4. Production Deployment

- Use HTTPS only
- Secure environment variables
- Remove default admin creation
- Set up proper logging
- Regular security updates
- Database connection security

5. Monitoring & Auditing

- Log authentication attempts
- Monitor failed login attempts
- Track user activities
- Set up alerts for suspicious activities
- Regular security audits

Troubleshooting

Common Issues

1. "Invalid token" Error

Causes:

- Token expired
- Wrong JWT secret
- Token malformed
- User deactivated

Solutions:

- Check token expiration
- Verify JWT_SECRET environment variable
- Use refresh token to get new access token
- Check user status in database

2. "Access token is required" Error

Causes:

- Missing Authorization header
- Wrong header format
- Token not included

Solutions:

- Add Authorization header: `Bearer <token>`
- Check header format (space after "Bearer")
- Include valid access token

3. "Password authentication failed" Error

Causes:

- Wrong username/password
- User doesn't exist
- User deactivated

Solutions:

- Verify credentials
- Check user exists in database
- Ensure user is active

4. "Admin access required" Error

Causes:

- User role is not 'admin'
- Wrong endpoint access

Solutions:

- Use admin account
- Check user role in database
- Verify endpoint permissions

5. Database Connection Issues

Causes:

- Wrong database credentials
- Database not running
- Network issues

Solutions:

- Check .env database settings
- Verify PostgreSQL is running
- Test database connection

Debug Steps

1. Check Server Logs

```bash
# Look for error messages in server console
npm run dev
```

2. Verify Environment Variables

```bash
# Check if .env is loaded correctly
console.log(process.env.JWT_SECRET);
```

3. Test Database Connection

```bash
# Run migration to test DB connection
npm run migrate create
```

4. Validate JWT Token

```javascript
// Decode token without verification (for debugging)
const jwt = require('jsonwebtoken');
const decoded = jwt.decode(token);
console.log(decoded);
```

5. Check User Status

```sql
-- Connect to database and check user
SELECT * FROM users WHERE username = 'admin';
```

References

Dependencies Used

- **express**: Web framework
- **jsonwebtoken**: JWT token handling
- **bcryptjs**: Password hashing
- **express-validator**: Input validation
- **helmet**: Security headers
- **express-rate-limit**: Rate limiting
- **cors**: Cross-origin requests
- **pg**: PostgreSQL client
- **dotenv**: Environment variables

Useful Commands

```bash
```

```bash
# Start development server
npm run dev

# Run database migration
npm run migrate create

# Reset database
npm run migrate reset

# Run tests (when implemented)
npm test
```

Environment Variables Reference

```bash
# Server
PORT=3000
NODE_ENV=development

# Database
DB_HOST=localhost
DB_PORT=5432
DB_NAME=supervision_app
DB_USER=postgres
DB_PASSWORD=your_password

# JWT
JWT_SECRET=your_jwt_secret
JWT_REFRESH_SECRET=your_refresh_secret
JWT_EXPIRES_IN=24h
JWT_REFRESH_EXPIRES_IN=7d

# CORS
ALLOWED_ORIGINS=http://localhost:3000,http://localhost:8080

# Rate Limiting
RATE_LIMIT_WINDOW_MS=900000
RATE_LIMIT_MAX_REQUESTS=100
```

Last Updated: January 2025

Version: 1.0

Author: Supervision App Development Team

This documentation covers all aspects of the authentication and authorization system. Keep it updated as new features are added.