

Autenticación en Serena API

Este documento describe el sistema de autenticación implementado en la API de Serena para garantizar la seguridad de los datos y el acceso controlado a los recursos.

Visión General

Serena utiliza JWT (JSON Web Tokens) para la autenticación y autorización. Este enfoque proporciona:

- Autenticación sin estado (stateless)
- Gestión eficiente de sesiones
- Posibilidad de especificar roles y permisos
- Facilidad de implementación en clientes diversos

Flujo de Autenticación

1. Registro de Usuario

Los usuarios pueden ser registrados únicamente por administradores. No existe un endpoint público para registro de usuarios.

2. Inicio de Sesión

1. El usuario envía sus credenciales (email y contraseña) al endpoint `/api/auth/login`
2. El sistema verifica las credenciales contra la base de datos
3. Si son válidas, genera un token JWT firmado
4. El token se devuelve al cliente para su almacenamiento y uso posterior

Ejemplo de solicitud:

```
json
POST /api/auth/login
{
  "email": "usuario@ejemplo.com",
  "password": "contraseña123"
}
```

Ejemplo de respuesta:

```
json

{
  "success": true,
  "data": {
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
    "user": {
      "id": 1,
      "first_name": "Usuario",
      "last_name": "Ejemplo",
      "email": "usuario@ejemplo.com",
      "role": "familiar"
    }
  },
  "message": "Inicio de sesión exitoso"
}
```

3. Uso del Token

Para acceder a endpoints protegidos, el cliente debe incluir el token en el encabezado de autorización:

```
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

4. Renovación del Token

Los tokens tienen un tiempo de expiración (por defecto 24 horas). Para renovar un token sin requerir nuevas credenciales:

1. El cliente envía el token actual al endpoint `/api/auth/refresh`
2. Si el token es válido y no ha expirado, se genera un nuevo token
3. El nuevo token se devuelve al cliente

Ejemplo de solicitud:

```
POST /api/auth/refresh
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

Ejemplo de respuesta:

```
json
{
  "success": true,
  "data": {
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
  },
  "message": "Token renovado"
}
```

5. Cierre de Sesión

La naturaleza sin estado de JWT significa que no hay un verdadero "cierre de sesión" en el lado del servidor. Para cerrar sesión:

1. El cliente elimina el token almacenado localmente
2. Opcionalmente, el cliente puede notificar al servidor para registrar la salida

Estructura del Token JWT

El token JWT utilizado en Serena contiene la siguiente información:

Encabezado (Header)

```
json
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Payload (Claims)

```
json
{
  "sub": "1",           // ID del usuario
  "name": "Usuario Ejemplo", // Nombre del usuario
  "email": "usuario@ejemplo.com",
  "role": "familiar", // Rol del usuario
  "iat": 1609459200, // Issued at (cuándo se emitió)
  "exp": 1609545600 // Expiration time (cuándo expira)
}
```

Firma (Signature)

La firma se genera usando el algoritmo especificado en el encabezado (HS256) con una clave secreta conocida solo por el servidor.

Roles y Permisos

Serena implementa un sistema de roles para controlar el acceso a diferentes recursos:

1. Admin:

- Acceso completo a todas las funcionalidades
- Puede crear/editar/eliminar cualquier recurso
- Puede gestionar usuarios y asignar roles

2. Familiar:

- Acceso a los pacientes vinculados a su cuenta
- Puede ver todos los datos de sus pacientes
- Puede gestionar cuidadores para sus pacientes
- No puede ver datos de pacientes no vinculados

3. Profesional:

- Acceso a los pacientes vinculados a su práctica
- Puede ver datos de salud pero no de gastos
- No puede modificar configuración de pacientes

Middleware de Autenticación

El backend implementa middleware para:

1. **Verificar Token:** Valida la autenticidad y expiración del token JWT
2. **Verificar Rol:** Comprueba si el usuario tiene el rol necesario
3. **Verificar Acceso:** Valida el acceso a recursos específicos (ej. pacientes)

Ejemplo de uso en rutas:

```

python

# Ruta protegida que requiere autenticación
@app.route('/api/patients', methods=['GET'])
@jwt_required
def get_patients():
    # Código para obtener pacientes...

# Ruta que requiere rol específico
@app.route('/api/users', methods=['GET'])
@jwt_required
@role_required('admin')
def get_users():
    # Código para obtener usuarios...

# Ruta que verifica acceso a un paciente específico
@app.route('/api/patients/<int:patient_id>', methods=['GET'])
@jwt_required
@patient_access_required
def get_patient(patient_id):
    # Código para obtener detalles del paciente...

```

Seguridad

Almacenamiento de Contraseñas

Las contraseñas nunca se almacenan en texto plano. Se utiliza BCrypt para:

- Generar hashes seguros de las contraseñas
- Incluir "salt" único para cada contraseña
- Implementar factor de trabajo ajustable (costo)

```

python

# Generar hash de contraseña
password_hash = bcrypt.generate_password_hash(password).decode('utf-8')

# Verificar contraseña
is_valid = bcrypt.check_password_hash(user.password_hash, password)

```

Protección contra Ataques

1. Limitación de Tasa (Rate Limiting):

- Implementada para prevenir ataques de fuerza bruta
- Límite de 5 intentos de login fallidos por minuto por IP

2. Protección CSRF:

- Tokens CSRF para formularios sensibles
- Validación de origen en solicitudes

3. Caducidad de Tokens:

- Tiempo de vida limitado (24 horas por defecto)
- Posibilidad de revocar tokens en caso de compromiso

Flujo de Recuperación de Contraseña

1. Usuario solicita reset en `/api/auth/forgot-password` proporcionando su email
2. Sistema genera token de reset temporal y envía email
3. Usuario hace clic en enlace con token y establece nueva contraseña
4. Sistema verifica token y actualiza la contraseña

Ejemplo de código para la implementación:

```
python
```

```
@app.route('/api/auth/forgot-password', methods=['POST'])
def forgot_password():
    # Recibir email
    email = request.json.get('email')

    # Buscar usuario
    user = User.query.filter_by(email=email).first()
    if not user:
        # No revelar si el email existe o no por seguridad
        return jsonify({
            'success': True,
            'message': 'Si el email está registrado, recibirás instrucciones para restablecer tu contraseña'
        })

    # Generar token
    reset_token = secrets.token_urlsafe(32)
    user.reset_token = reset_token
    user.token_expires = datetime.utcnow() + timedelta(hours=1)
    db.session.commit()

    # Enviar email con Link de reset
    send_reset_email(user.email, reset_token)

    return jsonify({
        'success': True,
        'message': 'Si el email está registrado, recibirás instrucciones para restablecer tu contraseña'
    })
```

Implementación en el Frontend

El cliente React debe implementar:

1. Almacenamiento del Token:

- Guardar en localStorage o cookies seguras
- Opcionalmente en memoria para mayor seguridad

2. Interceptor de Peticiones:

- Añadir automáticamente el token a todas las peticiones
- Manejar respuestas 401 (Unauthorized)
- Intentar renovar token cuando expire

3. Protección de Rutas:

- Componentes de orden superior para verificar autenticación
- Redirección a login si no hay token válido

Ejemplo de implementación con Axios:

```
javascript

// Interceptor para añadir token
axios.interceptors.request.use(config => {
  const token = localStorage.getItem('token');
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});

// Interceptor para manejar errores de autenticación
axios.interceptors.response.use(
  response => response,
  async error => {
    const originalRequest = error.config;

    // Si es error 401 y no hemos intentado renovar
    if (error.response.status === 401 && !originalRequest._retry) {
      originalRequest._retry = true;

      try {
        // Intentar renovar token
        const response = await axios.post('/api/auth/refresh');
        const { token } = response.data.data;

        // Guardar nuevo token
        localStorage.setItem('token', token);

        // Reintentar petición original
        return axios(originalRequest);
      } catch (refreshError) {
        // Si falla la renovación, redirigir a Login
        localStorage.removeItem('token');
        window.location.href = '/login';
        return Promise.reject(refreshError);
      }
    }
    return Promise.reject(error);
  }
);
```

Componente Protegido en React

jsx

```
// Componente de orden superior para proteger rutas
const ProtectedRoute = ({ children, requiredRole = null }) => {
  const { isAuthenticated, user, loading } = useAuth();
  const navigate = useNavigate();

  useEffect(() => {
    if (!loading && !isAuthenticated) {
      navigate('/login');
    }
  });

  if (!loading && requiredRole && user.role !== requiredRole) {
    navigate('/unauthorized');
  }
}, [isAuthenticated, loading, navigate, requiredRole, user]);

if (loading) {
  return <LoadingSpinner />;
}

return isAuthenticated ? children : null;
};

// Uso en rutas
<Routes>
  <Route path="/login" element={<Login />} />
  <Route
    path="/dashboard"
    element={
      <ProtectedRoute>
        <Dashboard />
      </ProtectedRoute>
    }
  />
  <Route
    path="/admin"
    element={
      <ProtectedRoute requiredRole="admin">
        <AdminPanel />
      </ProtectedRoute>
    }
  />
</Routes>
```

Consideraciones para Producción

En entorno de producción, se implementan medidas adicionales:

1. **HTTPS Obligatorio:** Todas las comunicaciones deben usar HTTPS

2. **Configuración de Cookies:**

- HttpOnly: Previene acceso por JavaScript
- Secure: Solo se envía por HTTPS
- SameSite: Protección contra CSRF

3. **Headers de Seguridad:**

- Content-Security-Policy
- X-Content-Type-Options
- X-Frame-Options

Pruebas y Debugging

Para facilitar las pruebas y debugging, se puede:

1. Crear un endpoint de debug (solo en desarrollo):

```
python

@app.route('/api/auth/debug', methods=['GET'])
@jwt_required
def debug_token():
    # Solo disponible en entorno de desarrollo
    if app.config['FLASK_ENV'] != 'development':
        return jsonify({'success': False, 'error': 'Endpoint no disponible'}), 403

    current_user = get_jwt_identity()
    return jsonify({
        'success': True,
        'data': {
            'user_id': current_user,
            'token_info': get_jwt()
        }
    })
```

2. Añadir logs detallados para eventos de autenticación:

```
python
```

```
app.logger.info(f"Inicio de sesión exitoso: usuario_id={user.id}, ip={request.remote_addr}"  
app.logger.warning(f"Intento de acceso denegado: ruta={request.path}, ip={request.remote_ad
```

