Authentication Endpoints Documentation

WearableApi - Login & Register System

🏗️ Architecture Overview

Design Patterns Implemented

1. **Service Layer Pattern**
   - Business logic separated from controllers
   - Services: `AuthenticationService`, `UserFactory`
   - Location: `api/services/`

2. **Factory Pattern**
   - `UserFactory.create_user()` creates users with role-specific profiles
   - Atomic transactions ensure data integrity
   - Extensible for new user types

3. **Strategy Pattern**
   - Authentication logic encapsulated in `AuthenticationService`
   - Permission handling via `AllowAny` for public endpoints

SOLID Principles

- ✅ **Single Responsibility**: Each service has one purpose
- ✅ **Open/Closed**: Easy to extend with new roles
- ✅ **Dependency Inversion**: Views depend on service abstractions

---

📋 API Endpoints

1. Register Endpoint

URL: `POST /api/usuarios/register/`
Authentication: None (Public endpoint)

Request Body (Consumidor)

```
{
  "nombre": "John Doe",
  "email": "john@example.com",
  "password": "SecurePass123",
  "telefono": "1234567890",
  "rol": "consumidor",
  "edad": 30,
```

```
  "peso": 70.5,
  "altura": 175.0,
  "genero": "masculino"
}
```

Request Body (Administrador)

```
{
  "nombre": "Jane Admin",
  "email": "jane@example.com",
  "password": "AdminPass456",
  "telefono": "0987654321",
  "rol": "administrador",
  "area_responsable": "IT Department"
}
```

Success Response (201 Created)

```
{
  "message": "User registered successfully",
  "user_id": 1,
  "email": "john@example.com",
  "rol": "consumidor"
}
```

Error Response (400 Bad Request)

```
{
  "error": "Email already exists",
  "email": ["This email is already registered"]
}
```

---

2. Login Endpoint

URL: `POST /api/usuarios/login/`
Authentication: None (Public endpoint)

Request Body

```
{
  "email": "john@example.com",
```

```
  "password": "SecurePass123"
}
```

Success Response - Consumidor (200 OK)

```
{
  "user_id": 1,
  "nombre": "John Doe",
  "email": "john@example.com",
  "telefono": "1234567890",
  "rol": "consumidor",
  "consumidor_id": 1,
  "edad": 30,
  "peso": 70.5,
  "altura": 175.0,
  "genero": "masculino",
  "bmi": 23.1,
  "created_at": "2025-11-01T12:00:00-07:00"
}
```

Success Response - Administrador (200 OK)

```
{
  "user_id": 2,
  "nombre": "Jane Admin",
  "email": "jane@example.com",
  "telefono": "0987654321",
  "rol": "administrador",
  "administrador_id": 1,
  "area_responsable": "IT Department",
  "created_at": "2025-11-01T12:00:00-07:00"
}
```

Error Response (401 Unauthorized)

```
{
  "error": "Invalid credentials"
}
```

---

3. Update Profile Endpoint

URL: `PATCH /api/usuarios/{id}/profile/`
Authentication: Required (IsAuthenticated)

Request Body (Partial Update)

```
{
  "nombre": "Updated Name",
  "telefono": "9999999999",
  "password": "NewPassword123",
  "edad": 31,
  "peso": 72.0
}
```

Success Response (200 OK)

```
{
  "message": "Profile updated successfully",
  "user": {
    "id": 1,
    "nombre": "Updated Name",
    "email": "john@example.com",
    "telefono": "9999999999",
    "rol": "consumidor"
  }
}
```

✅ Validation Rules

Email Validation

- ✅ Must be unique
- ✅ Valid email format
- ✅ Normalized to lowercase
- ❌ Max length: 255 characters

Password Validation

- ✅ Minimum length: 6 characters
- ✅ Not too common (e.g., "123456")
- ❌ Hashed using Django's `make_password`

Consumidor Fields

- **edad**: 1-120 years
- **peso**: 1-300 kg
- **altura**: 50-250 cm
- **genero**: `masculino`, `femenino`, `otro`

Administrador Fields

- **area_responsable**: Max 200 characters

🧪 Testing

Using cURL

Register Consumidor

```
curl -X POST http://localhost:8000/api/usuarios/register/ \
  -H "Content-Type: application/json" \
  -d '{
    "nombre": "Test User",
    "email": "test@example.com",
    "password": "TestPass123",
    "telefono": "1234567890",
    "rol": "consumidor",
    "edad": 30,
    "peso": 70.5,
    "altura": 175.0,
    "genero": "masculino"
  }'
```

Login

```
curl -X POST http://localhost:8000/api/usuarios/login/ \
  -H "Content-Type: application/json" \
  -d '{
    "email": "test@example.com",
    "password": "TestPass123"
  }'
```

Using Python Test Suite

```
# Make sure server is running
python manage.py runserver

# In another terminal
python testers/test_authentication.py
```

The test suite includes:

- ✅ Register consumidor
- ✅ Register administrador
- ✅ Duplicate email validation

- ✅ Field validation (weak password, invalid edad)
- ✅ Login success
- ✅ Login wrong password
- ✅ Login user not found

🗁 Code Structure

```
api/
├── models/
│   └── user.py              # Usuario, Consumidor, Administrador
├── services/
│   ├── __init__.py
│   ├── auth_service.py      # AuthenticationService
│   └── user_factory.py      # UserFactory
├── serializers.py           # LoginSerializer, RegisterSerializer,
UserProfileSerializer
├── views.py                 # UsuarioViewSet with @action endpoints
└── urls.py                  # Router configuration
```

Service Layer Components

AuthenticationService

```
# Usage
success, user_data, error = AuthenticationService.authenticate(email, password)

if success:
    return Response(user_data, status=200)
else:
    return Response({'error': error}, status=401)
```

UserFactory

```
# Usage
usuario, success, message = UserFactory.create_user(validated_data)

if success:
    logger.info(f"User created: {usuario.email}")
else:
    logger.error(f"Failed: {message}")
```

🔒 Security Features

1. **Password Hashing**
    - Uses Django's `make_password` (PBKDF2 by default)
    - Passwords never stored in plain text
2. **Email Normalization**
    - Emails converted to lowercase
    - Prevents duplicate accounts with case variations
3. **Input Validation**
    - Comprehensive serializer validation
    - Cross-field validation (consumidor-specific fields)
    - Error messages for each field
4. **Atomic Transactions**
    - User creation is atomic (all or nothing)
    - Prevents orphaned records

🚀 Quick Start

1. **Start Django Server**

```
python manage.py runserver
```

2. **Register a User**

```
curl -X POST http://localhost:8000/api/usuarios/register/ \
  -H "Content-Type: application/json" \
  -d
'{"nombre":"Test","email":"test@test.com","password":"pass123","rol":"consum
idor","edad":25,"genero":"masculino"}'
```

3. **Login**

```
curl -X POST http://localhost:8000/api/usuarios/login/ \
  -H "Content-Type: application/json" \
  -d '{"email":"test@test.com","password":"pass123"}'
```

📝 Notes

- **Timezone**: All timestamps use `America/Tijuana` timezone
- **Permissions**: Register and login are public (AllowAny)

- **Profile Updates**: Require authentication
- **BMI Calculation**: Automatically calculated for consumidores (peso / (altura/100)^2)

Common Issues

1. "Email already exists"

- Check if user already registered
- Email comparison is case-insensitive

2. "Invalid credentials"

- Verify email and password are correct
- Both login failures (wrong password, user not found) return same message for security

3. "Validation errors"

- Check field requirements:
  - Password: min 6 characters
  - Edad: 1-120
  - Peso: 1-300
  - Altura: 50-250

Last Updated: November 2025
Architecture: Service Layer Pattern with Factory & Strategy Patterns
Framework: Django REST Framework 3.x