# Documentación Técnica: Sistema de Autenticación y Seguridad

## Índice


1 Arquitectura del Sistema
2 Implementaciones de Seguridad
3 Ejemplos de Uso
4 Guía de Integración
5 Mejores Prácticas


## 1. Arquitectura del Sistema

### 1.1 Modelo de Usuario Personalizado

```
# usuarios/models.py
from django.contrib.auth.models import AbstractUser
from django.utils.translation import gettext_lazy as _
class Usuario(AbstractUser):
    email = models.EmailField(_('correo electrónico'), unique=True)
    telefono = models.CharField(max_length=15, blank=True)
    fecha_registro = models.DateTimeField(auto_now_add=True)
    reset_password_token = models.CharField(max_length=100,
blank=True, null=True)
    reset_password_expires = models.DateTimeField(blank=True,
null=True)
    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = ['username']
```

### 1.2 Serializers con Validación

```
# usuarios/serializers.py
class RegistroUsuarioSerializer(serializers.ModelSerializer):
    password = serializers.CharField(
        write_only=True,
        required=True,
        validators=[validate_password]
    )
    password2 = serializers.CharField(write_only=True,
required=True)
    class Meta:
        model = Usuario
        fields = ('email', 'username', 'password', 'password2',
                  'first_name', 'last_name', 'telefono')
    def validate(self, attrs):
        if attrs['password'] != attrs['password2']:
            raise serializers.ValidationError({
                "password": "Las contraseñas no coinciden"
            })
        return attrs
    def create(self, validated_data):
        validated_data.pop('password2')
```

```
        usuario = Usuario.objects.create_user(**validated_data)
        return usuario
```

## 2. Implementaciones de Seguridad

### 2.1 Rate Limiting

```
# usuarios/views.py
class LoginRateThrottle(AnonRateThrottle):
    rate = '5/minute'  # 5 intentos por minuto
class EmailRateThrottle(AnonRateThrottle):
    rate = '3/hour'    # 3 emails por hora
@api_view(['POST'])
@permission_classes([AllowAny])
@throttle_classes([LoginRateThrottle])
def login_view(request):
    email = request.data.get('email')
    password = request.data.get('password')

    # Verificar intentos fallidos
    key = f'login_attempts_{email}'
    attempts = cache.get(key, 0)

    if attempts >= 5:
        logger.warning(f'Múltiples intentos fallidos de login para
{email}')
        return Response(
            {'error': 'Demasiados intentos fallidos. Intente más
tarde.'},
            status=status.HTTP_429_TOO_MANY_REQUESTS
        )
```

### 2.2 Sistema de Tokens JWT

```
# settings.py
SIMPLE_JWT = {
    'ACCESS_TOKEN_LIFETIME': timedelta(minutes=60),
    'REFRESH_TOKEN_LIFETIME': timedelta(days=1),
    'ROTATE_REFRESH_TOKENS': False,
    'BLACKLIST_AFTER_ROTATION': True,
    'UPDATE_LAST_LOGIN': False,
    'ALGORITHM': 'HS256',
    'SIGNING_KEY': SECRET_KEY,
    'AUTH_HEADER_TYPES': ('Bearer',),
}
# usuarios/views.py
@api_view(['POST'])
@permission_classes([IsAuthenticated])
def logout_view(request):
    try:
        tokens =
OutstandingToken.objects.filter(user_id=request.user.id)
```

```
            for token in tokens:
                BlacklistedToken.objects.get_or_create(token=token)
            return Response({"message": "Logout exitoso"})
        except Exception as e:
            logger.error(f'Error en logout: {str(e)}')
            return Response(
                {"error": "Error al cerrar sesión"},
                status=status.HTTP_500_INTERNAL_SERVER_ERROR
            )
```

### 2.3 Sistema de Logging

```
# settings.py
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'formatters': {
        'verbose': {
            'format': '{levelname} {asctime} {module} {message}',
            'style': '{',
        },
    },
    'handlers': {
        'file': {
            'level': 'WARNING',
            'class': 'logging.FileHandler',
            'filename': 'debug.log',
            'formatter': 'verbose',
        },
    },
    'loggers': {
        'usuarios': {
            'handlers': ['file'],
            'level': 'WARNING',
            'propagate': True,
        },
    },
}
```

## 3. Ejemplos de Uso

### 3.1 Login

```
# Ejemplo usando requests en Python
import requests
def login(email, password):
    response = requests.post(
        'http://localhost:8000/usuarios/login/',
        json={
            'email': email,
            'password': password
        }
```

```
    )

    if response.status_code == 200:
        tokens = response.json()
        return tokens['access'], tokens['refresh']
    else:
        raise Exception(response.json()['error'])
# Ejemplo de uso
try:
    access_token, refresh_token = login('usuario@ejemplo.com',
'contraseña123')
    print(f'Access Token: {access_token}')
except Exception as e:
    print(f'Error: {str(e)}')
```

### 3.2 Recuperación de Contraseña

```
# Ejemplo usando requests en Python
def solicitar_reset_password(email):
    response = requests.post(
        'http://localhost:8000/usuarios/solicitar_reset_password/',
        json={'email': email}
    )
    return response.json()
def confirmar_reset_password(token, new_password):
    response = requests.post(
        'http://localhost:8000/usuarios/reset_password_confirm/',
        json={
            'token': token,
            'new_password': new_password,
            'new_password2': new_password
        }
    )
    return response.json()
# Ejemplo de uso
try:
    # Paso 1: Solicitar reset
    result = solicitar_reset_password('usuario@ejemplo.com')
    print('Revisa tu email para el token')

    # Paso 2: Confirmar reset con el token recibido
    token = input('Ingresa el token recibido: ')
    result = confirmar_reset_password(token, 'nueva_contraseña123')
    print('Contraseña actualizada exitosamente')
except Exception as e:
    print(f'Error: {str(e)}')
```

### 3.3 Uso con Flutter/Dart

```
// api_service.dart
class ApiService {
  final String baseUrl = 'http://localhost:8000';
```

```
  Future<Map<String, dynamic>> login(String email, String password)
async {
    try {
      final response = await http.post(
        Uri.parse('$baseUrl/usuarios/login/'),
        headers: {'Content-Type': 'application/json'},
        body: jsonEncode({
          'email': email,
          'password': password,
        }),
      );

      if (response.statusCode == 200) {
        return jsonDecode(response.body);
      } else {
        throw Exception(jsonDecode(response.body)['error']);
      }
    } catch (e) {
      throw Exception('Error de conexión');
    }
  }

  Future<Map<String, dynamic>> getUserProfile(String token) async {
    try {
      final response = await http.get(
        Uri.parse('$baseUrl/usuarios/me/'),
        headers: {
          'Authorization': 'Bearer $token',
          'Content-Type': 'application/json',
        },
      );

      if (response.statusCode == 200) {
        return jsonDecode(response.body);
      } else {
        throw Exception('Error al obtener perfil');
      }
    } catch (e) {
      throw Exception('Error de conexión');
    }
  }
}
```

# 4. Guía de Integración

## 4.1 Configuración del Cliente

```
// Ejemplo de configuración en Axios (JavaScript)
const api = axios.create({
  baseURL: 'http://localhost:8000',
  timeout: 5000,
  headers: {
```

```
    'Content-Type': 'application/json',
  }
});
// Interceptor para agregar token
api.interceptors.request.use(
  (config) => {
    const token = localStorage.getItem('access_token');
    if (token) {
      config.headers.Authorization = `Bearer ${token}`;
    }
    return config;
  },
  (error) => {
    return Promise.reject(error);
  }
);
// Interceptor para refrescar token
api.interceptors.response.use(
  (response) => response,
  async (error) => {
    const originalRequest = error.config;

    if (error.response.status === 401 && !originalRequest._retry) {
      originalRequest._retry = true;

      try {
        const refreshToken = localStorage.getItem('refresh_token');
        const response = await
axios.post('/usuarios/token/refresh/', {
          refresh: refreshToken
        });

        localStorage.setItem('access_token', response.data.access);

        originalRequest.headers.Authorization =
          `Bearer ${response.data.access}`;
        return api(originalRequest);
      } catch (err) {
        // Token de refresco expirado, redirigir a login
        window.location.href = '/login';
      }
    }

    return Promise.reject(error);
  }
);
```

## 4.2 Manejo de Errores

```
# usuarios/views.py
from rest_framework.exceptions import APIException
class MaxLoginAttemptsExceeded(APIException):
    status_code = 429
```

```
    default_detail = 'Demasiados intentos fallidos. Intente más
tarde.'
    default_code = 'max_login_attempts_exceeded'
@api_view(['POST'])
@permission_classes([AllowAny])
@throttle_classes([LoginRateThrottle])
def login_view(request):
    try:
        email = request.data.get('email')
        if not email:
            raise ValidationError({'email': 'Este campo es
requerido'})

        # Verificar intentos fallidos
        key = f'login_attempts_{email}'
        attempts = cache.get(key, 0)

        if attempts >= 5:
            raise MaxLoginAttemptsExceeded()

        # ... resto del código ...

    except ValidationError as e:
        return Response(e.detail,
status=status.HTTP_400_BAD_REQUEST)
    except MaxLoginAttemptsExceeded as e:
        return Response({'error': str(e)}, status=e.status_code)
    except Exception as e:
        logger.error(f'Error inesperado en login: {str(e)}')
        return Response(
            {'error': 'Error interno del servidor'},
            status=status.HTTP_500_INTERNAL_SERVER_ERROR
        )
```

## 5. Mejores Prácticas

### 5.1 Configuración de Seguridad en Producción

```
# settings.py
SECURE_SSL_REDIRECT = True
SESSION_COOKIE_SECURE = True
CSRF_COOKIE_SECURE = True
SECURE_BROWSER_XSS_FILTER = True
SECURE_CONTENT_TYPE_NOSNIFF = True
X_FRAME_OPTIONS = 'DENY'
SECURE_HSTS_SECONDS = 31536000
SECURE_HSTS_INCLUDE_SUBDOMAINS = True
SECURE_HSTS_PRELOAD = True
# Configuración de CORS
CORS_ALLOWED_ORIGINS = [
    "https://tuapp.com",
    "https://api.tuapp.com",
```

```
]
CORS_ALLOW_CREDENTIALS = True
CORS_EXPOSE_HEADERS = ['Content-Type', 'X-CSRFToken']
```

### 5.2 Validaciones Personalizadas

```
# usuarios/validators.py
from django.core.exceptions import ValidationError
import re
def validate_strong_password(password):
    if len(password) < 8:
        raise ValidationError(
            'La contraseña debe tener al menos 8 caracteres'
        )

    if not re.search(r'[A-Z]', password):
        raise ValidationError(
            'La contraseña debe contener al menos una mayúscula'
        )

    if not re.search(r'[a-z]', password):
        raise ValidationError(
            'La contraseña debe contener al menos una minúscula'
        )

    if not re.search(r'[0-9]', password):
        raise ValidationError(
            'La contraseña debe contener al menos un número'
        )

    if not re.search(r'[!@#$%^&*(),.?":{}|<>]', password):
        raise ValidationError(
            'La contraseña debe contener al menos un carácter
especial'
        )
# Uso en serializers
class RegistroUsuarioSerializer(serializers.ModelSerializer):
    password = serializers.CharField(
        write_only=True,
        required=True,
        validators=[validate_password, validate_strong_password]
    )
```

### 5.3 Monitoreo y Alertas

```
# usuarios/signals.py
from django.db.models.signals import post_save
from django.dispatch import receiver
from django.core.mail import send_mail
@receiver(post_save, sender=Usuario)
def alert_on_multiple_failed_attempts(sender, instance, created,
**kwargs):
    key = f'login_attempts_{instance.email}'
```

```
attempts = cache.get(key, 0)

if attempts >= 3:
    # Enviar alerta al administrador
    send_mail(
        'Alerta de Seguridad',
        f'Múltiples intentos fallidos para {instance.email}',
        'noreply@tuapp.com',
        ['admin@tuapp.com'],
        fail_silently=False,
    )
```

Esta documentación proporciona una guía detallada de la implementación y uso del sistema de autenticación. ¿Necesitas que profundice en algún aspecto específico?