VoyaAI API Security Guide

1. Token-Based Authentication

Use Django REST Framework's TokenAuthentication. Each user receives a token on login, which must be sent in the header of each request:

```
Authorization: Token user_token_here
```

2. Enforce HTTPS

Use HTTPS in production to encrypt all data between the client and your API. Use Let's Encrypt or a provider like Cloudflare to install SSL certificates.

3. CORS Restrictions

Configure django-cors-headers to allow only your frontend domain:

```
CORS_ALLOWED_ORIGINS = ['https://your-frontend.com']
```

Avoid using CORS_ALLOW_ALL_ORIGINS = True.

4. Permissions for Protected Routes

Use DRF's IsAuthenticated permission class to restrict access to sensitive data. Always link itinerary access to a logged-in user.

5. Throttling and Rate Limiting

Limit how often users or anonymous clients can call your API to prevent abuse:

```
REST_FRAMEWORK = {
  'DEFAULT_THROTTLE_CLASSES': [...],
  'DEFAULT_THROTTLE_RATES': {'user': '100/hour', 'anon': '10/hour'}
}
```

6. Input Validation and Sanitization

Always validate user input using Django serializers. Sanitize all inputs, especially those used in AI prompts or database queries.

7. Secure Environment Variables

Store sensitive values like API keys and database URLs in .env files. Use python-decouple or django-environ to load them securely into your settings.

8. CSRF Considerations

If you're using session-based authentication, ensure CSRF protection is enabled. With token-based auth, CSRF is typically not required.

9. Use Proxies or API Gateways (Advanced)

For high-traffic apps, consider using Cloudflare, AWS API Gateway, or NGINX to provide an additional layer of rate-limiting and protection.

10. Monitoring and Logging

Set up logging for authentication events and failed requests. Tools like Sentry or AWS CloudWatch can alert you to suspicious activity.