Coding & Development / API Development

Generate JWT, OAuth, or API key authentication middleware for Express, Django, or FastAPI.
Difficulty: Advanced
Model: GPT-4 / Claude / Gemini
Use Case: Authentication, Middleware, API Security
Updated: May 2026
Why This Prompt Exists
Most APIs have authentication — but implementing it correctly is easy to get wrong.

You get:

- hardcoded secrets in code (security risk)
- no token validation (expiration, signature)
- inconsistent auth headers (Bearer vs. Basic)
- no user context attached to request
- no error handling for invalid tokens

But authentication middleware is not optional.

It is the gatekeeper of your API.

- Extract token from headers (Authorization: Bearer)
- Validate token (signature, expiration, issuer)
- Attach user to request object (req.user)
- Handle errors (401 Unauthorized, 403 Forbidden)
- Optionally check user permissions/roles

Without proper middleware, your API is vulnerable.

This framework forces AI to generate secure authentication middleware.

The Prompt

```
Assume the role of a security engineer who builds authentication
middleware.

Your task is to generate authentication middleware code.

Generate:

1. IMPORTS
   - JWT library, environment variables, etc.

2. TOKEN EXTRACTION
   - Get token from Authorization header
   - Handle missing or malformed header

3. TOKEN VALIDATION
   - Verify signature
   - Check expiration
   - Verify issuer/audience (if applicable)

4. USER ATTACHMENT
   - Decode user from token
   - Attach to request object (req.user)

5. ERROR HANDLING
   - Invalid token → 401
```

   - Missing token → 401
   - Expired token → 401 (with refresh hint)

6. MIDDLEWARE EXPORT
   - Function to use in route handlers

INPUTS:

Framework:
[EXPRESS / FASTAPI / DJANGO / OTHER]

Authentication Type:
[JWT / API KEY / OAUTH2]

Token Source:
[HEADER / COOKIE / QUERY PARAM]

User Data in Token:
[E.G., "userId, email, role"]

Secret Management:
[ENV VAR / VAULT / HARDCODED (NOT RECOMMENDED)]

RULES:
- Never hardcode secrets — use environment variables
- Validate token signature (don't trust client)
- Check expiration (reject expired tokens)
- Handle missing token gracefully (401)
- Attach user to request for downstream handlers

```
- Use async/await for database lookups (if needed)
- Log auth failures for monitoring (but don't leak details)
```

How To Use It

- Never hardcode secrets — use environment variables.
- Validate token signature — don’t trust client-side tokens.
- Check expiration — reject expired tokens.
- Attach user to request for downstream handlers.
- Test middleware with valid, expired, and malformed tokens.

Example Input

Framework: EXPRESS

Authentication Type: JWT

Token Source: HEADER (Authorization: Bearer)

User Data in Token: userId, email, role

Secret Management: ENV VAR (process.env.JWT_SECRET)

Why It Works
Most auth middleware is buggy or insecure.

This framework improves outcomes by forcing:

- token extraction (standard pattern)
- signature validation (security)
- expiration checking (freshness)
- user attachment (context)
- error handling (proper status codes)

Great auth middleware doesn’t just check tokens — it validates, attaches, and fails gracefully.

Build Better AI Systems

Subscribe for advanced prompt engineering, AI coding tools, API development frameworks, and practical strategies for developers and engineers.

Carefully engineered prompts for people doing real work.

Share this:

- Share on Facebook (Opens in new window) Facebook
- Share on X (Opens in new window) X

See also The API Client SDK Generator