API Keys

- API Keys are unique secret identifiers used to authenticate client applications accessing an API.

Management :- 1) Secure generation → Hashing → storage → Rotation
↓
Monitoring ← Revocation

Prompt Injection Attack

- When we integrate LLM into backend, flow :

User → Backend → LLM → Backend → Response
↓ ↑
Tool / DB / API

- New LLM may call tools, query DB, access internal APIs, process hidden instructions.
- Prompt injection is when a user inserts malicious instructions into input that override or manipulate AI's intent.

Simple :- Attackers tricks the AI to ignore your system instruction.

- Since LLMs have access to all tools & DB, if prompt injection succeeds, attackers may :
  - Extract secrets
  - Access internal data
  - Bypass restrictions
  - Trigger unauthorized actions.
- Normally in backend systems we :-
  - validate JSON
  - sanitize SQL
  - escape HTML

But here attack is semantic. Malicious content is not code its natural language.

Types

- Instruction override → "Ignore previous rules"
- Data exfiltration attempt → 'Reveal hidden system prompt'
- Tool manipulation → 'Call this function with these arguments'
- Indirect injection (via documents) → Malicious instructions hidden in PDFs or web page that model reads. (Dangerous in RAG system).

How to defend against Prompt Injection (Backend)

1) Strict Tool Authorization →
   - Validate permission & user identity
   - Role checks.

2) Separate system & User Prompts

3) Output Validation Layer →
   - Do not execute model output.
   - Parse structured output, validate fields
   - Reject suspicious instructions.

4) Principle of least Privilege →
   - LLM should access minimal data.
   - Have limited scope & no access to production credentials directly.

PII Masking

- Data like Full name, Contact, email, Bank details are called Personally Identifiable Information.)
- If this data is sent to 3rd party, to AI models or exposed in responses it can cause privacy violation, data breach etc.

= PII masking means hiding / obfuscating sensitive personal data. so it cannot be directly identified.

Applications

1) In API responses → returning card no. like xxxx xxxx 3456
2) In Logs → User Login failed for jxxx@gmail.com.
3) Before sending data to AI

PII Masking	Encryption	Hashing
• Hides part of data visually • still readable, used for display safety	• fully reversible with Key • Used for secure storage	• One way transformation • Used for passwords.

PII masking Implementation

1) Regex based masking
   - Detect patterns like email format, credit cards, phone no.

2) Structured field masking

   If your API knows field names:

   { email : "john.@ gmail.com"
   }

   Mask specific fields before response (more reliable than regex)

3) Middleware based masking

   Request → Middleware → Mask PII → Logging

   Response → Mask sensitive fields → Send to client.

In AI systems

- Prompts & may contain PII, AI responses may repeat PII, Embeddings may store PII.
- If not masked:
  - Model leakage
  - Compliance violations.

- User → Backend
- Access control
- PII detection
- Mask if required
- Send to AI or logs
- Validate output
- Mask before response.

Mistakes

- Masking only in frontend
- Forgetting logs.
- Hardcoding masking rules.
- Not masking AI prompts