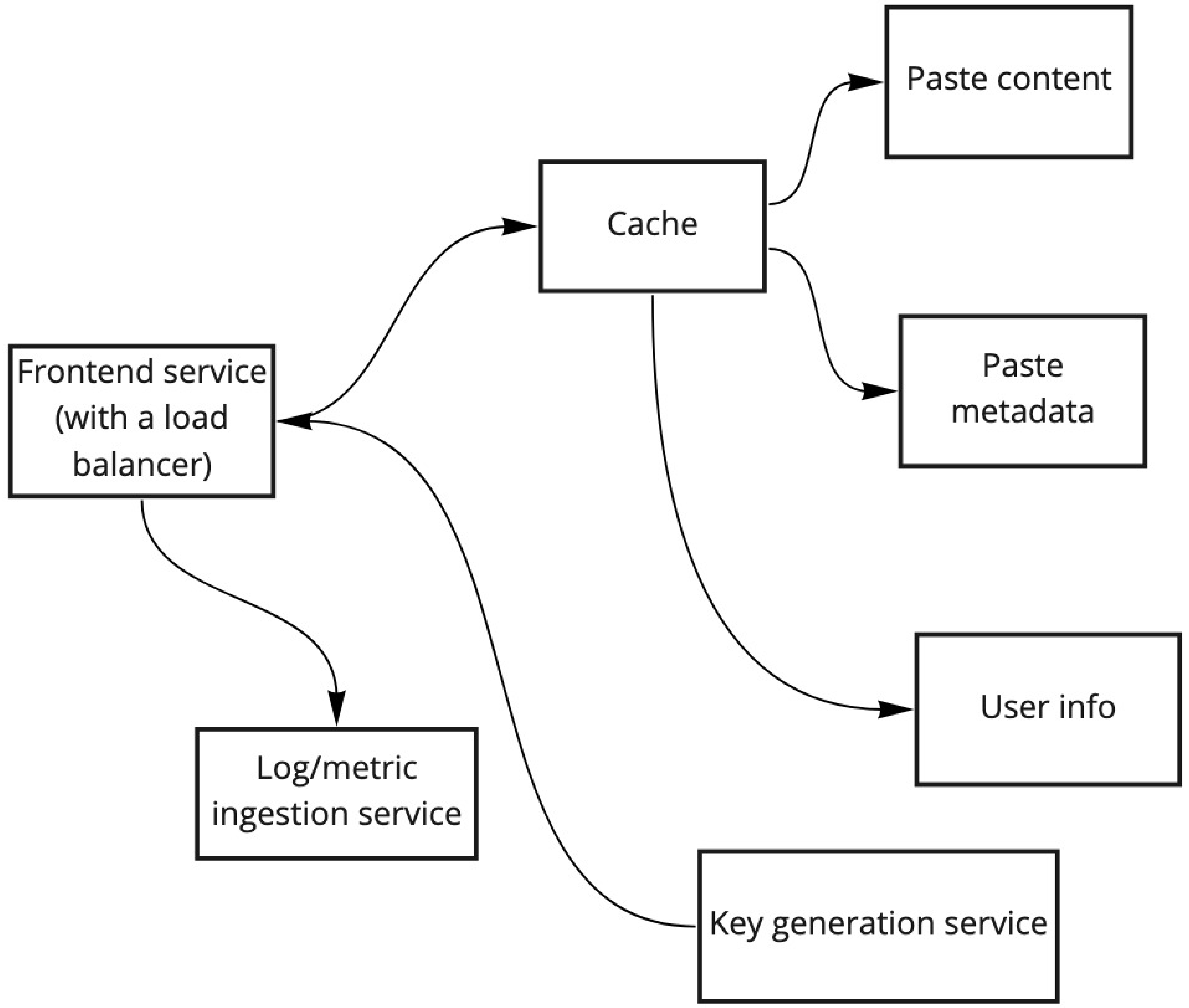
Functional requirements:
- Text/images -> URL. (Max 10 MB.)
- TTL: expiry time for URLs.
- Don't allow edits.
- Stretch goals:
  - Password protected content.
  - Custom URLs.

Non-functional requirements:
- Highly available and durable.
- Strong consistency = read-after-write.

Estimations:
- Assumptions:
  - 100M users.
  - 5 pastes per user per month.
  - Write to read ratio: 1:10
  - Average paste size: 100 KB.
  - 20% content gets 80% traffic.
- Storage:
  - Per month: 100M * 5 * * 100KB = 5 TB.
  - Cache size: 1 TB.
  - For 5 years: 5 * 60 = 300 TB.
- Request rate:
  - Writes: 100M * 5 / 2.5M = 200 writes per second.
  - Reads: 2k per second.



Paste content
Cache
Frontend service (with a load balancer)
Paste metadata
User info
Log/metric ingestion service
Key generation service

Data stores:
- Paste content:
  - Text or images.
  - Encrypted by user password for password protected pastes.
  - Encrypted-at-rest by default.
  - Could be compressed - subject to load testing.
  - TTL.
- Paste metadata:
  - NoSQL DB.
  - Keyed by paste-id. "Value" could contain user-id, timestamp, password-protection, expiration time etc.
- User info:

URL generation:
- UUIDs: 128 bits in size = 32 hex characters. Plus, 4 hyphens, so 36 character/byte long IDs.
- SHA/MD5 hash of user content + user-id -> 128 bits of info.
  - [a-zA-Z0-9_.] - 64 characters.
  - 6 bits per character.
  - Want 8 character long IDs = 64^8
  - 8*6 = 48 bits out of 128 bits that we generated by the hash.
  - In case of collisions, "+ random number".
- Key generation service.

Cache usage:
- Write to cache async if we got a cache miss on read.
- (Recently created content might be more popular.) We always write to cache whenever users create content.

How to store data:
- Update content first, metadata second.