Multiple login & Singleton problem.

Use object graph instead

tokens can be stored in keychains. each domain for each user

private data
- User data
- API key (Identifies the application)

Anything that is private should not be in the app

=> People can reverse engineer and get it

FetchResultController -> One which keeps listening to CoreData Changes

Monolith, SOA, Single point of failure

Distributed system

CAP - Theory

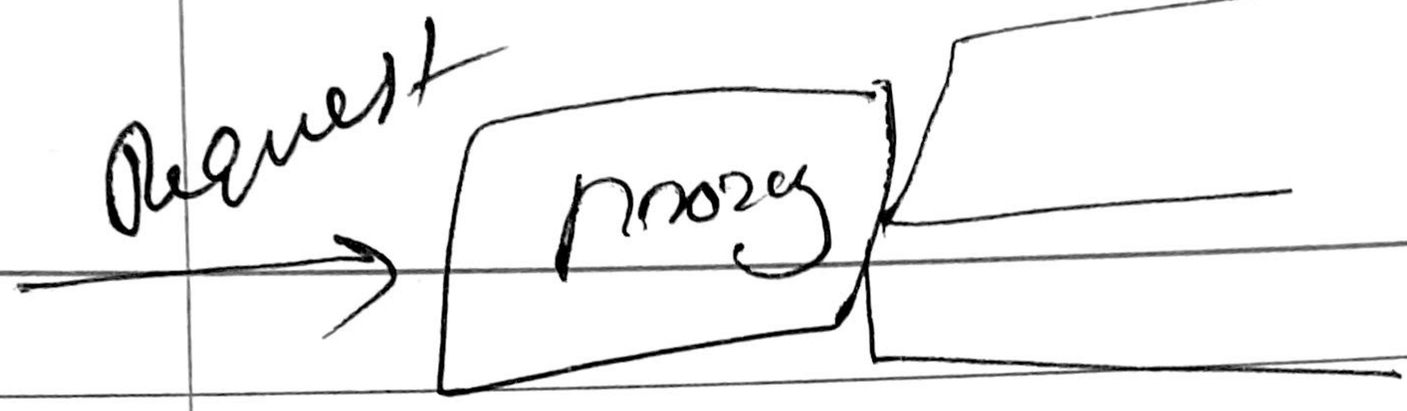
Consistency — Availability — Partition tolerant

Load balancing →

Caching → performance booster, feature of DB

1. DB level caching
2. Application level caching
   @cacheable (spring framework)
3. Web service level caching

(4) client caching



Rate limiting → If there's a spike in Request, it starts dropping the request (It can be a hack)

Kafka - Messaging Queue - Resilience
- making process asynchronous
- don't want client to wait

Consistency - Data should be consistent in all nodes

Availability - Client should always get a respo se

Partition tolerance - Client should not know one of the node are off