SE 464
Study Question Activity

Design Scenario:

Suppose you need to architect a logging system that uses CassandraDB for storage.

How would you configure CassandraDB for this task?

Some Hand-Holding (Prompting Questions)

1. What kind of database is CassandraDB? What properties does it have?
2. What is the relative frequency/intensity of read and write operations for this application?

More Hand-Holding

1. CassandraDB is a quorum-based system.
2. Logging is write-intensive, and has relatively few reads.

Design Scenario:

Suppose you are running an online video game service. You have a MySQL database storing various player information, such as usernames and passwords. You notice that every weekday at 5:30PM, you have a large influx of players. This is causing the database to be heavily overloaded.

How would you alter the database to support this large influx of players?

Some Hand-Holding (Prompting Questions)

1. What kind of database is MySQL? What can it provide to users to allow for faster reads?
2. What is the relative frequency/intensity of read and write operations for this application?
3. What can we use to ensure database traffic is distributed evenly?

More Hand-Holding

1. MySQL supports Read-Replicas
2. Logging in is a read intensive operation
3. What can we use to ensure traffic is distributed across database replicas?

Design Scenario:

Suppose you're building a phone app that allows users to send digital postcards to their friends.

How would you process a user request to send a postcard, prioritizing user experience?

Some Hand-Holding (Prompting Questions)

1. What are the two main styles of handling a request?
2. Of these two styles, which one favours better user experience?

More Hand-Holding

1. There are synchronous requests and asynchronous requests.
2. Asynchronous requests allow a user to immediately perform another action.

Q: Create a real world problem where a design pattern could be used to effectively solve it. Provide the design pattern’s name, a brief overview of how it solves the problem, and any potential trade offs.

Q: Create a real world problem where a design pattern could be used to effectively solve it. Provide the design pattern's name, a brief overview of how it solves the problem, and any potential trade offs.

A: Below is an example, this question is open-ended:

Real World Problem:

You are working on a brand new social media platform. One feature you're tasked with working on is allowing for users to see new posts of people they follow in real time。

Design Pattern Solution: Observer Pattern

How it solves the problem: Users are publishers that create content. Followers are other users that subscribe to the publishers, other users. When a user (publisher) creates a new post, they will notify all their followers (subscribers).

Trade-Offs: May be difficult to scale if there are a large number of subscribers.

Q: Your company is working on a financial services API which will facilitate transactions between customers’ debit accounts and seller/service providers. The API will be integrated into the websites of any clients that decide to offer transactions using your service and need to work securely with their existing infrastructure. How might the use of RPCs to implement this API be beneficial, and what drawbacks need to be taken into consideration when designing the API?

Q: Your company is working on a financial services API which will facilitate transactions between customers' debit accounts and seller/service providers. The API will be integrated into the websites of any clients that decide to offer transactions using your service and need to work securely with their existing infrastructure. How might the use of RPCs to implement this API be beneficial, and what drawbacks need to be taken into consideration when designing the API?

A: The language neutrality of RPCs will make it much easier for clients to integrate the API into their systems, and the abstracted network communication will make it easier to work with when they start implementing it. This will make it less likely that developers make mistakes implementing the API and may be a factor in whether they become a client at all. However, the design of the API should ensure that errors can be easily handled and are clear. Additionally, there should be robust testing capabilities to ensure that clients can check that the API is implemented correctly. Finally, The API should make use of robust encryption to ensure that customers data is not exposed when travelling over the network.

Q: You are part of a team looking to create a retro video games e-commerce website. For such a project, should your team use a monolithic design or a microservices design? Give reasons to support your choice.

Q: You are part of a team looking to create a retro video games e-commerce website. For such a project, should your team use a monolithic design or a microservices design? Give reasons to support your choice.

A: A microservices design would be optimal to create an e-commerce website.
It allows the website to be organized into several services and databases (e.g. inventory, accounting, shipping, ...). Agree on the network-level API for each service. Assign team members to each service and build them independently. It is also easier to scale than a monolithic design and can ensure better data security.

Q: You are part of a team that is debating whether to prioritize the development of new features or improving existing NFPs in a software product. Evaluate the pros and cons of each approach.

Q: You are part of a team that is debating whether to prioritize the development of new features or improving existing NFPs in a software product. Evaluate the pros and cons of each approach.

A: Prioritizing the development of new features can attract more users and enhance market competitiveness. However, neglecting existing NFPs can lead to performance issues, security vulnerabilities, and increased complexity and result in losing current users. The decision should consider factors like user feedback, market demands, and the system’s current state

Q: Your company is trying to decide on whether to use a NoSQL-type key-value store or traditional relational database for OLAP workloads. Which one would you suggest and why?

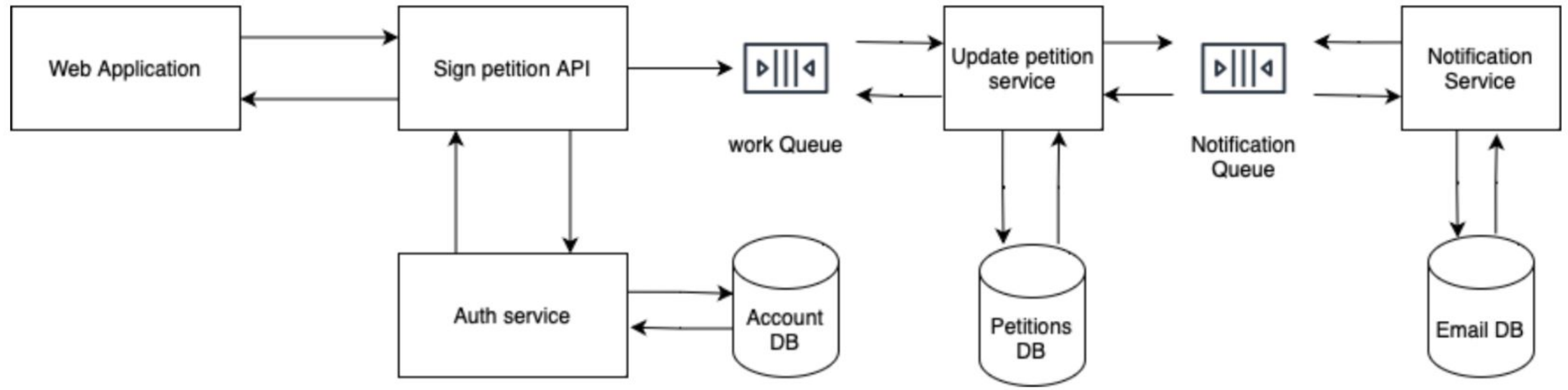
Q: Your company is trying to decide on whether to use a NoSQL-type key-value store or traditional relational database for OLAP workloads. Which one would you suggest and why?

A: Since OLAP workloads involve heavy use of scans over entire collections/tables across the database and do not typically feature cross-referencing relations or foreign keys until the whole collection is aggregated, a relational database might incur processing costs that a NoSQL-type key-value storage design would not. Furthermore, in OLAP workloads, indices aren’t of much use and the more easily scalable and distributed architecture of NoSQL databases would offer more throughput on metrics and analytics.

Q: You are the lead architect at your company. The signing service has already been created for a petition signing webapp, but you have been tasked to design the workflow, starting from a request from the webapp to the service. Draw an architecture diagram for this, including the use of a message queue.

Q: You are the lead architect at your company. The signing service has already been created for a petition signing webapp, but you have been tasked to design the workflow, starting from a request from the webapp to the service. Draw an architecture diagram for this, including the use of a message queue.

A:



Q: You are designing a file-submission service for your company. The requirements are:
(a) Multiple submission which can be edited up to the deadline must be allowed.
(b) Attempts to edit/alter submissions past the deadline must be denied.
(c) Files uploaded must secure (there must be an authentication system for viewing/editing submissions).
(d) Submissions must be tagged with the original time of submission.

Q: You are designing a file-submission service for your company. The requirements are:
(a) Multiple submission which can be edited up to the deadline must be allowed.
(b) Attempts to edit/alter submissions past the deadline must be denied.
(c) Files uploaded must secure (there must be an authentication system for viewing/editing submissions).
(d) Submissions must be tagged with the original time of submission.

A: Describe which requirements are tailored to either a synchronous or asynchronous implementation. Include information on how unexpected behavior such as file upload errors or network connectivity issues during uploads will be handled, and how users will be notified of these issues. Justify how your architectural decision will be suitable for these unexpected behaviors.

Q: Explain how the Three-Tiered Pattern could be implemented for a web application?

Q: Explain how the Three-Tiered Pattern could be implemented for a web application?

A: Using the Three-Tiered Pattern, the web application can be organized into three tiers:
(a) Front tier - Refers to the user interface, which could run on a web browser or as a desktop application.
(b) Middle tier - Contains all the main functionality (processes info from front tier + adds/modifies or deletes data from the back tier, e.g. using Java + uses API calls to connect to the back tier.
(c) Back tier - Where information is stored/managed, e.g. PostgreSQL, MongoDB.

Q: You’re working on a cross-platform architecture with a service API. Now, you want to add a new platform. What do you need to do in order to add this new platform successfully?

Q: You’re working on a cross-platform architecture with a service API. Now, you want to add a new platform. What do you need to do in order to add this new platform successfully?

A: In order to add a new platform, one must ensure that it follows the API protocol (REST, GraphQL, etc) used by the service API when interacting with it. It must also follow any other communication steps required by the service API such as authentication. Additionally, there may also be platform-specific features you will want to add support for on the backend (push notifications for mobile, etc).