Bengaluru Connect

I would like to follow a hybrid approach i.e to use both Relational and Non-Relational databases for different parts of the platform.

My analysis on three requirements:-

- Requirement A - Ticket Sales : This part of the system requires transactions to take place i.e if any error/bug occurs in between the transaction, it has to be revoked back to its previous state. If everything works fine(all the desired operations being performed as planned, no bugs/issues in between the transaction) then we can commit and end the transaction.
  It is better to keep this module in the Relational database. I feel so because:-
  - Every Transaction needs to be atomic , also it has to follow all ACID properties. Relational databases have better transaction support compared to non relational databases. We can introduce @Transactional on functions performing ticket-sales.
  - Since Ticket-Sales seems to have a definite schema structure over a period of longer duration, it is better to have a pre-defined schema (Table). For which relational databases can be a better option.
  - In terms of CAP theorem, the availability and consistency should work together in this case, which suggests that a relational database can be a better option.

- Requirement B - News Feed :- As mentioned, this particular module requires high availability(atleast the feed, where subscribed posts need to be shown). Since the news feed has a huge and enormous amount of data which grows every moment, it requires both vertical and horizontal scaling. Vertical scaling if new posts/news come up every moment(the database needs to scale up vertically in this case). Horizontal scaling if the same post/news-instance grows(means new updates come on the same/previous instance).A feed can contain n number of posts. Every post can contain n number of comments.Every such comment can have n number of replies. We can observe that it doesn't have a schema-structure. Also we can observe that it has a hierarchical structure/tree based structure.
  It is better to keep this module in a Non-Relational database. I feel so because:-
  - Since the module requires both vertical and horizontal scaling, NoSql databases are better for horizontal scaling.For vertical scaling we can add more resources like more RAM/more pods to handle vertical scaling.
  - Since the News-feed has multiple posts, where each post has n number of attributes associated with it, it clearly has a tree based structure. I feel NoSql databases are the ultimate choice.

- Also that the system keeps growing with past history attached with present news, it may be necessary to partition the database document in future. Hence its a better choice to opt for NoSql database.
- Considering the CAP theorem , the availability and partition tolerance should work together, which suggests that a NoSql database can be a better option.

- Requirement C - User Profiles:- As mentioned, this module requires more flexibility. Flexibility for users to add/delete their bio, link/unlink other social media accounts.Link portfolios and add/delete pictures. The structure of the portfolio will evolve over time. Since this module requires high partition tolerance (for re-structuring the feed and profile) and high consistency(the profile needs to be consistent with changing/updating pictures and other social media links).
  It is better to keep this module in a Non-Relational database. I feel so because:-
  - Since the module requires high re-structuring of database-document, it should be schema-less to facilitate this. A NoSql database is the ultimate choice for schema-less and continuously changing document/structure.
  - Also the partition tolerance comes into picture when a user wants to restructure their profile/feed separately. In this case a NoSql database is a better option.
  - In terms of CAP theorem, the consistency and partition tolerance should work together in this case, which suggests that a NoSql database can be a better option.

According to me, the Biggest risk in the entire Platform is:-

- Risk : Managing the transactions during the Ticket Sales(Requirement-A). This should be synchronous and locked(locking will also be risky, what if two users click the seat selection option at the same instance? Which user should I favour for the locking system) at the same time. Even the optimistic locking fails here(checking the version variable, where the previous version is the same for both the users).
- Managing the risk : When two users are booking the same seat at the same time-stamp(Here two locks try to come-up simultaneously. We can tackle this race condition by using some database constraints. Like allocating the lock to the user based on his user-id, type of user. Message queues can be used for better processing of events. Let's say one user gets rejected (due to some custom set database constraint) , even this rejection of the seat shall be informed as quickly as possible so that he can select a different seat.