Architecture & Data Model & Trade-Offs

AWS Lambda is a good choice to serve API requests for this app due to the scope of this app. AWS Lambda takes care of automatic scaling and takes a lot of effort off of developers' hands. One important limitation is that it's not suitable for long-running executions. A lot of necessary features have been omitted in this assignment such as payment and credit check. Containerized backend servers may be required later when we develop more services that require higher traffic, or when we need some operations with long processing times.

For data storage, I chose Supabase, which is a branch of Postgres for good ACID-compliance support of relational databases. I took advantage of Postgres's pessimistic lock on a user to resolve race conditions. The potential downside of relational databases is performance, compared to non-relational databases, which are easily scalable horizontally.

Scaling & Performance

This app by nature is a read-heavy system where more requests are for fetching applications/transactions, compared to applying for cash advance or changing application states. One way to improve performance for a read heavy system is database indexing, and some of the candidate columns for indexing are:

- user_id of applications table
- application_id and type of transactions table

These columns are frequently accessed, but rarely changed. Another way to increase performance is sharding. Since applications and transactions are scoped to a user, we can use the user_id as a sharding key to shard databases into multiple copies. Storing a single user's information, and their relative application/transaction into the same shard will help distribute database load without performance overhead trying to access information across different shards.

Success Criteria

In order for this feature to be "successful", a user needs to be able to:

- apply for cash advance any time, 24/7
- see all their applications/transactions with the correct states
- be informed with correct error messages (not enough credit available, try again later, etc)

From technical point, this means:

- ensuring no downtime
- instant UI update, displaying loading icon rather than incorrect/outdated information
- covering all edge cases

Potential Extensions

- We can apply automatic credit limit increase for a user who has repaid a certain number of times within a short period of time. This motivates the user to repay quickly and consistently.
- Prioritizing users/applications with larger tips can motivate users to tip more. The larger the tip is, the faster the service will be.