Why Lazy Loading Works When @Transactional is Used

Background:
By default, JPA relationships can be loaded either *eagerly* (immediately) or *lazily* (when accessed). Eager loading loads all related entities immediately when the parent entity is fetched. Lazy loading delays the loading of related entities until they are actually accessed in code.

What Changed:

- We modified all entity relationships in the Account domain class to use FetchType.LAZY.
- We added @Transactional at the class level of AccountService, meaning all public methods in this service run within a transaction.

Why the Application Still Works with Lazy Loading:

- Lazy loading requires an **open Hibernate session**, which is only available during a **transaction**.
- Since all service methods are marked @Transactional, a session remains open while the code is accessing the lazy relationships.
- This allows the lazy properties to be fetched successfully when needed — even though they weren't loaded initially.

What Happens Without @Transactional:

- When the session is closed (which happens at the end of a non-transactional method), any attempt to access a lazily-loaded property throws a LazyInitializationException because the session is no longer available.
- This demonstrates the importance of having an active transaction when working with lazy-loaded data.

Conclusion:
We no longer need eager loading because:

- Eager loading fetches unnecessary data upfront and can hurt performance.
- Lazy loading improves efficiency and works reliably **as long as we access relationships within a transactional context**, like in AccountService.