UNIT OF WORK PATTERN:

* It’s an extension of Repository pattern.
* One unit of work involves all the operations(for eg:CRUD) in single transaction.

REPOSITORY PATTERN:

* A class defined as an entity with all the possible operations on that entity.
* Disadvantages: Problem arises when we add a repository for another entity. In that case, both repositories will generate and maintain their own instance of the DbContext. This may lead to issues in the future, since each DbContext will have its own in-memory list of changes of the records, of the entities, that are being added/updated/modified, in a single transaction/operation. In such a case, if the SaveChanges of one of the repository fails and other one succeeds, it will result in database in-consistency.

SOLUTION:

To avoid this, we will add an intermediate layer between the controller and the repository. This layer will act as a centralized store for all the repositories to receive the instance of the DbContext. This will ensure that, for a unit of transaction, that spans across multiple repositories, should either complete for all entities or should fail entirely, as all of them will share the same instance of the DbContext.

* Add a new class and this class generate repository instances and receive instance of DbContext. (Include all the operations related to Basket)
* Execute the save changes using the instance of this new class.