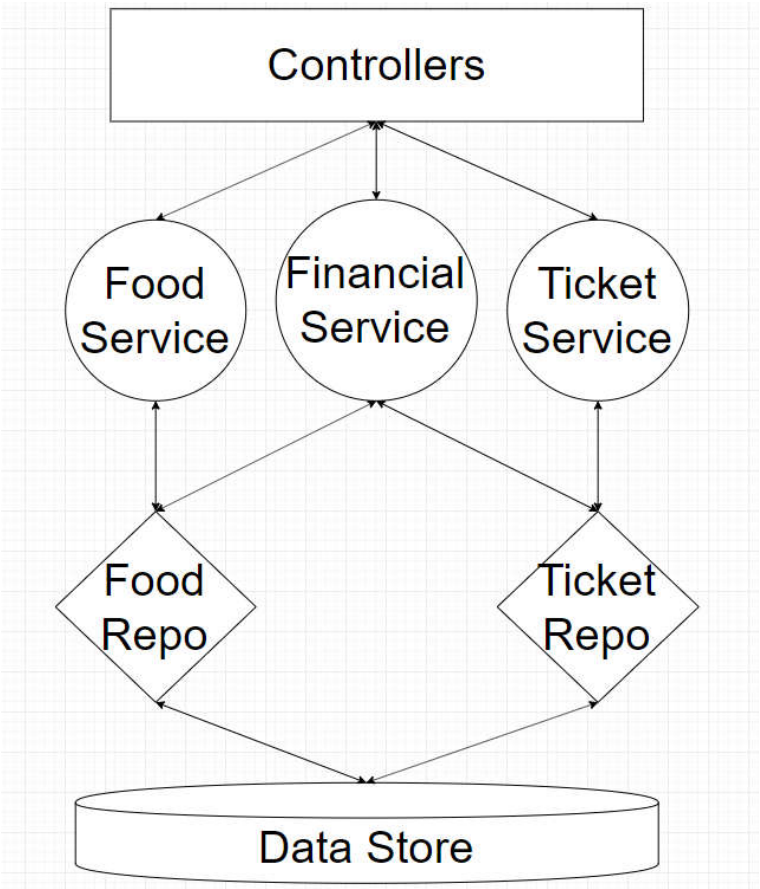
Repository-Service pattern



The diagram points out the major benefit to using this pattern: clear and consistent separation between the layers of the architecture. This gives us the ability to change one layer with minimal impact to the others, and with clear direction as to what each layer contains, we can do so quickly and with a minimum of code.

Drawback of This Pattern

The Repository-Service Pattern is a great pattern for situations in which you need to query for data from a complex data store or need some layers of separation between what happens for single models vs combinations of models. That said, it has one primary drawback that needs to be taken into account.

That drawback is simply this: it's a LOT of code, some of which might be totally unnecessary. The TicketService and FoodService classes from earlier do nothing except inherit from their corresponding Repositories. You could just as easily remove these classes and have the Repositories injected into the Controllers.

I personally will argue that any real-world app will be sufficiently complicated so as to warrant the additional Service layer, but it's not a hill I'll die on.

Summary

The Repository-Service Pattern is a great way to architect a real-world, complex application. Each of the layers (Repository and Service) have a well defined set of concerns and abilities, and by keeping the layers intact we can create an easily-modified, maintainable program architecture. There is one major drawback, but in my opinion it doesn't impact the pattern enough to stop using it.