My Thoughts and Notes

Reporting Structure:

- *"The number of reports is determined to be the number of directReports for an employee and all of their direct reports"*
    - Does this keep going all the way down the line?
    - Or does it just stop after an employee's direct reports and their direct reports?
    - **I eventually decided that the solution is recursive, I just found that wording to be a bit confusing as to how far the direct reports search should go.**
- Reporting Structure is very heavily tied to Employee, and doesn't persist any data, so there was no need to make any extra Services, Repositories, or Data files for it. Just the Controller and Model were fine.
    - Also added another couple methods to EmployeeService since it needs employee data to calculate the total number of reports.

Compensation:

- I ended up deciding to make a separate repository for Compensation because it was entirely it's own data type. In hindsight, I could've very easily just added another field to EmployeeContext as:
    - public DbSet<Compensation> Compensations {get; set;}
- I wouldn't have had to worry as much about keeping track of and maintaining another Data file, but I wanted to keep the original design going since a Compensation is an entirely different data type than an Employee.
- I did my best to mirror the implementation of Employee and adhere to MVC for Compensation. This involved making my own context, repository, service, and controller for Compensation.
    - Interfaces were also added where appropriate

Tests:

- The design for testing was well laid out in EmployeeControllerTests, and was super easy to follow.
- I created a new test file for each of the new controllers I created.
- There is 100% test coverage across all of the new APIs I created.
- There's also probably a better way of running the tests. Right now the tests are creating an httpClient and testServer for each test class. Maybe we could refactor that creation somewhere else so we only have to do it once