Some of my learnings from the course

- Fresh to test driven development.
- Mob vs navigator vs driver teamwork.
- Naming convention for readability, and implicit documentation.
- The idea of structuring code, ie. in other languages too.

Example 01: Creating indexes in MongoDb database

```
public class AccessControlEntryRepository : BaseRepository<AccessControlEntry>, IProvideAccessControlEntries
{
        protected override CreateIndexModel<AccessControlEntry> IndexModel
        {
            get
            {
                var indexBuilder = new IndexKeysDefinitionBuilder<AccessControlEntry>();
                return new CreateIndexModel<AccessControlEntry>(keys, options);
            }
        }
}

protected BaseRepository(IMongoCollection<TEntity> collection, ...)
{
            Entities = collection;
            EnsureIndexes();
}

private void EnsureIndexes()
{
            // Creates one index based on the IndexModel input
            Entities.Indexes.CreateOne(IndexModel);
}
```

Problem:
- Tight coupling & using getter.
- Only one index.
- Creates index upon class instantiation.

- Creating multiple indexes?
- Requires update to BaseRepository.

- Hence, cost of change is high.

Solution 01: Creating indexes in MongoDb database

```
public class AccessControlEntriesStartupTask : IStartupTask
    {
        ...
        public async Task ExecuteAsync(CancellationToken cancellationToken = default)
        {
            await _accessControlEntriesRepository.CreateDatabaseIndexes();
        }
    }
```

Solution:

- Decouple.
- Index created by startup task (ie. as a "query").
- Separation of concerns.

- Repository more agnostic
- More flexible.

```
public async Task CreateDatabaseIndexes()
{
    var indexBuilder = new IndexKeysDefinitionBuilder<AccessControlEntry>();
    var indexModelCreators = new List<CreateIndexModel<AccessControlEntry>>
    {
        new CreateIndexModel<AccessControlEntry>(keys, options),
        ...
    };
    await Entities.Indexes.CreateManyAsync(indexModelCreators);
}
```

- Expanding functionality -> startup task + repository method.
- Hence, low cost of change.

Example 02: Avoid using else and too many dots

```
public async Task<bool> InsertOneAccessView<TView>(TView view) where TView : BaseAccessView
{
    var existing = await Entities.OfType<TView>().Aggregate().Match(x => x.Name == view.Name).FirstOrDefaultAsync().ConfigureAwait(false);

    if (existing != null)
    {
        var filter = Builders<TView>.Filter.Where(x => x.Name == view.Name);
        view.Modified = DateTime.UtcNow;
        var replaced = await Entities.OfType<TView>().ReplaceOneAsync(filter, view).ConfigureAwait(false);
        return replaced.IsAcknowledged;
    }
    else
    {
        view.Created = DateTime.UtcNow;
        view.CreatedBy = "Access control service";
        await Entities.InsertOneAsync(view).ConfigureAwait(false);
    }

    return true;
}
```

Too many dots.

Unnecessary complication

Thanks for the attention!

- Any questions?