**Задание 1.** Разработать консольную программу, позволяющую принять от пользователя 3 строки, и записать эти 3 строки в 3 файла в папке, из которой запущена программа. 1-я строка в файл task1.txt, 2-я строка в файл taskN1.txt, 3-я строка в файл task\_1.txt.

Запись в файлы должна производиться в параллельных потоках, по завершении которых пользователю должно показаться сообщение Success, если все прошло без ошибок, либо Failure, если была хотя бы одна ошибка хотя бы в одном потоке.

Использование различных имен файлов для различных строк реализовать используя наследование, полиморфизм и паттерн "шаблонный метод".

Вывод запросов на ввод строки должен производиться объектами классов, реализующих полиморфное поведение. Результат пользовательского ввода также должен храниться в этих объектах. Запросы на ввод строки должны выводиться последовательно, один за другим, в виде "Input string 1: ", "Input string 2: ", "Input string 3: ". Пользователь должен увидеть сообщения о вводе в "натуральном" порядке (1, 2, 3) независимо от того, какой из объектов каким по очереди запрашивает ввод (т.е. если требования поменяются так, что 1-я строка должна записываться в файл taskN1.txt, а 2-я в файл task1.txt, то от программиста потребуется только поменять порядок работы с соответствующими объектами, но не менять код полиморфных классов).

Все произошедшие ошибки залогировать в файл task1.log, используя любую библиотеку для логирования в .NET. Библиотеку подключить используя NuGet. Для возможности тестирования логирования написать генератор случайных ошибок, генерирующий ошибки с вероятностью 1/3 вместо записи любой из строк в файл. Ошибка (случайная или реальная) при записи одной их строк не должна помешать записи других строк.

Подсказки и пожелания:

- При реализации многопоточности, сравнить методы Task.WaitAll, Task.WhenAll, Parallel.ForEach - какие из них гарантированно обеспечивают многопоточность? Какие только при определенных условиях? Какой подход в целом целесообразнее?

- Разобраться с понятием async метод, с работой await. Легче будет использовать версию C# >= 7.1 (там можно написать async Main метод).

- Популярные библиотеки для логирования - NLog и log4net. Можно выбрать любую другую.

- Нет большой необходимости создавать много проектов и файлов для решения этой задачи, в принципе позволительно обойтись один файлом / классом, с вложенными в него подклассами. Но можно и создать несколько независимых файлов, если так проще или удобнее.

- Номера (1, 2, 3), выводящиеся пользователю при запросе ввода, стоит хранить в статическом поле базового класса и увеличивать на 1 после каждого запроса ввода. В базовом же классе стоит реализовать почти весь функционал по запросу ввода, хранению ввода, записи в файл, генерации ошибки, логированию и т.д., только имена файлов стоит отдать на откуп наследникам.

- Наступление случайного события с вероятностью 1/3 можно симулировать, используя класс Random.

**Задание 2.**  Разрабатываем систему управления персоналом школы

Первая версии системы должна представлять собой веб сайт, который при открытии показывает список сотрудников школы в виде таблицы с заголовками. Информация о сотруднике состоит из следующих полей:

- Имя

- Отчество

- Фамилия

- Дата рождения

- Email

- Контактный телефон

Все эти поля должны выводиться в соответствующих колонках таблицы, за исключением Полного имени. Полное имя - это колонка, где выводятся Имя, Отчество и Фамилия в любом адекватном формате. Отдельных колонок для этих трех полей не требуется.

Также на странице должна располагаться кнопка, запускающая функционал создания нового сотрудника (в любом виде: поп-ап, новая страница, что угодно еще). Для создания сотрудника необходимо заполнить простую форму из 6 полей, обозначенных выше. Необходимо определить адекватные правила валидации этих полей (длина, формат, обязательно ли присутствие поля в заполненной форме и т.п.).

Первичная валидация должна проходить на стороне клиента (в браузере) и не пропускать данные на сервер, если они не валидны. При этом, если каким-то образом невалидные данные все же достигнут сервера, на сервере также должна происходить валидация и в случае невалидности данных в браузер должен уйти соответствующий ответ, позволяющий отобразить пользователю полную информацию о том, что пошло не так.

При успешном сохранении информации о новом сотруднике, пользователь должен возвращаться на страницу со списком сотрудников, где уже будет отображен вновь добавленный сотрудник.

На серверной стороне должна быть реализована типичная 3-х слойная архитектура - DAL, Business, Presentation. Каждый слой должен содержаться в отдельном проекте. Presentation должен представлять из себя ASP.NET MVC приложение (можно ASP.NET Core), business - набор интерфейсов и классов, реализующих исключительно логику приложения и вызывающих DAL для получения данных и сохранения изменений в них. Слои должны связываться друг с другом через Dependency Injection (любой инструмент, можно встроенный в ASP.NET Core, можно сторонний, подключенный через NuGet).

При необходимости использовать различные классы для моделей в каждом из слоев, можно воспользоваться библиотекой AutoMapper (через NuGet) для упрощения кода, перебрасывающего данные из объектов одного типа в похожие объекты другого типа.

DAL должен быть реализован с использованием Entity Framework, Code First подход. Кроме модели данных, в DAL должна присутствовать, как минимум, первичная миграция для создания необходимых таблиц в базе данных. Кроме того, необходимо выбрать стратегию обновления базы данных, не подразумевающую регулярного пересоздания базы на старте приложения, чтобы не допустить потерю данных при выпуске последующих версий приложения.

Реализовать подход к использованию контекста Entity Framework, при котором сохранение изменений производится единожды по окончании обработки запроса (паттерн Unit Of Work). Т.е. не вызывается SaveChanges в каждом методе бизнес слоя / репозиториев (использование репозиториев не обязательно), а вызывается один раз в логике, запускающейся после каждого HTTP запроса. Для этого, в частности, необходимо регистрировать контекст в DI контейнере как "один на http запрос".

Данные должны сохраняться в SQL базу данных. Проще всего использовать Microsoft SQL Server Developer Edition, последнюю версию.

Колонки в таблицах в базе данных должны по максимуму контролировать валидность данных, попадающих в нее. Т.е. не должны допускаться строки большей, чем допустимо, длины, не должно быть возможно не заполнить обязательное для заполнения поле и т.п. - не стоит "перегибать палку" и пытаться провалидировать все; только самые базовые валидации, не требующие особых усилий на стороне базы.