Тестовое задание для Web C# Developer

**Проблема бизнеса:**

В компанию поступают заказы от клиентов в разных форматах и из разных источников. Необходимо автоматизировать приём заказов в существующую ERP систему.

Форматы:  xlsx, xml, csv, json

Источники: email, FTP, web API клиента компании, Telegram

**Постановка задачи:**

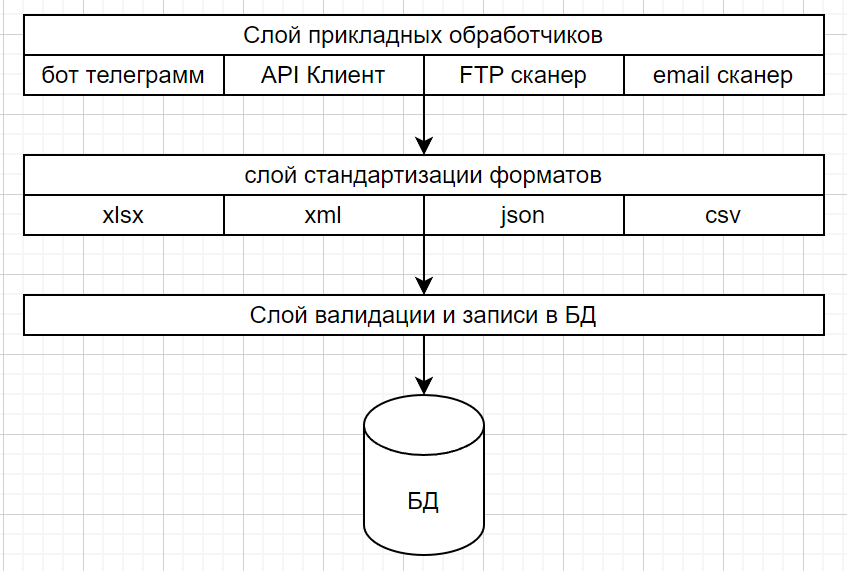
1. Описать реализацию задачи, перечень необходимых модулей, связи между ними, выбор технологий для каждого модуля. Детализация на усмотрение. Схемы, изображения приветствуются.
2. Реализовать на выбор один из модулей, либо несколько с целью демонстрации их коммуникации и внутреннего устройства.

**Пожелания к реализации:**

1. C#, .Net 6
2. Следование соглашениям о наименовании для C#.
3. Хранить данные в БД. Тип баз подбирать в соответствии с задачей.
4. Использовать технологию внедрения зависимостей.
5. Unit тесты. Использовать NUnit либо XUnit

Система контроля версий: GIT

Архитектура системы



1. **Прикладные обработчики** занимаются обнаружением поступивших данных. Их задача обнаружить поступившую информацию и передать ее в следующий слой. В этом слое так же имеет смысл проверить, все ли нужные поля документа заполнены. Каждый из обработчиков может быть реализован как самостоятельной программой или сервисом, так и библиотекой. Кроме списка товаров в заказе обработчики должны передать информацию о том, кто прислал данные (ник в телеграмме, e-mail отправителя, папку на FTP). В идеале каждый обработчик должен отчитаться об успешной или ошибочной операции, но не понятно, как это сделать с FTP.
2. **Слой стандартизации форматов** может быть реализован отдельным сервисом, библиотекой или контроллером. Задача этого слоя превратить документ и вспомогательные данные в объект данных.  
   *Технологии:* предполагаю данные передавать через POST запросы в виде JSON, и, если есть файлы, то добавить их к POST запросу и разбирать контроллерами.
   1. JSON разбирается контроллером автоматически
   2. XML может быть разобран через XML Serializer - <https://learn.microsoft.com/ru-ru/dotnet/api/system.xml.serialization.xmlserializer?view=net-6.0>
   3. Xlsx можно разобрать с помощью EPPlus
   4. Csv можно разобрать через CsvHelper
3. **Слой валидации** должен проверить данные на соответствие тому, что уже лежит в БД и либо отдать данные на запись, либо вернуть ошибку с подробным описанием на предыдущий слой.
4. **База данных** должна быть реляционной по следующим причинам:
   1. Должна сохраняться структура заказа и проверка данных на валидность
   2. В задаче четко видна структура сущностей реляционной БД: товар-заказ-клиент

Конкретную СУБД следует выбирать из тех, с которыми работали разработчики и тех, что уже куплены, при условии, что БД удовлетворяет требованиям к производительности. С учетом фразы из ТЗ **«Необходимо автоматизировать приём заказов в существующую ERP систему.»** своя база данных вообще может отсутствовать, а данные будут передаваться в сервис ERP системы.

Эффективнее всего сделать базу данных в виде интерфейса с одной реализацией, чтобы потом можно было легко ее заменять, в том числе и на сервис ERP системы.

Реализация:

1. **Слой прикладных обработчиков** в реализации выполняет Swagger – приемлемая замена Web API клиента на этапе тестирования
2. **Слой стандартизации форматов** реализован в виде контроллера, поскольку он автоматически принимает данные в JSON. В реализации исправлен контроллер OrderController. В нем создается внедрение зависимостей и данные пропускаются через валидацию и отправляются в БД.
3. **Слой валидации** должен проверить данные на соответствие тому, что уже лежит в БД и либо записать данные, либо вернуть ошибку с подробным описанием на предыдущий слой. Этот слой можно разделить на два слоя – собственно валидацию и запись.
4. **База данных** сделана на интерфейсе IDatabaseAccess и одной его реализации через EF
5. **Тесты.** Сделаны всего два теста для метода валидации – как пример умения писать тесты и использовать Moq, если нужно, то готов все написанное покрыть тестами.