# Intro

## Про задания

Задания под \* не являются обязательными и не влияют на итоговую оценку. Их следует выполнять только в случае, если останется свободное время.

Если формулировка задания вам не понятна, напишите об этом инструктору (но лучше в общий чат группы, чтобы инструктору не повторять несколько раз одно и то же).

Если вы нашли ошибку, опечатку в задании, пожалуйста, напишите об этом инструктору.

## Про оформление

Для работы мы будем использовать публичный репозиторий GitHub.

<https://github.com/>

Именовать репозиторий следует так:

EPAM\_Ext\_Lab\_<Квартал старта группы>\_<Год>\_<Имя\_ Фамилия>.

Например, EPAM\_Ext\_Lab\_Q2\_2018\_Ivan\_Ivanov.

Что касается именования проектов решений, рекомендую нумеровать их по порядковому номеру задания.

## Документирование кода

Каждый метод должен иметь описание (<summary>). Поскольку теги описания генерируются автоматически при наборе «///», то рекомендуется озаботиться о документировании кода после реализации всех сигнатур методов.

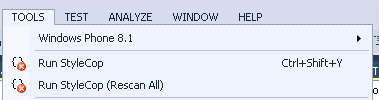
Комментарии к коду не приветствуются, поскольку мы руководствуемся принципов самодокументируемости кода (Код должен быть прост и понятен без комментариев. Если код требует комментариев, есть смысл подумать о другой реализации.). Понятно, что есть исключения (сложная логика кода; интуитивно непонятное решение, базирующееся на правилах бизнеса).

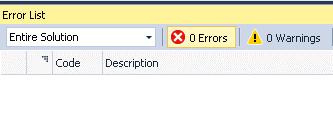
Однако, если есть какой-то вопрос по определенной строке, то допустимо задать его в комментарии.

## Про отправку работ на проверку

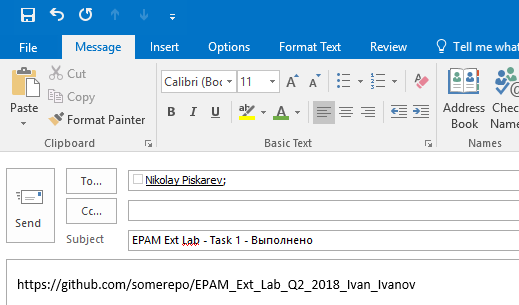
После того, как вы считаете, что выполнили все требования задания ЕЩЕ РАЗ проверьте всю функциональность вашего решения, проверьте, что все тесты завершились успешно. Если решение будет падать при запуске, такая работа будет получать сразу **0** баллов.

Запустите проверку кода StyleCop. Никаких предупреждений быть не должно.





После необходимо написать письмо вашему инструктору:



Обязательно вставьте ссылку на ваш репозиторий.

По дате письма можно будет понять, когда вы выполнили задание.

После отправки письма редактировать код в репозитории нельзя (только санкционировав редактирование с инструктором)\*. В противном случае, работа будет оцениваться в **0** баллов.

*\* Ограничение вынужденное, поскольку были студенты, которые, чтобы успеть в дедлайн, просто отправляли письма, не заливая код в репозиторий.*

## Критерии оценки

Система оценки работ трехбальная и довольно проста:

2 балла – все требования задания выполнены, существенный нареканий нет (по стилю кода, по документированию и др.).

1 балл – все требования задания выполнены, но можно кое-что улучшить. Есть минорные баги (не влияющие на основную функциональность решения). Обычно такие задания отправляются на доработку, если есть в этом смысл. Самая распространенная оценка.

0 баллов – не выполнено одно и более требований задания, критические и мажорные баги (неотловленное падение приложения или ошибки при каком-либо действии пользователя (ожидаемом или нет)), пропуск дедлайна.

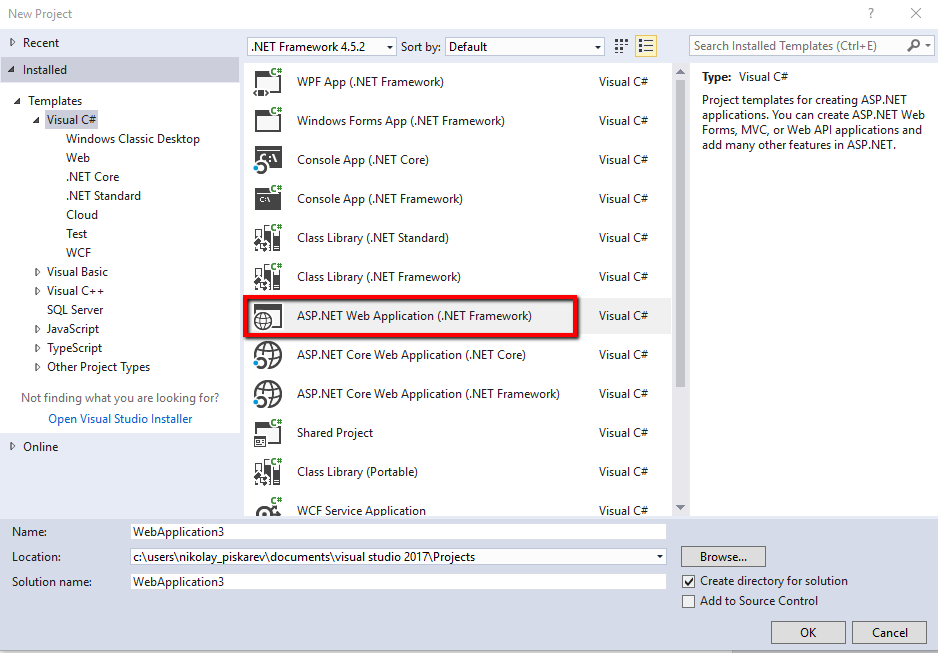
При наличии у студента **трех** несданных работ (нулей) – отчисление из группы.

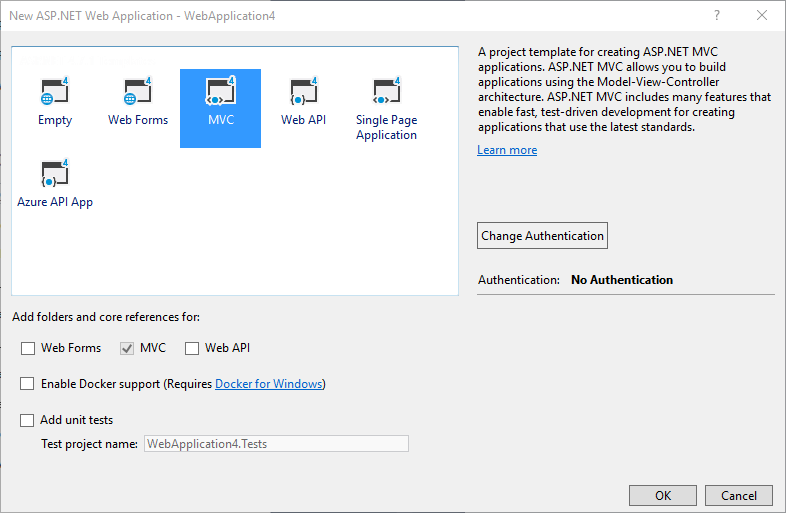
**В связи с этим, если вы участвуете в каких-либо мероприятиях, болеете либо произошли какие-либо другие ситуации, которые мешают вам сдать задание вовремя – напишите инструктору и попросите продлить дедлайн.**

Форс-мажорность ситуации определяются инструктором по его личному убеждению.

# Задание 1. Introduction to ASP.Net MVC

1. Создать Web-приложение из шаблона MVC.

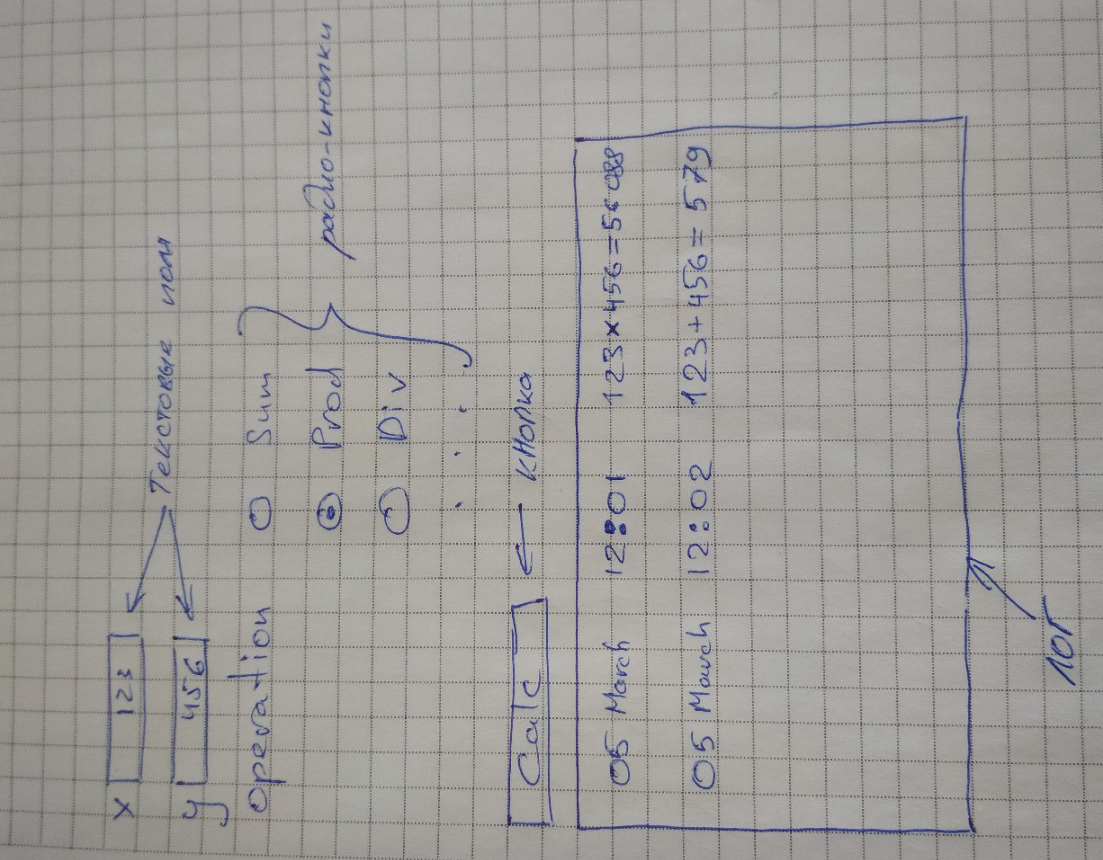




1. Создать контроллер CalculatorController с методами:

* Sum(int x, int y) – сумма двух слагаемых
* Prod(int x, int y) – умножение двух слагаемых
* И другие операции по желанию.

1. Создать представление Calculartor.cshtml. Разместить на нем два текстовых поля с лейблами «x» и «y» для ввода соответствующих переменных. Также разместить список радио-кнопок для выбора текущего действия (см. рисунок) и кнопку «Calc». По нажатию на кнопку должно производиться вычисление соответствующей операции над переменными, введенными в поля «x» и «y».
2. Результат операции должен записаться в лог в виде Дата Время X Операция Y = Результат (см. рисунок).
3. Удалить все неиспользуемые экшены и контроллеры.

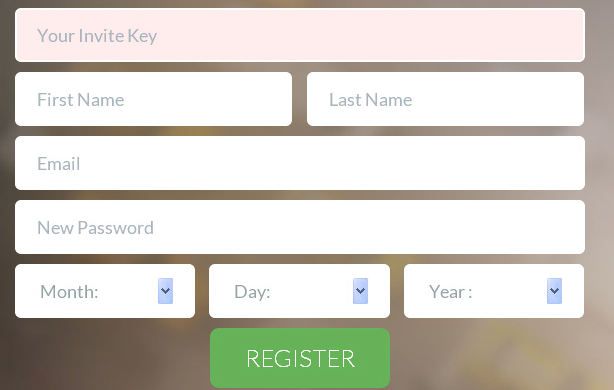


# Задание 2. HTML, CSS

Необходимо сверстать макет будущего сайта.

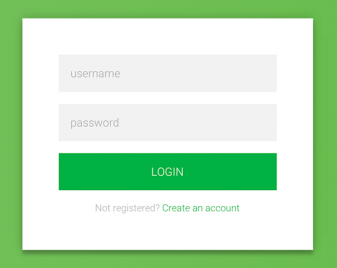
## 2.1. Форма регистрации

Несколько текстовых полей, расположенных друг под другом и кнопка. Для почты и пароля использовать специальный тип у input.



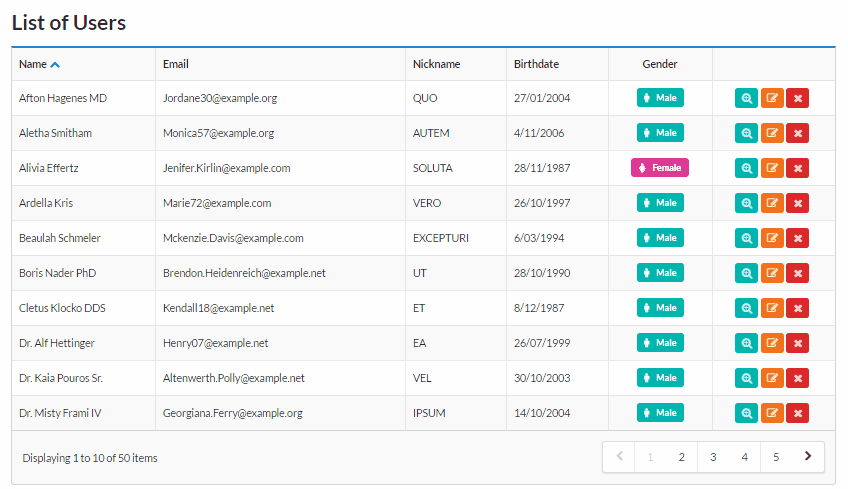
## 2.2. Форма авторизации

Два текстовых поля, расположенных друг под другом и кнопка. Для почты и пароля использовать специальный тип у input.



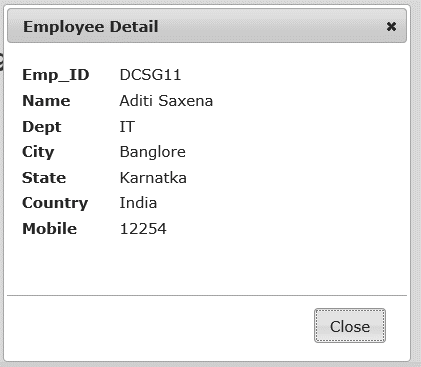
## 2.3. Страница со списком

В качестве элементов колонок могут быть кнопки, выпадающие списки и изображения. Изображения должны иметь фиксированную величину. Кнопки должны быть расположены в ряд.



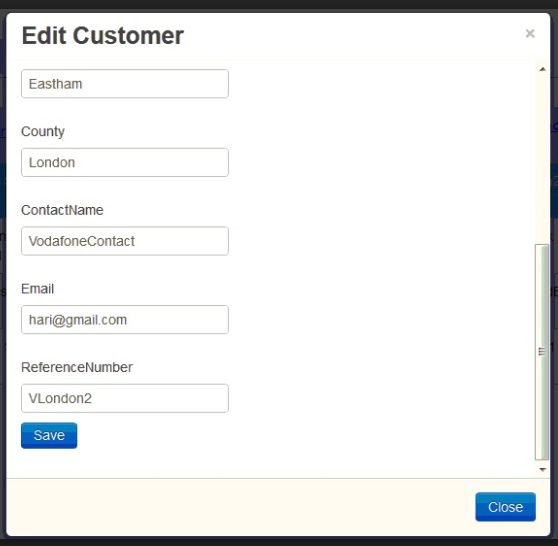
## 2.4. Страница с показом детальной информации

Только текст без возможности изменить что-либо.



## 2.5. Страница с редактированием.

Текстовые поля с возможностью редактирования.

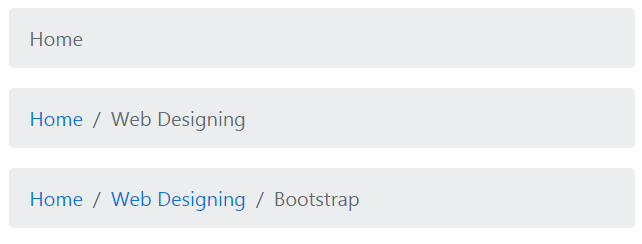


## 2.6. Модальное окно

См. предыдущие изображения (2.4 и 2.5).

## 2.7. Хлебные крошки



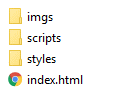


## Требования к реализации

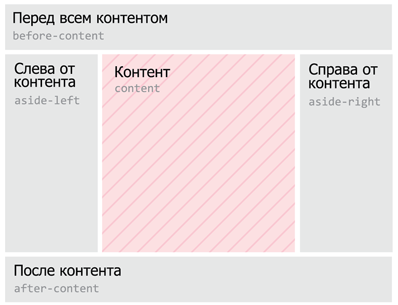
Все страницы должны иметь единый стиль (цвет кнопок, ориентация контролов на странице, шрифт итп).

Скачивание из интернета готовых макетов не допускается.

Итоговое решение должно содержать следующую структуру: для картинок, скриптов (если есть) и стилей – собственные папки, необходимое количество html файлов с макетами страниц. Пустые папки создавать не нужно.



1. Отверстать шаблон каркаса верстки и разместить в нем некоторые реализованные элементы (форма авторизации, список, хлебные крошки, какой-то текстовый блок).
2. \*Предусмотреть изменение местоположения элементов в случае узкого экрана (телефон/планшет)



1. Посмотреть, как выглядят макеты в Internet Explorer и других браузерах.

# Задание 3. Basics of C#

Задания ниже выполнить в виде консольного приложения.

Реализация заданий должна находиться в отдельном классе, отличном от класса Program.

При запуске консоли пользователю должна быть предоставлена возможность выбрать запускаемое задание. После завершения выполнения задания необходимо спросить у пользователя, хочет ли он **повторить текущее** задание, либо **выбрать другое**, либо **выйти** из приложения.

Для удобства тестирования рекомендуется в качестве тестовых значений использовать случайно сгенерированные входные значения, ограниченные определенными рамками.

*Например, если нужно вывести пирамиду из звездочек, то можно ограничить число рядов от 1 до 50.*

*В случае необходимости ввести строку, необходимо предложить пользователю использовать строку по умолчанию или ввести собственную строку.*

Все задания должны корректно обрабатывать любые возникшие в процессе работы ошибки. Все некорректные действия пользователя должны отлавливаться и на них должны выводиться понятные сообщения об ошибке.

*Например, если пользователь пытается ввести отрицательное число в размерность массива, ему нужно указать, что «Размерность может быть только больше 0!».*

В случае непредвиденной ошибки пользователю должно выводиться что-то вроде «Извините, произошла ошибка». Сам текст ошибки и её stack trace необходимо записать в лог (отдельный файл).

## Задание 3.1

Написать программу, которая определяет площадь прямоугольника со сторонами a и b. Если пользователь вводит некорректные значения (отрицательные, или 0), должно выдаваться сообщение об ошибке. Возможность ввода пользователем строки вида «абвгд», или нецелых чисел игнорировать.

## Задание 3.2

Написать программу, которая запрашивает с клавиатуры число N и выводит на экран следующее «изображение», состоящее из N строк:



## Задание 3.3

Написать программу, которая запрашивает с клавиатуры число N и выводит на экран следующее «изображение», состоящее из N строк:



## Задание 3.4

Написать программу, которая запрашивает с клавиатуры число N и выводит на экран следующее «изображение», состоящее из N треугольников:

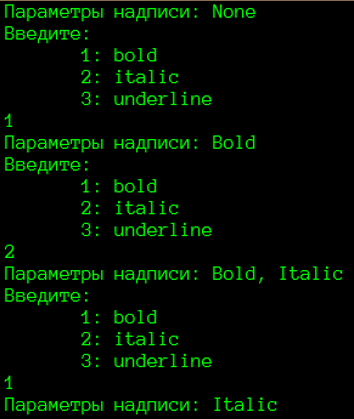


## Задание 3.5

Если выписать все натуральные числа меньше 10, кратные 3, или 5, то получим 3, 5, 6 и 9. Сумма этих чисел будет равна 23. Напишите программу, которая выводит на экран сумму всех чисел меньше 1000, кратных 3, или 5.

## Задание 3.6

Для выделения текстовой надписи можно использовать выделение жирным, курсивом и подчёркиванием. Предложите способ хранения информации о выделении надписи и напишите программу, которая позволяет назначать и удалять текстовой надписи выделение:



## Задание 3.7

Написать программу, которая генерирует случайным образом элементы массива (число элементов в массиве и их тип определяются разработчиком), определяет для него максимальное и минимальное значения, сортирует массив и выводит полученный результат на экран.

*Примечание: LINQ запросы и готовые функции языка (Sort, Max и т.д.) использовать в данном задании запрещается.*

## Задание 3.8

Написать программу, которая заменяет все положительные элементы в трёхмерном массиве на нули. Число элементов в массиве и их тип определяются разработчиком.

## Задание 3.9

Написать программу, которая определяет сумму неотрицательных элементов в одномерном массиве. Число элементов в массиве и их тип определяются разработчиком.

## Задание 3.10

Элемент двумерного массива считается стоящим на чётной позиции, если сумма номеров его позиций по обеим размерностям является чётным числом (например, [1,1] – чётная позиция, а [1,2] - нет). Определить сумму элементов массива, стоящих на чётных позициях.

## Задание 3.11

Написать программу, которая определяет среднюю длину слова во введенной текстовой строке. Учесть, что символы пунктуации на длину слов влиять не должны. Регулярные выражения не использовать. И не пытайтесь прописать все ручками. Используйте стандартные методы класса String.

## Задание 3.12

Написать программу, которая удваивает в первой введенной строки все символы, принадлежащие второй введенной строке.

**Пример:**

Введите первую строку: написать программу, которая

Введите вторую строку: описание

Результирующая строка: ннааппииссаать ппроограамму, коотоораая

## Задание 3.13

Проведите сравнительный анализ скорости работы классов String и StringBuilder для операции сложения строк:

string str = "";

StringBuilder sb = new StringBuilder();

int N = 100;

for (int i = 0; i < N; i++)

{

str += "\*";

}

for (int i = 0; i < N; i++)

{

sb.Append("\*");

}

# Задание 4. Basics of OOP

## 4.1 Нарисовать диаграмму классов

Тебе была выдана тема финального проекта. Необходимо разработать диаграмму классов для указанной области.

## 4.2 Реализовать иерархию классов

По разработанной в предыдущем задании диаграмме реализовать иерархию классов в отдельном проекте (который будет заготовкой будущего DAL).

Предусмотреть возможность вынесение определенных сущностей в папки и подпапки.

# Задание 5. Generics and Collections

Используя наработки по DAL реализовать следующий интерфейс:

interface IBaseRepository<T> where T : class, new()

{

T Get(int id);

List<T> GetAll();

bool Save(T entity);

bool Delete(int id);

}

В качестве хранилища данных использовать списки.

Продемонстрировать работу всех методов в классе с тестами.

# Задание 6. Delegates and events

## Задание 6.1

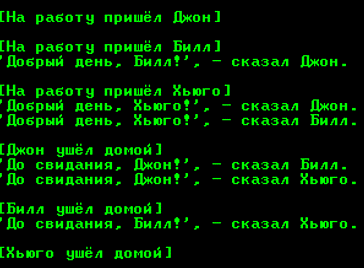
Написать программу, выполняющую сортировку массива строк по возрастанию длины. Если строки состоят из равного числа символов, их следует отсортировать по алфавиту. Реализовать метод сравнения строк отдельным методом, передаваемым в сортировку через делегат*.*

## Задание 6.2

Написать программу, описывающую небольшой офис, в котором работают сотрудники – объекты класса Person, обладающие полем имя (Name). Каждый из сотрудников содержит пару методов: приветствие сотрудника, пришедшего на работу (принимает в качестве аргументов объект сотрудника и время его прихода) и прощание с ним (принимает только объект сотрудника). В зависимости от времени суток, приветствие может быть различным: до 12 часов – «Доброе утро», с 12 до 17 – «Добрый день», начиная с 17 часов – «Добрый вечер». Каждый раз при входе очередного сотрудника в офис, все пришедшие ранее его приветствуют. При уходе сотрудника домой с ним также прощаются все присутствующие. Вызов процедуры приветствия/прощания производить через групповые делегаты. Факт прихода и ухода сотрудника отслеживается через генерируемые им события. Событие прихода описывается делегатом, передающим в числе параметров наследника EventArgs, явно содержащего поле с временем прихода.

Продемонстрировать работу офиса при последовательном приходе и уходе сотрудников.

Пример:



# Задание 7. Extensions and LINQ

## Задание 7.1

Напишите расширяющий метод, который определяет сумму элементов массива*.*

## Задание 7.2

Напишите расширяющий метод, который определяет, является ли строка положительным целым числом. Методы Parse и TryParse не использовать.

## Задание 7.3

Написать методы поиска элемента в массиве (например, поиск всех положительных элементов в массиве) в виде:

1. Метода, реализующего поиск напрямую;
2. Метода, которому условие поиска передаётся через делегат;
3. Метода, которому условие поиска передаётся через делегат в виде анонимного метода;
4. Метода, которому условие поиска передаётся через делегат в виде лямбда-выражения;
5. LINQ-выражения

Сравнить скорость выполнения вычислений.

# Задание 8. SQL

## Цель выполнения

Получить практические навыки работы с Query Analyzer для выполнения операторов языка Transact SQL. Освоить применение различных команд DML для получения, изменения и удаления данных в БД. Научиться разрабатывать запросы с применением следующих операторов и конструкций языка Transact SQL:

* DISTINCT, ORDER BY, IN, BETWEEN, LIKE, EXISTS, CASE;
* агрегатных функций SUM, COUNT и предложений GROUP BY и HAVING;
* соединений таблиц и самосоединений, конструкций Inner и Outer JOIN;
* простых и коррелированных подзапросов;

Научиться разрабатывать хранимые процедуры и функции и вызывать их. Научиться применять и использовать курсоры

## Ограничения

При выполнения задания использовать требования, изложенные в руководстве «Database Modeling Guidelines» EPAM Systems

## Список заданий

1. **Работа с типами данных Date, NULL значениями, трехзначная логика. Возвращение определенных значений в результатах запроса в зависимости от полученных первоначальных значений результата запроса. Высветка в результатах запроса только определенных колонок.**
   1. Выбрать в таблице Orders заказы, которые были доставлены после 6 мая 1998 года (колонка ShippedDate) включительно и которые доставлены с ShipVia >= 2. Формат указания даты должен быть верным при любых региональных настройках, согласно требованиям статьи “Writing International Transact-SQL Statements” в Books Online раздел “Accessing and Changing Relational Data Overview”. Этот метод использовать далее для всех заданий. Запрос должен высвечивать только колонки OrderID, ShippedDate и ShipVia.   
      Пояснить почему сюда не попали заказы с NULL-ом в колонке ShippedDate.
   2. Написать запрос, который выводит только недоставленные заказы из таблицы Orders. В результатах запроса высвечивать для колонки ShippedDate вместо значений NULL строку ‘Not Shipped’ – использовать системную функцию CASЕ. Запрос должен высвечивать только колонки OrderID и ShippedDate.
   3. Выбрать в таблице Orders заказы, которые были доставлены после 6 мая 1998 года (ShippedDate) не включая эту дату или которые еще не доставлены. В запросе должны высвечиваться только колонки OrderID (переименовать в Order Number) и ShippedDate (переименовать в Shipped Date). В результатах запроса высвечивать для колонки ShippedDate вместо значений NULL строку ‘Not Shipped’, для остальных значений высвечивать дату в формате по умолчанию.
2. **Использование операторов IN, DISTINCT, ORDER BY, NOT**
   1. Выбрать из таблицы Customers всех заказчиков, проживающих в USA и Canada. Запрос сделать с только помощью оператора IN. Высвечивать колонки с именем пользователя и названием страны в результатах запроса. Упорядочить результаты запроса по имени заказчиков и по месту проживания.
   2. Выбрать из таблицы Customers всех заказчиков, не проживающих в USA и Canada. Запрос сделать с помощью оператора IN. Высвечивать колонки с именем пользователя и названием страны в результатах запроса. Упорядочить результаты запроса по имени заказчиков.
   3. Выбрать из таблицы Customers все страны, в которых проживают заказчики. Страна должна быть упомянута только один раз и список отсортирован по убыванию. Не использовать предложение GROUP BY. Высвечивать только одну колонку в результатах запроса.
3. **Использование оператора BETWEEN, DISTINCT**
   1. Выбрать все заказы (OrderID) из таблицы Order Details (заказы не должны повторяться), где встречаются продукты с количеством от 3 до 10 включительно – это колонка Quantity в таблице Order Details. Использовать оператор BETWEEN. Запрос должен высвечивать только колонку OrderID.
   2. Выбрать всех заказчиков из таблицы Customers, у которых название страны начинается на буквы из диапазона b и g. Использовать оператор BETWEEN. Проверить, что в результаты запроса попадает Germany. Запрос должен высвечивать только колонки CustomerID и Country и отсортирован по Country.
   3. Выбрать всех заказчиков из таблицы Customers, у которых название страны начинается на буквы из диапазона b и g, не используя оператор BETWEEN. С помощью опции “Execution Plan” определить какой запрос предпочтительнее 3.2 или 3.3 – для этого надо ввести в скрипт выполнение текстового Execution Plan-a для двух этих запросов, результаты выполнения Execution Plan надо ввести в скрипт в виде комментария и по их результатам дать ответ на вопрос – по какому параметру было проведено сравнение. Запрос должен высвечивать только колонки CustomerID и Country и отсортирован по Country.
4. **Использование оператора LIKE**
   1. В таблице Products найти все продукты (колонка ProductName), где встречается подстрока 'chocolade'. Известно, что в подстроке 'chocolade' может быть изменена одна буква 'c' в середине - найти все продукты, которые удовлетворяют этому условию. Подсказка: результаты запроса должны высвечивать 2 строки.
5. **Использование агрегатных функций (SUM, COUNT)**
   1. Найти общую сумму всех заказов из таблицы Order Details с учетом количества закупленных товаров и скидок по ним. Результат округлить до сотых и высветить в стиле 1 для типа данных money. Скидка (колонка Discount) составляет процент из стоимости для данного товара. Для определения действительной цены на проданный продукт надо вычесть скидку из указанной в колонке UnitPrice цены. Результатом запроса должна быть одна запись с одной колонкой с названием колонки 'Totals'.
   2. По таблице Orders найти количество заказов, которые еще не были доставлены (т.е. в колонке ShippedDate нет значения даты доставки). Использовать при этом запросе только оператор COUNT. Не использовать предложения WHERE и GROUP.
   3. По таблице Orders найти количество различных покупателей (CustomerID), сделавших заказы. Использовать функцию COUNT и не использовать предложения WHERE и GROUP.
6. **Явное соединение таблиц, самосоединения, использование агрегатных функций и предложений GROUP BY и HAVING** 
   1. По таблице Orders найти количество заказов с группировкой по годам. В результатах запроса надо высвечивать две колонки c названиями Year и Total. Написать проверочный запрос, который вычисляет количество всех заказов.
   2. По таблице Orders найти количество заказов, cделанных каждым продавцом. Заказ для указанного продавца – это любая запись в таблице Orders, где в колонке EmployeeID задано значение для данного продавца. В результатах запроса надо высвечивать колонку с именем продавца (Должно высвечиваться имя полученное конкатенацией LastName & FirstName. Эта строка LastName & FirstName должна быть получена отдельным запросом в колонке основного запроса. Также основной запрос должен использовать группировку по EmployeeID.) с названием колонки ‘Seller’ и колонку c количеством заказов высвечивать с названием 'Amount'. Результаты запроса должны быть упорядочены по убыванию количества заказов.
   3. По таблице Orders найти количество заказов, cделанных каждым продавцом и для каждого покупателя. Необходимо определить это только для заказов сделанных в 1998 году. В результатах запроса надо высвечивать колонку с именем продавца (название колонки ‘Seller’), колонку с именем покупателя (название колонки ‘Customer’) и колонку c количеством заказов высвечивать с названием 'Amount'. В запросе необходимо использовать специальный оператор языка T-SQL для работы с выражением GROUP (Этот же оператор поможет выводить строку “ALL” в результатах запроса). ***Группировки должны быть сделаны по ID продавца и покупателя.*** Результаты запроса должны быть упорядочены по продавцу, покупателю и по убыванию количества продаж. В результатах должна быть сводная информация по продажам. Т.е. в резульирующем наборе должны присутствовать дополнительно к информации о продажах продавца для каждого покупателя следующие строчки:

**Seller Customer Amount**  
ALL ALL <общее число продаж>  
<имя> ALL <число продаж для данного продавца>  
ALL <имя> <число продаж для данного покупателя>  
<имя> <имя> <число продаж данного продавца для даннного покупателя>

* 1. Найти покупателей и продавцов, которые живут в одном городе. Если в городе живут только один или несколько продавцов или только один или несколько покупателей, то информация о таких покупателя и продавцах не должна попадать в результирующий набор. Не использовать конструкцию JOIN. В результатах запроса необходимо вывести следующие заголовки для результатов запроса: ‘Person’, ‘Type’ (здесь надо выводить строку ‘Customer’ или ‘Seller’ в завимости от типа записи), ‘City’. Отсортировать результаты запроса по колонке ‘City’ и по ‘Person’.
  2. Найти всех покупателей, которые живут в одном городе. В запросе использовать соединение таблицы Customers c собой - самосоединение. Высветить колонки CustomerID и City. Запрос не должен высвечивать дублируемые записи. Для проверки написать запрос, который высвечивает города, которые встречаются более одного раза в таблице Customers. Это позволит проверить правильность запроса.
  3. По таблице Employees найти для каждого продавца его руководителя, т.е. кому он делает репорты. Высветить колонки с именами 'User Name' (LastName) и 'Boss'. В колонках должны быть высвечены имена из колонки LastName. Высвечены ли все продавцы в этом запросе?

1. **Использование Inner JOIN**
   1. Определить продавцов, которые обслуживают регион 'Western' (таблица Region). Результаты запроса должны высвечивать два поля: 'LastName' продавца и название обслуживаемой территории ('TerritoryDescription' из таблицы Territories). Запрос должен использовать JOIN в предложении FROM. Для определения связей между таблицами Employees и Territories надо использовать графические диаграммы для базы Northwind.
2. **Использование Outer JOIN**
   1. Высветить в результатах запроса имена всех заказчиков из таблицы Customers и суммарное количество их заказов из таблицы Orders. Принять во внимание, что у некоторых заказчиков нет заказов, но они также должны быть выведены в результатах запроса. Упорядочить результаты запроса по возрастанию количества заказов.
3. **Использование подзапросов**
   1. Высветить всех поставщиков колонка CompanyName в таблице Suppliers, у которых нет хотя бы одного продукта на складе (UnitsInStock в таблице Products равно 0). Использовать вложенный SELECT для этого запроса с использованием оператора IN. Можно ли использовать вместо оператора IN оператор '=' ?
4. **Коррелированный запрос**
   1. Высветить всех продавцов, которые имеют более 150 заказов. Использовать вложенный коррелированный SELECT.
5. **Использование EXISTS**
   1. Высветить всех заказчиков (таблица Customers), которые не имеют ни одного заказа (подзапрос по таблице Orders). Использовать коррелированный SELECT и оператор EXISTS.
6. **Использование строковых функций**
   1. Для формирования алфавитного указателя Employees высветить из таблицы Employees список только тех букв алфавита, с которых начинаются фамилии Employees (колонка LastName ) из этой таблицы. Алфавитный список должен быть отсортирован по возрастанию.
7. **Разработка функций и процедур**
   1. Написать процедуру, которая возвращает самый крупный заказ для каждого из продавцов за определенный год. В результатах не может быть несколько заказов одного продавца, должен быть только один и самый крупный. В результатах запроса должны быть выведены следующие колонки: колонка с именем и фамилией продавца (FirstName и LastName – пример: Nancy Davolio), номер заказа и его стоимость. В запросе надо учитывать Discount при продаже товаров. Процедуре передается год, за который надо сделать отчет, и количество возвращаемых записей. Результаты запроса должны быть упорядочены по убыванию суммы заказа. Процедура должна быть реализована с использованием оператора SELECT и БЕЗ ИСПОЛЬЗОВАНИЯ КУРСОРОВ. Название функции соответственно GreatestOrders. Необходимо продемонстрировать использование этих процедур. Также помимо демонстрации вызовов процедур в скрипте Query.sql надо написать отдельный ДОПОЛНИТЕЛЬНЫЙ проверочный запрос для тестирования правильности работы процедуры GreatestOrders. Проверочный запрос должен выводить в удобном для сравнения с результатами работы процедур виде для определенного продавца для всех его заказов за определенный указанный год в результатах следующие колонки: имя продавца, номер заказа, сумму заказа. Проверочный запрос не должен повторять запрос, написанный в процедуре, - он должен выполнять только то, что описано в требованиях по нему.  
      ВСЕ ЗАПРОСЫ ПО ВЫЗОВУ ПРОЦЕДУР ДОЛЖНЫ БЫТЬ НАПИСАНЫ В ФАЙЛЕ Query.sql – см. пояснение ниже в разделе «Требования к оформлению».
   2. Написать процедуру, которая возвращает заказы в таблице Orders, согласно указанному сроку доставки в днях (разница между OrderDate и ShippedDate). В результатах должны быть возвращены заказы, срок которых превышает переданное значение или еще недоставленные заказы. Значению по умолчанию для передаваемого срока 35 дней. Название процедуры ShippedOrdersDiff. Процедура должна высвечивать следующие колонки: OrderID, OrderDate, ShippedDate, ShippedDelay (разность в днях между ShippedDate и OrderDate), SpecifiedDelay (переданное в процедуру значение). Необходимо продемонстрировать использование этой процедуры.
   3. Написать процедуру, которая высвечивает всех подчиненных заданного продавца, как непосредственных, так и подчиненных его подчиненных. В качестве входного параметра функции используется EmployeeID. Необходимо распечатать имена подчиненных и выровнять их в тексте (использовать оператор PRINT) согласно иерархии подчинения. Продавец, для которого надо найти подчиненных также должен быть высвечен. Название процедуры SubordinationInfo. В качестве алгоритма для решения этой задачи надо использовать пример, приведенный в Books Online и рекомендованный Microsoft для решения подобного типа задач. Продемонстрировать использование процедуры.
   4. Написать функцию, которая определяет, есть ли у продавца подчиненные. Возвращает тип данных BIT. В качестве входного параметра функции используется EmployeeID. Название функции IsBoss. Продемонстрировать использование функции для всех продавцов из таблицы Employees.
8. **Работа по финальному проекту**

На основе диаграммы классов проработайте архитектуру базы данных вашего финального проекта.

Напишите скрипт создания сущностей пользователя, ролей и зависимых сущностей (достаточных для выполнения CRUD операций над пользователями и выдачи им определенных ролей)

\*Напишите скрипт создания оставшихся сущностей вашей диаграммы.

## Требования к оформлению

1. В качестве результата выполненного задания необходимо представить файлы, указанные ниже в списке. Перед сдачей задания на проверку обязательно проверьте, что скрипты можно запускать несколько раз и это не вызывает ошибки.
2. Задание преставляет собой:

* Query.sql - в нем должны быть приведены все требуемые запросы и примеры использования разработанных процедур и функций. В файле должны быть выполнены все необходимые действия для использования базы Northwind при запуске на выполнение этого файла. **Перед каждым запросом должен быть комментарий, содержащий номер запроса и текст описания задания. Если в задании есть вопрос, то ответ должен быть дан также в комментарии к запросу**.
* Procedure.sql – в нем должны быть исходные текста процедур и функций задания. В файле должны быть выполнены все необходимые действия по инсталляции процедур и функций в базу Northwind при выполнении этого файла. Инсталляция должна учитывать тот факт, что процедуры с данным именем могут пристутствовать в базе. **Перед каждой процедурой или функцией должен быть комментарий, соответствующий ее номеру в задании. ПРИМЕРЫ использования процедур должны быть приведены в файле Query.sql**

# Задание 9. ADO.Net

## Формулировка задания

Нашей задачей является разработка части DAL (Data Access Layer) к базе вашего будущего финального проекта. При этом необходимо соблюдать ряд требований:

* DAL предоставляет объектный интерфейс. Т.е. данные которые он принимает и выдает на вход являются обычными [POCO (plain old CLR object)](https://en.wikipedia.org/wiki/Plain_Old_CLR_Object) объектами
* DAL должен быть спроектирован таким образом, чтобы его можно было легко заменить на fakes или mocks при модульном тестировании бизнес-слоя
* DAL должен поддерживать подключение к произвольной базе (строки подключения нигде не хардкодятся).
* DAL должен быть спроектирован максимально в provider independent стиле. При этом должна быть оставлена возможность заменить некоторые методы на специфические для конкретной СУБД

## Задание 9.1

Реализовать UserRepository, который должен наследоваться от BaseRepository, реализующего следующий интерфейс:

interface IBaseRepository<T> where T : class, new()

{

T Get(int id);

List<T> GetAll();

bool Save(T entity);

bool Delete(int id);

}

Для создания реализации репозитория рекомендуется придерживаться следующей схемы:

class User

{

}

interface IBaseRepository<T> where T : class, new()

{

T Get(int id);

List<T> GetAll();

bool Save(T entity);

bool Delete(int id);

}

interface IUserRepository : IBaseRepository<User> {

}

class UserRepository : IUserRepository

{

public bool Delete(int id)

{

throw new NotImplementedException();

}

public User Get(int id)

{

throw new NotImplementedException();

}

public List<User> GetAll()

{

throw new NotImplementedException();

}

public bool Save(User entity)

{

throw new NotImplementedException();

}

}

## Задание 9.2

Добавить в существующий проект тестов класс UserRepositoryTests, в котором реализовать проверку корректности работы всех методов репозитория пользователей.

## Задание 9.3

Подключить реализованный репозиторий через инъекцию зависимостей к вашему финальному проекту.

## Задание 9.4

Напишите хранимую процедуру GetAllUsers, принимающую на вход количество возвращаемых записей. Добавте в репозиторий метод, вызывающий эту хранимку.

## IoC контейнер

Контейнер необходимо инициализировать в Global.asax MVC приложения в методе Application\_Start:

protected void Application\_Start()

{

//some code...

DependencyResolver.SetResolver(new NinjectDependencyResolver(new StandardKernel()));

//some code...

}

public class NinjectDependencyResolver : IDependencyResolver

{

private IKernel kernel;

public NinjectDependencyResolver(IKernel kernelParam)

{

this.kernel = kernelParam;

this.AddBindings();

}

public object GetService(Type serviceType)

{

return this.kernel.TryGet(serviceType);

}

public IEnumerable<object> GetServices(Type serviceType)

{

return this.kernel.GetAll(serviceType);

}

private void AddBindings()

{

this.kernel.Bind<IUserRepository>().To<UserRepository>();

}

}

# Задание 10. ASP.Net MVC

## Задание 10.1

Реализуйте следующие контроллеры для административной части вашего приложения:

* UserController (CRUD)
* RoleController (CRUD)

## Задание 10.2

Реализуйте представления для всех экшенов контроллеров из задания 10.1.

## Задание 10.3

Добавьте оставшиеся контроллеры со всеми необходимыми экшенами, которые могут понадобиться для реализации всего функционала финальной работы.

Сами экшены реализовывать не нужно. Достаточно сделать заглушки на них (например, возвращать пустую или заполненную дефолтными значениями запись вью-модели).

Разработайте и реализуйте все необходимые для функционирования экшенов вью-модели.

## Задание 10.4

Создайте кастомный роутинг для следующих методов:

|  |  |
| --- | --- |
| /create-<model> | Создание новой записи. Например, /create-user, /create-order итп |
| /<model>/<id> | Получение конкретной записи по уникальному идентификатору. Например, /user/10, /order/1023 |
| /<model>/<id>/edit | Редактирование записи. Например, /user/10/edit, /order/1023/edit |
| /<model>/<id>/delete | Удаление записи. Например, /user/10/delete, /order/1023/delete |
| /<model>**s** | Страница со списком всех записей определенной модели. Например, /user**s**, /order**s** |
| /<model>**s**/<string> | Поиск по определенной строке в списке всех записей. По сути фильтр. Например, /users/John, /orders/4473729 |

В дальнейшем придерживайтесь этого роутинга для всех новых контроллеров.

# Задание 11. FrontEnd

## Задание 11.1

Подключите к вашему MVC приложению разработанные ранее макеты страниц.

## Задание 11.2

Реализуйте валидации при помощи Unobtrusive Validation JQuery.

Валидация корректности заполнения формы должна происходить до её отправки на сервер. После успешной валидации на клиенте в экшене контроллера должна происходить проверка ModelState на валидность.

<https://habr.com/post/181758/>

<https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions/creating-a-mvc-3-application-with-razor-and-unobtrusive-javascript>

## Задание 11.3

Просмотр, создание и обновление записей реализовать в модальных окнах.

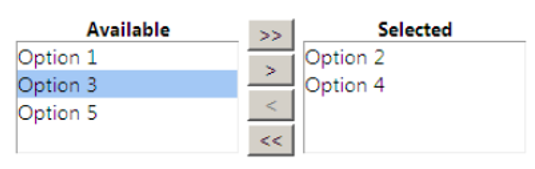
## Задание 11.4

В работе повсеместно применять стили bootstrap.

<https://getbootstrap.com/>

## Задание 11.5\*

Создать страницу с приблизительно таким внешним видом



При нажатии кнопок  и  выделенные элементы из одного списка перекидываются в другой. При этом выделение с них снимается и они добавляются в конец списка. Если ни один из элементов не выделен, выводится сообщение.

При нажатии кнопок  и  все элементы из одного списка перекидываются в другой.

Сами данные должны получаться через ajax-запросы с сервера. В качестве которого можете использовать примитивное MVC приложение с двумя массивами в качестве хранилищ для строк.

**Примечание**

Предусмотреть возможность нахождения на одной странице нескольких таких групп элементов со схожим функционалом.

## Задание 11.6\*

Создать сайт, позволяющий просматривать несколько страниц последовательно друг за другом. Содержимое страниц значения не имеет.

Каждая страница показывается в течение определённого времени, по истечении которого происходит переход на следующую страницу. Последовательность страниц жёстко задана. На странице должен быть размещён обратный отсчёт до перехода на следующую страницу.

При достижении последней страницы выводится запрос о повторении пролистывания страниц или завершении работы. Если выбрано пролистывание, то пользователь перенаправляется на первую страницу, если же выбрано завершение пролистывания, то окно браузера закрывается.

На каждой странице пользователь может:

1) Перейти на предыдущую страницу при нажатии на определённую кнопку.

2) Приостановить обратный отсчёт и повторно запустить его.

# Задание 12. Final task.

Реализуйте оставшуюся функциональность вашей финальной работы.

## Про задание

Вам выдана тема проекта. Задача реализовать веб-приложение, реализующее указанный в теме функционал (далее «функциональные требования»).

Помимо основной клиентской части приложения, решение должно содержать административную часть, в которой администраторы приложения могут выполнять отмеченные голубым цветом функции.

Также необходимо предоставить **тестовые данные** для приложения (список ролей, пользователей и другие специфичные для данной темы записи). Задание без тестовых данных не рассматривается.

Если формулировка задания вам не понятна, сообщите об этом любым удобным способом.

Если вы нашли ошибку, опечатку в задании, пожалуйста, напишите нам об этом.

## Про оформление

Для работы мы будем использовать публичный репозиторий GitHub.

<https://github.com/>

## Документирование кода

Каждый метод должен иметь описание (<summary>). Поскольку теги описания генерируются автоматически при наборе «///», то рекомендуется озаботиться о документировании кода после реализации всех сигнатур методов.

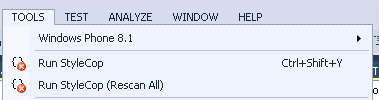
Комментарии к коду не приветствуются, поскольку мы руководствуемся принципов самодокументируемости кода (Код должен быть прост и понятен без комментариев. Если код требует комментариев, есть смысл подумать о другой реализации.). Понятно, что есть исключения (сложная логика кода; интуитивно непонятное решение, базирующееся на правилах бизнеса).

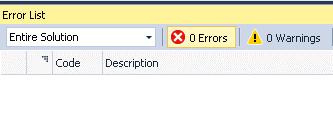
Однако, если есть какой-то вопрос по определенной строке, то допустимо задать его в комментарии.

## Про отправку работ на проверку

После того, как вы считаете, что выполнили все требования задания ЕЩЕ РАЗ проверьте всю функциональность вашего решения, проверьте, что все тесты завершились успешно. Если решение будет падать при запуске, такая работа будет получать сразу **0** баллов.

Запустите проверку кода StyleCop. Никаких предупреждений быть не должно (за исключением предупреждений об отсуствии документации).





Напишите сопроводительное письмо тому, кто выдал вам задание (инструктору или HR специалисту). В письме необходимо указать ссылку на ваш репозиторий с решением, логины/пароли администраторов, пользователей, необходимых для тестирования всего функционала приложения.

После отправки сообщения о том, что работа выполнена редактировать код в репозитории нельзя. В противном случае, работа будет оцениваться в **0** баллов.

## Про дедлайн

Рекомендованный срок выполнения задания – 1 неделя. Срок может быть продлен по усмотрению инструктора или HR специалиста.

## Критерии оценки

Каждое из требований оценивается по трёхбальной шкале.

2 балла –требование задания выполнено, существенный нареканий нет (по стилю кода, по документированию и др.).

1 балл –требование задания выполнено, но можно кое-что улучшить. Есть минорные баги (не влияющие на основную функциональность решения). Обычно такие задания отправляются на доработку, если есть в этом смысл. Самая распространенная оценка.

0 баллов – требование не выполнено, критические и мажорные баги (неотловленное падение приложения или ошибки при каком-либо действии пользователя (ожидаемом или нет)).

Максимум за всё задание: 14 \* 2 = 28 баллов.

## Рекомендации

Имейте в виду, что в любом случае будет оцениваться всё решение целиком. Будет проверяться работоспособность указанного в теме функционала, его отказоустойчивость при типичных и нетипичных действиях пользователя.

Не нужно уделять какому-то из требований приоритетное значение. Требования выделены исключительно для упрощения системы выставления оценки.

Также отметим, что, если не получается реализовать все функциональные требования задания, лучше выбрать некоторые и реализовать их до конца, а не отправлять на проверку обрывки функциональности, которые не работают.

## Требования по реализации

1. Многоуровневая архитектура решения.
   1. Наличие явно выраженного слоя DAL (Data Access Layer)

При этом необходимо соблюдать ряд требований:

• DAL предоставляет объектный интерфейс. Т.е. данные которые он принимает и выдает на вход являются обычными POCO (plain old CLR object) объектами

• DAL должен быть спроектирован таким образом, чтобы его можно было легко заменить на fakes или mocks при модульном тестировании бизнес-слоя

• DAL должен поддерживать подключение к произвольной базе (строки подключения нигде не хардкодятся).

• DAL должен быть спроектирован максимально в provider independent стиле. При этом должна быть оставлена возможность заменить некоторые методы на специфические для конкретной СУБД.

* 1. Наличие проекта с тестами для слоя DAL.
  2. Использование IoC (Inversion of Control) контейнера, соответствующего принципу Dependency Injection[[1]](#footnote-1).

1. Собственная база данных, реализованная в виде SSDT проекта[[2]](#footnote-2).
2. Применение технологии ASP.Net MVC для серверной и клиентской сторон приложения.
   1. Использование разных Layouts для административной и пользовательской части приложения.
   2. Применение PartialViews.
   3. Использование ViewModels для передачи моделей на представления.
   4. Просмотр, создание и обновление записей реализовать в модальных окнах (для открытия/закрытия окон использовать JS).
3. Создайте кастомный роутинг для следующих методов:

|  |  |
| --- | --- |
| /create-<model> | Создание новой записи. Например, /create-user, /create-order итп |
| /<model>/<id> | Получение конкретной записи по уникальному идентификатору. Например, /user/10, /order/1023 |
| /<model>/<id>/edit | Редактирование записи. Например, /user/10/edit, /order/1023/edit |
| /<model>/<id>/delete | Удаление записи. Например, /user/10/delete, /order/1023/delete |
| /<model>**s** | Страница со списком всех записей определенной модели. Например, /user**s**, /order**s** |
| /<model>**s**/<string> | Поиск по определенной строке в списке всех записей. По сути фильтр. Например, /users/John, /orders/4473729 |

В дальнейшем придерживайтесь этого роутинга для всех новых контроллеров.

1. Аутентификация на основе Forms, реализованная самостоятельно, без использования стандартной базы данных membership.
2. Использование JQuery для валидации форм на клиенте, для анимации на клиенте итд (модальные окна, разворачивание списков итп).
3. Использование стилей Bootstrap фреймворка (или любой другой библиотеки или фреймворка).
4. Чистый код (отсутствие хардкода, повторений кода итп)[[3]](#footnote-3).
5. Плюсом будет, если будут существовать тесты на само MVC приложение. Т.е. будут протестированы:

* Бизнес-логика
* Маршрутизация
* Представления
* Контроллеры

1. Если нет опыта работы с IoC контейнерами, то это требование можно пропустить. [↑](#footnote-ref-1)
2. Если нет опыта работы с SSDT проектом допускается добавление в репозиторий SQL скрипта, разворачивающего вашу базу данных на произвольной машине (создание схемы БД + наполнение таблиц тестовыми данными). [↑](#footnote-ref-2)
3. Для соблюдения этого требования сильно поможет проверка решения через StyleCop. [↑](#footnote-ref-3)