Тестовые задания на .Net-разработчика

Задания на клиентскую разработку

Разработать генератор форм для web-приложения на основе JSON-массива. В JSON-массиве содержится тип элемента, его стилизация, маска ввода, правила валидации значений и отображаемые значения. Проверочные JSON-массивы:



Структура входных данных

Входные данные в виде JSON-массива:

* "form" – это блок формы содержащий массив элементов и сообщение пользователю по результатам взаимодействия с формой;
* "name" – имя формы (пример: "two", "one");
* "items" – массив элементов формы;
* "postmessage" – сообщение в случае успешного взаимодействия.

Все элементы формы содержат атрибут "type", он определяет какой элемент описывается. Рассмотрим элементы которые содержаться в json-массивах задания.

Тип filler

«Наполнитель» - это произвольный текст, поддерживает html разметку. Элемент содержит атрибут "message", который содержит размещаемый контент.

Тип text

Поле для ввода текста. Содержит атрибуты:

* "name" – имя элемента;
* "placeholder" – при проверке, значение поля не должно быть равно значению плейсхолдера. Необязательный атрибут;
* "required" – это свойство, указывающее на обязательность заполнения элемента, принимает значения true / false;
* "validationRules" – это правила валидации. Содержит атрибуты:
  + "type" – это тип проверки. Может принимать следующие значения:
  + text – поле должно содержать любой текст;
  + tel – поле должно содержать номер телефона;
  + email – поле должно содержать e-mail адрес;
  + select – проверка выбора элемента списка;
  + radio – проверка выбора опции;
  + checkbox – проверка установки флажка.
* "value" – это предустановленное значение элемента, может быть пустым;
* "label" – описание поля, произвольный текст, может быть пустым;
* "class" – css-класс элемента;
* "disabled" – активен или нет элемент, принимает значения true / false.

Тип textarea

Поле для ввода многострочного текста. Атрибуты типа textarea аналогичны атрибутам типа text, описанного выше.

Тип checkbox

Поле для выбора. Атрибуты типа checkbox аналогичны атрибутам типа text, описанного выше, за исключением атрибута "value". Также содержит дополнительный атрибут "checked", указывающий первоначальное состояние флажка. Он принимает значение true/false.

Тип button

Кнопка. Содержит атрибуты:

* "class" – css-класс кнопки;
* "text" – текст кнопки.

Тип select

Выпадающий список. Атрибуты типа select аналогичны атрибутам типа text, описанного выше. Но имеется один дополнительный атрибут:

* "options" – список выпадающих значений, каждое значение которого содержит атрибуты:
  + "value" – значение элемента списка;
  + "text" – текст элемента списка;
  + "selected" – активен или нет элемент по умолчанию, принимает значение true / false.

Тип radio

Набор опций. Атрибуты типа radio аналогичны атрибутам типа text, описанного выше. Но имеется один дополнительный атрибут:

* "items" – список элементов набора опций. Каждый элемент содержит:
  + "value" – значение опции, которое будет передано на сервер;
  + "label" – текст опции, который видит пользователь;
  + "checked" – активна или нет опция, принимает значение true / false.

Задания на верстку

При выполнении заданий не запрещается использовать css-фреймворки, js-плагины/библиотеки, однако использование javascript должно быть минимальным. Блоки должны быть независимыми, а значит верстка не должна ломаться при удалении какого-либо блока, либо добавления аналогичного, а также при заполнении блока контентом или удалении контента.

Макет доступен по [ссылке](https://www.figma.com/file/K8zVAzVPxXcrChBn8CMbCfB5/DIRECTUM---тестовое-задание-на-верстку?node-id=2:285).

Элементы, представленные на макете:

1. Баннер с логотипом.
2. Текстовый блок.
3. Контентный блок с типизированными элементами

4. Футер с логотипом. При удалении контентного блока (3), футер должен остаться прижатым к низу страницы.

Задания на серверную разработку

Необходимо разработать систему локализации некоторого сервиса.

Требования:

* Локализованное значение строки должно быть доступно по коду строки локализации и переданному объекту CultureInfo.
* Если при запросе локализованного значения строки объект CultureInfo не был передан, то необходимо использовать текущую культуру потока.
* В качестве источника строк могут выступать как ресурсы текущей сборки, так и сторонние источники (БД, XML-файл или иное). Необходимо предусмотреть возможность подключения произвольного источника строк.
* Запрашиваемая строка локализации может находится в любом из подключенных источников данных.
* Разные источники могут иметь пересечения по кодам строк, система должна корректно определять из какого источника брать строку в каждом конкретном случае.

Реализация

Код строки локализации – любой уникальный идентификатор на ваше усмотрение.

Для работы с локализациями необходимо разработать класс LocalizationManager, который предоставляет следующие методы:

* GetString – возвращает значение строки локализации по её коду для переданной культуры;
* RegisterSource – регистрирует источник строк локализаций.

Для проверки работоспособности системы локализации достаточно сделать простое консольное приложение или написать модульные тесты.

Задания на анализ кода

Дан исходный код функции. Необходимо написать, что выполняет код; указать на недостатки исходного кода и предложить свою реализацию алгоритма.

static void Func1(ref KeyValuePair<int, string>[] a, int key, string value)

{

Array.Resize(ref a, a.Length + 1);

var keyValuePair = new KeyValuePair<int, string>(key, value);

a[a.Length - 1] = keyValuePair;

for (int i = 0; i < a.Length; i++)

{

for (int j = a.Length - 1; j > 0; j--)

{

if (a[j - 1].Key > a[j].Key)

{

KeyValuePair<int, string> x;

x = a[j - 1];

a[j - 1] = a[j];

a[j] = x;

}

}

}

}