**Установка и настройка редактора CKEditor + CKFinder на ASP.NET MVC**

Дата публикации: 04.05.2019. Категория: [ASP.NET MVC](https://alekseev74.ru/lessons/show/aspnet-mvc)

Последнее обновление: 18.02.2020

**Сегодня разберемся, как интегрировать и настроить очень популярный редактор контента CKEditor в стандартное приложение ASP.NET MVC. Также установим и настроим не менее популярный менеджер файлов CKFinder.**

Официальная страница разработчика: <https://ckeditor.com/>

CKEditor - это очень популярный на сегодняшний день визуальный WYSIWYG редактор HTML-содержимого.

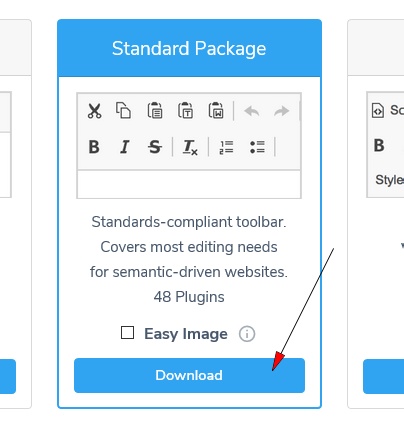
CKFinder - это менеджер файлов, который позволяет получить доступ к файловой системе на сервере и управлять файлами из браузера клиента.

Обычно оба этих компонента идут в связке, то есть менеджер файлов интегрируется в редактор.

CKEditor представлен в двух версиях. Версия 4 как более ранняя и, соответственно, более распространенная. И версия 5, новая, реализующая модульную архитектуру, с множеством нововведений.

**Настройка CKEditor**

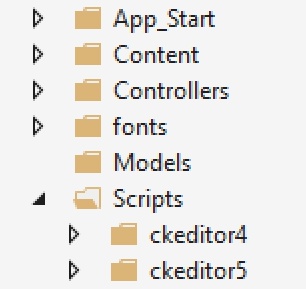
Я в своих проектах использую 4-ую версию, ее функционала вполне хватает для решения любых задач. Скачаем архив с редактором и добавим его себе в проект.

[](https://alekseev74.ru/files/upload/images/lq3pgcvo.jpg)

*Скачать архив с CKEditor 4*

[](https://alekseev74.ru/files/upload/images/lgjmwh40.jpg)

*Скачать архив с CKEditor 5*

[](https://alekseev74.ru/files/upload/images/djl0m0od.jpg)

*Копируем файлы редактора и добавляем их в проект*

Добавить **CKEditor**очень просто. Это происходит в несколько этапов:

1. подключаем через тег <script> JS-библиотеку на страницу;
2. определяем HTML-тег <textarea>, который будет использоваться в качестве контейнера для CKEditor;
3. используем метод CKEDITOR.replace(); для подключения редактора к ранее определенной <textarea>.

<div>

<script src="~/Scripts/ckeditor4/ckeditor.js"></script>

<div>

@using (Html.BeginForm("Index", "Home", FormMethod.Post))

{

<div>

<textarea id="content1" name="content1"></textarea>

</div>

<div>

<input type="submit" value="Отправить" />

</div>

}

</div>

</div>

<script>

document.addEventListener("DOMContentLoaded", function (event) {

var editor = CKEDITOR.replace('content1');

});

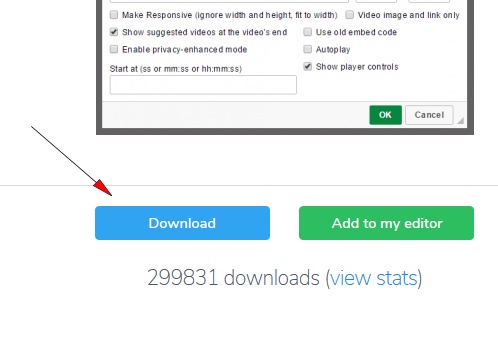
</script>

Все готово! В приведенном выше примере CKEditor подключается к текстовой области **'content1'** внутри HTML-формы. Далее содержимое этого поля будет отправлено на сервер.

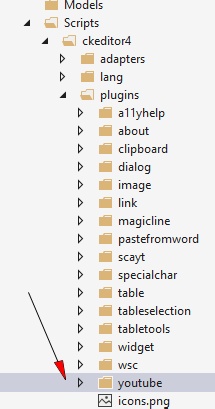
Стоит отметить, что для CKEditor существует множество плагинов, который расширяют его стандартный функционал. Также плагины можно писать самому (более подробно этот процесс описывается в официальной документации на сайте). Давайте добавим к редактору плагин, который позволяет встраивать на страницу видео с видеохостинга **Youtube.**

На момент написания статьи все плагины можно найти в отдельном разделе на сайте. Интересующий нас плагин находится по адресу: <https://ckeditor.com/cke4/addon/youtube>

1. скачиваем zip-архив

[](https://alekseev74.ru/files/upload/images/cpooyirr.jpg)

1. помещаем содержимое архива в папку **plugins**

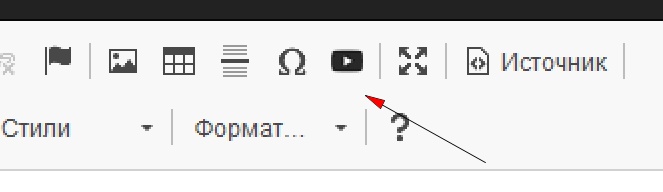
[](https://alekseev74.ru/files/upload/images/bnhaydyr.jpg)

1. подключаем файл через настройку в файле конфигурации **config.js**

***config.js***

config.extraPlugins = 'youtube';

 После этого на панели инструментов редактора появится новая кнопка, позволяющая работать с новым плагином.

[](https://alekseev74.ru/files/upload/images/dbum04xd.jpg)

*Новый плагин Youtube был успешно подключен к редактору*

**Настройка CKFinder**

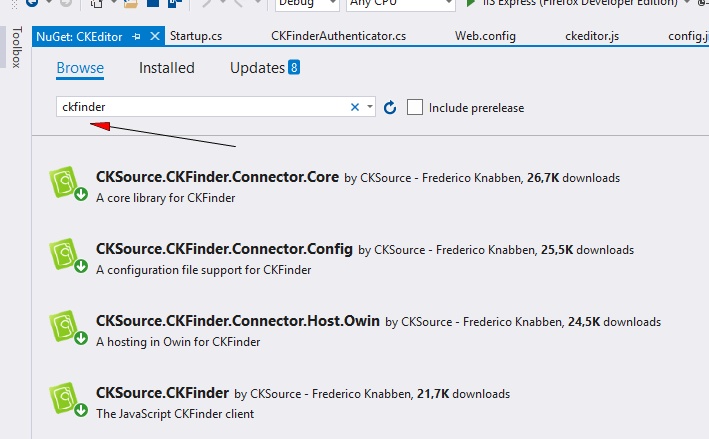
Теперь разберемся с менеджером файлов CKFinder. Так как файлы (например, картинки) у нас располагаются на сервере, то помимо подключения этого компонента на клиенте, то есть в браузере пользователя, нам также будет необходимо настроить его со стороны сервера, а конкретно подключить некоторые библиотеки и реализовать несколько методов.

Официальная документация по настройке коннектора для ASP.NET приложений находится по адресу: <https://ckeditor.com/docs/ckfinder/ckfinder3-net/>

**Важно! Со слов разработчиков, они пока что не планируют адаптировать CKFinder под ASP.NET Core. "Unfortunately, we do not plan to develop CKFinder for .Core, at least at the moment. Currently, we are working on CKFinder for Java which was highly requested by our users."**

В отличии от редактора CKEditor, который мы настраивали на сайте и скачивали вручную, CKFinder мы можем добавить в проект прямо в Visual Studio через [NuGet Manager](https://alekseev74.ru/lessons/show/visual-studio/nuget-manager-first-look). Минимальный необходимый для работы набор пакетов выглядит так:

1. **CKSource.CKFinder** - JavaScript-клиент, подключается на стороне клиента
2. **CKSource.CKFinder.Connector.Core** - основная и ключевая библиотека, которая обеспечивает работу коннектора CKFinder на сервере
3. **CKSource.CKFinder.Connector.Config** - эта библиотека позволяет настраивать CKFinder через общий [файл конфигурации приложения Web.config.](https://alekseev74.ru/lessons/show/aspnet-mvc/web-config-basics). Обратите внимание, как изменится конфиг-файл после установки этого пакета, теперь там находятся все настройки для CKFinder.
4. **CKSource.CKFinder.Connector.Host.Owin** - эта библиотека позволяет подключить коннектор как OWIN-компонент в приложении. На серверной стороне CKFinder работает как раз следуя этой спецификации, т.е. Open Web Interface for .NET
5. **Microsoft.Owin.Host.SystemWeb** - сразу же установим OWIN хост или сервер
6. **CKSource.FileSystem.Local** - этот компонент добавляет поддержку работы с файловой системой на сервере

[](https://alekseev74.ru/files/upload/images/lcb4l3hk.jpg)

*Через поиск в NuGet Manager можно найти нужные нам пакеты*

После установки всех необходимых компонентов можно приступить к настройке серверной стороны.

Во-первых, CKFinder требует, чтобы доступ к файловой системе был только у аутентифицированных пользователей, то есть при каждом запросе на сервер через этот менеджер файлов мы должны сообщать ему, имеет ли право данный пользователь получить доступ к файлам на сервере или нет.

Для этого нам нужно реализовать специальный интерфейс, а в нем метод, который аутентифицирует все входящие запросы.

***CKFinderAuthenticator.cs***

public class CKFinderAuthenticator : IAuthenticator

{

public Task<IUser> AuthenticateAsync(ICommandRequest commandRequest, CancellationToken cancellationToken)

{

var user = new User(true, new List<string>());

return Task.FromResult((IUser) user);

}

}

В приведенном выше примере мы просто каждый раз возвращаем фейкового пользователя, который всегда аутентифицирован (флаг true) и не имеет никаких ролей (пустой лист). То есть у нас всегда есть права на доступ к файловой системе. В реальном проекте к этому вопросу, конечно, нужно подойти более серьезно и совместить этот метод с существующей системой аутентификации и авторизации пользователей.

Далее, если в приложении еще не создан файл **Startup.cs**, то создаем его. В этом файле мы завершим настройку серверной части CKFinder. Финальная версия файла выглядит так:

***Startup.cs***

[assembly: OwinStartup(typeof(CKEditor.Startup))]

namespace CKEditor

{

public class Startup

{

public void Configuration(IAppBuilder app)

{

//регистрируем файловую систему для коннектора

FileSystemFactory.RegisterFileSystem<LocalStorage>();

//объявляем маршрут в приложении и сопоставляем его с коннектором

//клиентская JS-библиотека CKFinder ожидает увидеть коннектор именно по этому маршруту

app.Map("/ckfinder/connector", SetupConnector);

}

//объявляем метод по настройке и инициализации коннектора

private static void SetupConnector(IAppBuilder app)

{

//создаем экземпляры необходимых классов

var connectorFactory = new OwinConnectorFactory();

var connectorBuilder = new ConnectorBuilder();

var customAuthenticator = new CKFinderAuthenticator();

connectorBuilder

.LoadConfig() //подгружаем конфигурацию из файла Web.config

.SetAuthenticator(customAuthenticator) //устанавливаем ранее определенный аутентификатор

.SetRequestConfiguration((request, config) => { config.LoadConfig(); }); //определяем конфигурацию для каждого отдельного запроса

//создаем экземпляр коннектора

var connector = connectorBuilder.Build(connectorFactory);

//добавляем коннектор в pipeline

app.UseConnector(connector);

}

}

}

На этом настройка серверной части завершена. Теперь интегрируем CKFinder-клиент с CKEditor на странице.

<div>

<script src="~/Scripts/ckeditor4/ckeditor.js"></script>

<script src="~/Scripts/ckfinder/ckfinder.js"></script>

<div>

@using (Html.BeginForm("Index", "Home", FormMethod.Post))

{

<div>

<textarea id="content1" name="content1"></textarea>

</div>

<div>

<input type="submit" value="Отправить" />

</div>

}

</div>

</div>

<script>

document.addEventListener("DOMContentLoaded", function (event) {

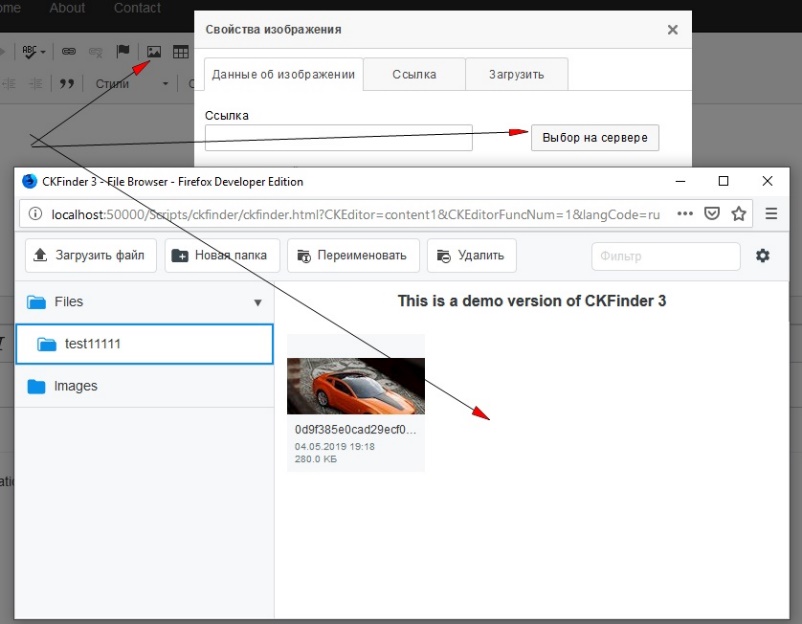
var editor = CKEDITOR.replace('content1');

CKFinder.setupCKEditor(editor);

});

</script>

Если все скрипты подключены правильно и на стороне сервера коннектор настроен, то мы получим доступ к файловой системе через CKFinder из браузера пользователя.

[](https://alekseev74.ru/files/upload/images/11fwlzbu.jpg)

*CKFinder в действии*

В зависимости от настроек коннектора в файле **Web.config**мы можем менять разрешения на манипуляции с файлами и папками (создание, перемещение и т.д.).

Отмечу, что мы рассмотрели только базовую конфигурацию CKFinder. В дополнение к этому можно прикрутить логирование, кэширование файлов, различные плагины (например, добавлять водяной знак к загруженным изображениям) и т.д. Более подробно об этом можно почитать в официальной документации.