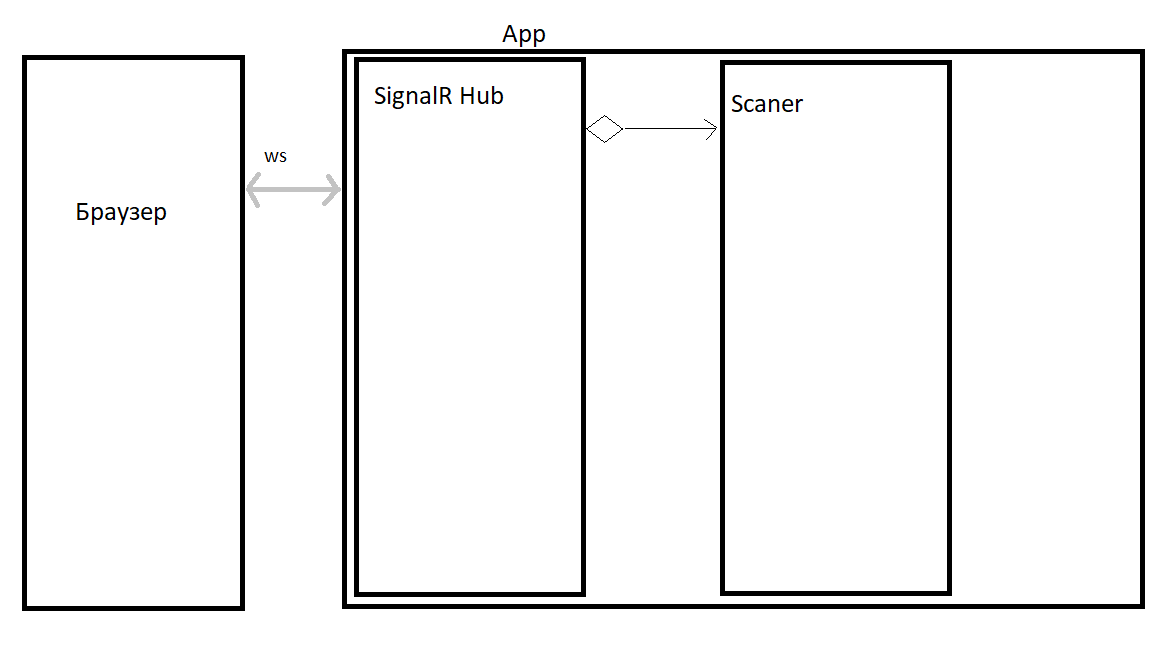
-----------------------------------------------------------------------------------------------------------------------

ДОПОЛНИТЕЛЬНО

-----------------------------------------------------------------------------------------------------------------------

**1.Предположения и комментарии:**

* Тут много что еще нужно менять если использовать в проде. Поэтому я буду дополнительно писать о тех улучшениях, которые я не сделал, но сделал бы в случае дальнейшего использования программы.
* Алгоритм поиска в ширину предполагает закидывание в очередь исполнения дочерних нод и полностью обработки одного уровня дерева, но так как у нас может использоваться более чем 1 поток теоретически некоторые ноды могут долго загружаться и нарушить алгоритм. Поэтому делаю в 2 этапа: 1 – закачка всех нод уровня и добавление этих нод в очередь на парсинг, 2 – парсинг закачанного и закидывание ссылок в очередь на загрузку. При таком раскладе у нас пропадает привязка к очереди, и мы можем хранить ноды в любой коллекции. Следствием того что мы храним все данные есть следующий пункт.
* Сейчас при работе приложения все данные хранятся в оперативной памяти, поэтому очень большие запросы могут быть проблемой. По-хорошему тут нужно писать в БД или как вариант можно отойти от алгоритма + обработанные данные не сохранять (только отображение), это уменьшит использование оперативной памяти.
* В графе могут быть циклы, поэтому будем проверять и избегать зацикливания. Для этого создаем dictionary <url, id>. Добавляем туда новые url и присваиваем им id. Если найдется такой url – не добавляем в очередь на закачку.
* Ссылки на другие страницы ищу в теге <a>, потому что полно ссылок на другие файлы (картинки, js).
* По заданию сказано, что ссылкой считать подстроку, которая начинается с подстроки HTTP:// поэтому относительные ссылки я пропускаю.
* Не проверяю на расширения, могут грузиться больше файлы типа pdf что тормозит систему. Вообще можно использовать библиотеки для парсинга (к примеру, anglesharp) или придется изменять регулярное выражение.
* Есть 2 прогресс бара один для текущего этапа закачки, второй для глобального процесса обработки (показывает сколько из общего числа сайтов уже были полностью обработаны).
* Остановка - используем токен отмены и не можем вернуться к выполнению.
* Пауза - используем токен отмены и можем вернуться к выполнению.
* Отображение ошибки минимальное и не всегда удачное, можно конечно выводить весь стек информации (в начале в свернутом состоянии, по нажатию на карточке сайта отображать полностью), который находиться в exception, это будет не красиво. Плюс помимо exception нужно смотреть на код ответа и, если он не 200, добавлять его в сообщение (этого я не делал).
* Использовал для реализации .Net Core, SignalR и веб GUI. На клиенте валидации данных нет, поэтому есть возможность все сломать.



* Предполагаю, что UX/UI тут вообще не важен поэтому сделал так что бы просто работало.
* Есть класс SiteScanner, он требует рефакторинга(вынести логику методов по загрузке и парсингу в другие классы), но как это сделать красиво не уверен. Проблема в том, что эти методы сильно зависят от нескольких коллекций и от событий, которые в них происходят.

**2.Тестирование:**

* Попробовал бы задать экстремальные параметры, к примеру 10000-1000000 страниц для парсинга.
* Попробовал бы задать не корректные параметры.
* Протестировал бы функциональность кнопок, действительно ли они отрабатывают так как должны.
* На корректность можно готовить свои локальные примеры с заранее желаемым результатом. И сравнивать ожидаемое с действительным(автоматически). В этом случае сложность в подготовке данных. Можно так же задать просто любой адрес, и вручную пройтись по результатам и проверить их корректность. Но искать текст на сайте с помощью ctrl + f - не вариант, потому что есть случаи, когда в html - есть, а на странице не отображается. Нужно смотреть через sources.

------------------------------------------------------------------------------------------------------------------------