Правительство Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Санкт-Петербургский государственный университет»

Сокольвяк Сергей Дмитриевич

HwProj на WebSharper: Маршрутизация и GUI

Курсовая работа

Научный руководитель:

доцент Литвинов Ю.В.

Санкт-Петербург 2018

Оглавление

Введение

На сегодняшний день, большинство веб-страниц являются так называемыми rich-client приложениями, которые включают большое количество JavaScript кода, который выполняется на клиентской стороне Вашего веб-приложения. Существует несколько способов для разработки rich-client веб-приложений, используя язык F#:

- Вы можете вручную объединять JavaScript код в файлы и обсуживать эти файлы как часть вашей серверной части веб-приложения, используя веб-фреймворки, такие как Suave или ASP.NET.
- Вы можете использовать WebSharper (http://websharper.com), чтобы написать обе части клиент-серверного приложения на языке F#. В таком случае, хостом веб-приложения может стать ваш собственный компьютер. Это возможно благодаря использованию Suave, WebSharper.Warp, или контейнера ASP.NET, такого как IIS.

Разработка rich-client приложений с помощью первой техники идет по довольно стандартному пути, где сторона клиента главным образом независима от F#. Целью моей работы было сосредоточится на второй технике, познакомиться с некоторыми основными особенностями WebSharper, включая Sitelet'ы, Pagelet'ы, Formlet'ы и Form'ы для разработки реактивных приложений WebSharper, используя пространство имен WebSharper.UI, реактивного расширения WebSharper и использовать полученные знания при разработке веб-приложения для проверки домашних работ HwProj2.
Основной упор в своей работе я делал на разработку пользовательского интерфейса веб-приложения, обслуживания и маршрутизации контента, возвращаемого пользователю.

Краткий обзор WebSharper

WebSharper – веб-инструмент с открытым исходным кодом, который предоставляет широкие возможности для быстрой разработки веб-приложений полностью на языке F#. WebSharper представляет собой систему состоящую из компилятора F#-to-JavaScript и более чем 50 библиотек, которые включают в себя веб-абстракции и расширения популярных библиотек JavaScript для разработки клиентской части приложения. В частности, WebSharper использует несколько преимуществ F#, сочетание которых позволяет получить уникальный программный опыт для развития веб-приложений:

• Клиентский и серверный код, помеченный специальными атрибутами, может быть создан в одном проекте F#. Клиентский контент, называемый Pagelet'ы, автоматически транслируется в JavaScript и возвращается клиенту в зависимости от того, как это определит соответствующий Sitelet.

• Структура веб-приложения строится с помощью Sitelet'ов, включающих в себя EndPoint'ы. Endpoint'ы, которые определяют различные конечные точки и методы доступа к ним. Sitelet'ы выступают в роли контроллеров, которые отвечают за отображение контента, соответствующего какой-либо конечной точке, причем этот контент возвращается асинхронно. Sitelet'ы вписываются в архитектуру ASP.NET MVC и могут легко адаптированы для использования вместе с Web-API в ASP.NET.

• HTML разметка в Pagelet'ах и Sitelet'ах может внедряться в ваше веб-приложения с помощью различных методов, включенных в пространство имен WebSharper.UI. WebSharper.UI предоставляет широкий функционал для создания пользовательского контента с возможность конструирования веб-страничек с помощью общих шаблонов, с поддержкой динамического создания узлов DOM в зависимости от полученного запроса.

• Различные пользовательские формы, такие как формы входа и регистрации могут быть выражены в чрезвычайно компактной форме в виде так называемых Formlet'ов или Form (для реактивного использования). Их функционал может быть серьезно расширен с помощью множества функций пространства имен WebSharper.Forms.Bootstrap, включающего в себя функции проверки входных данных и кастомизации полей ввода.

• В клиентском коде могут использоваться многие dotNET и F# библиотеки. Причем, их функционал сопоставлен с соответствующим функциональном JavaScript через сложные методы. Кроме того, вы можете обеспечить доступ к любому dotNET типу, описав явно, как транслировать его в JavaScript.

• Клиент может осуществлять асинхронные вызовы функций на сервере, например, для конструирования необходимого контента или для асинхронного выполнения больших вычислений, которые зачастую не могут быть выполнены только на мощной серверной машине.

Описание решения

Использование WebSharper.Sitelets на примере HwProj2

Для обслуживания и маршрутизации контента в веб-приложении, разрабатываемом с помощью технологии WebSharper, служат так называемые Sitelet'ы.

Преимущества Sitelet'ов:

• Они предоставляют бесшовную коммуникацию между клиентом и сервером, позволяя Вам вызывать клиентские методы для расширения функционала серверной части. Для вызова клиентских методов требуется использовать ключевое слово client, а название метода заключить внутри конструкции <@ …@>. В примере ниже, вызываются клиентские методы, конструирующие форму регистрации и входа.

```
div [] [
    div[attr.style "float:left; width:400px"][ h1[][text("Log In")]
                                               client <@ RegClient.AnonUser() @>]
    div[attr.style "float:right; width:400px"][ h1[][text("Register")]
                                                client <@ RegClient.RegUser() @>]
]
```

• Они предоставляют конечные точки для Вашего веб-приложения, в виде F# типа (так называемый EndPoint type), таким образом, что сразу перечисляются все возможные значения типа конечной точки. В примере ниже, также следует обратить внимание на URL адреса, которые привязаны к тому или иному значению конечной точки. Здесь в работу включается маршрутизатор (см. ниже).

```
type EndPoint =
    | [<EndPoint "/">] Start
    | [<EndPoint "/about">] About
    | [<EndPoint "/profile">] Profile
    | [<EndPoint "GET /course">] GetCourse of int
    | [<EndPoint "POST /course">] CreateCourse of data: string
    | [<EndPoint "GET /delete_course">] DeleteCourse of id: int
    | [<EndPoint "/edit">] EditCourse of int
    | [<EndPoint "/courses">] ListOfCourses
```

• Они обеспечивают безопасные связи с ресурсами серверной части, таким образом, Ваши гиперссылки всегда будут актуальными. Это сделано с помощью обсуживающего контекста (Context<Endpoint>), через который URL адрес может быть автоматически сгенерирован для любой конечной точки. В примере ниже используется метод ctx.Link, который генерирует URL адрес для конечной точки, переданной в качестве аргумента.

```
let MenuBarLogged (ctx: Context<EndPoint>) endpoint : Doc list =
        [li [if endpoint = Profile then yield attr.``class`` "active"]
            [a [attr.href (ctx.Link Profile)] [text "Profile"]];
         li [if endpoint = About then yield attr.``class`` "active"]
            [a [attr.href (ctx.Link About)] [text "About"]];
         li [if endpoint = ListOfCourses then yield attr.``class`` "active"]
            [a [attr.href (ctx.Link ListOfCourses)] [text "Courses"]];
         li [on.click (fun _ _ -> RegClient.LogOutUser())]
            [a [attr.href "#"] [text "Log Out"]]
        ]
```

• Они предоставляют механизм, способный отвечать на поступающие запросы и направлять пользователя на необходимую конечную точку (за это отвечает так называемый маршрутизатор), и сопоставлять конечным точкам содержание, которое будет возвращаться пользователю (за это отвечает контроллер). Вы можете также определить маршрутизаторы и контроллеры вручную. В примере ниже продемонстрирован контроллер, возвращающий пользовательский контент (представления) для различных конечных точек.

```
[<Website>]
let Main =
    Application.MultiPage (fun ctx endpoint ->
        match endpoint with
        | EndPoint.Start -> StartPage ctx
        | EndPoint.About -> AboutPage ctx
        | EndPoint.Profile -> ProfilePage ctx
        | EndPoint.ListOfCourses -> AllCoursesForUser ctx
        | EndPoint.DeleteCourse id ->
          OngoingCoursesManager.DeleteOngoingCourse id
          DeletePage ctx id
```

• Кроме того, они способны объединяться с шаблонами HTML, которые позволяют Вам конструировать возвращаемый пользователю контент (представления) по общему шаблону. В примере ниже, Templating.Main является функцией, создающий страничку по общему шаблону, но добавляющей некоторые детали, уникальные для каждой отдельной странички.

```
let ProfilePage (ctx: Context<_>) =
    let getUser = ctx.UserSession.GetLoggedInUser() |> Async.RunSynchronously
    let login =
        getUser
        |> (fun getUser -> match getUser with
                           | Some name -> name
                           | _ -> "" )
    let isTeacher = AccountManager.IsTeacher login
    Templating.Main ctx EndPoint.Profile "Profile" [
        h1 [] [text "Ваш профиль"]
        p [] [text "Ваш email: "]
        p [] [text login]
        p [] [text "Ваше имя: "]
        p [] [text "Peter Pen"]
        p [] [text "Ваш статус: "]
        p [] [text (match isTeacher with | true -> "Преподаватель" | false -> "Студент")]
    ]
```

Использование WebSharper.Forms на примере HwProj2

При разработке пользовательских форм в веб-приложении ключевую роль играют библиотеки WebSharper.Forms и WebSharper.Forms.Bootstrap. Первая предоставляет доступ к веб-формам, а вторая обеспечивает удобные сокращения для различных элементов управления пользовательским интерфейсом, используя популярную библиотеку JavaScript для начальной разработки Bootstrap.
Пространство имен WebSharper.Forms.Bootstrap.Controls содержит ряд вариантов управления пользовательским интерфейсом, таких как Button, Checkbox, Input, InputPassword или Radio, каждый из которых принимает список атрибутов WebSharper.UI для настройки внешнего вида пользовательской формы. Значения по умолчанию для каждого из них представлены в модуле Controls.Simple. Вы можете увидеть форму входа с использованием Forms и Forms.Bootstrap в примере ниже:

```
let AnonUser () =
    Form.Return(fun user pass -> (user, pass))
    <*> (Form.Yield ""
         |> Validation.IsNotEmpty "Enter an email")
    <*> (Form.Yield ""
         |> Validation.IsNotEmpty "Enter a password")
    |> Form.WithSubmit
    |> Form.Run (fun (user, pass) ->
        async {
            do! Server.LoginUser user pass
            return JS.Window.Location.Reload()
        } |> Async.Start
    )
    |> Form.Render (fun user pass submit ->
        form [] [
            Controls.Simple.InputWithError "Email" user submit.View
            Controls.Simple.InputPasswordWithError "Password" pass submit.View
            Controls.Button "Log in" [attr.``class`` "btn btn-primary"] submit.Trigger
            Controls.ShowErrors [attr.style "margin-top:1em;"] submit.View
        ])
```

Заключение

WebSharper несомненно является основным веб фреймворком для разработки на языке F#, который предоставляет широкие возможности для написания и использование клиентских скриптов, полностью на языке F#. WebSharper Sitelet'ы позволяют представлять целое веб-приложение в виде F# значений типа EndPoint на стороне сервера в хорошо структурированном виде. Кроме того, WebSharper предоставляет возможности для разработки пользовательского интерфейса, который представляется в виде строго типизированных значений F# на стороне клиента и веб-форм, каждую из которых можно кастомизировать, используя библиотеки JavaScript.
Однако, несмотря на богатый функционал, разработка проекта на платформе WebSharper является весьма трудоемкой задачей. Это обуславливается в первую очередь недостаточным количеством документации и обучающего материала. На русскоязычных форумах информации о WebSharper'е практически нет, на англоязычных - представлена в виде документации, в которой нет никаких практических пояснений и одной главы книги, которой нет в открытом доступе. Тем не менее, данный проект актуален тем, что может стать отправной точкой для программистов, желающих начать свое знакомство с разработкой веб-приложений с помощью WebSharper.

Список литературы

[1] URL: https://developers.websharper.com/docs/ дата обращения 25.03.2018

[2] URL: https://try.websharper.com/ дата обращения 03.04.2018

[3] Don Syme, Adam Granicz, Antonio Cisternino – Expert F# 4.0 [4-е издание, 2015, Apress], с.363, 415

[4] URL: https://www.w3schools.com/html/ дата обращения 15.04.2018

[5] URL: https://www.w3schools.com/css/ дата обращения 18.04.2018

[6] URL: https://www.w3schools.com/js/ дата обращения 21.04.2018

[7] URL: https://habr.com/post/351890/ дата обращения 25.04.2018