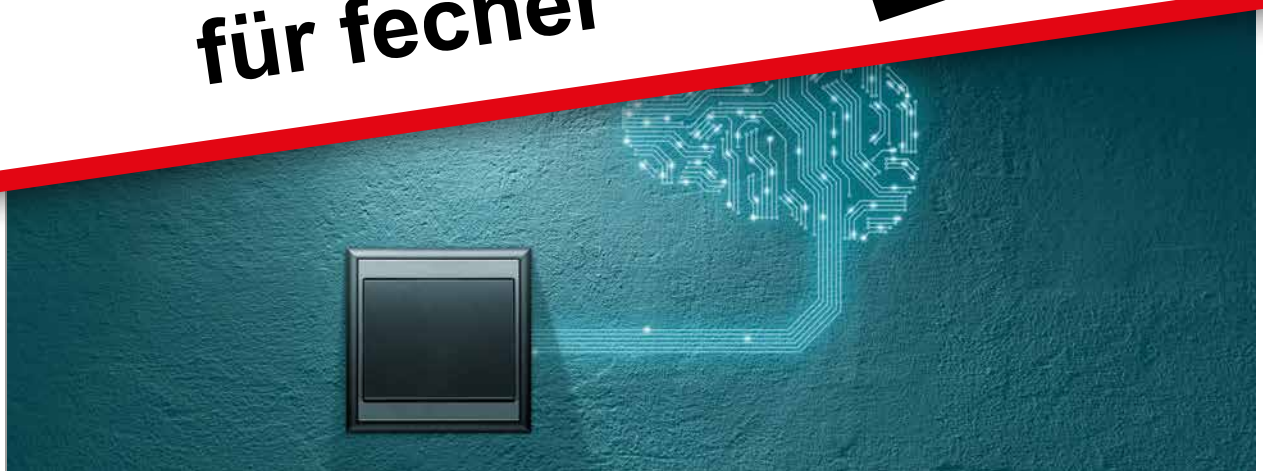
windows .developer Ausgabe 11.2021

Sonderdruck für fecher



Toolbasiertes Web-Enabling für WinForms-Anwendungen

Paradigmenwechsel (fast) auf Knopfdruck

In welchem Unternehmen gibt es sie nicht, die guten alten Geschäftsanwendungen, die den Laden am Laufen halten? Will man sie aber in Browser- oder Webanwendungen umwandeln, steht man häufig vor Problemen. Hier verspricht das Wisej-Framework Abhilfe: Aus einer vorhandenen WinForms-Anwendung soll durch den Austausch von Frontend-Komponenten unkompliziert eine moderne Webanwendung gemacht werden können. Ein Beispielprojekt zeigt die Knackpunkte auf.

von Jens Eidinger

Wisej ist für die schnelle Entwicklung komplexer Webanwendungen konzipiert – so heißt es auf der Website zum Produkt [1]. Damit dabei echte Realtime-Web-Applications entstehen, die sich für den Benutzer ebenso interaktiv wie eine Desktopanwendung gestalten, kümmert sich das Framework eigenständig um das benötigte HTML, um Statusmanagement, Backend-Services, Ajax Callbacks, Synchronisation, DOM, CSS, JavaScript, Sicherheit, Authentifizierung und Concurrency. Der Entwickler muss dafür kaum etwas tun; aus seiner Sicht läuft das Ganze weitgehend identisch mit der WinForms-Welt ab.

Zu gut, um wahr zu sein?

Das klingt gut, doch welchen Entwickler würden solche vollmundigen Marketingaussagen nicht skeptisch stimmen? Blicken wir also einmal hinter die Kulissen und beleuchten die Softwarearchitektur, die dieses gewohnte Verhalten in neuer Umgebung möglich macht. Wisej baut auf der Open-Source-JavaScript-Bibliothek Qooxdoo [2] auf und bindet deren HTML5-Widgets ein, die in ihrer Funktion den jeweiligen WinForms Controls entsprechen. Mit Wisej erhält man außerdem zusätzliche eigene Widgets. So gibt es etwa ein Grid Control mit .NET-konformer Anbindung an beliebige Datenquellen. Das Erscheinungsbild entspricht standardmäßig dem von WinForms, lässt sich aber über Themes frei gestalten.

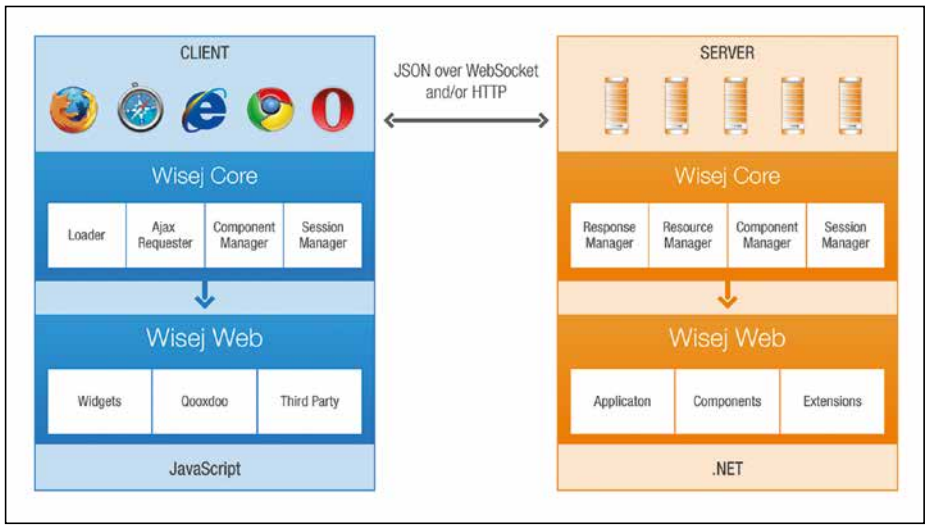
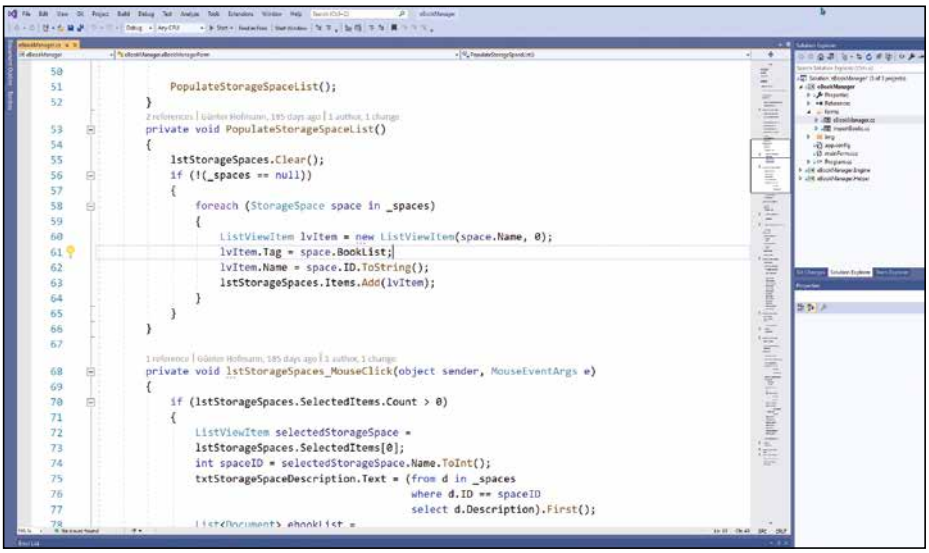


Abb. 1: Architektur von Wisej



```
        PopulateStorageSpaceList();
    }
    private void PopulateStorageSpaceList()
    {
        lstStorageSpaces.Clear();
        if (!(_spaces == null))
        {
            foreach (StorageSpace space in _spaces)
            {
                ListViewItem lvItem = new ListViewItem(space.Name, 0);
                lvItem.Tag = space.BookList;
                lvItem.Name = space.ID.ToString();
                lstStorageSpaces.Items.Add(lvItem);
            }
        }
    }

    private void lstStorageSpaces_MouseClick(object sender, MouseEventArgs e)
    {
        if (lstStorageSpaces.SelectedItems.Count > 0)
        {
            ListViewItem selectedStorageSpace =
            lstStorageSpaces.SelectedItems[0];
            int spaceID = selectedStorageSpace.Name.ToInt();
            txtStorageSpaceDescription.Text = (from d in _spaces
                                               where d.ID == spaceID
                                               select d.Description).First();
```

Abb. 2: Codestruktur des eBookManager

Alle Elemente bindet Wisej in einem durchgängigen Distributed Object Model (DOM) ein, das weitgehend dem Objektmodell der WinForms-Programmierung unter .NET entspricht. Dazu synchronisieren sich Browserclient und Serverkomponenten in Echtzeit über WebSockets mittels komprimierter JSON-Pakete. Auf der Serverseite sorgt Wisej per DOM für die Anbindung an die dortige Geschäftslogik und Datenbankdienste (**Abb. 1**).

Mit dieser internen Architektur von Wisej müssen sich Entwickler und Anwender aber nicht auseinandersetzen: Die Einbindung von Qooxdoo wie auch die permanente Synchronisation des DOM zwischen Client und Server erfolgen vollständig gekapselt innerhalb des Frameworks, ohne dass manuelle Eingriffe nötig sind. Dabei ist die Implementierung ausgesprochen effizient: Die anfangs an den Clientbrowser übertragenen JavaScript-Dateien umfassen gerade einmal rund 500 KB Code; die während der Laufzeit übertragenen Datenmengen beschränken sich auf wenige hundert Bytes pro Interaktion. Somit wird auch eine langsame Internetverbindung nicht zum Handicap.

Die Probe aufs Exempel

Nun wollen wir uns einmal ansehen, wie man damit in der Praxis eine WinForm-Anwendung ins Web migrieren kann. Da kein wirklicher Plattformwechsel stattfindet – auch die alte Anwendung läuft ja bereits auf .NET –, sprechen wir statt von Migration korrekt von „Web-Enabling“. Die Grundlage bildet eine gängige WinForms-Beispielapplikation: der eBookManager, eine Visual Studio Solution mit zwei Fenstern, zweischichtig als typisch monolithische Desktopapplikation entwickelt (**Abb. 2**). Sie besteht aus drei Projekten mit einem Main Project, das die wesentliche Funktionalität hinter jedem der Fenster realisiert. Es werden Events abonniert und die entsprechenden Handler implementiert. Es wird auf das Dateisystem zugegriffen. Und es werden Funktionen ausgeführt, die bei jeder Änderung in einem Datenfeld ein Update eines anderen Controls durchführen.

Wenn wir die Applikation starten, ist gut zu erkennen, was sie tut: Auf der linken Seite erscheinen sogenannte Storage Spaces, in denen E-Books verwaltet werden. Diese sind auf der lokalen Festplatte gespeichert und können in der Anwendung strukturiert abgelegt und mit zusätzlichen Informationen angereichert werden. Dafür sind unterschiedliche Datenfelder vorgesehen, beispielsweise vom Typ Text für Titel und Autor, vom Typ Zahl für den Preis oder auch ein Kalender-Control für das Erscheinungsdatum. Klickt man ein Buch an, wird die Datei jeweils von der lokalen Festplatte geöffnet. Schließlich kann man in einem modalen Dialog auf der Festplatte gespeicherte E-Books hinzufügen und dafür auch neue Storage Spaces anlegen. So weit die Grundfunktionalität (**Abb. 3**). Auch wenn die Anwendung übersichtlich erscheint, enthält sie doch einige Stolpersteine für das Web-Enabling, wegen der wir sie ausgewählt haben – dazu später mehr.

Immer hübsch der Reihe nach

Jetzt wollen wir erst einmal aus dem WinForms-Projekt ein Wisej-Projekt machen. Der erste Schritt ist, Wisej zu installieren und ein neues Projekt vom Typ WISEJ APPLICATION anzulegen. Aus dessen *CSPROJ*-Datei kopieren wir nun die Importe und *ProjectTypeGuids* in unsere eigene Datei und ändern dort zu guter Letzt den *OutputType* von *WinExe* auf *Library*. Damit sind in etwa die Zeilen in Listing 1 vorhanden.

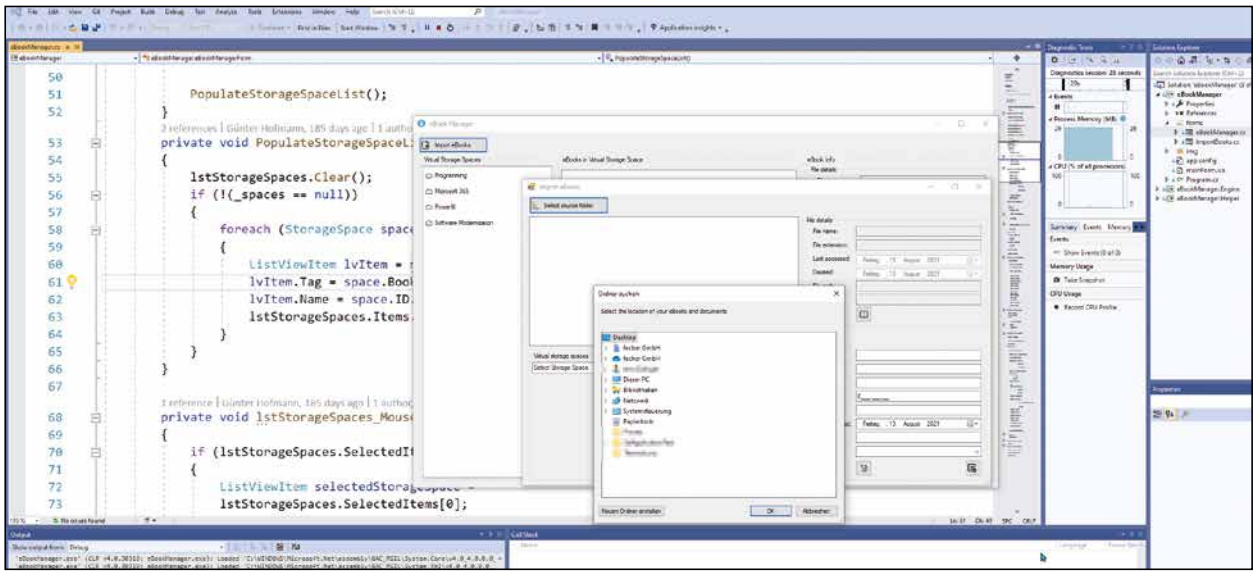


Abb. 3: Der eBook-Manager in Aktion

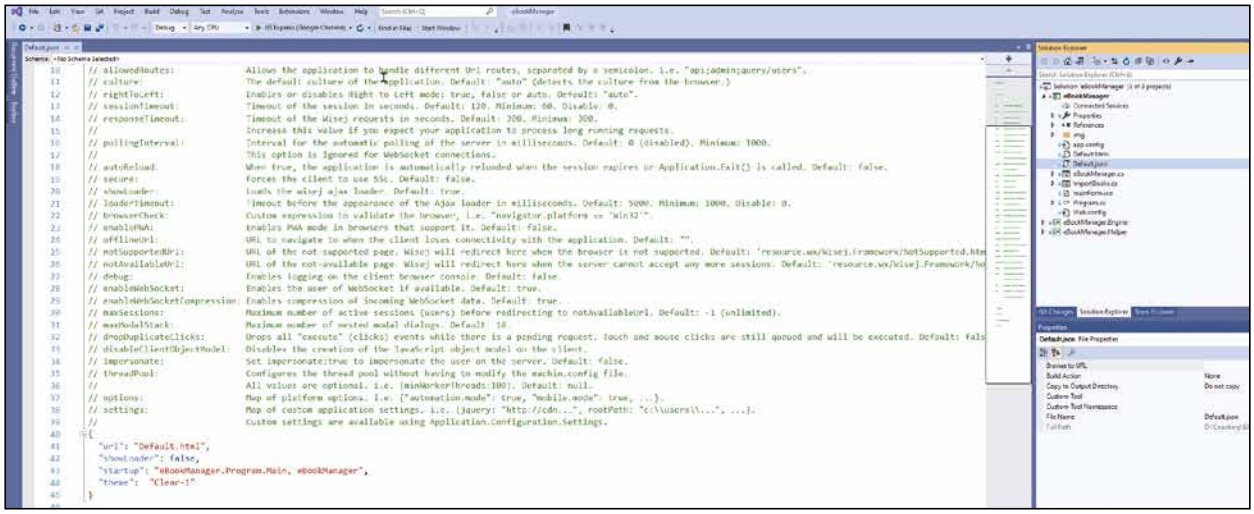


Abb. 4: Konfigurationseinstellungen in Default.json

Weiterhin benötigt jede Wisej-Applikation eine *Default.html*, eine *Default.json* und eine *Web.config*. Für alle drei nehmen wir die Vorlagen aus dem zuvor angelegten leeren Projekt und ziehen sie per Drag and Drop in das Main Project. *Default.json* öffnen wir gleich (**Abb. 4**), fügen am Ende die Start-up-Routine ein, mit der das Projekt bei Ausführung starten soll, und definieren *Clear-1* als das zu verwendende Theme.

Damit sind die Vorbereitungen weitgehend abgeschlossen – bis auf einen wichtigen Schritt: Überall im Quellcode steht noch *System.Windows.Forms* in den Usings. Das ist ein Fall für Suchen und Ersetzen, bei dem einfach jedes Vorkommen von *System.Windows.Forms* durch *Wisej.Web* ersetzt wird.

Jetzt heißt es noch schnell, den Aufruf der Software anzupassen, der in *program.cs* ganz im Stil einer WinForms-Applikation definiert ist. Dort ersetzen wir deshalb den vorhandenen Code:

```
static void Main()
{
  Application.EnableVisualStyles();
  Application.SetCompatibleTextRenderingDefault(false);
  Application.Run(new eBookManagerForm());
}
```

durch einen Wisej-konformen Aufruf:

```
static void Main()
{
  eBookManagerForm eBookManagerForm = new eBookManagerForm();
  eBookManagerForm.Show();
}
```

Listing 1

```
<Import Project=" $(VSToolsPath)\WebApplications\Microsoft.WebApplication.targets"
                                        Condition="$' (VSToolsPath)' != "" />
<Import Project="$(MSBuildExtensionsPath32)\Microsoft\VisualStudio\v14.0\
          WebApplications\Microsoft.WebbApplications.targets" Condition="false" />
<ProjectTypeGuids>{349c5851-65df-11da-9384-00065b846f21};{
                          fae04ec0-301f-11d3-bf4b-00c04f79efbc}</ProjectTypeGuids>
<OutputType>Library</OutputType>
```

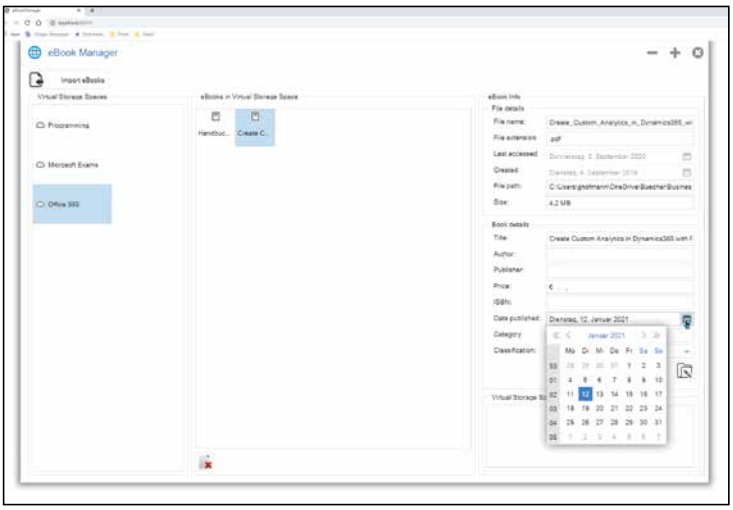


Abb. 5: Der eBookManager als Browseranwendung

Erst mal kompilieren

Nun sind wir an dem Punkt angekommen, das Projekt zum ersten Mal kompilieren und schauen zu können, welche Fehler noch nachbearbeitet werden müssen. In unserem Fall sind es 16 Stück. Sortiert man diese nach der Description, sieht man sehr schnell, dass sich allein acht Fehler auf die Verwendung von *UseVisualStyleBackColor* beziehen. Da die visuelle Erscheinung der Webanwendung ab sofort per Theme gesteuert wird, können wir mit Suchen und Ersetzen und einer Regular Expression alle Vorkommnisse von *UseVisualStyleBackColor* einfach direkt im Quellcode auskommentieren. Die weiteren Fehlermeldungen arbeiten wir einzeln ab. Es handelt sich um einige weitere visuelle Einstellungen, die im neuen Kontext nicht mehr relevant sind und ebenfalls auskommentiert werden können. Übrig bleiben eine Eigenschaft *TileSize*, die in Wisej *ItemSize* heißt und schnell umbenannt ist, sowie ein Icon, das ein Bitmap-Format erwartet, das durch Hinzufügen der Methode *.ToBitmap()* erzwungen werden kann.

Bin ich schon drin?

Nach diesen Korrekturen ist der Kompilierlauf erfolgreich und das Projekt lässt sich problemlos starten. Tatsächlich öffnet sich ein Browserfenster und es erscheint darin eine Anwendung, die unserem ursprünglichen eBookManager sehr ähnlich sieht (**Abb. 5**).

Wie gehabt finden wir auf der linken Seite die verschiedenen Storage Spaces und in der Mitte die zugehörigen E-Books. Auch die Funktionalität ist vorhanden. Wir können rechts die Informationen zu einem E-Book ändern und sehen die Auswirkung sofort im mittleren Bereich. Dabei wird die Tastaturbedienung voll unterstützt, die Datumsauswahl geschieht per Kalender-Control, und selbst der modale Dialog zum Hinzufügen eines E-Books verhält sich wie zuvor. Sogar die Fenstergröße lässt sich vom Benutzer durch einfaches Ziehen verändern und die Controls passen ihre Position entsprechend der vorgegebenen Verankerung an.

Trotzdem lässt sich das Erscheinungsbild noch verbessern. Dazu verwendet man den Visual Designer von Wisej, der sich bei dessen Installation automatisch in Visual Studio integriert hat (**Abb. 6**). Wenn wir das Fenster zum Bearbeiten öffnen, können wir den *FormBorderStyle* von *sizeable* auf *none* und den *WindowState* von *Normal* auf *Maximized* umstellen, damit das Anwendungsfenster das Browserfenster grundsätzlich ohne Umrandung komplett ausfüllt. Natürlich lassen sich mit dem Designer auch Controls verschieben, umdefinieren, weitere Controls einfügen oder in Zukunft ganz neue Fenster erstellen.

Noch lange nicht alles

Sollte das schon alles gewesen sein, was zum Web-Enabling einer WinForms-Anwendung erforderlich ist? Ganz sicher nicht! Denn auch wenn unser kleines Beispielprojekt jetzt zuverlässig und schnell im Browser läuft, ist es doch noch keine echte Webanwendung.

Da wären zunächst einmal die lokal gespeicherten E-Books. Zwar lassen sie sich auch in der Browser-App problemlos anzeigen, aber nur, solange diese lokal auf dem Rechner ausgeführt wird, in dessen Dateisystem die E-Books liegen. Auf jedem anderen Client wird sie nicht funktionieren. Soll die Anwendung nur firmenintern im Intranet verwendet werden, lässt sich relativ leicht Abhilfe schaffen, indem man die Speicherung auf einen für alle Nutzer verfügbaren Networkshare verlagert. Bei Einsatz im Web führt allerdings kein Weg an einem Online-Storage wie etwa Azure Blob Storage vorbei, in den die E-Books beim Einrichten per Upload gespeichert und aus dem sie zum Betrachten heruntergeladen werden. Für all das bringt Wisej die passenden Mechanismen mit, deren Einsatz aber Codeanpassungen erfordert.

Ein anderes Feld, in dem beim Web-Enabling regelmäßig Code angepasst werden muss, sind die statischen Variablen (Statics). Von WinForms-Anwendungen sind die Entwickler gewohnt, dass jede Instanz der Anwendung ihre eigene Ausführungsumgebung besitzt und jeder Benutzer daher seinen eigenen Satz statischer Variablen zur Verfügung hat. Das ist in einer Webumgebung aber nicht mehr der Fall. Hier bedient die Ausführungsumgebung alle gleichzeitigen Nutzer der Anwendung. Man muss sich also grundsätzlich Gedanken darüber machen, welche Informationen in Statics gespeichert sind, wo sich Probleme aus der gemeinsamen Nutzung ergeben und wie sich diese lösen lassen.

Auch Controls von Drittanbietern verursachen häufig Kopfschmerzen. Während Wisej für alle mitgelieferten WinForms-Standard-Controls entsprechende Äquivalente bietet, ist das für die zugekauften naturgemäß nicht der Fall. Setzt eine Anwendung auf solche Fremd-Controls, muss sehr genau überlegt werden, wie sich die entsprechende Funktionalität im Browser zur Verfügung stellen lässt. Steht eine Webversion des Controls zur Verfügung? Gibt es andere webfähige Controls, die als Alternative dienen können? Oder muss die Funktionalität möglicherweise sogar selbst nachprogrammiert werden? Wie auch immer die Entscheidung ausfällt, re-

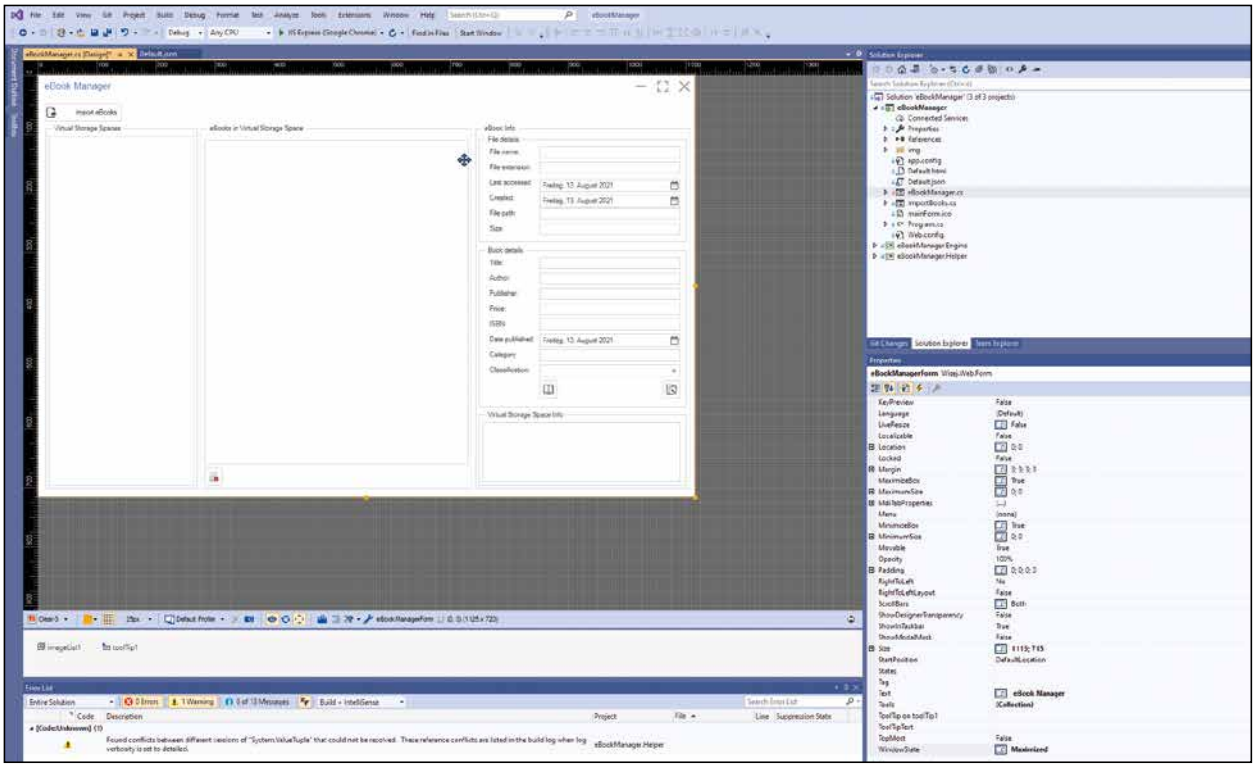


Abb. 6: Der Visual Designer von Wisej

sultieren daraus zwingend Codeanpassungen – entweder nimmt man sie an allen Stellen vor, an denen das Fremd-Control zum Einsatz kommt, oder man baut sich einen Wrapper, der die ursprüngliche Signatur nachbildet und dahinter die neue Lösung implementiert.

Schließlich ist auch der Aufwand für die manuelle Nachbearbeitung nicht zu vernachlässigen. Wie wir gesehen haben, sind die pro Fenster anfallenden Aufgaben meist überschaubar. Wer jedoch Anwendungen mit Hunderten oder Tausenden von Fenstern einem Web-Enabling unterzieht, kommt bald auf die Idee, diese Schritte lieber zu automatisieren. Im Unternehmen des Autors ist daher mit dem winformPORTER [3] ein hausinternes Werkzeug entstanden. Mit einem anpassbaren Satz von Transformationsregeln und einer Bibliothek zeitsparender Zusatzfunktionen bündelt es das Know-how aus rund 50 Web-Enabling-Projekten und wird mit jedem weiteren Projekt ergänzt. So hilft das Werkzeug, die kleinen syntaktischen Fallstricke zu umgehen, ein Instance-Manager löst das Problem der Statics grundlegend und für viele gängige Fremd-Controls sind bereits die möglichen Ersetzungen vorgesehen. Nicht zuletzt sind auch solche Funktionen enthalten, die dabei helfen, das Web-Enabling gleich mit einem Redesign der Benutzeroberfläche zu verbinden.

Fazit

Ob in einem kleinen Projekt manuell nachbearbeitet oder in größeren Projekten mit zusätzlicher Werkzeugunterstützung: Dank der Kompatibilität zu WinForms bietet Wisej eine robuste und effiziente Grundlage für das Web-Enabling vorhandener .NET-Anwendungen, ohne diese neu schreiben zu müssen. Verglichen mit einer Neuentwicklung benötigt ein solches Web-Enabling nur rund 10 bis 15 Prozent des Aufwands und garantiert so erhebliche Kosteneinsparungen sowie wesentlich schnellere Umsetzungszeiten.

Wichtig für die Weiterentwicklung: Im Gegensatz zu Blazor Server und anderen vergleichbaren Ansätzen, die ebenfalls eine Stateful-Architektur für die Webentwicklung unter .NET zur Verfügung stellen, kommt Wisej vollständig ohne JavaScript im Anwendungscode aus. Wer sich selbst von der Leistungsfähigkeit des Frameworks überzeugen möchte, findet unter www.wisej.com eine kostenlose Testversion mit vollem Leistungsumfang.

Jens Eidinger ist .NET-Entwickler, Trainer und Senior Consultant bei der fecher GmbH. Bereits seit zehn Jahren wirkt er tatkräftig bei der Modernisierung von Geschäftssoftware für internationale Unternehmen mit und schult die fecher-Kunden in der Verwendung von Wisej.

Links & Literatur

[1] https://wisej.com

[2] https://qooxdoo.org

[3] https://www.modernizing-applications.de/unsere-services/web-enabling/#winformporter