AmazingGeoRace

MOC5-Projekt

Stefan Kert

6. Januar 2016

1 Allgemeines

Die Plattform, welche für die Projektarbeit ausgewählt wurde, ist Windows Phone 8.1. Es gibt bei der Version 8.1 der Plattform zahlreiche Vorteile, wie zum Beispiel eine bessere Unterstützung des .Net Frameworks und im Allgemeinen ist es zukunftsorientierter gestaltet. Als IDE wurde Visual Studio 2015 in der Enterprise Version verwendet und getestet wurde die App auf einem Emulator mit WP 8.1 und einem Emulator mit WP 10.

1.1 Tombstoning in WP 8.1

Bei der Architektur von WP 8.1 hat sich Microsoft für einen ähnlichen Weg für das Lifecycle Management entschieden wie beim WP 8. Einer der größten Unterschiede ist die Tatsache, dass Apps nicht mehr direkt beendet oder deaktiviert werden, sondern in den sogenannten *Suspended*-Status versetzt werden. Bei Speicherknappheit, kann es vorkommen, dass die App denoch komplett beendet wird. Dies geschieht wie auch beim WP 8 ohne Vorwarnung und daher sollte sichergestellt werden, dass bei jedem Übergang in den *Suspended*-Status alle benötigten Daten gespeichert werden. Für diese Zwecke können in WP 8.1 die in der Klasse Application vorhandenen Methoden *OnLaunched* überschrieben, bzw. *OnSupsended* auf das *Suspended* Event gebunden werden. Dort wird ähnlich wie beim WP 8 auf ein *SessionState*-Dictionary zugegriffen, welches schließlich im System persistiert wird. Sobald das *OnLaunched* Event aufgerufen wird, können die Daten aus diesem Dictionary erneut geladen werden. Weiters gibt es die Möglichkeit zum Speichern der aktuell geöffneten Frame, sodass diese beim erneuten Öffnen wieder geöffnet wird.

2 Lösungsidee

Bei der Lösung wird auf eine möglichst gute Trennung der einzelnen Aspkete geachtet. Es wird dabei das in der .NET Programmierungsumgebung etablierte MVVM Pattern verwendet. Durch dieses wird eine gute Trennung zwischen View und Logik gewährleistet. Weiters wird eine eigene Klasse für die Interaktion mit dem Webservice implementiert.

2.1 Anwendungsarchitektur

Grundsätzlich besteht die AmazingGeoRace-App aus 3 Seiten und mehreren Dialogen:

- LoginPage - Hier wird dem User die Möglichkeit gegeben sich anzumelden. Dabei wird bei falschen Logindaten eine Fehlermeldung ausgegeben und bei erfolgreicher Anmeldung weitergeleitet. Nachdem erfolgreicher Anmeldung werden die Logindaten bei jedem Beenden der App persistiert und der Benutzer kann so eingeloggt bleiben.
- MainPage: Hier werden alle für den Benutzer vorhandenen Routen aufgelistet und der Benutzer kann sich eine Route aussuchen. Es wird dem Benutzer außerdem mitgeteilt, ob eine Route bereits erfolgreich abgeschlossen wurde. Dies erfolgt durch einen grünen Haken neben dem Namen der Route. Weiters gibt es die Möglichkeit alle Routen zurückzusetzen und sich auszuloggen.
- RaceDetailsPage: Nach der Auswahl einer Route wird auf diese Seite weitergeleitet. Auf dieser Seite hat der Benutzer die Möglichkeit die einzelnen Checkpoints auf einer Karte zu sehen. Weiters wird dem Benutzer der Name, sowie ein Hinweis für den aktuellen Checkpoint angezeigt. Wenn der Benutzer eine Lösung angeben möchte gibt es dafür einen Button. Durch ein tippen auf diesen Button kommt der Benutzer zum SolutionDialog wo er die Lösung angeben kann. Bei erfolgreicher Eingabe im SolutionDialog bekommt der Benutzer eine Erfolgsmeldung und der nächste Checkpoint wird angezeigt. Es werden außerdem für alle bereits passierten Checkpoints Punkte bzw. Linien zwischen den einzelnen Checkpoints auf der Karte eingezeichnet. Bei falscher Eingabe wird der Benutzer darüber informiert, dass die angegebene Lösung falsch war. Wenn der Benutzer den letzten Checkpoint erfolgreich freigeschalten hat bekommt er eine Erfolgsmeldung und das Interface der RaceDetailsPage ändert sich, sodass jetzt nur noch die Karte und ein Button zum Zurücksetzen der aktuellen Route vorhanden sind. Bei Tippen auf diesen Button wird die aktuelle Route zurückgesetzt und die Seite neu geladen und die Schnitzeljagd kann von neuem beginnen.

2.2 Implementierungsdetails

Architektur

Wie bereits erwähnt ist die Anwendung mit dem für .NET Applikationen üblichen MVVM Pattern umgesetzt, wo die View über das ViewModel bescheid weiß, das ViewModel aber keine Referenz auf dei View besitzt. Das ViewModel wiederum weiß bescheid, wie es die

Daten auslesen kann. Dies wird über den ServiceProxy erledigt, welcher wiederum auf den Webservice zugreift, welcher den Zugriff auf den Webservice kapselt. Ein weiterer wichtiger Bestandteil ist die Klasse Loginservice, welche weiderum vom ServiceProxy abhängt und die Funktionalität zum Authentifizieren und zum Speichern der Logindaten bietet. Die beschrieben Architektur ist in Abb. 1 dargestellt.

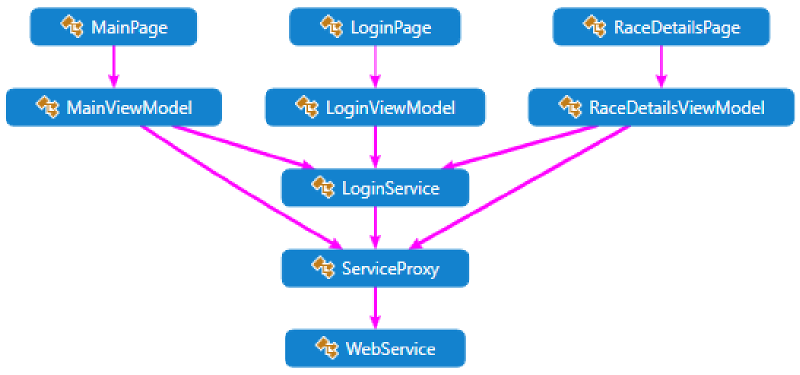


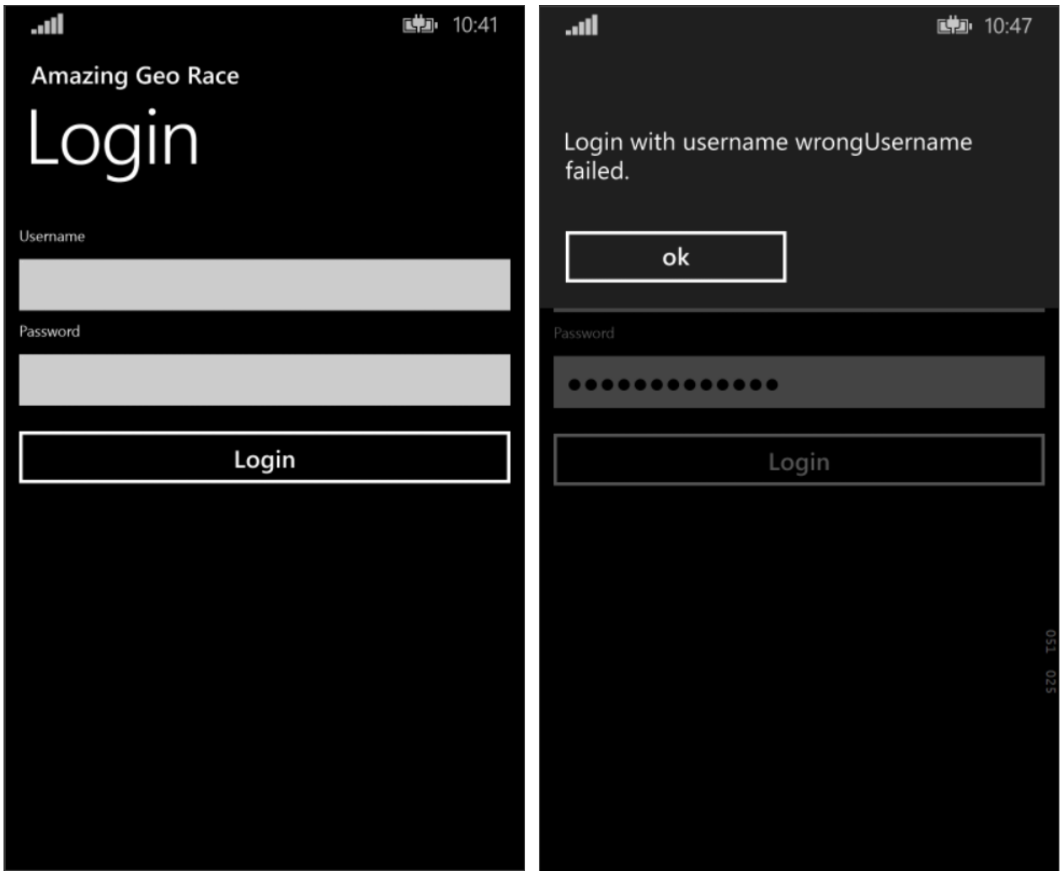
Abbildung 1: Architektur

3 Testfälle und Screenshots

In diesem Abschnitt werden Screenshots für die einzelnen Testfälle gezeigt. Diese sind dabei nach Seiten gruppiert.

3.1 LoginPage

Für die Loginpage gibt es nur zwei Testfälle. Einerseits, dass der Login erfolgreich war, dann wird die MainPage angezeigt. Andererseits, dass der Login nicht erfolgreich war, dann wird das in Abb. 2 rechts dargestellte Fenster angezeigt. Der Basisdialog zur Eingabe befindet sich in Abb. 2 links.



Login with username wrongUsername failed.

Abbildung 2: Login

3.2 MainPage

Beim erstmaligen Öffnen der Mainpage sollte das in Abb. 3 ganz links dargestellte Bild angezegit werden. Es sind noch keine Routen abgeschlossen und daher wird dem Benutzer auch kein Erfolgszeichen angezeigt. Das mittlere Bild stellt den Fall dar, dass der Benutzer bereits eine Route abgeschlossen hat, es wird ein grünes Häckchen angezeigt. Beim ganz rechten Bild handelt es sich um den Fall, dass der Benutzer die Routen zurückgesetzt hat. Bei Erfolg wird er darüber informiert, dass alle Routen zurückgesetzt sind. Beim Klicken des Logout Buttons wird der Benutzer zum Homescreen des WP weitergeleitet.

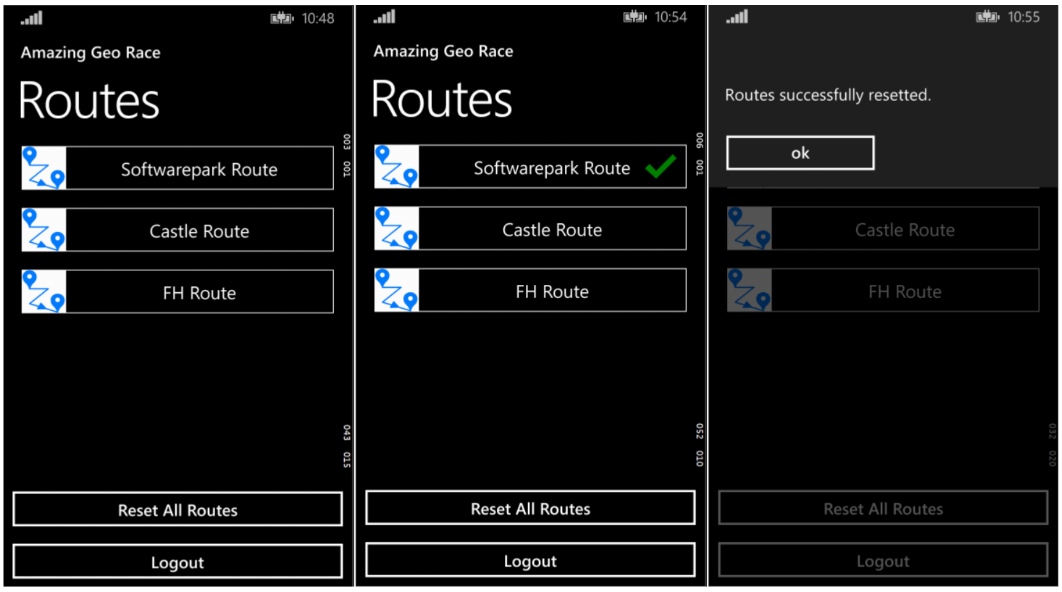


Abbildung 3: MainPage

3.3 RaceDetailsPage

Nachdem Auswählen einer Route auf der Mainpage wird dem Benutzer das in Abb. 4 ganz links angezeigte Fenster angezeigt, falls der Benutzer diese Route noch nicht erfolgreich abgeschlossen hat. Wenn alle Checkpoints freigeschalten wurden und das Ziel erreicht wurde, oder falls er eine bereits abgeschlossene Route noch einmal öffnet, wird dem Benutzer die in der Mitte der Abb. 4 gezeigte Seite angezeigt. Auf dieser Seite kann der Benutzer wiederum die aktuelle Route zurücksetzen. Bei Erfolg wird dem Benutzer die ganz rechts dargestellte Seite angezeigt. Wenn der Benutzer auf den *ProvideSolution* Button klickt, wird ihm der SolutionDialog angezeigt.

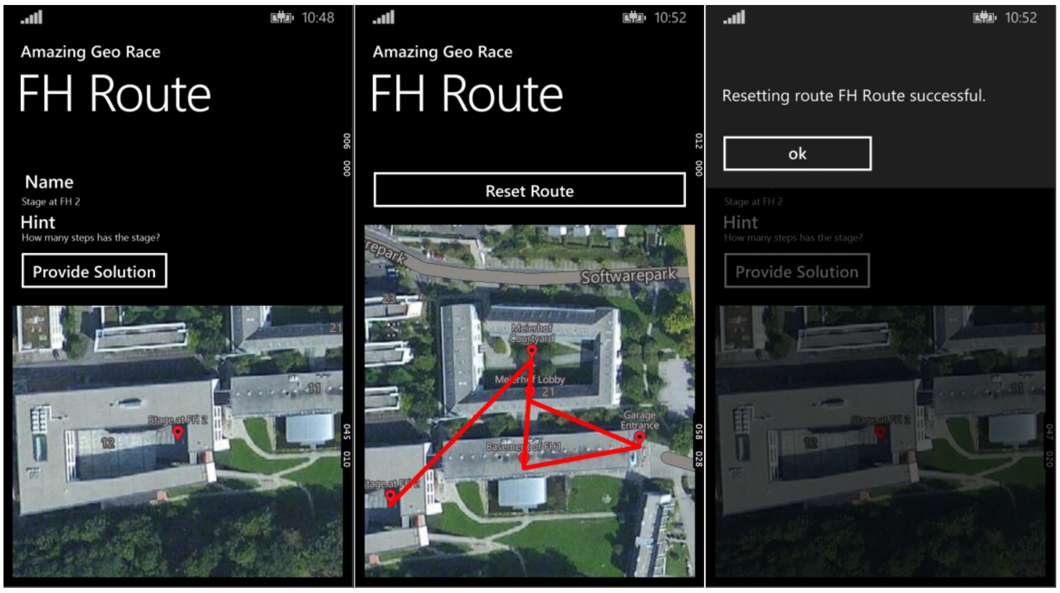
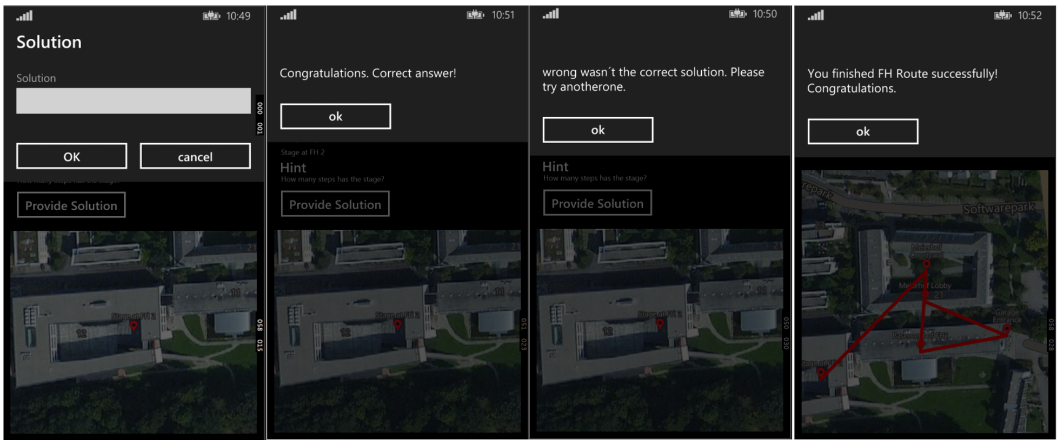


Abbildung 4: RaceDetailsPage

3.4 SolutionDialog

Beim Öffnen des SolutionDialogs wird dem Benutzer ein Overlay mit leerem Textfeld und zwei Buttons angezeigt, wo die Lösung eingegeben werden kann. Bei Erfolg wird dem Benutzer die Meldung *Congratulations. Correct answer!* ausgegeben. Bei einer falschen Lösung wird der Benutzer mit einer Fehlermeldung darüber informiert, dass diese Lösung falsch war. Sobald der Benutzer die Lösung für den letzten Checkpoint richtig angegeben hat, wird er darüber informiert, dass die Route erfolgreich abgeschlossen wurde. Diese Fälle sind in 5 dargestellt.


Congratulations. Correct answer!

wrong wasn´t the correct solution. Please try anotherone.

You finished FH Route successfully! Congratulations.

Abbildung 5: SolutionDialg

4 Quellcode

4.1 Commands

```
using System;
using System.Windows.Input;

namespace AmazingGeoRace.Commands
{
    public class RelayCommand: ICommand
    {
        private readonly Action<object> _action;
        private readonly Func<bool> _canExecute;


        public event EventHandler CanExecuteChanged;

        public RelayCommand(Action<object> action): this(action,
           () => true) {}

        public RelayCommand(Action<object> action, Func<bool>
           canExecute) {
            _action = action;
            _canExecute = canExecute;
        }

        public bool CanExecute(object parameter) {
            return _canExecute();
        }

        public void Execute(object parameter) {
            _action(parameter);
        }

        public void RaiseCanExecuteChanged() {
            var handler = CanExecuteChanged;
            handler?.Invoke(this, EventArgs.Empty);
        }
    }
}
```

Listing 1: RelayCommand.cs

4.2 Common

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using Windows.UI.Popups;

namespace AmazingGeoRace.Common
{
    public static class ExceptionHandling
    {
        public static async Task HandleException(Action
            methodToHandle) {
            try {
                methodToHandle();
            }
            catch (Exception ex) {
                var dialog = new MessageDialog(ex.Message,
                    "Error");
                await dialog.ShowAsync();
            }
        }

        public static async Task
            HandleExceptionForAsyncMethod(Func<Task>
            methodToHandle)
        {
            try
            {
                await methodToHandle();
            }
            catch (Exception ex)
            {
                var dialog = new MessageDialog(ex.Message,
                    "Error");
                await dialog.ShowAsync();
            }
        }
    }
}
```

Listing 2: ExceptionHandling.cs

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Windows.UI.Popups;

namespace AmazingGeoRace.Common
{
    public static class MessageBoxWrapper
    {
        public static async Task ShowAsync(string message) {
            var d = new MessageDialog(message);
            await d.ShowAsync();
        }

        public static async Task ShowOkAsync(string message)
        {
            var d = new MessageDialog(message);
            d.Commands.Add(new UICommand("OK") {
                Id = 0
            });
            await d.ShowAsync();
        }


        public static async Task ShowAsync(string message, string
           title) {
            var d = new MessageDialog(message, title);
            await d.ShowAsync();
        }
    }
}
```

Listing 3: MessageBoxWrapper.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Input;
using Windows.System;
using Windows.UI.Core;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Navigation;
using AmazingGeoRace.Commands;

namespace AmazingGeoRace.Common
{
    /// <summary>
    /// NavigationHelper aids in navigation between pages.  It
       provides commands used to
    /// navigate back and forward as well as registers for
       standard mouse and keyboard
    /// shortcuts used to go back and forward in Windows and the
```

```
   hardware back button in
/// Windows Phone.  In addition it integrates
   SuspensionManger to handle process lifetime
/// management and state management when navigating between
   pages.
/// </summary>
/// <example>
/// To make use of NavigationHelper, follow these two steps or
/// start with a BasicPage or any other Page item template
   other than BlankPage.
///
/// 1) Create an instance of the NavigationHelper somewhere
   such as in the
///     constructor for the page and register a callback for
   the LoadState and
///     SaveState events.
/// <code>
///     public MyPage()
///     {
///         this.InitializeComponent();
///         var navigationHelper = new NavigationHelper(this);
///         this.navigationHelper.LoadState +=
   navigationHelper_LoadState;
///         this.navigationHelper.SaveState +=
   navigationHelper_SaveState;
///     }
///
///     private async void navigationHelper_LoadState(object
   sender, LoadStateEventArgs e)
///     { }
///     private async void navigationHelper_SaveState(object
   sender, LoadStateEventArgs e)
///     { }
/// </code>
///
/// 2) Register the page to call into the NavigationHelper
   whenever the page participates
///     in navigation by overriding the <see
   cref="Windows.UI.Xaml.Controls.Page.OnNavigatedTo"/>
///     and <see
   cref="Windows.UI.Xaml.Controls.Page.OnNavigatedFrom"/>
   events.
/// <code>
///     protected override void
   OnNavigatedTo(NavigationEventArgs e)
///     {
///         navigationHelper.OnNavigatedTo(e);
```

```
    ///      }
    ///
    ///      protected override void
        OnNavigatedFrom(NavigationEventArgs e)
    ///      {
    ///          navigationHelper.OnNavigatedFrom(e);
    ///      }
    /// </code>
    /// </example>
    [Windows.Foundation.Metadata.WebHostHidden]
    public class NavigationHelper : DependencyObject
    {
        private Page Page { get; set; }
        private Frame Frame { get { return this.Page.Frame; } }

        /// <summary>
        /// Initializes a new instance of the <see
            cref="NavigationHelper"/> class.
        /// </summary>
        /// <param name="page">A reference to the current page
            used for navigation.
        /// This reference allows for frame manipulation and to
            ensure that keyboard
        /// navigation requests only occur when the page is
            occupying the entire window.</param>
        public NavigationHelper(Page page)
        {
            this.Page = page;

            // When this page is part of the visual tree make two
                changes:
            // 1) Map application view state to visual state for
                the page
            // 2) Handle hardware navigation requests
            this.Page.Loaded += (sender, e) =>
            {
#if WINDOWS_PHONE_APP
                Windows.Phone.UI.Input.HardwareButtons.BackPressed
                    += HardwareButtons_BackPressed;
#else
                // Keyboard and mouse navigation only apply when
                    occupying the entire window
                if (this.Page.ActualHeight ==
                    Window.Current.Bounds.Height &&
                     this.Page.ActualWidth ==
                        Window.Current.Bounds.Width)
                {
                    // Listen to the window directly so focus
                       isn't required
                    Window.Current.CoreWindow.Dispatcher.AcceleratorKeyActivated
                       +=
                        CoreDispatcher_AcceleratorKeyActivated;
                    Window.Current.CoreWindow.PointerPressed +=
                        this.CoreWindow_PointerPressed;
                }
#endif
            };

            // Undo the same changes when the page is no longer
               visible
            this.Page.Unloaded += (sender, e) =>
            {
#if WINDOWS_PHONE_APP
                Windows.Phone.UI.Input.HardwareButtons.BackPressed
                   -= HardwareButtons_BackPressed;
#else
                Window.Current.CoreWindow.Dispatcher.AcceleratorKeyActivated
                   -=
                    CoreDispatcher_AcceleratorKeyActivated;
                Window.Current.CoreWindow.PointerPressed -=
                    this.CoreWindow_PointerPressed;
#endif
            };
        }

        #region Navigation support

        RelayCommand _goBackCommand;
        RelayCommand _goForwardCommand;

        /// <summary>
        /// <see cref="RelayCommand"/> used to bind to the back
           Button's Command property
        /// for navigating to the most recent item in back
           navigation history, if a Frame
        /// manages its own navigation history.
        ///
        /// The <see cref="RelayCommand"/> is set up to use the
           virtual method <see cref="GoBack"/>
        /// as the Execute Action and <see cref="CanGoBack"/> for
           CanExecute.
        /// </summary>
        public RelayCommand GoBackCommand
        {
```

```
        get
        {
            if (_goBackCommand == null)
            {
                _goBackCommand = new RelayCommand(
                    (obj) => this.GoBack(),
                    () => this.CanGoBack());
            }
            return _goBackCommand;
        }
        set
        {
            _goBackCommand = value;
        }
    }
    /// <summary>
    /// <see cref="RelayCommand"/> used for navigating to the
       most recent item in
    /// the forward navigation history, if a Frame manages
       its own navigation history.
    ///
    /// The <see cref="RelayCommand"/> is set up to use the
       virtual method <see cref="GoForward"/>
    /// as the Execute Action and <see cref="CanGoForward"/>
       for CanExecute.
    /// </summary>
    public RelayCommand GoForwardCommand
    {
        get
        {
            if (_goForwardCommand == null)
            {
                _goForwardCommand = new RelayCommand(
                    (obj) => this.GoForward(),
                    () => this.CanGoForward());
            }
            return _goForwardCommand;
        }
    }

    /// <summary>
    /// Virtual method used by the <see
       cref="GoBackCommand"/> property
    /// to determine if the <see cref="Frame"/> can go back.
    /// </summary>
    /// <returns>
    /// true if the <see cref="Frame"/> has at least one entry
```

```
        /// in the back navigation history.
        /// </returns>
        public virtual bool CanGoBack()
        {
            return this.Frame != null && this.Frame.CanGoBack;
        }
        /// <summary>
        /// Virtual method used by the <see
           cref="GoForwardCommand"/> property
        /// to determine if the <see cref="Frame"/> can go
           forward.
        /// </summary>
        /// <returns>
        /// true if the <see cref="Frame"/> has at least one entry
        /// in the forward navigation history.
        /// </returns>
        public virtual bool CanGoForward()
        {
            return this.Frame != null && this.Frame.CanGoForward;
        }

        /// <summary>
        /// Virtual method used by the <see
           cref="GoBackCommand"/> property
        /// to invoke the <see
           cref="Windows.UI.Xaml.Controls.Frame.GoBack"/> method.
        /// </summary>
        public virtual void GoBack()
        {
            if (this.Frame != null && this.Frame.CanGoBack)
               this.Frame.GoBack();
        }
        /// <summary>
        /// Virtual method used by the <see
           cref="GoForwardCommand"/> property
        /// to invoke the <see
           cref="Windows.UI.Xaml.Controls.Frame.GoForward"/>
           method.
        /// </summary>
        public virtual void GoForward()
        {
            if (this.Frame != null && this.Frame.CanGoForward)
               this.Frame.GoForward();
        }

#if WINDOWS_PHONE_APP
        /// <summary>
        /// Invoked when the hardware back button is pressed. For
            Windows Phone only.
        /// </summary>
        /// <param name="sender">Instance that triggered the
            event.</param>
        /// <param name="e">Event data describing the conditions
            that led to the event.</param>
        private void HardwareButtons_BackPressed(object sender,
            Windows.Phone.UI.Input.BackPressedEventArgs e)
        {
            if (this.GoBackCommand.CanExecute(null))
            {
                e.Handled = true;
                this.GoBackCommand.Execute(null);
            }
        }
#else
        /// <summary>
        /// Invoked on every keystroke, including system keys
            such as Alt key combinations, when
        /// this page is active and occupies the entire window.
            Used to detect keyboard navigation
        /// between pages even when the page itself doesn't have
            focus.
        /// </summary>
        /// <param name="sender">Instance that triggered the
            event.</param>
        /// <param name="e">Event data describing the conditions
            that led to the event.</param>
        private void
            CoreDispatcher_AcceleratorKeyActivated(CoreDispatcher
            sender,
             AcceleratorKeyEventArgs e)
        {
            var virtualKey = e.VirtualKey;

            // Only investigate further when Left, Right, or the
                dedicated Previous or Next keys
            // are pressed
            if ((e.EventType ==
                CoreAcceleratorKeyEventType.SystemKeyDown ||
                 e.EventType ==
                    CoreAcceleratorKeyEventType.KeyDown) &&
                 (virtualKey == VirtualKey.Left || virtualKey ==
                    VirtualKey.Right ||
                 (int)virtualKey == 166 || (int)virtualKey == 167))
            {
```

```
            var coreWindow = Window.Current.CoreWindow;
            var downState = CoreVirtualKeyStates.Down;
            bool menuKey =
               (coreWindow.GetKeyState(VirtualKey.Menu) &
               downState) == downState;
            bool controlKey =
               (coreWindow.GetKeyState(VirtualKey.Control) &
               downState) == downState;
            bool shiftKey =
               (coreWindow.GetKeyState(VirtualKey.Shift) &
               downState) == downState;
            bool noModifiers = !menuKey && !controlKey &&
               !shiftKey;
            bool onlyAlt = menuKey && !controlKey &&
               !shiftKey;

            if (((int)virtualKey == 166 && noModifiers) ||
                (virtualKey == VirtualKey.Left && onlyAlt))
            {
                // When the previous key or Alt+Left are
                   pressed navigate back
                e.Handled = true;
                this.GoBackCommand.Execute(null);
            }
            else if (((int)virtualKey == 167 && noModifiers)
               ||
                (virtualKey == VirtualKey.Right && onlyAlt))
            {
                // When the next key or Alt+Right are pressed
                   navigate forward
                e.Handled = true;
                this.GoForwardCommand.Execute(null);
            }
        }
    }

/// <summary>
/// Invoked on every mouse click, touch screen tap, or
   equivalent interaction when this
/// page is active and occupies the entire window.  Used
   to detect browser-style next and
/// previous mouse button clicks to navigate between
   pages.
/// </summary>
/// <param name="sender">Instance that triggered the
   event.</param>
/// <param name="e">Event data describing the conditions
```

```
           that led to the event.</param>
        private void CoreWindow_PointerPressed(CoreWindow sender,
            PointerEventArgs e)
        {
            var properties = e.CurrentPoint.Properties;

            // Ignore button chords with the left, right, and
               middle buttons
            if (properties.IsLeftButtonPressed ||
               properties.IsRightButtonPressed ||
                properties.IsMiddleButtonPressed) return;

            // If back or foward are pressed (but not both)
               navigate appropriately
            bool backPressed = properties.IsXButton1Pressed;
            bool forwardPressed = properties.IsXButton2Pressed;
            if (backPressed ^ forwardPressed)
            {
                e.Handled = true;
                if (backPressed) this.GoBackCommand.Execute(null);
                if (forwardPressed)
                   this.GoForwardCommand.Execute(null);
            }
        }
#endif

        #endregion

        #region Process lifetime management

        private String _pageKey;

        /// <summary>
        /// Register this event on the current page to populate
           the page
        /// with content passed during navigation as well as any
           saved
        /// state provided when recreating a page from a prior
           session.
        /// </summary>
        public event LoadStateEventHandler LoadState;
        /// <summary>
        /// Register this event on the current page to preserve
        /// state associated with the current page in case the
        /// application is suspended or the page is discarded from
        /// the navigaqtion cache.
        /// </summary>
```

```
public event SaveStateEventHandler SaveState;

/// <summary>
/// Invoked when this page is about to be displayed in a
    Frame.
/// This method calls <see cref="LoadState"/>, where all
    page specific
/// navigation and process lifetime management logic
    should be placed.
/// </summary>
/// <param name="e">Event data that describes how this
    page was reached.  The Parameter
/// property provides the group to be displayed.</param>
public void OnNavigatedTo(NavigationEventArgs e)
{
    var frameState =
        SuspensionManager.SessionStateForFrame(this.Frame);
    this._pageKey = "Page-" + this.Frame.BackStackDepth;

    if (e.NavigationMode == NavigationMode.New)
    {
        // Clear existing state for forward navigation
            when adding a new page to the
        // navigation stack
        var nextPageKey = this._pageKey;
        int nextPageIndex = this.Frame.BackStackDepth;
        while (frameState.Remove(nextPageKey))
        {
            nextPageIndex++;
            nextPageKey = "Page-" + nextPageIndex;
        }

        // Pass the navigation parameter to the new page
        if (this.LoadState != null)
        {
            this.LoadState(this, new
                LoadStateEventArgs(e.Parameter, null));
        }
    }
    else
    {
        // Pass the navigation parameter and preserved
            page state to the page, using
        // the same strategy for loading suspended state
            and recreating pages discarded
        // from cache
        if (this.LoadState != null)
```

```
                {
                    this.LoadState(this, new
                       LoadStateEventArgs(e.Parameter,
                       (Dictionary<String,
                       Object>)frameState[this._pageKey]));
                }
            }
        }

        /// <summary>
        /// Invoked when this page will no longer be displayed in
           a Frame.
        /// This method calls <see cref="SaveState"/>, where all
           page specific
        /// navigation and process lifetime management logic
           should be placed.
        /// </summary>
        /// <param name="e">Event data that describes how this
           page was reached.  The Parameter
        /// property provides the group to be displayed.</param>
        public void OnNavigatedFrom(NavigationEventArgs e)
        {
            var frameState =
               SuspensionManager.SessionStateForFrame(this.Frame);
            var pageState = new Dictionary<String, Object>();
            if (this.SaveState != null)
            {
                this.SaveState(this, new
                   SaveStateEventArgs(pageState));
            }
            frameState[_pageKey] = pageState;
        }

        #endregion
    }

    /// <summary>
    /// Represents the method that will handle the <see
       cref="NavigationHelper.LoadState"/>event
    /// </summary>
    public delegate void LoadStateEventHandler(object sender,
       LoadStateEventArgs e);
    /// <summary>
    /// Represents the method that will handle the <see
       cref="NavigationHelper.SaveState"/>event
    /// </summary>
    public delegate void SaveStateEventHandler(object sender,
    SaveStateEventArgs e);

/// <summary>
/// Class used to hold the event data required when a page
    attempts to load state.
/// </summary>
public class LoadStateEventArgs : EventArgs
{
    /// <summary>
    /// The parameter value passed to <see
       cref="Frame.Navigate(Type, Object)"/>
    /// when this page was initially requested.
    /// </summary>
    public Object NavigationParameter { get; private set; }
    /// <summary>
    /// A dictionary of state preserved by this page during
       an earlier
    /// session.  This will be null the first time a page is
       visited.
    /// </summary>
    public Dictionary<string, Object> PageState { get;
       private set; }

    /// <summary>
    /// Initializes a new instance of the <see
       cref="LoadStateEventArgs"/> class.
    /// </summary>
    /// <param name="navigationParameter">
    /// The parameter value passed to <see
       cref="Frame.Navigate(Type, Object)"/>
    /// when this page was initially requested.
    /// </param>
    /// <param name="pageState">
    /// A dictionary of state preserved by this page during
       an earlier
    /// session.  This will be null the first time a page is
       visited.
    /// </param>
    public LoadStateEventArgs(Object navigationParameter,
       Dictionary<string, Object> pageState)
        : base()
    {
        this.NavigationParameter = navigationParameter;
        this.PageState = pageState;
    }
}
/// <summary>
```

```
    /// Class used to hold the event data required when a page
       attempts to save state.
    /// </summary>
    public class SaveStateEventArgs : EventArgs
    {
        /// <summary>
        /// An empty dictionary to be populated with serializable
           state.
        /// </summary>
        public Dictionary<string, Object> PageState { get;
           private set; }

        /// <summary>
        /// Initializes a new instance of the <see
           cref="SaveStateEventArgs"/> class.
        /// </summary>
        /// <param name="pageState">An empty dictionary to be
           populated with serializable state.</param>
        public SaveStateEventArgs(Dictionary<string, Object>
           pageState)
            : base()
        {
            this.PageState = pageState;
        }
    }
}
```

Listing 4: NavigationHelper.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Windows.Foundation.Collections;

namespace AmazingGeoRace.Common
{
    public class ObservableDictionary : IObservableMap<string,
       object>
    {
        private class ObservableDictionaryChangedEventArgs :
           IMapChangedEventArgs<string>
        {
            public
               ObservableDictionaryChangedEventArgs(CollectionChange
               change, string key)
            {
                this.CollectionChange = change;
                this.Key = key;
    }

    public CollectionChange CollectionChange { get;
       private set; }
    public string Key { get; private set; }
}

private Dictionary<string, object> _dictionary = new
   Dictionary<string, object>();
public event MapChangedEventHandler<string, object>
   MapChanged;

private void InvokeMapChanged(CollectionChange change,
   string key)
{
    var eventHandler = MapChanged;
    if (eventHandler != null)
    {
        eventHandler(this, new
           ObservableDictionaryChangedEventArgs(change,
           key));
    }
}

public void Add(string key, object value)
{
    this._dictionary.Add(key, value);
    this.InvokeMapChanged(CollectionChange.ItemInserted,
       key);
}

public void Add(KeyValuePair<string, object> item)
{
    this.Add(item.Key, item.Value);
}

public bool Remove(string key)
{
    if (this._dictionary.Remove(key))
    {
        this.InvokeMapChanged(CollectionChange.ItemRemoved,
           key);
        return true;
    }
    return false;
}
```

```
public bool Remove(KeyValuePair<string, object> item)
{
    object currentValue;
    if (this._dictionary.TryGetValue(item.Key, out
       currentValue) &&
        Object.Equals(item.Value, currentValue) &&
           this._dictionary.Remove(item.Key))
    {
        this.InvokeMapChanged(CollectionChange.ItemRemoved,
           item.Key);
        return true;
    }
    return false;
}

public object this[string key]
{
    get
    {
        return this._dictionary[key];
    }
    set
    {
        this._dictionary[key] = value;
        this.InvokeMapChanged(CollectionChange.ItemChanged,
           key);
    }
}

public void Clear()
{
    var priorKeys = this._dictionary.Keys.ToArray();
    this._dictionary.Clear();
    foreach (var key in priorKeys)
    {
        this.InvokeMapChanged(CollectionChange.ItemRemoved,
           key);
    }
}

public ICollection<string> Keys
{
    get { return this._dictionary.Keys; }
}

public bool ContainsKey(string key)
{
    return this._dictionary.ContainsKey(key);
}

public bool TryGetValue(string key, out object value)
{
    return this._dictionary.TryGetValue(key, out value);
}

public ICollection<object> Values
{
    get { return this._dictionary.Values; }
}

public bool Contains(KeyValuePair<string, object> item)
{
    return this._dictionary.Contains(item);
}

public int Count
{
    get { return this._dictionary.Count; }
}

public bool IsReadOnly
{
    get { return false; }
}

public IEnumerator<KeyValuePair<string, object>>
   GetEnumerator()
{
    return this._dictionary.GetEnumerator();
}

System.Collections.IEnumerator
   System.Collections.IEnumerable.GetEnumerator()
{
    return this._dictionary.GetEnumerator();
}

public void CopyTo(KeyValuePair<string, object>[] array,
   int arrayIndex)
{
    int arraySize = array.Length;
    foreach (var pair in this._dictionary)
    {
        if (arrayIndex >= arraySize) break;
```

```
                    array[arrayIndex++] = pair;
                }
            }
        }
    }
```

Listing 5: ObservableDictionary.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.Serialization;
using System.Text;
using System.Threading.Tasks;
using Windows.ApplicationModel;
using Windows.Storage;
using Windows.Storage.Streams;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;

namespace AmazingGeoRace.Common
{
    /// <summary>
    /// SuspensionManager captures global session state to
       simplify process lifetime management
    /// for an application.  Note that session state will be
       automatically cleared under a variety
    /// of conditions and should only be used to store
       information that would be convenient to
    /// carry across sessions, but that should be discarded when
       an application crashes or is
    /// upgraded.
    /// </summary>
    internal sealed class SuspensionManager
    {
        private static Dictionary<string, object> _sessionState =
            new Dictionary<string, object>();
        private static List<Type> _knownTypes = new List<Type>();
        private const string sessionStateFilename =
            "_sessionState.xml";

        /// <summary>
        /// Provides access to global session state for the
            current session.  This state is
        /// serialized by <see cref="SaveAsync"/> and restored by
        /// <see cref="RestoreAsync"/>, so values must be
            serializable by
```

```
/// <see cref="DataContractSerializer"/> and should be as
   compact as possible.  Strings
/// and other self-contained data types are strongly
   recommended.
/// </summary>
public static Dictionary<string, object> SessionState
{
    get { return _sessionState; }
}

/// <summary>
/// List of custom types provided to the <see
   cref="DataContractSerializer"/> when
/// reading and writing session state.  Initially empty,
   additional types may be
/// added to customize the serialization process.
/// </summary>
public static List<Type> KnownTypes
{
    get { return _knownTypes; }
}

/// <summary>
/// Save the current <see cref="SessionState"/>.  Any
   <see cref="Frame"/> instances
/// registered with <see cref="RegisterFrame"/> will also
   preserve their current
/// navigation stack, which in turn gives their active
   <see cref="Page"/> an opportunity
/// to save its state.
/// </summary>
/// <returns>An asynchronous task that reflects when
   session state has been saved.</returns>
public static async Task SaveAsync()
{
    try
    {
        // Save the navigation state for all registered
           frames
        foreach (var weakFrameReference in
           _registeredFrames)
        {
            Frame frame;
            if (weakFrameReference.TryGetTarget(out
               frame))
            {
                SaveFrameNavigationState(frame);
                }
            }

            // Serialize the session state synchronously to
               avoid asynchronous access to shared
            // state
            MemoryStream sessionData = new MemoryStream();
            DataContractSerializer serializer = new
               DataContractSerializer(typeof(Dictionary<string,
               object>), _knownTypes);
            serializer.WriteObject(sessionData,
               _sessionState);

            // Get an output stream for the SessionState file
               and write the state asynchronously
            StorageFile file = await
               ApplicationData.Current.LocalFolder.CreateFileAsync(sessionSta
               CreationCollisionOption.ReplaceExisting);
            using (Stream fileStream = await
               file.OpenStreamForWriteAsync())
            {
                sessionData.Seek(0, SeekOrigin.Begin);
                await sessionData.CopyToAsync(fileStream);
            }
        }
        catch (Exception e)
        {
            throw new SuspensionManagerException(e);
        }
    }

    /// <summary>
    /// Restores previously saved <see cref="SessionState"/>.
        Any <see cref="Frame"/> instances
    /// registered with <see cref="RegisterFrame"/> will also
       restore their prior navigation
    /// state, which in turn gives their active <see
       cref="Page"/> an opportunity restore its
    /// state.
    /// </summary>
    /// <param name="sessionBaseKey">An optional key that
       identifies the type of session.
    /// This can be used to distinguish between multiple
       application launch scenarios.</param>
    /// <returns>An asynchronous task that reflects when
       session state has been read.  The
    /// content of <see cref="SessionState"/> should not be
```

```
   relied upon until this task
/// completes.</returns>
public static async Task RestoreAsync(String
   sessionBaseKey = null)
{
    _sessionState = new Dictionary<String, Object>();

    try
    {
        // Get the input stream for the SessionState file
        StorageFile file = await
           ApplicationData.Current.LocalFolder.GetFileAsync(sessionStateF
        using (IInputStream inStream = await
           file.OpenSequentialReadAsync())
        {
            // Deserialize the Session State
            DataContractSerializer serializer = new
               DataContractSerializer(typeof(Dictionary<string,
               object>), _knownTypes);
            _sessionState = (Dictionary<string,
               object>)serializer.ReadObject(inStream.AsStreamForRead())
        }

        // Restore any registered frames to their saved
           state
        foreach (var weakFrameReference in
           _registeredFrames)
        {
            Frame frame;
            if (weakFrameReference.TryGetTarget(out
               frame) &&
               (string)frame.GetValue(FrameSessionBaseKeyProperty)
               == sessionBaseKey)
            {
                frame.ClearValue(FrameSessionStateProperty);
                RestoreFrameNavigationState(frame);
            }
        }
    }
    catch (Exception e)
    {
        throw new SuspensionManagerException(e);
    }
}

private static DependencyProperty
   FrameSessionStateKeyProperty =
     DependencyProperty.RegisterAttached("_FrameSessionStateKey",
         typeof(String), typeof(SuspensionManager), null);
private static DependencyProperty
    FrameSessionBaseKeyProperty =
     DependencyProperty.RegisterAttached("_FrameSessionBaseKeyParams",
         typeof(String), typeof(SuspensionManager), null);
private static DependencyProperty
    FrameSessionStateProperty =
     DependencyProperty.RegisterAttached("_FrameSessionState",
         typeof(Dictionary<String, Object>),
         typeof(SuspensionManager), null);
private static List<WeakReference<Frame>>
    _registeredFrames = new List<WeakReference<Frame>>();

/// <summary>
/// Registers a <see cref="Frame"/> instance to allow its
    navigation history to be saved to
/// and restored from <see cref="SessionState"/>.  Frames
    should be registered once
/// immediately after creation if they will participate
    in session state management.  Upon
/// registration if state has already been restored for
    the specified key
/// the navigation history will immediately be restored.
    Subsequent invocations of
/// <see cref="RestoreAsync"/> will also restore
    navigation history.
/// </summary>
/// <param name="frame">An instance whose navigation
    history should be managed by
/// <see cref="SuspensionManager"/></param>
/// <param name="sessionStateKey">A unique key into <see
    cref="SessionState"/> used to
/// store navigation-related information.</param>
/// <param name="sessionBaseKey">An optional key that
    identifies the type of session.
/// This can be used to distinguish between multiple
    application launch scenarios.</param>
public static void RegisterFrame(Frame frame, String
    sessionStateKey, String sessionBaseKey = null)
{
     if (frame.GetValue(FrameSessionStateKeyProperty) !=
         null)
     {
          throw new InvalidOperationException("Frames can
              only be registered to one session state key");
     }
```

```
        if (frame.GetValue(FrameSessionStateProperty) != null)
        {
            throw new InvalidOperationException("Frames must
               be either be registered before accessing frame
               session state, or not registered at all");
        }

        if (!string.IsNullOrEmpty(sessionBaseKey))
        {
            frame.SetValue(FrameSessionBaseKeyProperty,
               sessionBaseKey);
            sessionStateKey = sessionBaseKey + "_" +
               sessionStateKey;
        }

        // Use a dependency property to associate the session
           key with a frame, and keep a list of frames whose
        // navigation state should be managed
        frame.SetValue(FrameSessionStateKeyProperty,
           sessionStateKey);
        _registeredFrames.Add(new
           WeakReference<Frame>(frame));

        // Check to see if navigation state can be restored
        RestoreFrameNavigationState(frame);
    }

    /// <summary>
    /// Disassociates a <see cref="Frame"/> previously
       registered by <see cref="RegisterFrame"/>
    /// from <see cref="SessionState"/>.  Any navigation
       state previously captured will be
    /// removed.
    /// </summary>
    /// <param name="frame">An instance whose navigation
       history should no longer be
    /// managed.</param>
    public static void UnregisterFrame(Frame frame)
    {
        // Remove session state and remove the frame from the
           list of frames whose navigation
        // state will be saved (along with any weak
           references that are no longer reachable)
        SessionState.Remove((String)frame.GetValue(FrameSessionStateKeyProper
        _registeredFrames.RemoveAll((weakFrameReference) =>
        {
```

```
        Frame testFrame;
        return !weakFrameReference.TryGetTarget(out
           testFrame) || testFrame == frame;
    });
}

/// <summary>
/// Provides storage for session state associated with
   the specified <see cref="Frame"/>.
/// Frames that have been previously registered with <see
   cref="RegisterFrame"/> have
/// their session state saved and restored automatically
   as a part of the global
/// <see cref="SessionState"/>.  Frames that are not
   registered have transient state
/// that can still be useful when restoring pages that
   have been discarded from the
/// navigation cache.
/// </summary>
/// <remarks>Apps may choose to rely on <see
   cref="NavigationHelper"/> to manage
/// page-specific state instead of working with frame
   session state directly.</remarks>
/// <param name="frame">The instance for which session
   state is desired.</param>
/// <returns>A collection of state subject to the same
   serialization mechanism as
/// <see cref="SessionState"/>.</returns>
public static Dictionary<String, Object>
   SessionStateForFrame(Frame frame)
{
    var frameState = (Dictionary<String,
       Object>)frame.GetValue(FrameSessionStateProperty);

    if (frameState == null)
    {
        var frameSessionKey =
           (String)frame.GetValue(FrameSessionStateKeyProperty);
        if (frameSessionKey != null)
        {
            // Registered frames reflect the
               corresponding session state
            if
               (!_sessionState.ContainsKey(frameSessionKey))
            {
                _sessionState[frameSessionKey] = new
                   Dictionary<String, Object>();
```

```
                    }
                    frameState = (Dictionary<String,
                        Object>)_sessionState[frameSessionKey];
                }
                else
                {
                    // Frames that aren't registered have
                        transient state
                    frameState = new Dictionary<String, Object>();
                }
                frame.SetValue(FrameSessionStateProperty,
                    frameState);
            }
            return frameState;
        }

        private static void RestoreFrameNavigationState(Frame
            frame)
        {
            var frameState = SessionStateForFrame(frame);
            if (frameState.ContainsKey("Navigation"))
            {
                frame.SetNavigationState((String)frameState["Navigation"]);
            }
        }

        private static void SaveFrameNavigationState(Frame frame)
        {
            var frameState = SessionStateForFrame(frame);
            frameState["Navigation"] = frame.GetNavigationState();
        }
    }
    public class SuspensionManagerException : Exception
    {
        public SuspensionManagerException()
        {
        }

        public SuspensionManagerException(Exception e)
            : base("SuspensionManager failed", e)
        {

        }
    }
}
```

Listing 6: SuspensionManager.cs

4.3 Converters

```
using System;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Data;

namespace AmazingGeoRace.Converters
{
    public class BooleanToVisibilityConverter : IValueConverter
    {
        private object GetVisibility(object value)
        {
            if (!(value is bool))
                return Visibility.Collapsed;
            bool objValue = (bool)value;
            if (objValue)
            {
                return Visibility.Visible;
            }
            return Visibility.Collapsed;
        }
        public object Convert(object value, Type targetType,
           object parameter, string language)
        {
            return GetVisibility(value);
        }
        public object ConvertBack(object value, Type targetType,
           object parameter, string language)
        {
            throw new NotImplementedException();
        }


    }
}
```

Listing 7: BooleanToCollapsedConverter.cs

```
using System;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Data;

namespace AmazingGeoRace.Converters
{
    public class BooleanToCollapsedConverter : IValueConverter
    {
        private object GetVisibility(object value)
        {
            if (!(value is bool))
                return Visibility.Visible;
            bool objValue = (bool)value;
            if (objValue)
            {
                return Visibility.Collapsed;
            }
            return Visibility.Visible;
        }
        public object Convert(object value, Type targetType,
            object parameter, string language)
        {
            return GetVisibility(value);
        }
        public object ConvertBack(object value, Type targetType,
            object parameter, string language)
        {
            throw new NotImplementedException();
        }


    }
}
```

Listing 8: BooleanToVisibilityConverter.cs

4.4 Data

```
using System;
using System.Diagnostics;
using System.IO;
using System.Net;
using System.Net.Http;
using System.Runtime.Serialization.Json;
using System.Text;
using System.Threading.Tasks;
using Newtonsoft.Json;
using System.Net.Http.Headers;

namespace AmazingGeoRace.Data {

    public class WebserviceCallException: Exception
    {
        public WebserviceCallException() {}

        public WebserviceCallException(string message):
            base(message) {}
        public WebserviceCallException(string message, Exception
            innerException): base(message, innerException) {}
    }

    public static class WebService {
            private const string ServiceUrl =
                "https://demo.nexperts.com/MOC5/AmazingRaceService/AmazingRaceServ

                 public static async Task<T>
                     QueryDataFromService<T>(string endpoint) {
            var httpClient = new HttpClient();
            httpClient.DefaultRequestHeaders.CacheControl = new
                CacheControlHeaderValue { MaxAge = TimeSpan.Zero };
            var result = await
                httpClient.GetAsync(string.Format(ServiceUrl,
                endpoint));
            var content = await
                result.Content.ReadAsStringAsync();
                      if (result.IsSuccessStatusCode)
                          return
                              JsonConvert.DeserializeObject<T>(content);
                      throw new WebserviceCallException(content);

                 }

                 public static async Task<bool>
                     PostDataToService<TRequest>(string endpoint,
                     TRequest request) {
            var httpClient = new HttpClient();
            httpClient.DefaultRequestHeaders.CacheControl = new
                CacheControlHeaderValue { MaxAge = TimeSpan.Zero };
            var result = await
                httpClient.PostAsJsonAsync(string.Format(ServiceUrl,
                endpoint), request);
            var content = await
                result.Content.ReadAsStringAsync();
                      if (result.IsSuccessStatusCode)
                          return
                              JsonConvert.DeserializeObject<bool>(content);
            throw new WebserviceCallException(content);
        }
        }
}
```

Listing 9: WebService.cs

4.5 Domain

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using AmazingGeoRace.Models;

namespace AmazingGeoRace.Domain
{
    public class LoginService
    {
        public Credentials Credentials { get; set; }

        public bool IsAuthenticated() {
            return Credentials != null;
        }

        public async Task Login(string username, string password,
            Action onSucceeded, Action<Exception> onFailed) {
            var serviceProxy = new ServiceProxy();
            if (await serviceProxy.CheckCredentials(new
                Credentials(username, password))) {
                Credentials = new Credentials(username, password);
                onSucceeded();
            }
            else
                onFailed(new Exception($"Login with username
                    {username} failed."));
        }

        public void Logout()
        {
            Credentials = null;
            Application.Current.Exit();
        }
    }
}
```

Listing 10: LoginService.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using AmazingGeoRace.Data;
using AmazingGeoRace.Models;

namespace AmazingGeoRace.Domain
{
    public class ServiceProxy
    {
        public async Task<bool> CheckCredentials(Credentials
           credentials) {
            return await
               WebService.QueryDataFromService<bool>($"/CheckCredentials?userName
        }

        public async Task<IEnumerable<Route>>
           GetRoutes(Credentials credentials)
        {
            return await
               WebService.QueryDataFromService<IEnumerable<Route>>($"/GetRoutes?u
        }

        public async Task<bool>
           InformAboutVisitedCheckpoint(CheckpointRequest
           checkpoint)
        {
            return await
               WebService.PostDataToService("/InformAboutVisitedCheckpoint",
               checkpoint);
        }

        public async Task<bool> ResetAllRoutes(Request userData)
        {
            return await
               WebService.PostDataToService("/ResetAllRoutes",
               userData);
        }

        public async Task<bool> ResetRoute(RouteRequest
           routeToReset)
        {
            return await
               WebService.PostDataToService("/ResetRoute",
               routeToReset);
        }
    }
}
```

Listing 11: ServiceProxy.cs

4.6 Models

```
using System;
using System.Runtime.Serialization;
using Windows.Devices.Geolocation;

namespace AmazingGeoRace.Models
{
    [DataContract]
    public class Checkpoint
    {
        [DataMember]
        public Guid Id { get; set; }
        [DataMember]
        public int Number { get; set; }
        [DataMember]
        public string Name { get; set; }
        [DataMember]
        public string Hint { get; set; }
        [DataMember]
        public decimal Latitude { get; set; }
        [DataMember]
        public decimal Longitude { get; set; }

        public Geopoint Location => new Geopoint(new
           BasicGeoposition
        {
            Latitude = (double)Latitude,
            Longitude = (double)Longitude
        });
    }
}
```

Listing 12: Checkpoint.cs

```
using System;
using System.Runtime.Serialization;
using AmazingGeoRace.Domain;

namespace AmazingGeoRace.Models
{
    [DataContract]
    public class CheckpointRequest : Request {
        [DataMember]
        public Guid CheckpointId { get; private set; }
        [DataMember]
        public string Secret { get; private set; }
        public CheckpointRequest(Credentials credentials, Guid
            checkpointId, string secret): base(credentials) {
            CheckpointId = checkpointId;
            Secret = secret;
        }
    }
}
```

Listing 13: CheckpointRequest.cs

```
using System.Runtime.Serialization;
using AmazingGeoRace.Domain;

namespace AmazingGeoRace.Models
{
    [DataContract]
    public class Request {
        [DataMember]
        public string UserName { get; private set; }
        [DataMember]
        public string Password { get; private set; }

        public Request(Credentials credentials) {
            UserName = credentials.Username;
            Password = credentials.Password;
        }
    }
}
```

Listing 14: Request.cs

```
using System;
using System.Runtime.Serialization;

namespace AmazingGeoRace.Models
{
    [DataContract]
    public class Route {
        [DataMember]
        public Guid Id { get; set; }
        [DataMember]
        public string Name { get; set; }
        [DataMember]
        public Checkpoint[] VisitedCheckpoints { get; set; }
        [DataMember]
        public Checkpoint NextCheckpoint { get; set; }

        public bool Finished => NextCheckpoint == null;
    }
}
```

Listing 15: Route.cs

```
using System;
using System.Runtime.Serialization;
using AmazingGeoRace.Domain;

namespace AmazingGeoRace.Models {

    [DataContract]
        public class RouteRequest : Request {
                [DataMember]
                public Guid RouteId { get; private set; }

            public RouteRequest(Credentials credentials, Guid
               routeId): base(credentials) {
                RouteId = routeId;
            }
        }
}
```

Listing 16: RouteRequest.cs

4.7 ViewModels

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Input;
using Windows.UI.Popups;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using AmazingGeoRace.Commands;
using AmazingGeoRace.Common;
using AmazingGeoRace.Domain;

namespace AmazingGeoRace.ViewModels
{
    public class LoginViewModel : ViewModelBase
    {
        private LoginService LoginService { get; }
        private string _username;

        public string Username
        {
            get { return _username; }
            set { OnPropertyChanged(ref _username, value); }
        }

        public ICommand LoginCommand { get; set; }

        public LoginViewModel(LoginService loginService) {
            LoginService = loginService;
            LoginCommand = new RelayCommand(async obj => await
                OnLoginCommand(obj));
        }

        private async Task OnLoginCommand(object obj) {
            await ExceptionHandling.HandleException(async () => {
                var password = obj as string;
                if (string.IsNullOrEmpty(password)) {
                    await MessageBoxWrapper.ShowOkAsync("No
                        password given for user.");
                }
                else {
                    password = password.Trim();
                    await LoginService.Login(Username, password,
                        () => {
                         ((Frame)Window.Current.Content).Navigate(typeof(Views.Mai
                    }, async exception => {
                         await
                             MessageBoxWrapper.ShowOkAsync(exception.Message);
                    });
                }
            });
        }
    }
}
```

Listing 17: LoginViewModel.cs

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Threading.Tasks;
using System.Windows.Input;
using Windows.UI.Popups;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using AmazingGeoRace.Commands;
using AmazingGeoRace.Common;
using AmazingGeoRace.Domain;
using AmazingGeoRace.Models;

namespace AmazingGeoRace.ViewModels
{
    public class MainViewModel : ViewModelBase
    {
        private ServiceProxy ServiceProxy { get; }
        private LoginService LoginService { get; }
        public ICommand ShowRouteDetailsCommand { get; set; }
        public ICommand ResetAllRoutesCommand { get; set; }
        public ICommand LogoutCommand { get; set; }
        public ObservableCollection<Route> Routes { get; } = new
           ObservableCollection<Route>();

        public MainViewModel(ServiceProxy serviceProxy,
           LoginService loginService) {
            ServiceProxy = serviceProxy;
            LoginService = loginService;
            ShowRouteDetailsCommand = new RelayCommand(async obj
               => await OnShowRouteDetailsCommand(obj));
            ResetAllRoutesCommand = new RelayCommand(async obj =>
               await OnResetAllRoutesCommand());
            LogoutCommand = new RelayCommand(obj =>
               LoginService.Logout());
        }


        public async void Initialize() {
            await
               ExceptionHandling.HandleExceptionForAsyncMethod(async
               () => SetRoutes(await
               ServiceProxy.GetRoutes(LoginService.Credentials)));
        }

        public void SetRoutes(IEnumerable<Route> routes) {
            Routes.Clear();
            foreach (var route in routes) {
                Routes.Add(route);
            }
        }

        private async Task OnShowRouteDetailsCommand(object obj) {
            await ExceptionHandling.HandleException(() => {
                var route = obj as Route;
                if (route == null)
                    throw new Exception("No route selected.");
                ((Frame)Window.Current.Content).Navigate(typeof(Views.RaceDetails
                    route);
            });
        }

        private async Task OnResetAllRoutesCommand() {
            await ExceptionHandling.HandleException(async () => {
                await
                    ExceptionHandling.HandleExceptionForAsyncMethod(async
                    () => {
                     await ServiceProxy.ResetAllRoutes(new
                        Request(LoginService.Credentials));
                     await MessageBoxWrapper.ShowOkAsync("Routes
                        successfully resetted.");
                     SetRoutes(await
                        ServiceProxy.GetRoutes(LoginService.Credentials));
                });
            });
        }
    }
}
```

Listing 18: MainViewModel.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Input;
using Windows.Devices.Geolocation;
using AmazingGeoRace.Common;
using Windows.UI.Xaml.Controls.Maps;
using Windows.UI;
using Windows.Storage.Streams;
using Windows.UI.Xaml.Controls;
using AmazingGeoRace.Commands;
using AmazingGeoRace.Domain;
using AmazingGeoRace.Models;

namespace AmazingGeoRace.ViewModels
{
    public class RaceDetailsViewModel: ViewModelBase
    {
        public ServiceProxy ServiceProxy { get; set; }
        public LoginService LoginService { get; set; }
        public ICommand ShowUnlockCheckpointDialogCommand { get;
            set; }
        public ICommand ResetRouteCommand { get; set; }

        private bool _finished;
```

```
public bool Finished
{

    get { return _finished; }
    set { OnPropertyChanged(ref _finished, value); }
}

private Route _route;
public Route Route
{
    get { return _route; }
    private set { OnPropertyChanged(ref _route, value); }
}

public Checkpoint NextCheckPoint => Route.NextCheckpoint;

public MapControl Map { get; set; }

public RaceDetailsViewModel(ServiceProxy serviceProxy,
   LoginService loginService) {
    ServiceProxy = serviceProxy;
    LoginService = loginService;
    ShowUnlockCheckpointDialogCommand = new
       RelayCommand(obj => ShowUnlockCheckpointDialog());
    ResetRouteCommand = new RelayCommand(obj =>
       ResetRoute(Route));
}

public void SetRoute(Route route)
{
    Route = route;
    RefreshMap(GetMapElementsForCurrentRoute(), Route);
    if (Route.NextCheckpoint != null)
    {
        OnPropertyChanged("NextCheckPoint");
        Finished = false;
    }
    else {
        ShowSuccessMessage(route);
        Finished = true;
    }
}

public List<MapElement> GetMapElementsForCurrentRoute() {
    var elements = new List<MapElement>();
    var checkPoints = Route.VisitedCheckpoints.ToList();
    if (Route.NextCheckpoint != null) {
            elements.Add(GetMapIconForCheckPoint(Route.NextCheckpoint));
            checkPoints.Add(Route.NextCheckpoint);
        }
        elements.AddRange(Route.VisitedCheckpoints.Select(GetMapIconForCheckP
        elements.Add(GetLinesForCheckPoints(checkPoints));
        return elements;
    }

    public void RefreshMap(IEnumerable<MapElement> elements,
        Route route) {
        Map.MapElements.Clear();
        foreach (var element in elements) {
            Map.MapElements.Add(element);
        }
        Map.Center = route.NextCheckpoint?.Location ??
            route.VisitedCheckpoints.Last()?.Location;
    }

    private MapPolyline
        GetLinesForCheckPoints(IEnumerable<Checkpoint>
        checkPoints)
    {
        return new MapPolyline
        {
            Path = new Geopath(checkPoints.Select(x => new
                BasicGeoposition
            {
                Latitude = (double)x.Latitude,
                Longitude = (double)x.Longitude
            })),
            StrokeColor = Colors.Red,
            StrokeThickness = 5
        };
    }

    private MapIcon GetMapIconForCheckPoint(Checkpoint
        checkPoint)
    {
        return new MapIcon()
        {
            Location = checkPoint.Location,
            Title = checkPoint.Name,
            NormalizedAnchorPoint = new
                Windows.Foundation.Point(0.5, 0.95),
            Image =
                RandomAccessStreamReference.CreateFromUri(new
                Uri("ms-appx:///Assets/mappin.png"))
        };
    }

    private async void ResetRoute(Route route) {
        var result = await ServiceProxy.ResetRoute(new
            RouteRequest(LoginService.Credentials, route.Id));
        if (result) {
            var routes = await
                ServiceProxy.GetRoutes(LoginService.Credentials);
            SetRoute(routes.FirstOrDefault(x => x.Id ==
                route.Id));
            await MessageBoxWrapper.ShowOkAsync("Resetting
                route " + route.Name + " successful.");
            Finished = false;
        }
        else {
            await MessageBoxWrapper.ShowOkAsync("An error
                occurred when trying to reset route " +
                route.Name +".");
        }
    }

    private async void ShowSuccessMessage(Route route) {
        await MessageBoxWrapper.ShowOkAsync("You finished " +
            route.Name + " successfully! Congratulations.");
    }

    private async void ShowUnlockCheckpointDialog() {
        var dialog = new Views.SolutionDialog();
        var contentResult = await dialog.ShowAsync();
        if (contentResult == ContentDialogResult.Primary) {
            var result = await
                ServiceProxy.InformAboutVisitedCheckpoint(new
                CheckpointRequest(LoginService.Credentials,
                NextCheckPoint.Id, dialog.Solution));
            if (result) {
                await
                    MessageBoxWrapper.ShowOkAsync("Congratulations.
                    Correct answer!");
                var routes = await
                    ServiceProxy.GetRoutes(LoginService.Credentials);
                SetRoute(routes.FirstOrDefault(x => x.Id ==
                    Route.Id));
            }
            else {
                await
                    MessageBoxWrapper.ShowOkAsync(dialog.Solution
                              + " was not the correct solution. Please
                              try anotherone.");
                     }
                }
           }
      }
}
```

Listing 19: RaceDetailsViewModel.cs

```
using System;
using System.ComponentModel;
using System.Runtime.CompilerServices;

namespace AmazingGeoRace.ViewModels
{
     public class ViewModelBase : INotifyPropertyChanged
     {
          public event PropertyChangedEventHandler PropertyChanged;

          protected virtual void
              OnPropertyChanged([CallerMemberName] string
              propertyName = null) {
               PropertyChanged?.Invoke(this, new
                   PropertyChangedEventArgs(propertyName));
          }

          protected virtual void OnPropertyChanged<T>(ref T value,
              T newValue, [CallerMemberName] string propertyName =
              null)
          {
               if (Equals(value, newValue))
                    return;
               value = newValue;
               PropertyChanged?.Invoke(this, new
                   PropertyChangedEventArgs(propertyName));
          }
     }
}
```

Listing 20: ViewModelBase.cs

4.8 Views-Markup

```
<Page
     x:Class="AmazingGeoRace.Views.LoginPage"
     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```

```
xmlns:local="using:AmazingGeoRace"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
mc:Ignorable="d"
Background="{ThemeResource
   ApplicationPageBackgroundThemeBrush}"
   RequestedTheme="Light">

<Grid x:Name="LayoutRoot">
    <Grid.ChildrenTransitions>
        <TransitionCollection>
            <EntranceThemeTransition/>
        </TransitionCollection>
    </Grid.ChildrenTransitions>
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>

    <StackPanel Grid.Row="0" Margin="19,0,0,0">
        <TextBlock Text="Amazing Geo Race"
           Style="{ThemeResource TitleTextBlockStyle}"
           Margin="0,12,0,0"/>
        <TextBlock Text="Login" Margin="0,-6.5,0,26.5"
           Style="{ThemeResource HeaderTextBlockStyle}"
           CharacterSpacing="{ThemeResource
           PivotHeaderItemCharacterSpacing}"/>
    </StackPanel>

    <Grid Grid.Row="1" x:Name="ContentRoot" Margin ="10">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>
        <TextBlock Grid.Row="0">Username</TextBlock>
        <TextBox Grid.Row="1" Text="{Binding Username,
           Mode=TwoWay}"></TextBox>
        <TextBlock Grid.Row="2">Password</TextBlock>
        <PasswordBox Grid.Row="3"
           x:Name="Password"></PasswordBox>
        <Button Grid.Row="4" HorizontalAlignment="Stretch"
           Command="{Binding LoginCommand}"
           CommandParameter="{Binding ElementName=Password,
                Path=Password}">Login</Button>
        </Grid>
    </Grid>
</Page>
```

Listing 21: LoginPage.xaml

```
<Page
    x:Class="AmazingGeoRace.Views.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:AmazingGeoRace"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:converters="using:AmazingGeoRace.Converters"
    mc:Ignorable="d"
    Background="{ThemeResource
       ApplicationPageBackgroundThemeBrush}"
       RequestedTheme="Light">
    <Page.Resources>
        <converters:BooleanToVisibilityConverter
           x:Key="BoolToVis" />
    </Page.Resources>
    <Grid x:Name="LayoutRoot">

        <Grid.ChildrenTransitions>
            <TransitionCollection>
                <EntranceThemeTransition/>
            </TransitionCollection>
        </Grid.ChildrenTransitions>

        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>

        <!-- Title Panel -->
        <StackPanel Grid.Row="0" Margin="19,0,0,0">
            <TextBlock Text="Amazing Geo Race"
               Style="{ThemeResource TitleTextBlockStyle}"
               Margin="0,12,0,0"/>
            <TextBlock Text="Routes"  TextWrapping="Wrap"
               Style="{ThemeResource HeaderTextBlockStyle}"
               CharacterSpacing="{ThemeResource
               PivotHeaderItemCharacterSpacing}"/>
        </StackPanel>
```

```
<Grid Grid.Row="1" x:Name="ContentRoot" Margin ="10">
    <ListView ItemsSource="{Binding Routes}"
       x:Name="RoutesList"
       SelectionChanged="ListView_SelectionChanged">
        <ListView.ItemContainerStyle>
            <Style TargetType="ListViewItem">
                <Setter
                   Property="HorizontalContentAlignment"
                   Value="Stretch" />
            </Style>
        </ListView.ItemContainerStyle>
        <ListView.ItemTemplate>
            <DataTemplate>
                <Border Padding="5" Margin="10"
                   BorderBrush="Black"
                   BorderThickness="1"
                   HorizontalAlignment="Stretch">
                    <Grid HorizontalAlignment="Stretch">
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="Auto"/>
                        <ColumnDefinition Width="*"/>
                    </Grid.ColumnDefinitions>
                    <Image Grid.Column="0" Width="48"
                       Source="../Assets/route.png"/>
                        <Grid Grid.Column="1"
                           HorizontalAlignment="Stretch"
                           VerticalAlignment="Center">
                            <Grid.ColumnDefinitions>
                                <ColumnDefinition
                                   Width="2*"/>
                                <ColumnDefinition
                                   Width="Auto"/>
                                <ColumnDefinition
                                   Width="2*"/>
                            </Grid.ColumnDefinitions>
                            <TextBlock Grid.Column="1"
                               Text="{Binding Name}"
                               FontSize="20" />
                            <Image Grid.Column="2"
                               Visibility="{Binding
                               Finished,
                               Converter={StaticResource
                               BoolToVis},
                               FallbackValue=Hidden}"
                               HorizontalAlignment="Right"
                               Width="35"
```

```
                                        Source="../Assets/check.png"
                                        Margin="0,0,10,0"/>
                                 </Grid>
                             </Grid>
                         </Border>
                     </DataTemplate>
                 </ListView.ItemTemplate>
             </ListView>
         </Grid>
         <Button Grid.Row="2" Content="Reset All Routes"
            Margin="10,0" Command="{Binding
            ResetAllRoutesCommand}" HorizontalAlignment="Stretch"/>
         <Button Grid.Row="3" Content="Logout" Margin="10,0"
            Command="{Binding LogoutCommand}"
            HorizontalAlignment="Stretch"/>
     </Grid>
</Page>
```

Listing 22: MainPage.xaml

```
<Page
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:AmazingGeoRace"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:maps="using:Windows.UI.Xaml.Controls.Maps"
    xmlns:converters="using:AmazingGeoRace.Converters"
    x:Class="AmazingGeoRace.Views.RaceDetailsPage"
    mc:Ignorable="d"
    Background="{ThemeResource
       ApplicationPageBackgroundThemeBrush}"
       RequestedTheme="Light">
    <Page.Resources>
        <converters:BooleanToVisibilityConverter
           x:Key="BoolToVis"/>
        <converters:BooleanToCollapsedConverter
           x:Key="BoolToHidden"/>
    </Page.Resources>
    <Grid x:Name="LayoutRoot">
        <Grid.ChildrenTransitions>
            <TransitionCollection>
                <EntranceThemeTransition/>
            </TransitionCollection>
        </Grid.ChildrenTransitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
```

```
</Grid.RowDefinitions>

<StackPanel Grid.Row="0" Margin="19,0,0,0">
    <TextBlock Text="Amazing Geo Race"
       Style="{ThemeResource TitleTextBlockStyle}"
       Margin="0,12,0,0"/>
    <TextBlock TextWrapping="Wrap" Text="{Binding
       Route.Name, FallbackValue=Current Route}"
       Margin="0,-6.5,0,26.5" Style="{ThemeResource
       HeaderTextBlockStyle}"
       CharacterSpacing="{ThemeResource
       PivotHeaderItemCharacterSpacing}"/>
</StackPanel>

<Grid Grid.Row="1" x:Name="ContentRoot" Margin ="10">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>
        <Grid Grid.Row="0"  Margin ="10"
           Visibility="{Binding Finished,
           Converter={StaticResource BoolToHidden},
           FallbackValue=Visible}">
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="Auto"/>
            </Grid.RowDefinitions>
            <TextBlock Grid.Row="0" Text="Name"
               FontSize="20" Margin="5"
               Style="{ThemeResource
               TitleTextBlockStyle}"/>
            <TextBlock Grid.Row="1" Text="{Binding
               NextCheckPoint.Name, FallbackValue=Name}"
               TextWrapping="Wrap"/>
            <TextBlock Grid.Row="2"  Text="Hint"
               FontSize="20" Style="{ThemeResource
               TitleTextBlockStyle}"/>
            <TextBlock Grid.Row="3" Text="{Binding
               NextCheckPoint.Hint, FallbackValue=Hint}"
               TextWrapping="Wrap"/>
            <Button Grid.Row="4" Content="Provide
               Solution" Command="{Binding
               ShowUnlockCheckpointDialogCommand}" />
```

```
                </Grid>
                <Grid Grid.Row="0"  Margin ="10"
                   Visibility="{Binding Finished,
                   Converter={StaticResource BoolToVis},
                   FallbackValue=Collapsed}">
                    <Grid.RowDefinitions>
                        <RowDefinition Height="Auto"/>
                    </Grid.RowDefinitions>
                    <Button Grid.Row="0" Content="Reset Route"
                       Command="{Binding ResetRouteCommand}"
                       HorizontalAlignment="Stretch"/>
                </Grid>
                <maps:MapControl  x:Name="Map" Grid.Row="1"
                   ZoomLevel="18" Style="AerialWithRoads"
                   MapServiceToken="AoYHlDdX43W94h_H3MubTFFMGVMLuI_l8nL-udsJnPfi
            </Grid>
        </Grid>
    </Grid>
</Page>
```

Listing 23: RaceDetailsPage.xaml

```
<ContentDialog
    x:Class="AmazingGeoRace.Views.SolutionDialog"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:AmazingGeoRace"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    Title="Solution"
    PrimaryButtonText="OK"
    SecondaryButtonText="cancel" RequestedTheme="Light">
    <StackPanel VerticalAlignment="Stretch"
       HorizontalAlignment="Stretch">
        <TextBox x:Name="TbSolution" Header="Solution"/>
    </StackPanel>
</ContentDialog>
```

Listing 24: SolutionDialog.xaml

4.9 Views

```
using System.Diagnostics;
using Windows.UI.ViewManagement;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Navigation;
using AmazingGeoRace.Common;
```

```
// The Basic Page item template is documented at
   http://go.microsoft.com/fwlink/?LinkID=390556

namespace AmazingGeoRace.Views
{
    public sealed partial class LoginPage: Page
    {
        public LoginPage() {
            InitializeComponent();
            NavigationCacheMode = NavigationCacheMode.Disabled;
        }


        protected override void OnNavigatedTo(NavigationEventArgs
           e) {
            var viewmodel = App.Current.LoginViewModel;
            DataContext = viewmodel;
        }
    }
}
```

Listing 25: LoginPage.xaml.cs

```
using Windows.ApplicationModel;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Navigation;
using AmazingGeoRace.Models;
using AmazingGeoRace.ViewModels;

namespace AmazingGeoRace.Views
{
    public sealed partial class MainPage : Page
    {
        public MainPage()
        {
            this.InitializeComponent();
            this.NavigationCacheMode =
               NavigationCacheMode.Required;

            RoutesList.Items.Add(new Route {
                Name = "Test"
            });
        }

        /// <summary>
        /// Invoked when this page is about to be displayed in a
           Frame.
        /// </summary>
        /// <param name="e">Event data that describes how this
           page was reached.
        /// This parameter is typically used to configure the
           page.</param>
        protected override async void
           OnNavigatedTo(NavigationEventArgs e) {
            var viewModel = App.Current.MainViewModel;
            DataContext = viewModel;
            viewModel.Initialize();
        }

        private void ListView_SelectionChanged(object sender,
           SelectionChangedEventArgs e) {
            if (RoutesList.SelectedItem != null)
                App.Current.MainViewModel.ShowRouteDetailsCommand.Execute(RoutesL
        }
    }
}
```

Listing 26: MainPage.xaml.cs

```
using System.Collections.Generic;
using System.Linq;
using Windows.Phone.UI.Input;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Maps;
using Windows.UI.Xaml.Navigation;
using AmazingGeoRace.Models;

namespace AmazingGeoRace.Views
{
    public sealed partial class RaceDetailsPage: Page
    {

        public RaceDetailsPage() {
            this.InitializeComponent();
            HardwareButtons.BackPressed +=
               HardwareButtons_BackPressed;
        }

        private void HardwareButtons_BackPressed(object sender,
           BackPressedEventArgs e) {
            if (!Frame.CanGoBack)
                return;
            e.Handled = true;
            Frame.GoBack();
        }
        protected override void
            OnNavigatedFrom(NavigationEventArgs e) {
            HardwareButtons.BackPressed -=
                HardwareButtons_BackPressed;
        }

        protected override void OnNavigatedTo(NavigationEventArgs
            e) {
            var route = e.Parameter as Route;
            if (route == null) {
                Frame.Navigate(e.SourcePageType);
                return;
            }

            var viewModel = App.Current.RaceDetailsViewModel;
            viewModel.Map = Map;
            viewModel.SetRoute(route);
            DataContext = viewModel;
        }
    }
}
```

Listing 27: RaceDetailsPage.xaml.cs

```
using Windows.UI.Xaml.Controls;

namespace AmazingGeoRace.Views
{
    public sealed partial class SolutionDialog : ContentDialog
    {
        public string Solution => TbSolution.Text;

        public SolutionDialog() {
            this.InitializeComponent();
        }
    }
}
```

Listing 28: SolutionDialog.xaml.cs

4.10 App

```
<Application
    x:Class="AmazingGeoRace.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:AmazingGeoRace" RequestedTheme="Light">

</Application>
```

Listing 29: App.xaml

```
using AmazingGeoRace.Domain;
using AmazingGeoRace.ViewModels;
using System;
using System.Diagnostics;
using Windows.ApplicationModel;
using Windows.ApplicationModel.Activation;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Media.Animation;
using Windows.UI.Xaml.Navigation;
using AmazingGeoRace.Common;
using AmazingGeoRace.Views;

namespace AmazingGeoRace
{
    public sealed partial class App
    {
        private TransitionCollection _transitions;
        private readonly LoginService _loginService;

        internal RaceDetailsViewModel RaceDetailsViewModel { get;
           private set; }
        internal LoginViewModel LoginViewModel { get; private
           set; }
        internal MainViewModel MainViewModel { get; private set; }


        public new static App Current => (App)Application.Current;

        public App()
        {
            InitializeComponent();
            _loginService = new LoginService();
            var serviceProxy = new ServiceProxy();
            LoginViewModel = new LoginViewModel(_loginService);
            MainViewModel = new MainViewModel(serviceProxy,
               _loginService);
            RaceDetailsViewModel = new
               RaceDetailsViewModel(serviceProxy, _loginService);
            Suspending += OnSuspending;
        }
        private void RootFrame_FirstNavigated(object sender,
           NavigationEventArgs e)
        {
            var rootFrame = sender as Frame;
            if (rootFrame == null)
                return;
            rootFrame.ContentTransitions = _transitions ?? new
               TransitionCollection { new
               NavigationThemeTransition() };
            rootFrame.Navigated -= RootFrame_FirstNavigated;
        }

        protected override async void
           OnLaunched(LaunchActivatedEventArgs e) {
#if DEBUG
            if (Debugger.IsAttached) {
                DebugSettings.EnableFrameRateCounter = true;
            }
#endif
            var startPageType = typeof(LoginPage);
            Frame rootFrame = Window.Current.Content as Frame;

            if (rootFrame == null) {
                rootFrame = new Frame {
                    CacheSize = 1,
                    Language =
                       Windows.Globalization.ApplicationLanguages.Languages[0]
                };

                if (e.PreviousExecutionState ==
                   ApplicationExecutionState.Terminated) {
                    try {
                        await SuspensionManager.RestoreAsync();
                        if
                           ((bool)SuspensionManager.SessionState["Authenticated"
                           {
                            var username =
                               SuspensionManager.SessionState["Username"]
                               as string;
                            var password =
                               SuspensionManager.SessionState["Password"]
                               as string;
                            await _loginService.Login(username,
                               password, async () => {
                                startPageType = typeof(MainPage);
                                await
                                   MessageBoxWrapper.ShowOkAsync(string.Format("
                                have been logged in
                                automatically. With username
                                {0}.", username));
                        }, async (exception) => {
                            await
                                MessageBoxWrapper.ShowOkAsync(exception.Messag
                        });
                    }
                    await
                        SuspensionManager.RestoreAsync("frameSessionKey");
                }
                catch (Exception) {
                    Debug.WriteLine("Session restore
                        failed.");
                }
            }
            Window.Current.Content = rootFrame;
        }

        if (rootFrame.Content == null) {
            if (rootFrame.ContentTransitions != null) {
                _transitions = new TransitionCollection();
                foreach (var c in
                    rootFrame.ContentTransitions) {
                     _transitions.Add(c);
                }
            }

            rootFrame.ContentTransitions = null;
            rootFrame.Navigated += RootFrame_FirstNavigated;

            if (!rootFrame.Navigate(startPageType,
                e.Arguments)) {
                throw new Exception("Failed to create initial
                    page");
            }
        }
        Window.Current.Activate();
    }

    private async void OnSuspending(object sender,
        SuspendingEventArgs e) {
        var deferral = e.SuspendingOperation.GetDeferral();

        try {
            SuspensionManager.SessionState["Authenticated"] =
                _loginService.IsAuthenticated();
                if (_loginService.IsAuthenticated()) {
                    SuspensionManager.SessionState["Username"] =
                       _loginService.Credentials?.Username;
                    SuspensionManager.SessionState["Password"] =
                       _loginService.Credentials?.Password;
                }
                await SuspensionManager.SaveAsync();
            }
            catch (Exception) {
                Debug.WriteLine("Saving session failed.");
            }
            deferral.Complete();
        }
    }
}
```

Listing 30: App.xaml.cs