Tennis Challenge Design

The Tennis Challenge system consists of the following layers:

* InfoBoard: Windows Forms application that is used for displaying the “InfoBoard” dashboard web page. It integrates RFID capability since it is not supported by the browser alone.
* TennisWeb: ASP.NET MVC web application that implements the “InfoBoard” dashboard. Provides scheduling and tournament capability.
* TennisChallenge.Dal: The Data Access Layer for the system. Makes the underlying calls to the database.

The layers and dependencies are shown below:



The important components of the Tennis Challenge system and their dependencies are shown below:



frmMain is a Windows Forms control that contains a WebBrowser control (mainBrowser). The WebBrowser controller navigates to the InfoBoard URL and passes RFID input as JavScript events.

The InfoBoard URL is recognized by the TennisWeb ASP.NET MVC web application as a controller route. The InfoBoardController returns the corresponding view, InfoBoard\Index.cshtml. The view links to the tennis-challenge.js JavaScript file, which contains the main front end logic. It calls the MemberController and InfoBoardController to receive data and update the view based on the data. It also posts data from the view as the user makes requests through the user interface. The view also loads the manifest file for offline caching of content from the ResourcesController.

The MemberController and InfoBoard controller use the AccessorBase for database functionality.

# Features Added

1. Cache a single booking request offline for later submission
2. Automatic retry of booking request submission
3. Submit a cached booking request when connection is established
4. Create a booking request while offline
5. Cancel a booking request while offline
6. Cache user info for offline usage (needed for creating a booking request offline)
7. Cache member list for offline usage (needed for creating a booking request offline)
8. Open InfoBoard while offline (verified for Internet Explorer 10+, Chrome, Firefox, and InfoBoard Windows Forms application)

# Key Changes

1. In order to implement the capability to open InfoBoard offline, the TennisWeb application was modified to include a manifest so that the InfoBoard view could build an application cache.
   * Added ResourcesController. The controller contains one action, Manifest, which generates and returns a manifest file with all of the content files and controller routes that can be cached.
     1. Note: The page will attempt to use the cached files even when changes are made on the server. In order to recreate the cache, the manifest version number must be changes. This is part of the ManifestResult constructor. It is currently set to “1.0” so it should be incremented the next time there are file changes on the server.
   * Added ManifestResult. This is a custom class used to generate a manifest file.
   * Modified \_InfoBoardLayout.cshtml to use the manifest and render in the most current mode. Modified the html tag to include the manifest attribute. Removed the meta tag with content=”IE=9” attribute. The meta tag caused the page to render in Internet Explorer 9 mode which broke the application cache capability.
2. Configured registry so that the InfoBoard Windows application uses Internet Explorer 11 browser mode. The default setting when using a Windows Forms WebBrowser control is to use Internet Explorer 7 mode which will not work with the application cache feature. Included files in the “Registry” folder with the required registry settings so that the WebBrowser control will use Internet Explorer 11 mode. For more information, use the following links:

* <https://msdn.microsoft.com/en-us/library/ee330730.aspx>
* <http://www.codewrecks.com/blog/index.php/2011/06/06/witch-version-of-browser-is-used-by-the-webbrowser-control/>

1. Modified tennis-challenge.js to cache state offline and retry booking requests
   * Lines 3-21: Included JSON parser extension to parse dates as JavaScript date objects. This is used for getting state data from the cache.
   * Lines 197-199: Removed the logic to log off when cancelling a request. The call to log off broke offline capability.
   * Lines 261-271: modified to get existing state data if there is a connection error when opening a booking
   * Line 305: Set the cached state data when opening a booking request
   * Lines 321-368: Modified to set cached state data when the user is successfully verified. Moved the callback code from directly in the verifyUser function to a separate callback function (verifyUserCallback). This allows verifyUser to be called with a different callback function. Also, modified the error handler to get the cached state data.
   * Lines 535-548: Added retryBookingRequest function to retry a booking request on failure. Once connection is reestablished, the booking request will reach the server and complete the request. Added functions for getting and setting cached data. Also, added function to clear the cached data.
   * Lines 608-615: Modified to set cached booking request and booking member when calling sendBookingRequest. When the request is successful, the cache is cleared.
   * Lines 667-670: Returned functions so that they can be called outside of the model function
   * Lines 1606-1611: Modified init function to retry submitting a booking request if a cached booking request is found. This is retrying a booking request if a new session is started after a previous session had a cached booking request that was not submitted.
   * Lines 1660-1662: Modified modelStateChanged to load a cached booking member if one does not exist.
   * Lines 1813-1825: Added functions to get and set cached member list
   * Lines 1832-1843: Modified loadMemberList to cache the member list after it is loaded from the server. Moved presentation logic to separate function, presentMemberList. The member list can be presented from cache when there is an error.
   * Lines 1985-1987: Modified updateSingleDouble to get cached booking request info if one does not exist.
   * Lines 2055-2084: Added try catch to setBookingInfo since it was causing errors with certain data input.
   * Functions 2086-2097: Added functions for getting and setting cached booking request info
   * Line 2116: Modified bookTennisCourt to cache the booking request info.

# Additional Notes

1. The new version was only tested with username and password login and not a card reader. Recommend testing with a card reader to ensure that offline functionality works with a card reader.
2. Recommend modifying caching to better secure credentials. setCachedUserInfo (lines 565-567) save the userInfo object to local storage. Local storage uses clear text so would not be optimal in a production system. It should at least encrypt the user name and password. Ideally, the front end code would not store the user and password at all. Instead it would receive a session token from the server, which would be sent back to the server when connection is reestablished. Saving the user info was a result of the current server side implementation. The MemberController VerifyMember action is looking for RFID or username/password. It could probably be changed to look at the request to determine if it has an authorized session token.
3. Using a web browser (Internet Explorer, Chrome or Firefox) will successfully log in and resubmit booking request after losing connectivity. The Windows Forms application was able to function offline, but could not log in after reconnecting. Similar to the credential storage, this could be fixed by improving the authentication on the server side (such as the authentication token).
4. Suggest automating the registry changes. The InfoBoard application could include code to set the required registry keys to use Internet Explorer 11 mode. This could also be included in an installer for InoBoard application (if there is a plan to use an installer).