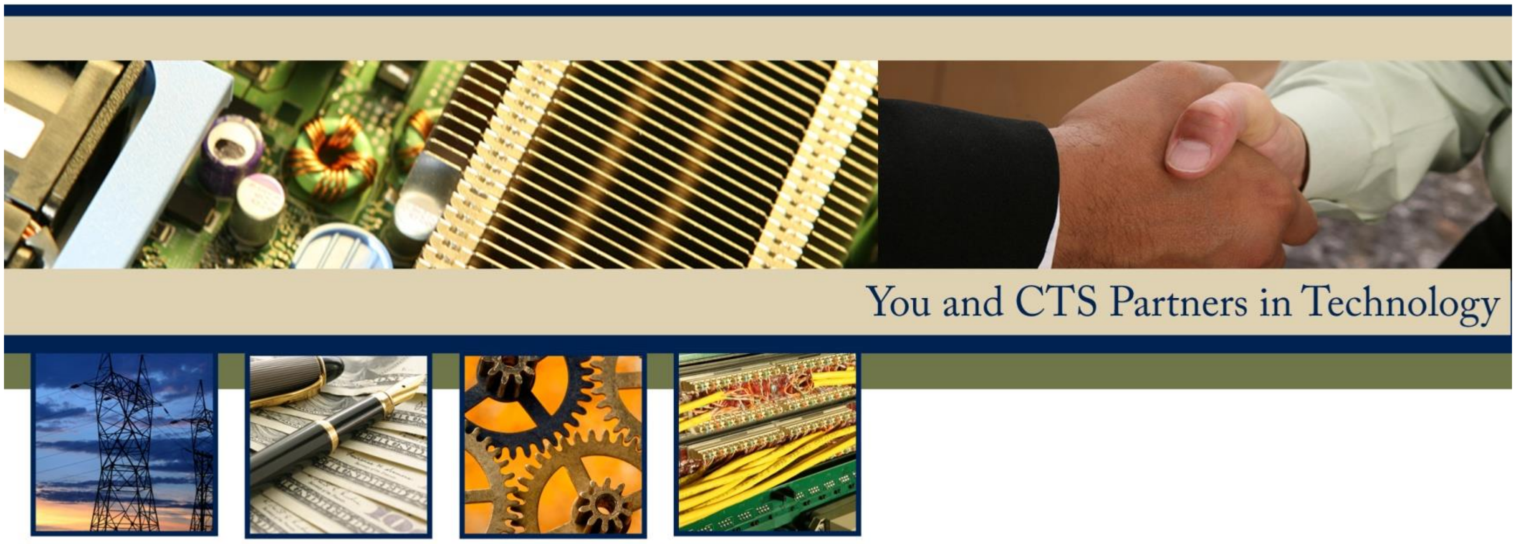


jQuery Training

Table of Contents

Revision History

Version	Author(s)	Description	Date
1.0	William King	Initial draft	2/21/2014
1.1	William	Updates for content; additional requirements	5/26/2014

1.0 Overview

You will implement the client side portion of a web-based library application. The application will allow the user to perform the following actions:

1. View all users
2. View all books
3. Check out a book to a user
4. Check in a book
5. Create a new user

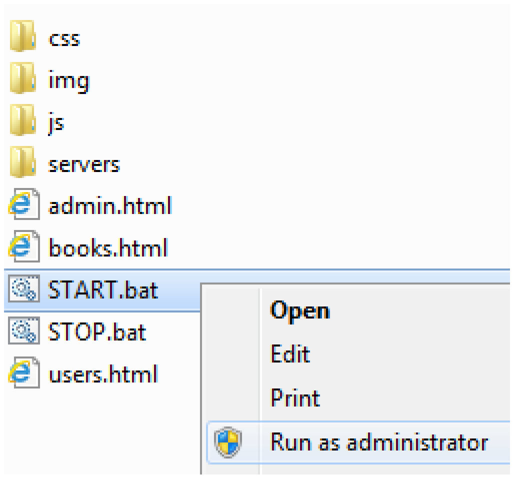
The instructor will provide files to use as a template. The files include everything required to create the web pages. You will need to create and implement the client side code necessary to satisfy the application requirements.

There is also a stand-alone file server and web service to allow all development to occur locally and without a dependency on any running servers. The library data exists in memory only. The application does not persist changes (i.e. new users, check outs, etc.) to disk.

With the exception of the provided plugins, jQuery is the only JavaScript library allowed for the project.

2.0 Getting Started

To launch your local servers, run START.bat as an administrator.

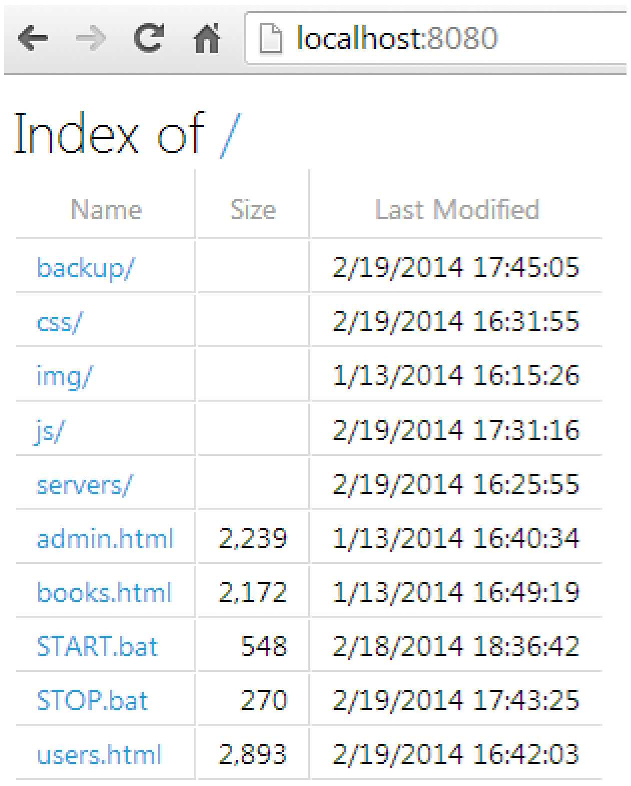


Two console windows will open. Each will display debug information. The file server will display the directory to which it is pointed. The web service will display operations as they occur, and will be useful to observe during development.

To stop your local servers, either close each console individually, or run STOP.bat as an administrator.

2.1. Verify File Server

The file server is, by default, pointing to the directory containing each of the HTML files. To verify it is working, open your browser and go to http://localhost:8080. If everything is working, you will see a directory listing.



Index of /

Name	Size	Last Modified
backup/		2/19/2014 17:45:05
css/		2/19/2014 16:31:55
img/		1/13/2014 16:15:26
js/		2/19/2014 17:31:16
servers/		2/19/2014 16:25:55
admin.html	2,239	1/13/2014 16:40:34
books.html	2,172	1/13/2014 16:49:19
START.bat	548	2/18/2014 18:36:42
STOP.bat	270	2/19/2014 17:43:25
users.html	2,893	2/19/2014 16:42:03

Calls to the web service will not succeed unless you are access the page using the local server. In other words, the URL will be:

http://localhost:8080/books.html

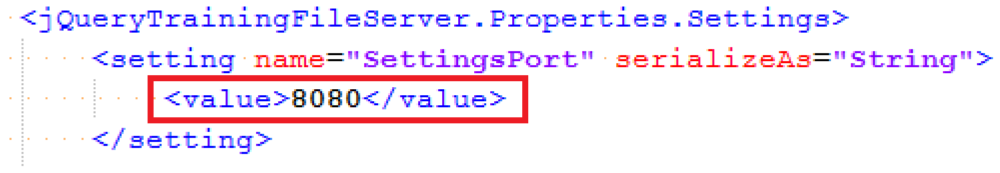
Not:

file:///C:/CTS/ jQuery/Project/books.html

If your computer is already using port 8080 and you need to adjust the file server, there are two required changes.

1. First, open the configuration file for the file server and change the **SettingsPort** value.

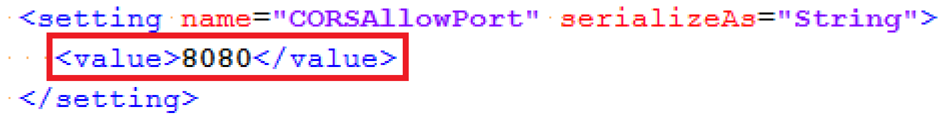
   \servers\jQueryTrainingFileServer\ jQueryTrainingFileServer.exe.config



```
<jQueryTrainingFileServer.Properties.Settings>
    <setting name="SettingsPort" serializeAs="String">
        <value>8080</value>
    </setting>
```

2. Then, open the web service configuration file and change **CORSAllowPort** to match the port used for the file server.

   \servers\jQueryTrainingWebService\jQueryTrainingWebService.exe.config



```
<setting name="CORSAllowPort" serializeAs="String">
  <value>8080</value>
</setting>
```

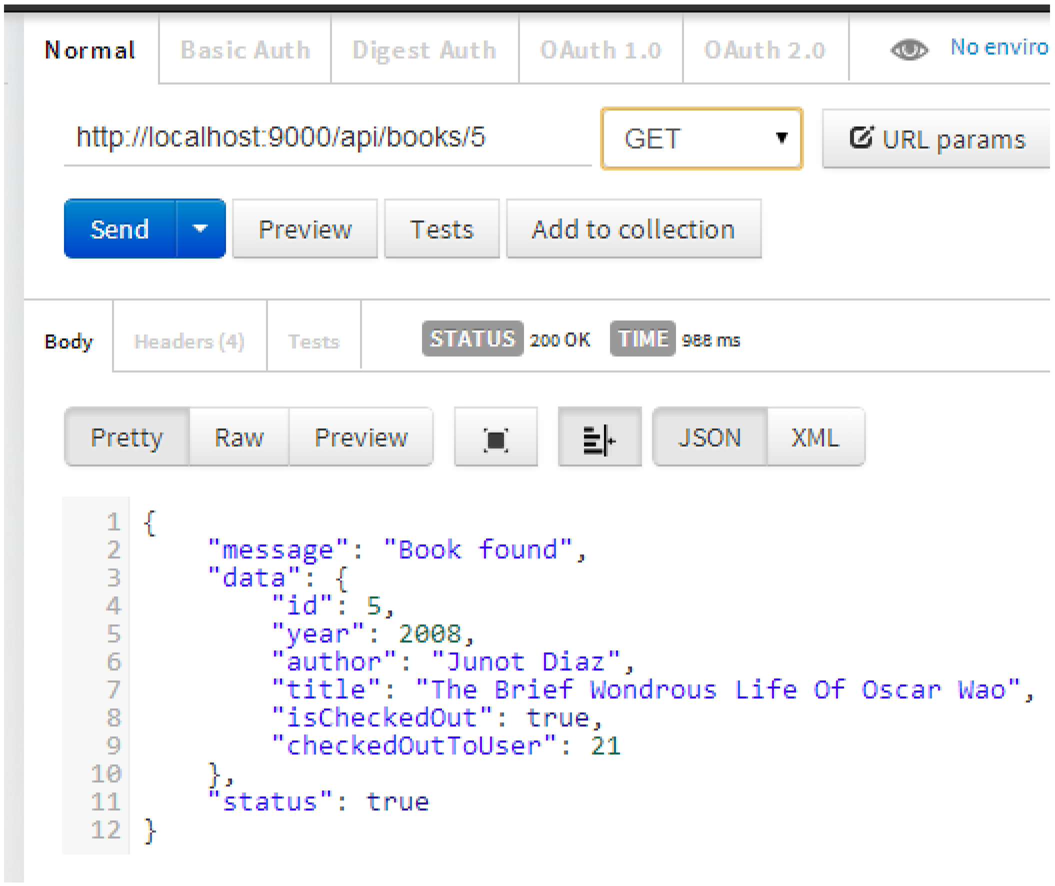
2.2. Verify Web Service

To test the web service, you will need to find and install a utility that allows making REST requests. The command line tool to use is cURL. There are different implementations available for Windows. There are also extensions available for Chrome. Two tested extensions are:

1. Postman:
   http://www.getpostman.com/

2. Advanced REST Client:
   https://chrome.google.com/webstore/detail/advanced-rest-client/hgmloofddffdnphfgcellkdfbfbjeloo

After installing the extension, enter **chrome://apps/** into Chrome's URL bar to view and launch installed extensions.



http://localhost:9000/api/books/5

GET

STATUS 200 OK TIME 988 ms

```
{
    "message": "Book found",
    "data": {
        "id": 5,
        "year": 2008,
        "author": "Junot Diaz",
        "title": "The Brief Wondrous Life Of Oscar Wao",
        "isCheckedOut": true,
        "checkedOutToUser": 21
    },
    "status": true
}
```

If your computer is already using port 9000 and you need to adjust the web service, open its configuration file and adjust the **ListenerPort**.

\servers\jQueryTrainingWebService\jQueryTrainingWebService.exe.config

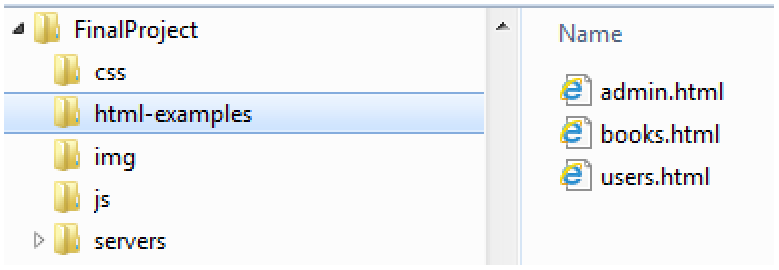
```
 <setting name="ListenerPort" serializeAs="String">
   <value>9000</value>
 </setting>
```

2.3. Web Service Delay Simulation

The web service application delays responses by a random number of milliseconds. Approximately 10% of all requests will take between three and six seconds to complete. All other requests will take between half of one second and two seconds. If needed, you can adjust these values in the configuration file.

2.4. Example Final Output

In the root of the project directory is the “html-examples” directory. This directory contains HTML pages you can use as a reference of generated content.



3.0 Requirements

There will be three pages in the library application:

1. Users - users.html
2. Books - books.html
3. Admin - admin.html

The provided files contain the structure of the page as well as example data filled in. Please create a copy of these files as backup before you begin work.

3.1. All Pages

As a rule, the application will display and hide messages and progress indicators using some type of transition (e.g. fade, slide). Additionally, the application loads user and books data after the Users or Books page has loaded. Depending on the web server delay, there will be nothing to see other than the page header and footer. It is up to the developer to determine a mechanism to ensure the user knows page data is loading.

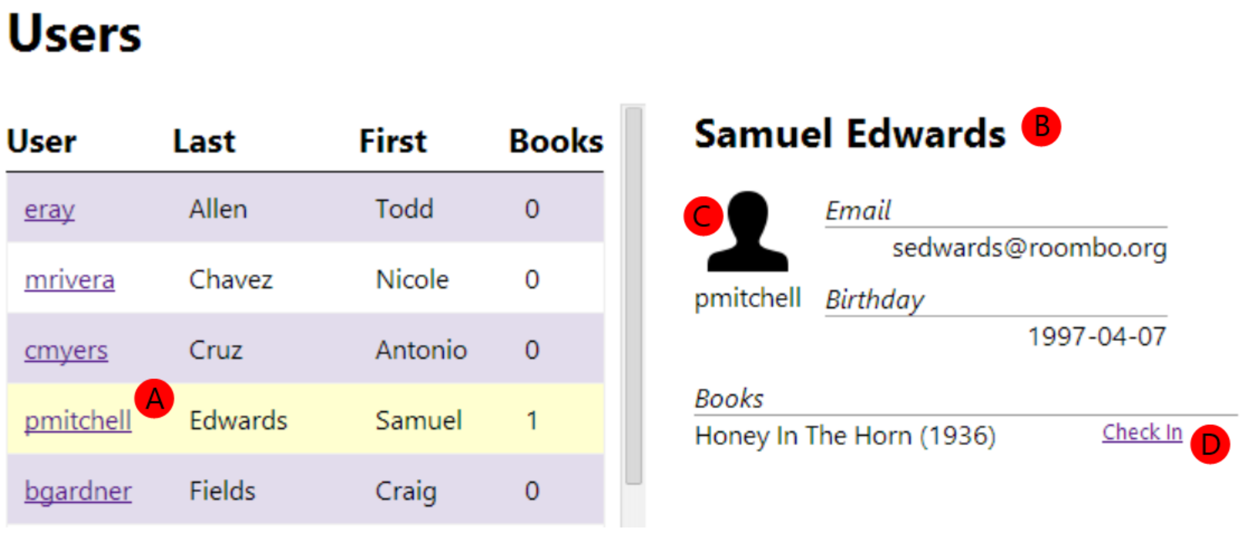
The HTML and CSS files delivered as part of the project contain the structure and style rules required to complete the project. The project aims to minimize time spent working on tasks outside of jQuery. Remember to examine the HTML markup while coding.

Also note: The users and books tables are not visible by default. Once their data is set, don’t forget to display them.

3.2. Users Page

The Users page displays all users in the system. It also allows checking in books for a user. The screen does not need to refresh in real time.

3.2.1. Screen



Users

User	Last	First	Books
eray	Allen	Todd	0
mrivera	Chavez	Nicole	0
cmyers	Cruz	Antonio	0
pmitchell	Edwards	Samuel	1
bgardner	Fields	Craig	0

Samuel Edwards

pmitchell

Email
sedwards@roombo.org

Birthday
1997-04-07

Books
Honey In The Horn (1936) Check In

The Users page will list all library users in a table on the left. The table will display four columns:

1. User – the user name
2. Last – the user's last name
3. First – the user's first name
4. Books – the number of books the user has checked out

The table is sortable using the provided "tablesorter" plugin. By default, the table will sort by the "User" column value, in ascending order. View the following page for an example of how to sort using the plugin: http://mottie.github.io/tablesorter/docs/index.html.

When the page loads, no user is selected by default and the area to the right of the table (B) is empty.

3.2.2. Actions

3.2.2.1. User Selection

When the application user selects a user name (A), the table row containing that user's information will have the "selected" class applied to it. There can only be one selected row at a time. The area to the right of the Users table (B) will update to display more information about the selected user.

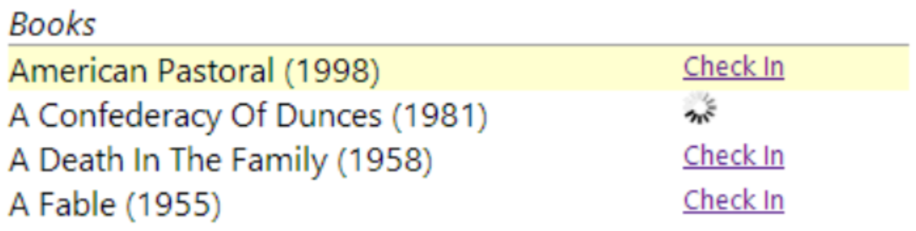
1. Full Name
2. Email Address
3. Birthday
4. Checked out books

There are three possible images to display (C) for the selected user. The application will display the image appropriate to the selected user's gender.

1. Male – *user-male-48x48.png*
2. Female – *user-female-48x48.png*
3. Unknown – *question-48x48.png*

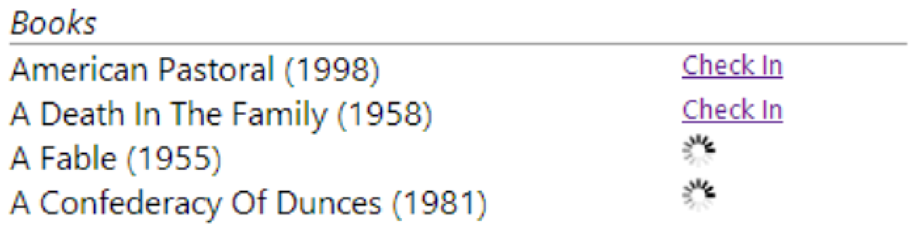
3.2.2.2. Book Check In

Each displayed book will have an associated "Check In" button (D). When the application user clicks "Check In", the application will perform that operation. During the check in process, the application will replace the clicked "Check In" button with a progress indicator (the *spinner.gif* image). When the check in process is complete, the application will remove the book from the list.



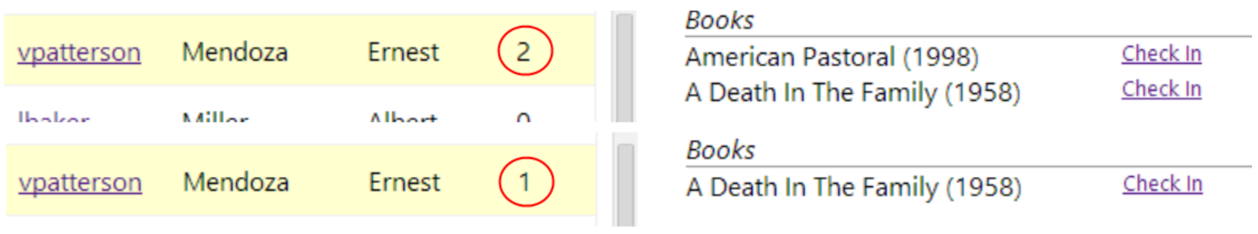
Books	
American Pastoral (1998)	Check In
A Confederacy Of Dunces (1981)	
A Death In The Family (1958)	Check In
A Fable (1955)	Check In

The application user can click "Check In" for a book while another book in the list is still in the check in process. Each selected book displays the progress indicator until the check in process is complete and the book removed.



Books	
American Pastoral (1998)	Check In
A Death In The Family (1958)	Check In
A Fable (1955)	
A Confederacy Of Dunces (1981)	

When the check in process is complete, the application will update the "Books" column value for the selected user to match the current number of books still checked out to that user.



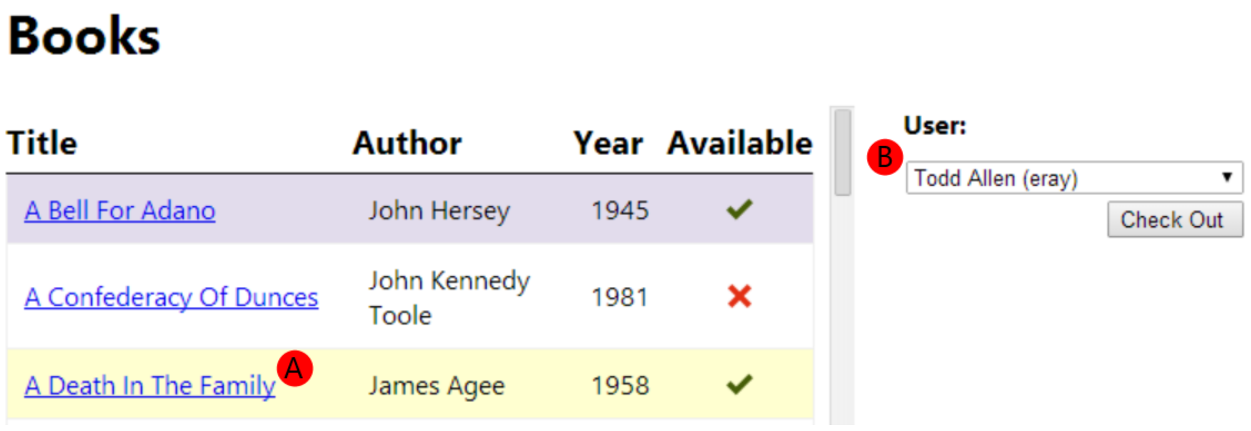
				Books	
vpatterson	Mendoza	Ernest	2	American Pastoral (1998)	Check In
				A Death In The Family (1958)	Check In
lbaker	Miller	Albert	0		

				Books	
vpatterson	Mendoza	Ernest	1	A Death In The Family (1958)	Check In

3.3. Books Page

The Books page will display all books in the system. It will also allow checking out books to a user. The screen does not need to refresh in real time.

3.3.1. Screen



Books

Title	Author	Year	Available
A Bell For Adano	John Hersey	1945	✓
A Confederacy Of Dunces	John Kennedy Toole	1981	✗
A Death In The Family	James Agee	1958	✓

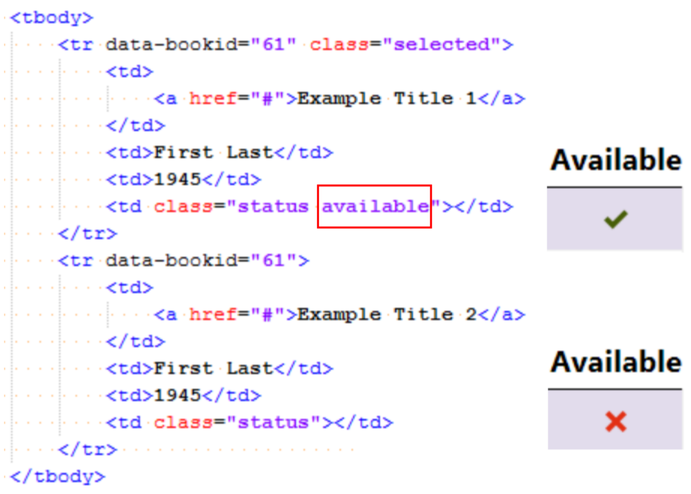
The Books page will list all library books in a table on the left. The table will display four columns:

1. Title – the book title
2. Author – the book’s author
3. Year – the book’s publication date
4. Available – the book’s availability

The Available column value will be one of two images, based on the books availability.

1. Book is checked in – *check-16x16.png*
2. Book is checked out – *x-16x16.png*

The default state for the Available table cell is not available (i.e. the book is checked out). The application will be display the check image by applying the “available” class value to the table cell.

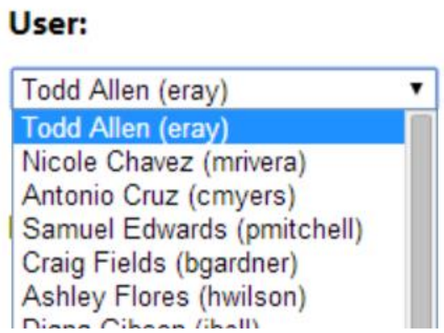


```
<tbody>
    <tr data-bookid="61" class="selected">
        <td>
            <a href="#">Example Title 1</a>
        </td>
        <td>First Last</td>
        <td>1945</td>
        <td class="status available"></td>
    </tr>
    <tr data-bookid="61">
        <td>
            <a href="#">Example Title 2</a>
        </td>
        <td>First Last</td>
        <td>1945</td>
        <td class="status"></td>
    </tr>
</tbody>
```

The table is sortable using the provided “tablesorter” plugin. By default, the table will sort by the book title, in ascending order. Use the following page for an example of how to sort using the plugin: http://mottie.github.io/tablesorter/docs/index.html.

There is no default book selection on page load, and the “Check Out” button is disabled.

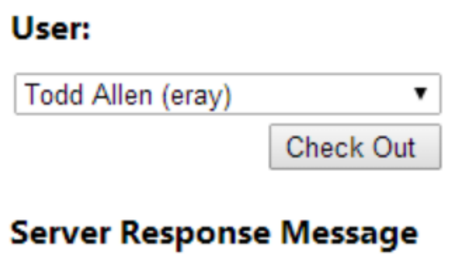
The page will display all system users in a selection list to the right of the table. The list will display each user using their full name and their user name in parentheses. The order of users is not important.



User:

Todd Allen (eray)
Todd Allen (eray)
Nicole Chavez (mrivera)
Antonio Cruz (cmyers)
Samuel Edwards (pmitchell)
Craig Fields (bgardner)
Ashley Flores (hwilson)

Beneath the user selection is a “Check Out” button and an area for the application to display server response messages. When the Books page is initially loaded, there will be no server response message to display and so the area will be empty.



3.3.2. Actions

3.3.2.1. “Available” Column Sorting

The application requires a custom parser to sort the table by the “Available” column value. The following page provides documentation and an example for creating a custom parser for the “tablesorter” plugin: http://mottie.github.io/tablesorter/docs/example-parsers.html.

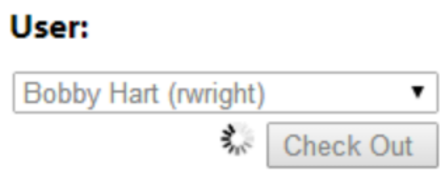
3.3.2.2. Book Selection

When the application user selects a book (Ⓐ), the table row containing that book’s information will have the “selected” class applied to it. There can only be one selected row at a time. The application will enable the “Check Out” button if the book is available to check out. The application disables the button if the book is not available to check out.

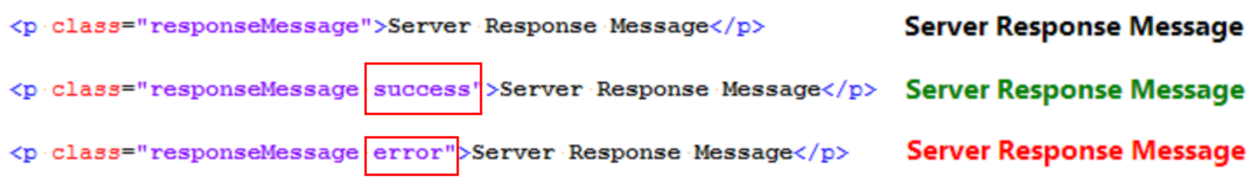
3.3.2.3. Book Check Out

When the user clicks the “Check Out” button, the application will check out the selected book to the selected user. During the checkout process:

1. Disable the “Check Out” button.
2. Disable ability to select a different user.
3. Display the hidden *spinner.gif* image next to the “Check Out” button.



The application will enable all previously disabled elements and hide the *spinner.gif image* when the checkout process is complete. In addition, the application will display the server response message using the “responseMessage” paragraph below the “Check Out” button. Apply the “success” or “error” class to adjust the displayed message color. The server’s response status determines the applied class.



```
<p class="responseMessage">Server Response Message</p>
<p class="responseMessage success">Server Response Message</p>
<p class="responseMessage error">Server Response Message</p>
```

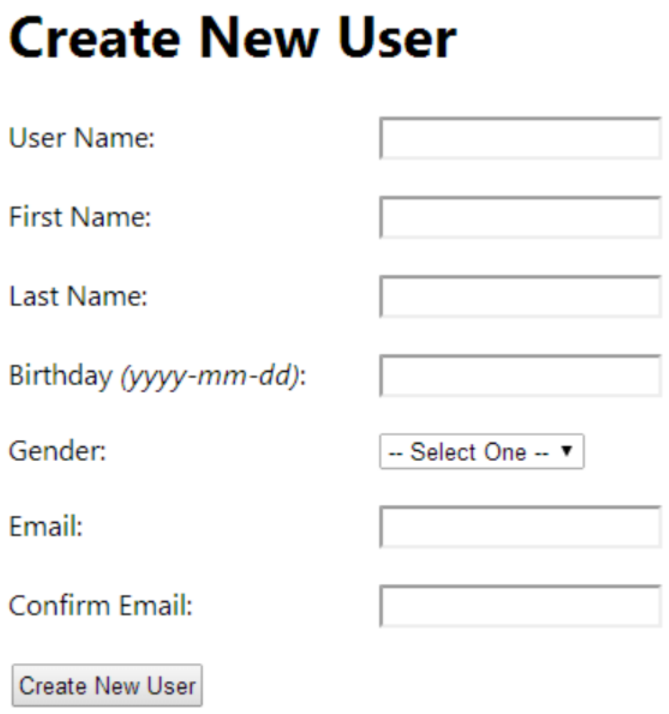
The application will only display one server response message at a time. It will clear any previous text before displaying the most recent message.

If the checkout process successfully completes, the application will update the Available image in the books table to indicate the book is no longer available.

3.4. Admin Page

The Admin page will provide the application user the ability to create new users in the library system.

3.4.1. Screen



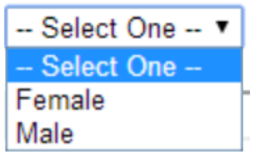
Create New User

User Name:	
First Name:	
Last Name:	
Birthday *(yyyy-mm-dd)*:	
Gender:	-- Select One -- ▾
Email:	
Confirm Email:	

The Admin page will provide input fields for entering new user information. The “Gender” input will be a drop down list with three possible selections:

1. -- Select One –
2. Female
3. Male

The default “Gender” selection on page load will be “-- Select One --”.

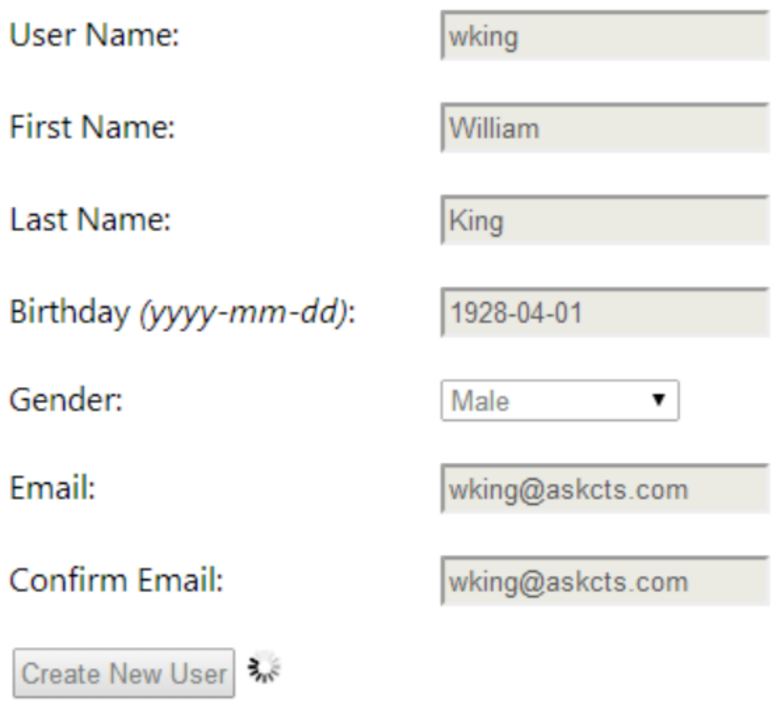


When the information is complete, the page will allow the application user to submit the form and create a new library user with the submitted data.

3.4.2. Actions

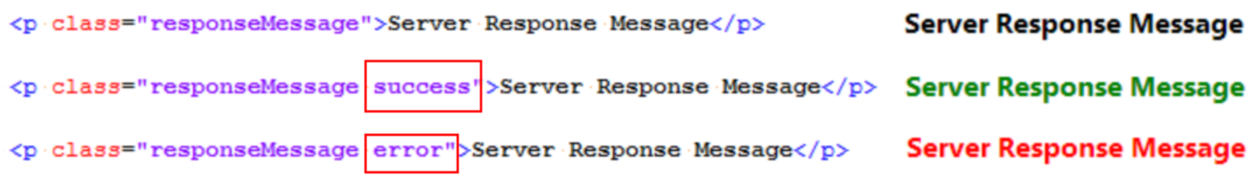
The application user clicks “Create New User” to submit the new library user data. During the creation process:

1. Disable the “Create New User” button.
2. Disable all text input fields.
3. Display the hidden *spinner.gif* image next to the “Create New User” button.



User Name:	wking
First Name:	William
Last Name:	King
Birthday *(yyyy-mm-dd)*:	1928-04-01
Gender:	Male
Email:	wking@askcts.com
Confirm Email:	wking@askcts.com

The application will enable all previously disabled elements and hide the *spinner.gif image* when the user creation process is complete. In addition, the application will display the server response message using the “responseMessage” paragraph below the “Check Out” button. Apply the “success” or “error” class to adjust the displayed message color. The server’s response status determines the applied class.



```
<p class="responseMessage">Server Response Message</p>
<p class="responseMessage success">Server Response Message</p>
<p class="responseMessage error">Server Response Message</p>
```

Server Response Message

Server Response Message

Server Response Message

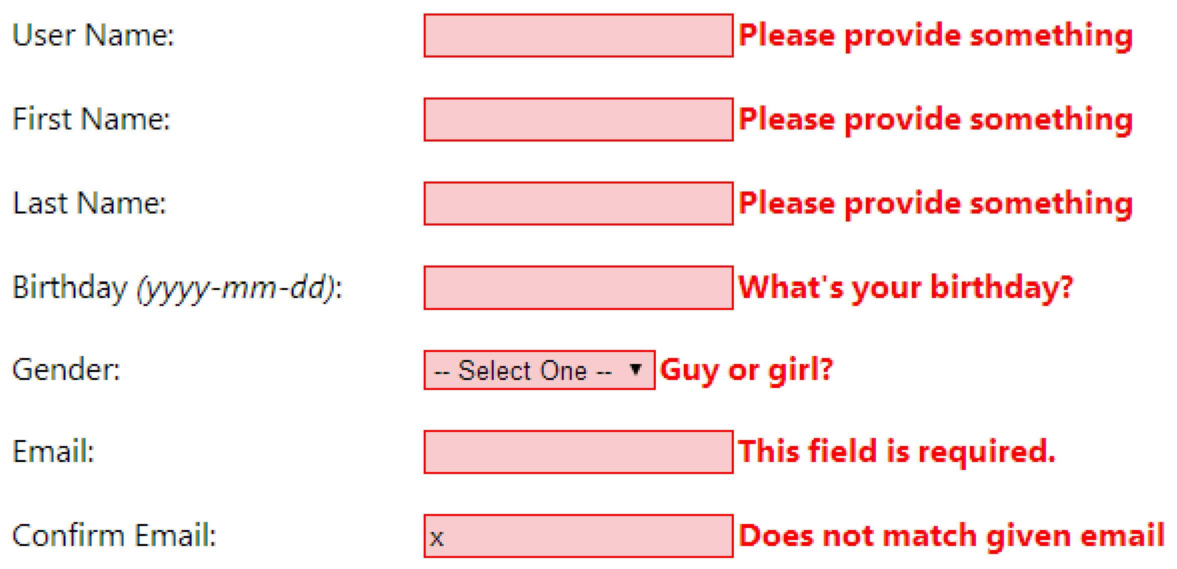
3.4.3. Data Validation

The application will use the *jquery.validate* plugin to perform data validation. The following table describes the validation rules and messages to apply.

Input	Rule	Error Message
User Name	Required	Please provide something
	At least 4 characters	Must be at least 4 characters
	No more than 12 characters	No more than 12 characters please
	Alphanumeric	*Validate plugin default*
First Name	Required	Please provide something
	At least 2 characters	Must be at least 2 characters
	No more than 15	That's way too long
	Letters Only	*Validate plugin default*
Last Name	Required	Please provide something
	At least 2 characters	Must be at least 2 characters
	No more than 15	That's way too long
	Letters Only	*Validate plugin default*
Birthday	Required	What's your birthday?
	Date Format	That's not your birthday
Email	Required	*Validate plugin default*
	Email Format	*Validate plugin default*
Confirm Email	Required	Confirm your email
	Matches Email Input	Does not match given email
Gender	Not Default	Guy or girl?

3.4.4. Expected Screen Example

The following screen is the expected result with no selected gender and inputs blank except for “Confirm Email”:



User Name:		Please provide something
First Name:		Please provide something
Last Name:		Please provide something
Birthday *(yyyy-mm-dd)*:		What's your birthday?
Gender:	-- Select One --	Guy or girl?
Email:		This field is required.
Confirm Email:	x	Does not match given email

4.0 Custom Plugin Requirements

The Users page table and the Books page table allow only single selections. You will create a jQuery plugin named "tableSingleSelection" to provide this functionality for both pages.

4.1. Selector

The plugin operates against table elements. When executed against an element that is not a table, the plugin will find the first nested table. If it finds no table, the plugin exits and does nothing.

4.2. Default Behavior

For each table the "tableSingleSelection" plugin operates against, it will attach a click event handler that will fire when a user provided selector is matched. The default selector will be "`a`" and the plugin will ensure that only events that originate from within the table body trigger the event handler.

The plugin accepts a callback function it will call on the handled click event. The provided function will always be called passing the same arguments as that received by the click event.

The plugin will apply the class "`selected`" to the row containing the selected element. It allows only a single row at one time to have this class.

4.3. User Options

The following table describes the available configuration options the user can provide when calling the plugin.

Option Name	Type	Default Value	Description
selectedClass	string	‘selected’	The class applied to the currently selected row.
targetSelector	string	‘a’	The selector to match for click events; plugin ensures only events within the table body are handled.
onSelected	function	$.noop	Called on a *new* row selection; arguments passed match those provided to the event handler. Not called on an already selected row.
isDisabled	function	$.noop	Called by the plugin on each click event handled; if the result is true, the event is ignored.

4.4. Example Calls

4.4.1. Default

Anchor element clicks will cause the plugin to apply the ‘selected’ class to their containing row.

```
$('table').tableSingleSelection();
```

4.4.2. Custom Selection Class

Anchor element clicks will cause the plugin to apply the 'redRow' class to their containing row.

```
$('table').tableSingleSelection({selectedClass:'redRow'});
```

4.4.3. Custom Target Selector

Button element clicks will cause their containing row to be marked selected. The plugin ignores anchor element clicks.

```
$('table').tableSingleSelection({targetSelector: 'button'});
```

4.4.4. Multiple Target Selectors

Click events from anchor elements and buttons will cause their containing row to be marked selected.

```
$('table').tableSingleSelection({targetSelector: 'a, button'});
```

4.4.5. Additional Click Event Handler

The plugin finds the first table within matched div elements. Click events trigger the console log statement. The console log statement will not print on subsequent calls to the same row. The user must select a new row.

```
$('div').tableSingleSelection({
    onSelected: function() {
        console.log("SELECTED");
    }
});
```

4.4.6. Disable Click Event Handling

The plugin ignore any click events while the result if `isDisabled` is true. In this case, the value for `tableSelectionDisabled` must be set to false in order for selections to occur.

```
var tableSelectionDisabled = true;
$('table').tableSingleSelection({
     isDisabled: function() {
          return tableSelectionDisabled;
     }
});
```

5.0 Web Service API

The following section describes the API for interacting with the library web service. It also describes the structure of object responses.

5.1. Objects

5.1.1. Common Response Object

All api requests will respond with an object of the following form:

```
{
    status: <boolean>,
    message: <string>,
    data: any valid JavaScript type
}
```

A true ‘status’ value means that the request completed without error. It does not indicate the success of a query or operation.

For example, a request to “/api/books/10” will return a response object with ‘data’ set to the book whose ID is 10. The value of ‘status’ will be true. If no such book exists, ‘data’ will be null, and status *will still be true*. On the other hand, you will generate an error if you attempt to check out a book already checked out. The service handler will catch that error and return an object with ‘status’ set to false.

5.1.2. User Object

```
{
    id:<number>,
    firstName:<string>,
    lastName:<string>,
    userName:<string>,
    email:<string>,
    password:<string>,
    gender:<string>,
    birthday:<string>,
    age:<number>,
    bookCount:<number>
    books:<array>,
}
```

5.1.3. Book Object

```
{
    id:<number>,
    year:<number>,
    author:<string>,
    title:<string>,
    isCheckedOut:<boolean>,
    checkedOutToUser:<number>
}
```

5.2. API

5.2.1. Users

Path	Type	POST Data	Description	Return Data
/api/users	GET		Returns all users	Array of user objects
/api/users/{user_id}	GET		Return user with provided id	User object
/api/users/{user_email}	GET		Return user with the provided email	User object

/api/users/new	POST	`{`
		`userName: <string>,`
		`firstName: <string>,`
		`lastName: <string>,`
		`email: <string>,`
		`gender: <string>,`
		`birthday: <string>`
		`}`
			Creates a new user using the provided POST data	User object

5.2.2. Books

Path	Type	Description	Return Data
/api/books	GET	Returns all books	Array of book objects
/api/books/{book_id}	GET	Return book with provided id	Book object

5.2.3. Library

Path	Type	POST Data	Description	Return Data

/api/library/checkout	POST	`{`
		`bookId: <num>,`
		`userId: <num>`
		`}`
			Check out the book with the provided book id to the user with the provided user id	none

/api/library/checkin	POST	`{`
		`id: <num>`
		`}`
			Check in the book with the provided id	none