```
<html lang="en">

<head>
  <Title>One single book of the Bible with Cross References</Title>
  <link rel="icon" type="image/x-icon" href="../images/favicon.ico">
</head>

<body>
<H1>One single Book of the Bible with its contents expanded.</H1>

<p id="context">
  In <b>BibleModel</b> a program can now fetch an entire book of the Bible,
  with or without the actual chapter and verse text.
</p>

<p id="what">
  This page simply reads that in and displays it as a Table and Text.
<p>
<h2>Details of the query execution:</h2>
<table style="border: thin double black">
  <tr><td><strong>Relative File URL </strong></td>   <td id="fileURL"></td></tr>
  <tr><td><strong>REST server URL </strong></td>     <td id="RESTURL"></td></tr>
  <tr><td><strong>Query URL </strong></td>           <td id="effectiveURL"></td></tr>
  <tr><td><strong>Response.type</strong></td>        <td id="responseType"></td></tr>
  <tr><td><strong>Response.url</strong></td>         <td id="responseUrl"></td></tr>
  <tr><td><strong>Response.status </strong></td>     <td id="responseStatus"></td></tr>
  <tr><td><strong>Response.ok</strong></td>          <td id="responseOK"></td></tr>
  <tr><td><strong>Response.statusText</strong></td>  <td id="responseStatusText"></td></tr>
  <tr><td><strong>Response.headers</strong></td>     <td id="responseHeaders"></td></tr>
```

```
30 </table>
31 <p>
32 <h2>One book of the Bible.</h2>
33 <table id="bookTable" style="border: thin double black">
34 </table>
35
36 <div id="BookDiv">
37   <p id="bookIntro"></p>
38   <p id="bookText"></p>
39 </div>
40
41
42 <h2>JSON Data read from books.json:</h2>
43 <p id="bookJson">
44 </p>
45
46 <script type="module">
47   let trace = true;
48   let booksArrayLocal = [];
49   let dataLibrary = "./defaultLibrary";
50   const preferencesLocalStorageTag = "BibleModel.prefs";
51   const preferences = JSON.parse(localStorage.getItem(preferencesLocalStorageTag));
52   if (preferences == null) alert("ERROR: No preferences set.  Please visit preferences page
   first.");
53   else {
54     dataLibrary = preferences["MyBiblePath"];
55   }
56
57
```

```
58   // Look for book selection and other details in the URL options.
59   let book = "1John";
60   if (preferences != null) book = preferences.Book;
61   let details = "contents";
62   let source = "file";
63   if (preferences != null) source = preferences.accessDataBy;
64
65   let url = document.URL;
66   if (trace) console.log("URL=" + url);
67   let urlOptions = url.split('?')[1];
68   if (trace) console.log("URL options=" + urlOptions);
69   if (urlOptions != undefined) {
70
71     let optionArray = urlOptions.split('&');
72     if (trace) console.log("URL option array=" + optionArray);
73
74     for (let i = 0; i < optionArray.length; i++) {
75       let item = optionArray[i].split('=');
76       if (item[0] === "book") {
77         book = item[1];
78       } else if (item[0] === "details") {
79         details = item[1];
80       } else if (item[0] === "source") {
81         source = item[1];
82       }
83     }
84     if (trace) console.log("book=" + book + " details=" + details + " source=" + source);
85   }
86
```

```
 87   const bookJsonPathname = `../${dataLibrary}/` + book + ".json";
 88   const bookJsonURL = `${preferences.RESTURL}:${preferences.RESTPort}/book/` + book + "/
    contents";
 89
 90   // - - - - - - - - - Get the list of all books - - - - - - -- - -
 91   const booksJsonPathname = `../${dataLibrary}/books.json`;
 92   const booksJsonURL = `${preferences.RESTURL}:${preferences.RESTPort}/books`;
 93
 94   let booksText = 'initialized text';
 95   let booksParagraphElement = document.getElementById('booksParagraph');
 96   let booksTableElement = document.getElementById('booksTable');
 97   let booksError;
 98   let enable = true;
 99   let booksTarget;
100
101   let headers = new Headers();   // may need to send some request headers
102   //- - - - decide if getting a file or live from the REST server - - - -
103   if (source == null || source === "file") {
104     booksTarget = booksJsonPathname;
105   } else {  // for now assume URL for the server.
106     booksTarget = booksJsonURL;
107   }
108   headers.append("Accept", "application/json");
109
110   console.log("books target=" + booksTarget);
111
112   if (enable === true) {
113     try {
114       console.log('Next step is to fetch books.json file.');
```

```
115       await fetch(booksTarget,
116               {
117                 headers: headers,
118                 mode: 'no-cors',
119                 method: 'GET'
120               }
121       ).then((response) => response.json())
122               .then((booksResult) => {
123                 console.log("booksJSON=", booksArrayLocal);
124                 booksArrayLocal = booksResult;
125                 booksText = JSON.stringify(booksArrayLocal);
126               });
127     } catch (error) {
128       console.log("Error trying to fetch books.json: ", error);
129       booksError = JSON.stringify(error);
130       console.log("Error fetching file ", booksJsonPathname, " : ", booksError);
131     }
132
133   } else
134     console.log('fetch disabled.');
135
136   //--------- a simple function to find the name for a number
137   function bookNumberToName(bookNumber) {
138     for (var eachBook of booksArrayLocal) {
139       if (eachBook.bookNumber === bookNumber)
140         return eachBook.name;
141     }
142     return bookNumber;
143   }
```

```
144
145   // - - - - - - - - - Get the book data itself - - - - - - - -- - -
146   let booksArray = booksArrayLocal;
147
148   //- - - - - - find placemarkers for insertion of the results
149
150   document.getElementById("fileURL").textContent = bookJsonPathname;
151   document.getElementById("RESTURL").textContent = bookJsonURL;
152
153   let bookJsonElement = document.getElementById('bookJson');
154   let bookTableElement = document.getElementById('bookTable');
155   let bookError;
156   let fetchResponse;
157   let responseType;
158   let responseUrl;
159   let responseCode;
160   let responseStatus;
161   let responseOK;
162   let responseStatusText;
163   let responseError;
164   let responseHeaders;
165
166   let target = null;
167   // - - - decide how to access both the file and from the rest server
168   if (source == null || source === "file") {
169     target = bookJsonPathname;
170   } else {  // for now assume URL for the server.
171     target = bookJsonURL;
172   }
```

```
173   console.log("target=" + target);
174   document.getElementById("effectiveURL").textContent = target;
175
176   // - - - - - Go get the data - - - - - -
177   let bookJsonData;  // the json data should be assigned to this
178   let bookJsonText;  // the string value of it should be left here.
179
180     // First, we setup a try {} catch {} block.
181     try {
182           // if anything explodes in this try block it will transfer
183           // control immediately to the catch block below.
184       console.log(`Next step is to fetch the data from ${target}.`);
185       //- - - - - - - - - - - The actual fetch request - - - - - - -
186       // WARNING: The following section of code is quite confusing by nature.
187       // So, I'm going to explain it at length for the benefit of those reading
188       // this kind of code for the first time, or someone like me who is writing
189       // this code and trying to figure out how it really works.
190       //
191       // So, here is the introduction:
192       // 1) Asynchronous processing is complex and non-linear, so you can't
193       //    just read the code and assume that it executes top to bottom.
194       // RANT: I really dislike JS Promise class.
195       // See:  https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/
    Global_Objects/Promise
196       // 2) You use this when some request (like Fetch) will take some time and you
197       //    don't (can't) make the rest of the browser freeze for a long time.
198       //    So, Fetch returns a Promise object, a promise that sooner or later
199       //    your request will either succeed or fail.  You can chain the actual
200       //    fetch request with a .then() method.  That method takes one or two
```

```
201         //    arguments that indicate what function to execute when the fetch completes
202         //    successfully, and if present, the second argument inidicates the function
203         //    that should be called if the Fetch fails.  One or the other of those
204         //    will be called when the fetch completes.
205         // 3) The natural way of coding this would be to write the Fetch request,
206         //    And one or two other functions named something like handleFetchResults,
207         //    and handleFetchFailure.  Each of those will have a single argument
208         //    with the details of the results.
209         //    function handleFetchResults(fetchResponse) {
210         //        <--- code to do something with the response. -->
211         //      }
212         // 4) If what you are doing in those two functions is non trivial then
213         //    coding it that way makes some sense.  But if it is trivial then it's
214         //    a lot of bother.   One solution to that is to have anonymous functions
215         //    that are basically functions without a name that are coded in the
216         //    place where you would normally put a call to that function.
217         //    This is what is called syntactic sugar.
218         //    See: https://en.wikipedia.org/wiki/Syntactic_sugar
219         //    In that page, you will find a JavaScript example of this very thing.
220         //    The problem is that this kind of sugar saves keystrokes but makes
221         //    the code harder to understand for novices.   To make matters worse
222         //    even the anonymous functions aren't simple enough, that they added
223         //    some more sugar called "arrow functions".
224         // I also dislike JS Arrow Functions.
225         // See: https://www.geeksforgeeks.org/javascript-anonymous-functions/
226         // 5) Another piece of syntactic sugar is function call chaining.
227         //    If you have an object and you want to call it several times
228         //    in a row without having to restate the object name.  One way
229         //    to do that is to make sure that each function returns the original
```

```
230       //    object as it's return value.   Then you can just make a chain.
231       //    See: https://x-team.com/magazine/javascript-method-chaining
232
233       // OK, here is the code.
234       // The fetch call sends a request to a web server somewhere in the world.
235
236       await //  the fetch is asynchronous normally, so WAIT for it to be done.
237               fetch(target,  // the url to be fetched
238                     {     // options needed to modify the simple fetch.
239                       headers: headers,  // A list of headers to add to the fetch
240                       mode: 'no-cors',  // Don't allow Cross Origin data.
241                       method: "GET"     // Get the URL data and return it.
242                     }  // end of the options parameter.
243                   )  // end of the arguments to the fetch call
244                // The fetch returns a promise object that we don't remember
245                // in a variable because we are going to just use it right
246                       // here by method chaining.  The fetch returned a promise
247                       // so that promise is assumed, and we call it's then
248                       // method.
249               .then(
250                       // Arrow function here which accepts a single argument
251                       // that is assumed to be an http response.
252                       // See: https://developer.mozilla.org/en-US/docs/Web/API/Response
253                       // The => arrow is sugar for the body of a function.
254                       // In this case all that it does is return the json object
255                       // from the response object.
256                       (response) => {
257                         fetchResponse = response.json();
258                         responseType = response.type;
```

```
259                           responseUrl = response.url;
260                           responseStatus = response.status;
261                           responseStatusText = response.statusText;
262                           responseOK = response.ok;
263                           responseHeaders = response.headers;
264                           console.log(`fetch done: response=${fetchResponse}`);
265                           console.log(`type=${responseType} status=${responseStatus} ok=${
    responseOK} code=${responseCode}`);
266                           return fetchResponse;
267                         }
268                )   // The end of the single argument to the then method.
269                         // So, that arrow function returns the json part of the response,
270                         // which is itself, another promise.  A promise that the data
271                         // will sooner or later be received and handled.  Some results
272                         // contain a great deal of data and it has to stream in over time.
273                         // So, this promise is that it will eventually be ready.
274                         // That promise will eventually be fulfilled, and when it does
275                         // this next 'then' will run and receive the json data.
276                .then(
277                          // The function to be called here is another anonymous arrow function
278                          // that takes the json data as it's argument.
279                         (bookResult) =>
280                           // in this case the code is a code block enclosed in {}
281                           {
282                             console.log("JSON=", bookResult);
283                             bookJsonData = bookResult;
284                             bookJsonText = JSON.stringify(bookJsonData);
285                           } // end of code block.
286                         // If all went well, the raw data and a text of it
```

```
287                         // has been safely stored away.
288                     );  // end of the then aruments, and finally end of the whole bloody
    statement.
289
290       } catch (error) {  // if we get here something bad happened in the try
291                           // block, then the error is passed in here.
292         console.log("Error trying to fetch books.json: ", error);
293         responseError = JSON.stringify(error);
294         console.log("Error fetching ", target, " : ", responseError);
295       }
296
297     // After the await of the fetch call and all the callbacks are done
298     // control will continue here, where we will populate the web page.
299
300     // Display the results on the web page.
301     bookJsonElement.innerHTML = bookJsonText;
302
303     document.getElementById("responseType").textContent = responseType;
304     document.getElementById("responseUrl").textContent = responseUrl;
305     document.getElementById("responseStatus").textContent = responseStatus;
306     document.getElementById("responseOK").textContent = responseOK;
307     document.getElementById("responseStatusText").textContent = responseStatusText;
308     // document.getElementById("responseCode").textContent = responseCode;
309     if ( responseError !== undefined)
310        document.getElementById("responseError").textContent = responseError;
311
312     // source of information
313     let bookData = bookJsonData;
314     let bookChapters = bookData.chapters;
```

```
315
316   // destination of information in the web page.
317   let bookDivElement = document.getElementById("BookDiv");
318   let bookIntroElement = document.getElementById("bookIntro");
319   let bookHeaderElement = document.createElement("h2");
320   bookIntroElement.appendChild(bookHeaderElement);
321
322   let bookNameElement = document.createTextNode(bookData.name + " : " + bookData.title);
323   let bookTitleHeaderElement = document.createElement("h2");
324   bookTitleHeaderElement.appendChild(bookNameElement);
325   bookIntroElement.appendChild(bookTitleHeaderElement);
326
327   let bookTextElement = document.getElementById("bookText");
328   // Now navigate across all the chapters and expand them into the html.
329   bookData.chapters.forEach((chap) => {
330     let header2 = document.createElement("h3");
331     header2.appendChild(document.createTextNode("Chapter " + chap.chapterNumber));
332     bookTextElement.appendChild(header2);
333     let verses = chap.verses;
334     verses.forEach((verse) => {
335       let xrefs = verse.xrefs;
336       let vn = verse.verseNumber;
337       let vt = verse.text;
338       let boldVerseNumberElement = document.createElement("STRONG");
339       boldVerseNumberElement.appendChild(document.createTextNode(vn + " "));
340       bookTextElement.appendChild(boldVerseNumberElement);
341       bookTextElement.appendChild(document.createTextNode(vt));
342       // bookTextElement.appendChild( document.createElement("br") );
343       if (xrefs.length > 0) {
```

```
344         let disclosureElement = document.createElement("details");
345         disclosureElement.appendChild(document.createElement("summary")
346                 .appendChild(document.createTextNode("Cross References")));
347         xrefs.forEach((xref) => {
348           disclosureElement.appendChild(document.createTextNode(
349                   "    [" +
350                   xref.xrefNumber + "=" +
351                   bookNumberToName(xref.targetBook) + " " +
352                   xref.targetChapter + ":" +
353                   xref.targetVerse));
354           if (xref.targetEndId > 0) {
355             disclosureElement.appendChild(document.createTextNode(
356                     "-" +
357                     //   xref.targetEndBook+" " +
358                     xref.targetEndChapter + ":" +
359                     xref.targetEndVerse));
360           }
361           disclosureElement.appendChild(document.createTextNode("]"));
362         });
363         bookTextElement.appendChild(disclosureElement);
364         // bookTextElement.appendChild( document.createElement("br") );
365       }
366     });
367   });
368
369 </script>
370
371 </body>
372
```

```
373 </html>
374
```