```
 1 // Knockout View Model
 2 function ViewModel() {
 3     self.fileData = '';
 4     self.physicalFrames = ko.observable(16);
 5     self.currentIndex = ko.observable(0);
 6     self.fileInputNumber = ko.observable(1);
 7     self.lines = ko.observableArray([]);
 8     self.modelPcbs = [];
 9     self.pcbsObservable = ko.observableArray([]);
10     self.modelPageTables = [];
11     self.pageTablesObservable = ko.observableArray([]);
12     self.shouldShowMemoryRequestTitle = ko.observable(
   false);
13     self.shouldShowPhysMem = ko.observable(false);
14     self.shouldShowPCBTitle = ko.observable(false);
15     self.shouldShowPageTable = ko.observable(false);
16     self.shouldShowFreeFrameList = ko.observable(false);
17     self.requestedFile = ko.observable("input3a.data");
18     self.errorBanner = ko.observable("");
19     self.successBanner = ko.observable("");
20     self.physicalMemoryObservable = ko.observableArray([])
   ;
21     self.orderedProcesses = ko.observableArray([]);
22     self.showFooter = ko.observable(false);
23     self.showStats = ko.observable(false);
24     self.pages = ko.observable(0);
25     self.memRefs = ko.observable(0);
26     self.pageFaults = ko.observable(0);
27     self.processStatsObservable = ko.observableArray([]);
28     self.modelStats = [];
29
30     /**
31      * Retrieves all of the data from the file and sends
   it to the controller.
32      * Must be done in model as 404 error can only be
   checked after page load.
33      */
34     self.getAllDataFromFile = function() {
35         self.clearFields();
36         // The request for the file
37         let rawFile = new XMLHttpRequest();
38
39         // Clear out the current session
40         self.fileData = '';
41         self.lines([]);
```

```
42
43          // Open the file
44          rawFile.open("GET", "inputs/" + self.requestedFile
    (), true);
45
46          // If file is ready, then loop through line by
    line and push into the array
47          rawFile.onreadystatechange = function ()
48          {
49              if(rawFile.readyState === 4)
50              {
51                  if(rawFile.status === 200 || rawFile.
    status === 0)
52                  {
53                      self.fileData = rawFile.responseText.
    split('\n');
54                      /** @namespace self.fileData */
55                      for (let i = 0; i < self.fileData.
    length -1; i++){
56                          self.lines.push(self.fileData[i]);
57                      }
58                  }
59              }
60          };
61
62          // Send the file
63          rawFile.send(null);
64
65          // Check if the file was not found and update my
    error banner
66          rawFile.onloadend = function() {
67              if(rawFile.status === 404) {
68                  setError("File does not exist or is
    invalid. Please try again.");
69                  self.fileData = '';
70                  self.lines([]);
71                  setMemoryRequests(self.lines());
72                  self.shouldShowMemoryRequestTitle(false);
73                  self.shouldShowPCBTitle(false);
74                  self.pcbsObservable([]);
75                  self.fileInputNumber(1);
76              } else {
77                  self.errorBanner("");
78                  self.shouldShowMemoryRequestTitle(true);
79                  setMemoryRequests(self.lines());
```

```
 80                 self.fileInputNumber(2);
 81                 let cells = document.getElementById("
    memoryRequests").getElementsByTagName("td");
 82                 cells[0].style.backgroundColor = "#6ebcce
    ";
 83                 self.showFooter(true);
 84             }
 85         }
 86     };
 87
 88     /**
 89      * Clears the fields to restart the simulation
 90      */
 91     self.clearFields = function() {
 92         self.fileData = '';
 93         setMemoryRequests(self.lines());
 94         self.shouldShowMemoryRequestTitle(false);
 95         self.shouldShowPCBTitle(false);
 96         self.lines([]);
 97         self.pcbsObservable([]);
 98         self.modelPcbs = [];
 99         self.pageTablesObservable([]);
100         self.modelPageTables = [];
101         self.shouldShowPageTable(false);
102         self.shouldShowFreeFrameList(false);
103         self.fileInputNumber(1);
104         self.successBanner("");
105         self.currentIndex(0);
106         self.physicalFrames = ko.observable(16);
107         self.shouldShowPhysMem(false);
108         self.physicalMemoryObservable([]);
109         self.orderedProcesses([]);
110         self.showStats(false);
111         self.pages(0);
112         self.memRefs(0);
113         self.pageFaults(0);
114         self.processStatsObservable([]);
115         self.modelStats = [];
116         colorCell('freeFrameList', 0, 'clear', 'all');
117         clearController();
118     };
119
120     /**
121      * Runs the simulation either one step, one fault, or
    to completion.
```

```
122      * @param step moves the simulation one step
123      * @param fault moves the simulation one fault
124      * @param completion runs the simulation to
    completion
125      */
126     self.run = function (step, fault, completion) {
127         if (step) {
128             if (!(self.currentIndex() >= self.lines().
    length)){
129                 if (self.currentIndex() !== 0) {
130                     colorCell('loaded', 0, 'gray', self.
    currentIndex()-1);
131                 }
132                 colorCell('loaded', 0, 'blue', self.
    currentIndex());
133                 // Calls the controller to move forward
134                 let success = moveOneStep();
135                 if (success === -1) {
136                     colorCell('loaded', 0, 'red', self.
    currentIndex());
137                 }
138             } else {
139                 setSuccess("Simulation Complete");
140                 $('html, body').animate({ scrollTop: 0 },
     'fast');
141             }
142         }
143         else if (fault) {
144             let done = false;
145             while (!done) {
146                 if (!(self.currentIndex() >= self.lines()
    .length)){
147                     if (self.currentIndex() !== 0) {
148                         colorCell('loaded', 0, 'gray',
    self.currentIndex()-1);
149                     }
150                     colorCell('loaded', 0, 'blue', self.
    currentIndex());
151                     // Calls the controller to move
    forward
152                     let success = moveOneStep();
153                     if (success === -1) {
154                         colorCell('loaded', 0, 'red',
    self.currentIndex());
155                         done = true;
```

```
156                     }
157                     else if (success === 1) {
158                         done = true
159                     }
160                 } else {
161                     setSuccess("Simulation Complete");
162                     $('html, body').animate({ scrollTop:
    0 }, 'fast');
163                     done = true;
164                 }
165             }
166         }
167         else if (completion) {
168             let done = false;
169             while (!done) {
170                 if (!(self.currentIndex() >= self.lines()
    .length)){
171                     if (self.currentIndex() !== 0) {
172                         colorCell('loaded', 0, 'gray',
    self.currentIndex()-1);
173                     }
174                     colorCell('loaded', 0, 'blue', self.
    currentIndex());
175                     // Calls the controller to move
    forward
176                     let success = moveOneStep();
177                     if (success === -1) {
178                         colorCell('loaded', 0, 'red',
    self.currentIndex());
179                         done = true;
180                     }
181                 } else {
182                     setSuccess("Simulation Complete");
183                     $('html, body').animate({ scrollTop:
    0 }, 'fast');
184                     done = true;
185                 }
186             }
187         }
188
189     };
190
191     /**
192      * Colors in the cell of a table.
193      * @param table The table type to be changed
```

```
194       * @param tableNumber The table number that should be
     accessed
195       * @param color The new color of the cell
196       * @param index The index of the cell
197       */
198      function colorCell(table, tableNumber, color, index)
    {
199          // Color the memory requests table
200          if (table === 'loaded') {
201              let cells = document.getElementById("
    memoryRequests").getElementsByTagName("td");
202              if (color === 'gray') {
203                  cells[index].style.backgroundColor = "#
    868b94";
204              }
205              if (color === 'blue') {
206                  cells[index].style.backgroundColor = "#
    6ebcce";
207              }
208              if (color === 'red') {
209                  cells[index].style.backgroundColor = "#
    f44336";
210              }
211          }
212
213          // Color the most recent process in the current
    order list
214          if (table === "currentOrder") {
215              let cells = document.getElementById("
    processOrder").getElementsByTagName("td");
216
217              if (color === 'clear') {
218                  if (index % 2 === 0) {
219                      cells[index].style.backgroundColor =
    "";
220                  } else {
221                      cells[index].style.backgroundColor =
    "#f2f2f2";
222                  }
223              }
224              if (color === 'yellow') {
225                  cells[index].style.backgroundColor = "#
    fffb83";
226              }
227              if (color === 'gray') {
```

```
228                 cells[index].style.backgroundColor = "#
    868b94";
229             }
230         }
231 
232         // Color a PCB table
233         if (table === 'pcb') {
234             if (color === 'clear') {
235                 for (let i = 0; i < self.modelPcbs.length
    ; i++) {
236                     let cells = document.getElementById("
    pcb"+i).getElementsByTagName("tbody");
237                     cells[0].style.backgroundColor = "";
238                 }
239             } else {
240                 let cells = document.getElementById("pcb"
     + tableNumber).getElementsByTagName("tbody");
241                 if (color === 'yellow') {
242                     cells[index].style.backgroundColor =
    "#fffb83";
243                 }
244             }
245         }
246 
247         // Color the logical cell of a page table
248         if (table === 'pageTableLogical') {
249             let cell = document.getElementById("pgtV_" +
    tableNumber + "_" + index);
250             if (color === 'yellow') {
251                 cell.style.backgroundColor = "#fffb83";
252             }
253             if (color === 'clear') {
254                 if (index % 2 === 0) {
255                     cell.style.backgroundColor = "";
256                 } else {
257                     cell.style.backgroundColor = "#f2f2f2
    ";
258                 }
259             }
260         }
261 
262         // Color the physical cell of a page table
263         if (table === 'pageTablePhysical') {
264             let cell = document.getElementById("pgtP_" +
    tableNumber + "_" + index);
```

```
265             if (color === 'yellow') {
266                 cell.style.backgroundColor = "#fffb83";
267             }
268             if (color === 'clear') {
269                 if (index % 2 === 0) {
270                     cell.style.backgroundColor = "";
271                 } else {
272                     cell.style.backgroundColor = "#f2f2f2
    ";
273                 }
274             }
275         }
276 
277         // Color the Free Frame List
278         if (table === 'freeFrameList') {
279             let cells = document.getElementById("
    freeFrameTable").getElementsByTagName("td");
280             if (color === 'gray') {
281                 cells[index].style.backgroundColor = "#
    868b94";
282             }
283             if (color === 'yellow') {
284                 cells[index].style.backgroundColor = "#
    fffb83";
285             }
286             if (color === 'clear') {
287                 if (index === 'all') {
288                     for(let i = 0; i < cells.length; i++)
     {
289                         if (i % 2 === 0) {
290                             cells[i].style.
    backgroundColor = "";
291                         } else {
292                             cells[i].style.
    backgroundColor = "#f2f2f2";
293                         }
294                     }
295                 } else {
296                     if (index % 2 === 0) {
297                         cells[index].style.
    backgroundColor = "";
298                     } else {
299                         cells[index].style.
    backgroundColor = "#f2f2f2";
300                     }
```

```
301                 }
302             }
303         }
304
305         // Colors the frame number on the physical memory
306         if (table === 'physicalMemoryFrameNumber') {
307             let cell = document.getElementById('frameNum_
    ' + index);
308             if (color === 'gray') {
309                 cell.style.backgroundColor = "#868b94";
310             }
311             if (color === 'yellow') {
312                 cell.style.backgroundColor = "#fffb83";
313             }
314             if (color === 'clear') {
315                 if (index % 2 === 0) {
316                     cell.style.backgroundColor = "";
317                 } else {
318                     cell.style.backgroundColor = "#f2f2f2
    ";
319                 }
320             }
321         }
322
323         // Colors the frame number on the physical memory
324         if (table === 'physicalMemoryInUseBy') {
325             let cell = document.getElementById('frameUse_
    ' + index);
326             if (color === 'gray') {
327                 cell.style.backgroundColor = "#868b94";
328             }
329             if (color === 'yellow') {
330                 cell.style.backgroundColor = "#fffb83";
331             }
332             if (color === 'clear') {
333                 if (index % 2 === 0) {
334                     cell.style.backgroundColor = "";
335                 } else {
336                     cell.style.backgroundColor = "#f2f2f2
    ";
337                 }
338             }
339         }
340     }
341
```

```
342     /**
343      * Sets an error message on the page.
344      * @param message The error message to display
345      */
346     function setError(message) {
347         self.errorBanner(message);
348     }
349 
350     /**
351      * Sets a success message message on the page.
352      * @param message The error message to display
353      */
354     function setSuccess(message) {
355         self.successBanner(message);
356         self.showFooter(false);
357     }
358 
359     /**
360      * Adds a new PCB table to the page
361      * @param processName The PCB information to display
362      */
363     function addPCB(processName) {
364         self.modelPcbs.push(processName);
365         self.pcbsObservable.push(processName);
366         highlightPCB(processName);
367     }
368 
369     /**
370      * Highlights the PCB being accessed
371      * @param processName The PCB information being
    highlighted
372      */
373     function highlightPCB(processName) {
374         let tableNumber = self.modelPcbs.indexOf(
    processName);
375         colorCell('pcb', tableNumber, 'yellow', 0);
376     }
377 
378     /**
379      * Updates a page table to the page.
380      * @param processName The name of the process
381      * @param table The page table data structure
382      */
383     function updatePageTable(processName, table) {
384         let tableNumber = self.modelPcbs.indexOf(
```

```
384 processName);
385         self.modelPageTables[tableNumber] = table;
386         self.pageTablesObservable(self.modelPageTables[0]
    ); //Displays if it is the first element
387         self.pageTablesObservable(self.modelPageTables);
    //Displays all others
388     }
389 
390     /**
391      * Updates the stats observable.
392      * @param processName The name of the process
393      * @param stats the stats that have been changed
394      */
395     function updateStats(processName, stats) {
396         let tableNumber = self.modelPcbs.indexOf(
    processName);
397         self.modelStats[tableNumber] = stats;
398         self.processStatsObservable(self.modelStats[0]);
    //Displays if it is the first element
399         self.processStatsObservable(self.modelStats); //
    Displays all others
400     }
401 
402     /**
403      * Updates the free frame list table on the page.
404      * @param usedList The frames that have been used
405      * @param index the cell to highlight in the table
406      */
407     function updateFreeFrameList(usedList, index) {
408         for (let i = 0; i < usedList.length; i++) {
409             colorCell("freeFrameList", 0, "gray",
    usedList[i]);
410         }
411         colorCell("freeFrameList", 0, "yellow", index);
412         self.shouldShowFreeFrameList(true);
413     }
414 
415     /**
416      * Updates the current index.
417      * @param index The new index number
418      */
419     function updateIndex(index) {
420         self.currentIndex(index);
421     }
422 
```

```
423     /**
424      * Shows or hides the physical memory table.
425      * @param shouldShow whether the table should be
    shown or not
426      */
427     function showPhysMemory(shouldShow) {
428         self.shouldShowPhysMem(shouldShow);
429     }
430 
431     /**
432      * Basic setter method for the controller to update
    the physical memory through the model.
433      * @param physicalMemoryList the list to display
434      */
435     function updatePhysicalMemory(physicalMemoryList) {
436         self.physicalMemoryObservable(physicalMemoryList)
    ;
437     }
438 
439     /**
440      * Basic setter method for the controller to update
    the order of processes through the model.
441      * @param orderedProcesses the list to display
442      */
443     function updateOrderedProcesses(orderedProcesses) {
444         self.orderedProcesses(orderedProcesses);
445     }
446 
447     /**
448      * Sets the number of total pages.
449      * @param totalPages the number of pages
450      */
451     function updatePagesNumber(totalPages) {
452         self.pageFaults(totalPages);
453     }
454 
455     /**
456      * Sets the number of memory references.
457      * @param totalMemRefs the number of memory
    references
458      */
459     function updateMemRefs(totalMemRefs) {
460         self.pageFaults(totalMemRefs);
461     }
462 
```

```
463     /**
464      * Sets the number of page faults.
465      * @param pageFaults the number of page faults
466      */
467     function updatePageFaults(pageFaults) {
468         self.pageFaults(pageFaults);
469     }
470 
471     // Expose functions for controller
472     ViewModel.setError = setError;
473     ViewModel.addPCB = addPCB;
474     ViewModel.updatePageTable = updatePageTable;
475     ViewModel.updateFreeFrameList = updateFreeFrameList;
476     ViewModel.updateIndex = updateIndex;
477     ViewModel.colorCell = colorCell;
478     ViewModel.highlightPCB = highlightPCB;
479     ViewModel.showPhysMemory = showPhysMemory;
480     ViewModel.updatePhysicalMemory = updatePhysicalMemory
    ;
481     ViewModel.updateOrderedProcesses =
    updateOrderedProcesses;
482     ViewModel.updatePagesNumber = updatePagesNumber;
483     ViewModel.updateMemRefs = updateMemRefs;
484     ViewModel.updatePageFaults = updatePageFaults;
485     ViewModel.updateStats = updateStats;
486 }
487 
488 let myViewModel = new ViewModel();
489 
490 ko.applyBindings(myViewModel);
491 
```