# Create Task Written Responses

## 2a. Video

The program is intended to be used as an eBook reader and it was written entirely in JavaScript. The video illustrates the programs ability to resize the page in accordance to the browser window along with having an “actions bar” that has several functions such as a night mode and the ability to go to the next and previous page.

## 2b. Development Process

At the beginning I struggled with using the built in JavaScript File API to import standard JSON formatted files and load them synchronously so the other portions of the application can handle the data as a raw JavaScript object. I later figured out how to use the API by looking up some appropriate documentation. Since I was the only one programming the application, I had to know how each and every segment worked so I would be able to freely modify it however I see fit.

## 2c. Algorithm

|  |
| --- |
| resize: function() {  var windowSize = [  window.innerWidth,  window.innerHeight  ]   pageSize = renderer.object.meta.size;   if (windowSize[0] < pageSize[0] || windowSize[1] < pageSize[1]) {  if (windowSize[0] < pageSize[0] && windowSize[1] < pageSize[1]) {  var biggest;  var page;   if (windowSize[0] > windowSize[1]) {  biggest = windowSize[0];  page = pageSize[0];  }   else if (windowSize[0] < windowSize[1]) {  biggest = windowSize[1];  page = pageSize[1];  }   renderer.factor = page/biggest;  }   else if (windowSize[0] < pageSize[0] && !(windowSize[1] < pageSize[1])) {  renderer.factor = pageSize[0]/windowSize[0];  }   else if (windowSize[1] < pageSize[1] && !(windowSize[0] < pageSize[0])) {  renderer.factor = pageSize[1]/windowSize[1];  }  }   renderer.element.style.width = renderer.object.meta.size[0]/renderer.factor + 'px';  renderer.element.style.height = renderer.object.meta.size[1]/renderer.factor + 'px';   for (var c = 0; c < renderer.element.children.length; c++) {  var pageItem = renderer.object.pages[renderer.currentPage][c];  if (renderer.element.children[c].getAttribute('id') != 'nav' && pageItem != undefined) {  if (pageItem.type == 'text') {  renderer.element.children[c].style.fontSize =  pageItem.font.size/renderer.factor + 'px';   if (pageItem.size != undefined) {  var width = '';  var height = '';   if (pageItem.size[0] != undefined) {  width = pageItem.size[0]/renderer.factor + 'px';  }   if (pageItem.size[1] != undefined) {  height = pageItem.size[1]/renderer.factor + 'px';  }   renderer.element.children[c].style.width = width;  renderer.element.children[c].style.height = height;  }  }   if (pageItem.type == 'image') {  if (pageItem.size.mode == 'proportional') {  renderer.element.children[c].style.width =  (pageItem.size.dimensions[0] \* pageItem.size.scale)/renderer.factor + 'px';  renderer.element.children[c].style.height =  (pageItem.size.dimensions[1] \* pageItem.size.scale)/renderer.factor + 'px';  }  }   renderer.element.children[c].style.left =  (pageItem.position[0]/renderer.factor) + 'px';  renderer.element.children[c].style.top =  (pageItem.position[1]/renderer.factor) + 'px';  }  }  } |

## Explanation

The purpose of this algorithm was to proportionally scale the page of the ebook to match the window size so no scrollbars are visible. I accomplished this by going through a series of if else checks that determines what dimensions of the widow is larger than the corresponding dimensions of the page so it can scale one of the other appropriately.

## 2d. Abstraction

|  |
| --- |
| invert: function(hex) {  function padZero(string, length) {  length = length || 2;  var zeros = new Array(length).join('0');  return (zeros + string).slice(-length);  }   if (hex.indexOf('#') === 0) {  hex = hex.slice(1);  }   if (hex.length === 3) {  hex = hex[0] + hex[0] + hex[1] + hex[1] + hex[2] + hex[2];  }   if (hex.length !== 6) {  console.log(hex);  console.error('invalid hex value');  }   var rgb = [  (255 - parseInt(hex.slice(0, 2), 16)).toString(16),  (255 - parseInt(hex.slice(2, 4), 16)).toString(16),  (255 - parseInt(hex.slice(4, 6), 16)).toString(16)  ]   return '#' + padZero(rgb[0]) + padZero(rgb[1]) + padZero(rgb[2]); } |

## Explanation

This is a simple method that takes a hexadecimal color as input, and outputs a different hexadecimal color that is the mathematically opposite color as the input. Whenever I use this method all that I care about is that I'm giving it a color and getting a different color. I don't care how it works as long as it does work.