Criterion C: Development

Table of Contents

Programming languages, frameworks and libraries used

- JavaScript
- HTML
- CSS
- Bootstrap
- JQuery
- Firebase
- AES encryption-JS
- Scrypt

Documents

The Solution is split between three documents. The HTML file contains the layout and structure of the Solution. The CSS file contains the style of the solution, and essentially makes the Solution look good. The JavaScript file handles all the responsive elements of the Solution, and makes the Solution work.

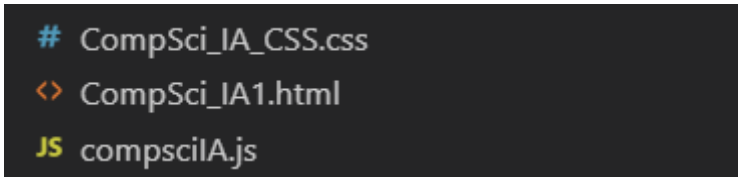

Figure 1: Documents

In addition to these documents, several other documents from outside libraries were used. The libraries used are: Bootstrap, JQuery, AES encryption-JS and Scrypt. The additional framework used is Firebase. These were linked at the top of the HTML file.

```
<!-- Bootstrap CSS -->
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css" integrity="sha384-MCw98/S

<!-- Bootstrap JS -->
<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfR
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js" integrity="sha384-ZMP7rVo3mIykV+2+9J3UJ4
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js" integrity="sha384-ChfqqxuZUCnJSK3+MXmPNIyE6

<link rel="stylesheet" type="text/css" href="CompSci_IA_CSS.css">

<!-- Firebase-->
<script src="https://www.gstatic.com/firebasejs/5.7.2/firebase.js"></script>


<!-- AES Encryption in Javascript -->
<script type="text/javascript" src="https://cdn.rawgit.com/ricmoo/aes-js/e27b99df/index.js"></script>

<!-- Password to key -->
<script src="https://raw.githubusercontent.com/ricmoo/scrypt-js/master/scrypt.js" type="text/javascript"></script>


<script src="compsciIA.js"></script>

<!-- end of working section -->
```

Figure 2: Links to Bootstrap, JQuery, Firebase, AES Encryption, Scrypt, and my own JS code

Starting Screen

The first screen that the client sees when opening the application was designed using HTML and CSS. The figures below show this screen:

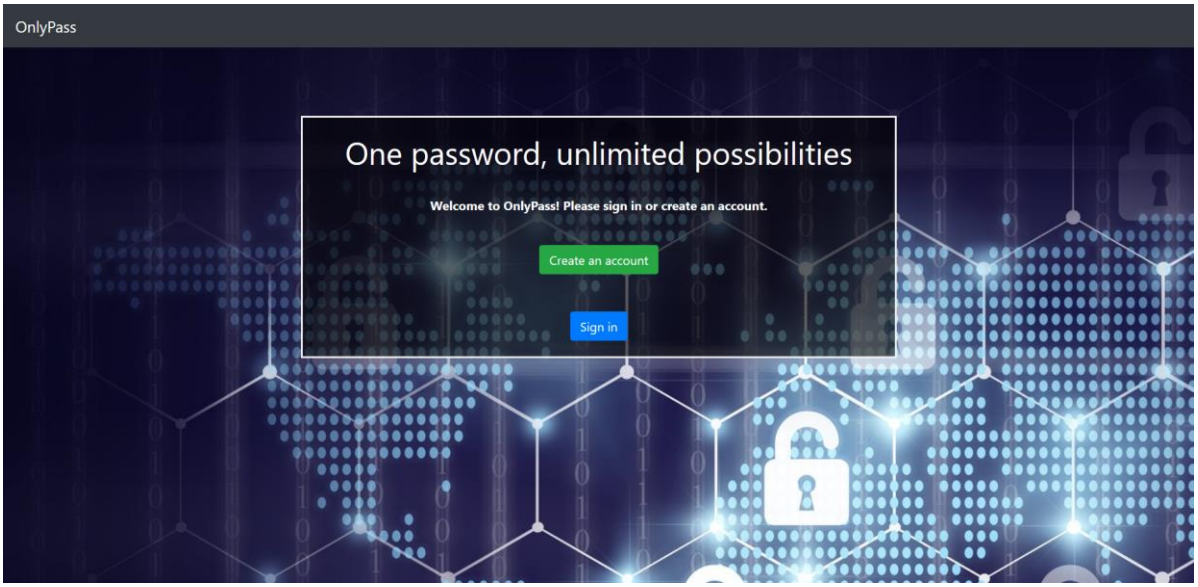

Figure 3: Starting Screen

The header at the top of this screen, where OnlyPass is displayed, was made using HTML and Bootstrap classes. The <nav> element was used, along with the classes navbar, navbar-expand-sm, bg-dark, and navbar-dark. Navbar-expand-sm is used to make the navbar mobile responsive. The other classes are used to make the navbar look the way it does.

```
<body id="body">
  <div>
  <nav class="navbar navbar-expand-sm bg-dark navbar-dark">
    <ul class="navbar-nav">
    <li class="nav-item">
      <a class="navbar-brand" href="#">OnlyPass</a>
    </li>
    </ul>
  </nav>
  </div>
```

Figure 4: Navbar HTML

The rest of the HTML code that deals with the starting screen is shown in figure 5, and the corresponding CSS in figure 6:

```
<div id="firstScreen">
  <br>
  <div class="container main-window">
    <h1 id="heading" class="">One password, unlimited possibilities</h1>
    <br>
    <p id="intro" class="">Welcome to OnlyPass! Please sign in or create an account.</p>
    <br>
    <div class="container">
      <button id="register_main" class="btn btn-success" onclick="document.getElementById('register').style.display='block'">Crea
    </div>
    <br><br>
    <div class="container">
      <button id="signin_main" class="btn btn-primary" onclick="openSignIn();">Sign in</button>
    </div>
  </div>
</div>
```

Figure 5: Starting screen HTML

```
.main-window {
    border: 3px solid #f1f1f1;
    background-color: rgb(0,0,0); /* Fallback color */
    background-color: rgba(0,0,0, 0.6); /* Black w/opacity/see-through */
    color: white;
    font-weight: bold;
    position: absolute;
    top: 40%;
    left: 50%;
    transform: translate(-50%, -50%);
    z-index: 2;
    width: 50%;
    padding: 20px;
    text-align: center;
}
```

Figure 6: Starting screen CSS

The entire HTML code for this section is enclosed in a element, for organization purposes. The rest of the relevant HTML is then given the class main-window. In the CSS, this class is defined to have a white border, a black background but without 100% opacity, and to be centered, among other things. Trivially, some text is displayed, and there are two buttons, which when clicked open the register or the sign in interfaces.

Register and sign in screens

The register and the sign in interfaces are pop-ups (modals), which are displayed on top of the starting page.

```css
/* The Modal (background) */
.modal {
  display: none; /* Hidden by default */
  position: fixed; /* Stay in place */
  z-index: 3; /* Sit on top */
  left: 0;
  top: 0;
  width: 100%; /* Full width */
  height: 100%; /* Full height */
  overflow: auto; /* Enable scroll if needed */
  background-color: rgb(0,0,0); /* Fallback color */
  background-color: rgba(0,0,0,0.4); /* Black w/ opacity */
  padding-top: 60px;
}
/* Modal Content/Box */
.modal-content {
  margin: 5% auto 15% auto; /* 5% from the top, 15% from the bottom and centered */
  border: 10px solid #888;
  width: 80%; /* Could be more or less, depending on screen size */
}
```

Figure 7: Pop-ups CSS

The pop-ups are implemented by having all HTML elements with the modal class be hidden by default, and by having a higher z-index than all other elements, which results in the pop-ups being on top of everything else, once they are unhidden. The pop-ups appear once the relevant aforementioned buttons are clicked, and the second line (overall line 165) in Figure 8 shows what code is used to make them appear. The second part of the openSignIn function in Figure 8 is an event listener, and listens for the enter key being pressed (13 is the keycode for enter). If the enter key is pressed while the sign-in pop-up is open, the program attempts to sign the user in by running the signIn() function, which will be discussed in greater detail later. The variable signInOpen (overall line 166) is set used so that pressing the enter key will only trigger the signIn() function if the sign in screen is open. The variable is of type Boolean, and is true if the sign in screen is open, and false otherwise.

```
function openSignIn(){
    document.getElementById('signin').style.display='block';
    signInOpen = true;

        // detect enter keypress
        $(document).keypress(function(e) {
            var keycode = (e.keyCode ? e.keyCode : e.which);
            if (keycode == '13') {
                if (signInOpen === true){
                    signIn();
                }
            }
        });
}
```

Figure 8: openSignIn function

```
<!-- Sign in screen-->
<div class="modal" id="signin">
  <form class="modal-content animate">
    <br>
    <div class="container">
      <div class="form-group">
        <br>
        <label for="email">Email address:</label>
        <input type="email" class="form-control" id="email" required>
      </div>
      <div class="form-group">
        <label for="pwd">Password:</label>
        <input type="password" class="form-control" id="pwd" required>
      </div>
      <div class="checkbox">
        <label><input type="checkbox"> Remember me</label>
      </div>
      <div class="error" id="error_incor">Incorrect username or password</div>
      <br>
    </div>
    <br>

  <div style="background-color:#f1f1f1; padding: 16px;">
    <button type="button" onclick="closeSignIn();" class="btn btn-danger">Cancel</button>
    <button type="button" style="float:right" class="btn btn-primary" id="button_signin">Sign in</button>
  </div>

  </form>
</div>
```

Figure 9: Sign in screen HTML

```
<!-- Create an account screen-->
<div class="modal" id="register">
  <form class="modal-content animate">
    <br>
    <div class="container">
      <div class="form-group">
      <label for="email">Email address:</label>
      <input type="email" class="form-control" id="email2" required>
      </div>
      <div class="form-group">
      <label for="pwd">Password:</label>
      <input type="password" class="form-control" id="pwd2" required>
      </div>
      <div class="form-group">
      <label for="pwd">Confirm password:</label>
      <input type="password" class="form-control" id="cpwd" required>
      </div>
      <div class="checkbox">
      <label><input type="checkbox" id="terms"> I have read the Terms and Conditions and agree to them.</label>
      </div>
      <div class="checkbox">
      <label><input type="checkbox"> You may send me occasional promotional material.</label>
      </div>
      <div class="error" id="error_match">Passwords don't match</div>
      <div class="error" id="error_short">Password is too short</div>
      <div class="error" id="error_number">Password must include a number</div>
      <div class="error" id="error_letter">Password must include a letter</div>
      <div class="error" id="error_terms">You must accept the Terms and Conditions</div>
      <br>
    </div>
    <br>

  <div style="background-color:#f1f1f1; padding: 16px;">
    <button type="button" onclick="closeRegister();" class="btn btn-danger">Cancel</button>
    <button type="submit" style="float:right" class="btn btn-success" id="button_register">Create an account</button>
  </div>

  </form>
</div>
```

Figure 10: Register screen HTML

Figures 9 and 10 show the HTML of the sign in and register screens, respectively. Both contain a form, which the user fills out. Both also contain errors, which are hidden by default by giving them the class error, which is defined in CSS to cause elements to be hidden by default. Furthermore, the class also causes elements to appear in red, to draw the user's attention to the error. Both screens contain two buttons, one to close the pop-up and one to sign in or register. The button clicks are handled directly (using onclick) for the cancel buttons, and Figure 11 shows the event listeners which handle the sign in and register buttons. Since the application is a single-page application, these are used to prevent the form from being submitted, thus preventing the page from being reloaded.

```
129  //Make it so the form isn't submitted when the Sign In button is clicked, but signIn() is run instead
130  $(document).ready(function() {
131      $("#button_signin").on("click", function(ev){
132          console.log("login clicked");
133          ev.preventDefault();
134          signIn();
135      });
136
137  });
138
139  //Make it so the form isn't submitted when the Create an Account button is clicked, but createUser() is run instead
140  $(document).ready(function() {
141      $("#button_register").on("click", function(ev){
142          console.log("register clicked");
143          ev.preventDefault();
144          createUser();
145      });
146
147  });
```

Figure 11: Event listeners for sign in and register buttons

Once the createUser() function is triggered, first all errors currently displayed will be hidden, so that there is a clean slate. Next, it will be checked whether the password and confirm password entries match. Moreover, the password will be checked against several criteria, namely whether it is sufficiently long and whether it contains both letters and numbers. Finally, it will be checked whether the user has accepted the Terms and conditions. Should any of these checks fail, the relevant error message will be displayed and the function will stop execution. Otherwise, the function will attempt to create a user.

```
47  //Create a user
48  function createUser(){
49      hideErrorsOnRegister();
50      var email = $("#email2").val();
51      var pwd = $('#pwd2').val();
52      if (pwd !== $('#cpwd').val()){
53          $("#error_match").css("display","block")
54          //alert("Passwords don't match!");
55          return;
56      } else if (pwd.length < 8) {
57          $("#error_short").css("display","block")
58          //alert("Password is too short!");
59          return;
60      } else if (pwd.match(/\d/) === null){
61          $("#error_number").css("display","block")
62          //alert("Password must include a number!");
63          return;
64      } else if (pwd.match(/\D/) === null) {
65          $("#error_letter").css("display","block")
66          //alert("Password must include a letter");
67          return;
68      } else if ($('#terms').is(':checked') === false) {
69          $("#error_terms").css("display","block")
70          //alert("You must accept the Terms and Conditions!");
71          return;
72      }
73      console.log($('#terms'));
74
75
76
77      console.log("trying to create user");
78      newUser = true;
79
80      firebase.auth().createUserWithEmailAndPassword(email, pwd).catch(function(error) {
81      // Handle Errors here.
82
83      var errorCode = error.code;
84      var errorMessage = error.message;
85      console.log("error creating user");
86      alert(errorMessage);
87      newUser = false;
88      //ADD SUCCESS MESSAGE
89      return;
90      // ...
91
92      });
93
94      }
```

Figure 12: createUser function

User authentication is done via Firebase. This framework makes it possible to use predefined functions (e.g. line 80) to create user accounts and authenticate users. Firebase also serves the purposes of a database.

Firebase and user authentication

Firebase first has to be initialized, which is show in the figure below. A link to the relevant Firebase project is established, and a connection with the online database is created. The fact that the

database is online makes it easier to sync passwords across devices. The potential drawback of unsecure storage is removed by only sending already encrypted data to Firebase. The encryption key is never sent.

```
4   // Initialize Firebase
5   var config = {
6       apiKey: "AIzaSyBt71Qd9Y9B56iua99Khs41NQxthxuysXc",
7       authDomain: "compsciia-f176a.firebaseapp.com",
8       databaseURL: "https://compsciia-f176a.firebaseio.com",
9       projectId: "compsciia-f176a",
10      storageBucket: "compsciia-f176a.appspot.com",
11      messagingSenderId: "567193459229"
12  };
13  firebase.initializeApp(config);
14
15  // Initialize Cloud Firestore through Firebase
16  var db = firebase.firestore();
17
18  // Disable deprecated features
19  db.settings({
20      timestampsInSnapshots: true
21  });
22
```

Figure 13: Initialization of Firebase

Once a new user account is created, the user has to sign in. This is done through the sign in screen as previously discussed, and the signIn() function is run.

```
99  //Sign in an existing user
100 function signIn(){
101     signInOpen = false;
102     $("#error_incor").css('display','none');
103     var email = $("#email").val();
104     var pwd = $("#pwd").val();
105     console.log("trying to log in",email,pwd);
106     firebase.auth().signInWithEmailAndPassword(email, pwd).catch(function(error) {
107         // Handle Errors here.
108         //alert("Incorrect username or password");
109         signInOpen = true;
110         $("#error_incor").css('display','block');
111         var errorCode = error.code;
112         var errorMessage = "This is an error";
113
114         // ...
115     });
116 }
```

Figure 14: signIn() function

A potential error message is first hidden, and then another predefined Firebase function is run. If there is no matching email and password pair, an error is displayed. Otherwise, the user is authenticated. There is an event listener that listens for a change of the authentication state, shown in Figures 15 and 16 below:

```
186   //Do this when a user signs in
187   var userId = null;
188   firebase.auth().onAuthStateChanged(function(user) {
189       if (user) {
190           if (newUser === false){
191           console.log("user signed in");
192           document.getElementById("signin").style.display='none';
193           document.getElementById("firstScreen").style.display='none';
194           document.getElementById("intro").style.display='none';
195           document.getElementById("intro").style.display='heading';
196           document.getElementById("secondScreen").style.display='inline';
197           document.getElementById("encryption_buttons").style.display='inline';
198           document.getElementById("save_status").style.display='inline';
199           $('body').css('background-image', '');
200
201           var name = user.displayName;
202           var email = user.email;
203
204
205           var user = firebase.auth().currentUser;
206           userId = user.uid;
207           console.log(userId);
```

Figure 15: Authentication listener (part 1)

Once the authentication listener detects that a user has signed in, all elements of the starting screen are hidden, and the main interface of the program is displayed.

```
docRef.get().then(function(doc) {
    if (doc.exists) {
        console.log("Document data:", doc.data());
        var docData = doc.data();
        console.log(docData);
        decrypt(docData.encrypted_data);

        //For all password entries, display those
        for (i = 0; i < passwords.length; i++){

            var table = document.getElementById('password_list');
            $("#password_list").attr('contenteditable', true);
            var row = table.insertRow(rowNumber);
            rowNumber++;
            var cell0 = row.insertCell(0); //document.createElement('td');
            var cell1 = row.insertCell(1);
            var cell2 = row.insertCell(2);
            var cell3 = row.insertCell(3);
            var cell4 = row.insertCell(4);
            var cell5 = row.insertCell(5);
            cell0.innerHTML = passwords[i].name;
            cell1.innerHTML = passwords[i].website;
            cell2.innerHTML = passwords[i].password;
            cell3.innerHTML = currentDate;
            cell4.innerHTML = "<button class='btn btn-primary'>Change</button>";
            cell5.innerHTML = "<button class='btn btn-danger' onclick='deleteEntry(this)'>Remove</button>";

            //row.appendChild(cell0);

            //cell0.setAttribute("class","editableField");

            cell0.className = "editableField";
            console.log(cell0.className);

        }
```

Figure 16: Authentication listener (part 2)

The program then obtains the encrypted password list, and runs the function to decrypt and parse it (see decryption section). Next, there is a loop which displays all the different passwords for different websites in an orderly table. Figure 17 shows the results that are displayed.

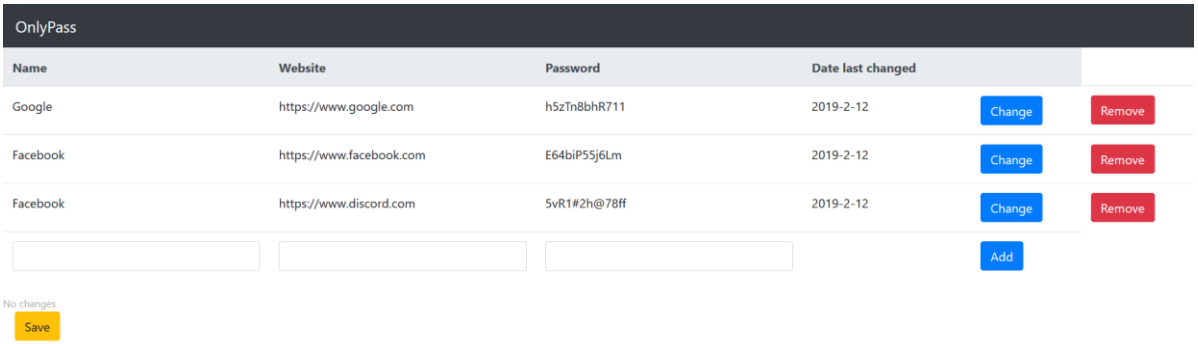

Figure 17: Password list screen

Making changes to the password list

Adding an entry

```
function addEntry(name, website, password){

    //backend --> adds the new entry to the array that is encrypted and stored
    var entry = new passwordEntry(name, website, password);
    passwords.push(entry);
    console.log(passwords);

    //frontend --> adds the new entry to the display
    var table = docu any getElementById('password_list');
    var row = table.insertRow(rowNumber);
    rowNumber++;
    var cell0 = row.insertCell(0);
    var cell1 = row.insertCell(1);
    var cell2 = row.insertCell(2);
    var cell3 = row.insertCell(3);
    var cell4 = row.insertCell(4);
    var cell5 = row.insertCell(5);
    cell0.innerHTML = name;
    cell1.innerHTML = website;
    cell2.innerHTML = password;
    cell3.innerHTML = currentDate;
    cell4.innerHTML = "<button class='btn btn-primary'>Change</button>";
    cell5.innerHTML = "<button class='btn btn-danger' onclick='deleteEntry(this)'>Remove</button>";
}
```

Figure 18: addEntry() function

Removing an entry

```
function deleteEntry(r){
    var i = r.parentNode.parentNode.rowIndex;
    passwords.splice(i-1,1);
    document.getElementById('password_table').deleteRow(i);
    rowNumber--;
    console.log(passwords);

}
```

Figure 19: deleteEntry() function

Encryption

AES-256 encryption is used. This is one of the most secure forms of encryption available and is therefore used as the client specifically wished for the encryption to be robust and as secure as possible.

Decrypting

```
468  //decrypts into plaintext
469  function decrypt(input){
470      var key = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 ];
471      var encryptedBytes = aesjs.utils.hex.toBytes(input);
472
473      // The counter mode of operation maintains internal state, so to
474      // decrypt a new instance must be instantiated.
475      var aesCtr = new aesjs.ModeOfOperation.ctr(key, new aesjs.Counter(5));
476      var decryptedBytes = aesCtr.decrypt(encryptedBytes);
477
478      // Convert our bytes back into text
479      var decryptedText = aesjs.utils.utf8.fromBytes(decryptedBytes);
480      console.log(decryptedText);
481      // "Text may be any length you wish, no padding is required."
482      //document.getElementById('decrypted_text').innerHTML=decryptedText;
483
484      //convert string to an array with password objects
485      parsePasswords(decryptedText);
486  }
```

Figure 20: decrypt() function

Encrypting

```
420  function encrypt() {
421
422      //convert password objects in the array to a string
423      stringifyPasswords();
424
425      // An example 128-bit key (16 bytes * 8 bits/byte = 128 bits)
426      var key = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 ];
427
428      // Convert text to bytes
429      var text = stringPasswords;
430      var textBytes = aesjs.utils.utf8.toBytes(text);
431
432      // The counter is optional, and if omitted will begin at 1
433      var aesCtr = new aesjs.ModeOfOperation.ctr(key, new aesjs.Counter(5));
434      var encryptedBytes = aesCtr.encrypt(textBytes);
435
436      // To print or store the binary data, you may convert it to hex
437      var encryptedHex = aesjs.utils.hex.fromBytes(encryptedBytes);
438      console.log(encryptedHex);
439      // "a338eda3874ed884b6199150d36f49988c90f5c47fe7792b0cf8c7f77eeffd87
440      //  ea145b73e82aefcf2076f881c88879e4e25b1d7b24ba2788"
441      //document.getElementById('a').innerHTML=encryptedHex;
442
443      var docRef = db.collection("users").doc(userId);
444      var setWithMerge = docRef.set({
445          encrypted_data: encryptedHex
446      }, { merge: true });
447
448
449
450      docRef.get().then(function(doc) {
451          if (doc.exists) {
452              console.log("Document data:", doc.data());
453              var docData = doc.data();
454              console.log(docData);
455              decrypt(docData.encrypted_data);
456
457
458          } else {
459              // doc.data() will be undefined in this case
460              console.log("No such document!");
461          }
462      }).catch(function(error) {
463          console.log("Error getting document:", error);
464      });
465
466  }
```

Figure 21: encrypt() function

The most important elements of the code have been highlighted here, but the entire code is available in the appendix.