

```

1 import React, { Component } from "react";
2 import { BrowserRouter as Router, Switch, Route, Link } from "react-router-dom";
3 import * as queryString from "query-string";
4
5 /**
6  * ./stitch is our MongoDB Stitch/Atlas interface
7  */
8 import { StitchService, LogItem, LogItemType } from "./stitch";
9
10 import "./styles.css";
11 import "./app.css";
12 import { ObjectId } from "bson";
13
14 const stitch = new StitchService();
15
16 /**
17  * Root componenet, holds everything in the app
18  */
19 export default class app extends Component {
20   constructor(props) {
21     super();
22     this.state = {};
23   }
24
25   render() {
26     return (
27       <Router>
28         <div>
29           <div className="header">
30             <h1>
31               <span style={{ color: "red" }}>C</span>
32               <span style={{ color: "green" }}>o</span>
33               <span style={{ color: "blue" }}>l</span>
34               <span style={{ color: "black" }}>o</span>
35               <span style={{ color: "pink" }}>u</span>
36               <span style={{ color: "brown" }}>r</span>
37               Password
38             </h1>
39           </div>
40           <Switch>
41             <Route path="/learn" component={Learn} />
42             <Route path="/password" component={Password} />
43             <Route path="/report" component={Report} />
44             <Route path="/" component={Homepage} />
45           </Switch>
46         </div>
47       </Router>
48     );
49   }
50 }
51
52 /**
53  * Homepage component, hold 2 buttons including "I am a participant"
54  * This component leads people to the /learn page
55  */
56 class Homepage extends React.Component {
57   constructor(props) {
58     super(props);
59     this.state = {};
60   }
61

```

```

62 render() {
63   return (
64     <div className="container main">
65       <h2>What would you like to do?</h2>
66       <div className="buttons">
67         <button className="bg-accent">
68           <Link to="/learn" className="color-inverse">
69             I am a participant
70           </Link>
71         </button>
72         <button>
73           <Link to="/report" className="color-primary">
74             FOR PROJECT TEAM USE ONLY
75           </Link>
76         </button>
77       </div>
78     </div>
79   );
80 }
81 }
82
83 /**
84  * Learn page. We don't really need this but we were making lots of changes
85  * to the code base and in the interest of not breaking things. We kept it.
86  * This page starts a new pathway for password scheme testing.
87  */
88 class Learn extends React.Component {
89   constructor(props) {
90     super(props);
91     this.state = {};
92   }
93
94   /**
95    * Authenticates the user, assigns a participant number and begins
96    * testing of the password scheme
97    * @param {string} type what type of password should we start with?
98    *                      === "email"
99    */
100   startPasswordScheme(type) {
101     // alert when something goes wrong
102     const failHandler = (err) => {
103       console.error(err);
104       alert("Something went wrong :(");
105     };
106     stitch
107       // authenticate the user
108       .login()
109       .then((stitchUser) => {
110         stitch
111           // create a new progress document in Atlas
112           .startProgress(stitchUser)
113           .then((progress) => {
114             stitch
115               // log that a participant started
116               .postLog(
117                 new LogItem(
118                   new Date(),
119                   new ObjectId(stitchUser.id),
120                   "-",
121                   LogItemType.CREATE_START,
122                   navigator.userAgent,

```

```

123         progress._id
124     )
125 )
126 .then(
127     () =>
128         // navigate to the password scheme testing url
129         (window.location.href = `/password?
action=create&type=${type}&progressId=${progress._id}`)
130     )
131     .catch(failHandler);
132 })
133     .catch(failHandler);
134 })
135     .catch(failHandler);
136 }
137
138 render() {
139     return (
140         <div id="learn" className="container main">
141             <p>Pick a type of password</p>
142             <button
143                 onClick={() => this.startPasswordScheme("email")}
144                 className="border-accent"
145             >
146                 Create for Email
147             </button>
148         </div>
149     );
150 }
151 }
152
153 /**
154  * Password scheme testing page. Handles creating passwords, confirming
155  * passwords and testing them. (The whole process)
156  */
157 class Password extends React.Component {
158     // define some constants (for the coloured circles)
159     blankColour = { name: "", id: 9 };
160     availableColours = [
161         { name: "#263238", id: 0 }, // black
162         { name: "#2196f3", id: 1 }, // blue
163         { name: "#795548", id: 2 }, // brown
164         { name: "#4caf50", id: 3 }, // green
165         { name: "#f48fb1", id: 4 }, // pink
166         { name: "#9c27b0", id: 5 }, // purple
167         { name: "#d32f2f", id: 6 }, // red
168         { name: "#ffeb3b", id: 7 }, // yellow
169     ];
170
171     constructor(props) {
172         super(props);
173
174         // get query options
175         const query = queryString.parse(props.location.search);
176
177         this.state = {
178             userName: "",
179             progressId: query.progressId,
180             action: query.action,
181             type: query.type,
182             progress: null,

```

```

183     circles: [],
184     password: [],
185     editingCircle: null,
186     confirmedPassword: null,
187     numTries: 0,
188   };
189
190   // bind functions
191   this.initCircles = this.initCircles.bind(this);
192   this.renderCircle = this.renderCircle.bind(this);
193   this.refreshPage = this.refreshPage.bind(this);
194   this.imReady = this.imReady.bind(this);
195   this.confirmPassword = this.confirmPassword.bind(this);
196   this.submitPassword = this.submitPassword.bind(this);
197
198   stitch
199     // get progress document from Atlas
200     .getProgress(this.state.progressId)
201     .then((progress) => {
202       if (!progress) return alert("progress not found!");
203       this.setState({ progress: progress }, () => {
204         if (this.state.action !== "create") {
205           // post some logs
206           stitch
207             .postLog(
208               new LogItem(
209                 new Date(),
210                 this.state.progress.userId,
211                 this.state.type,
212                 this.state.action === "confirm"
213                   ? LogItemType.CONFIRM_SHOW
214                   : LogItemType.TEST_SHOW,
215                 "-",
216                 this.state.progress._id
217               )
218             )
219             .catch((err) => {
220               console.error(err);
221               alert("Something went wrong when updating logs");
222             });
223         }
224         // generate new password if we're in the create stage
225         this.initCircles();
226       });
227     })
228     .catch((err) => {
229       console.error(err);
230       alert("something went wrong");
231     });
232 }
233
234 // build a password array (not necessarily a real one, can be blank)
235 buildCircles(blank) {
236   const circles = [];
237   for (let i = 0; i < 7; ++i) {
238     circles.push({
239       pos: i + 1,
240       colour: blank
241         ? this.blankColour
242         : this.availableColours[
243             Math.floor(Math.random() * this.availableColours.length)

```

```

244         ],
245     });
246 }
247 return circles;
248 }
249
250 /**
251  * Serialize a password for storage in Atlas.
252  * Also used for verifying a password -- if 2 serialized passwords match,
253  * then the passwords are the same
254  * @returns {string}
255  */
256 serializeCircles(circles) {
257     return circles.reduce((acc, cur) => (acc += cur.colour.id), "");
258 }
259
260 /**
261  * Converts a serialized password back into an object array
262  * @param {string} text
263  */
264 deserializeCircles(text) {
265     const circles = [];
266     for (let i = 0; i < 7; ++i) {
267         const colourId = parseInt(text.charAt(i));
268         circles.push({
269             pos: i + 1,
270             colour:
271                 this.availableColours.find((c) => c.id === colourId) ||
272                 this.blankColour,
273         });
274     }
275     return circles;
276 }
277
278 // geenrate / retrieve password for display
279 initCircles() {
280     const failHandler = (err) => {
281         console.error(err);
282         alert("Something went wrong");
283     };
284
285     const progress = this.state.progress;
286     const passwordName = this.state.type + "Password";
287     const password = progress[passwordName];
288     console.log(password);
289
290     // if a password has been made, just retrieve it
291     if (password) {
292         this.setState({
293             circles: this.deserializeCircles(password),
294             password: this.buildCircles(true),
295         });
296     } else {
297         // else create a new password, save it and add a log
298         progress[passwordName] = this.serializeCircles(this.buildCircles());
299         stitch
300             .updateProgress(progress)
301             .then(() => {
302                 stitch
303                     .postLog(
304                         new LogItem(

```

```

305         new Date(),
306         progress.userId,
307         this.state.type,
308         LogItemType.CREATE_PASSWORD,
309         progress[passwordName],
310         progress._id
311     )
312 )
313 .then(() =>
314     this.setState({
315         progress: progress,
316         circles: this.deserializeCircles(progress[passwordName]),
317         password: this.buildCircles(true),
318     })
319 )
320 .catch(failHandler);
321 })
322 .catch(failHandler);
323 }
324 }
325
326 /**
327  * Swap an existing password for a new one
328  */
329 refreshPage() {
330     const progress = this.state.progress;
331     const circles = this.buildCircles();
332     const newPassword = this.serializeCircles(circles);
333     progress[this.state.type + "Password"] = newPassword;
334
335     const failHandler = (err) => {
336         console.error(err);
337         alert("something went wrong");
338     };
339
340     stitch
341     .updateProgress(progress)
342     .then(() =>
343         stitch
344         // post a log that a password's been reset
345         .postLog(
346             new LogItem(
347                 new Date(),
348                 progress.userId,
349                 this.state.type,
350                 LogItemType.CREATE_RESET,
351                 newPassword,
352                 progress._id
353             )
354         )
355         .then(() =>
356             this.setState({
357                 circles,
358                 editingCircle: null,
359             })
360         )
361         .catch(failHandler)
362     )
363     .catch(failHandler);
364 }
365

```

```

366 /**
367  * Move on to the password-confirmation stage where the user inputs the
368  * password again to show they've memorised it
369  */
370 imReady() {
371   stitch
372     .postLog(
373       new LogItem(
374         new Date(),
375         this.state.progress.userId,
376         this.state.type,
377         LogItemType.CREATE_READY,
378         "-",
379         this.state.progress._id
380       )
381     )
382     .then(
383       () =>
384         (window.location.href = `/password?
action=confirm&type=${this.state.type}&progressId=${this.state.progressId}`)
385       )
386     .catch((err) => {
387       console.error(err);
388       alert("Something went wrong");
389     });
390 }
391
392 /**
393  * Function for changing the react state to editing a specific circle's
394  * colour
395  */
396 editCircle(circle) {
397   circle.refreshing = true;
398   this.setState({ editingCircle: circle }, () =>
399     setTimeout(() => {
400       circle.refreshing = false;
401       this.setState({ editingCircle: circle });
402     }, 75)
403   );
404 }
405
406 /**
407  * Checks that the entered password matches the actual password during the
408  * password confirmation stage.
409  */
410 confirmPassword() {
411   // serialize the passwords for verification
412   const enteredPassword = this.serializeCircles(this.state.password);
413   const actualPassword = this.serializeCircles(this.state.circles);
414
415   const failHandler = (err) => {
416     console.error(err);
417     alert("something went wrong");
418   };
419
420   if (enteredPassword !== actualPassword) {
421     // if the entered password is incorrect, post a log and let them try
422     // again
423     return stitch
424       .postLog(
425         new LogItem(

```

```

426         new Date(),
427         this.state.progress.userId,
428         this.state.type,
429         LogItemType.CONFIRM_INCORRECT,
430         enteredPassword,
431         this.state.progress._id
432     )
433 )
434 .then(() => alert("Sorry, the password you entered is incorrect"))
435 .catch(failHandler);
436 }
437
438 // if the entered password is correct, post a log and allow them to
439 // confirm the password again, return to the learn stage, or move on
440 stitch
441     .postLog(
442         new LogItem(
443             new Date(),
444             this.state.progress.userId,
445             this.state.type,
446             LogItemType.CONFIRM_CORRECT,
447             enteredPassword,
448             this.state.progress._id
449         )
450     )
451     .then(() => {
452         alert("You correctly entered the password!");
453
454         const action = prompt(
455             "What would you like to do next?\n" +
456             "Enter 'next' to skip to the next stage\n" +
457             "Enter 'learn' to see the current password again\n" +
458             "Enter 'confirm' to try entering the current password again"
459         );
460
461         console.log(action);
462
463         if (!action) return;
464
465         if (action === "learn") {
466             return (window.location.href = `/password?
action=create&type=${this.state.type}&progressId=${this.state.progressId}`);
467         } else if (action === "confirm") {
468             return this.setState({
469                 password: this.buildCircles(true),
470                 editingCircle: null,
471             });
472         }
473
474         // if they're moving on to the next stage, select the next password
475         // to be confirmed; or if they've all been confirmed, randomly select
476         // a type of password to be testing.
477         let type;
478         if (this.state.type === "email") {
479             type = "banking";
480         } else if (this.state.type === "banking") {
481             type = "shopping";
482         } else {
483             type = ["email", "banking", "shopping"][
484                 Math.floor(Math.random() * 3)
485             ];

```



```

486         return (
487             stitch
488                 // log that the confirm process is complete
489                 .postLog(
490                     new LogItem(
491                         new Date(),
492                         this.state.progress.userId,
493                         "-",
494                         LogItemType.CONFIRM_COMPLETE,
495                         "-",
496                         this.state.progress._id
497                     )
498                 )
499                 .then(
500                     () =>
501                         // navigate to the testing stage
502                         (window.location.href = `/password?
action=test&type=${type}&progressId=${this.state.progressId}`)
503                     )
504                 .catch(failHandler)
505             );
506         }
507         // move on to the next password
508         window.location.href = `/password?
action=create&type=${type}&progressId=${this.state.progressId}`;
509     })
510     .catch(failHandler);
511 }
512
513 /**
514  * Submit a password entered during the testing stage
515  */
516 submitPassword() {
517     const that = this;
518     // serialize the passwords for verification
519     const enteredPassword = this.serializeCircles(this.state.password);
520     const actualPassword = this.serializeCircles(this.state.circles);
521
522     const failHandler = (err) => {
523         console.error(err);
524         alert("something went wrong");
525     };
526
527     if (enteredPassword !== actualPassword) {
528         // if the entered password is incorrect, check how many tries the user
529         // has left
530         if (this.state.numTries < 2) {
531             // if we haven't used up our 3 tries, log the error and display the
532             // password again
533             stitch
534                 .postLog(
535                     new LogItem(
536                         new Date(),
537                         this.state.progress.userId,
538                         this.state.type,
539                         LogItemType.TEST_FAIL,
540                         enteredPassword,
541                         this.state.progress._id
542                     )
543                 )
544                 .then(() => {

```

```

545         alert(
546             `Sorry, the password you entered is incorrect\nYou have ${
547                 2 - this.state.numTries
548             } tries left`
549         );
550         this.setState({ numTries: this.state.numTries + 1 });
551     })
552     .catch(failHandler);
553 } else {
554     // if we've used up our 3 tries, log the failure and move on to the
555     // next password
556     stitch
557         .postLog(
558             new LogItem(
559                 new Date(),
560                 this.state.progress.userId,
561                 this.state.type,
562                 LogItemType.TEST_FAIL,
563                 enteredPassword,
564                 this.state.progress._id
565             )
566         )
567         .then(() =>
568             alert(
569                 "sorry, the password you entered is incorrect\nYou've used up all your
tries, taking you to the next password now."
570             )
571         )
572         .then(continueToNext)
573         .catch(failHandler);
574 }
575 } else {
576     stitch
577         .postLog(
578             new LogItem(
579                 new Date(),
580                 this.state.progress.userId,
581                 this.state.type,
582                 LogItemType.TEST_PASS,
583                 enteredPassword,
584                 this.state.progress._id
585             )
586         )
587         .then(() => alert("You correctly entered the password!"))
588         .then(continueToNext)
589         .catch(failHandler);
590 }
591 return;
592
593 function continueToNext() {
594     const buildParamName = (type) =>
595         "tested" + type.charAt(0).toUpperCase() + type.substr(1);
596
597     const progress = that.state.progress;
598
599     // mark the password type (email/banking/shopping) as tested
600     progress[buildParamName(that.state.type)] = true;
601
602     stitch
603         .updateProgress(progress)
604         .then(() => {

```

```

605 // filter out password types we've already tested
606 const types = ["shopping", "email", "banking"].filter(
607   (t) => !progress[buildParamName(t)]
608 );
609
610 // if we still have password types left to test, randomly pick one
611 // and test it
612 if (types.length > 0) {
613   window.location.href = `/password?action=test&type=${
614     types[Math.floor(Math.random() * types.length)]
615   }&progressId=${that.state.progressId}`;
616 } else {
617   // else log that we're done the password-scheme process
618   stitch
619     .postLog(
620       new LogItem(
621         new Date(),
622         that.state.progress.userId,
623         "-",
624         LogItemType.FINISH,
625         "-",
626         that.state.progress._id
627       )
628     )
629     .then(() => {
630       alert("Congrats, you've completed the process!");
631       // navigate to the questionnaire
632       window.location.href =
633         "https://hotsoft.carleton.ca/comp3008limesurvey/index.php/318265?
newtest=Y&lang=en";
634     })
635     .catch(failHandler);
636   }
637 })
638 .catch((err) => {
639   console.error(err);
640   alert("something went wrong");
641 });
642 }
643 }
644
645 /**
646  * Renders an HTML circle on the screen
647  * @param {{id: number, name: string, pos: number}} circle
648  */
649 renderCircle(circle) {
650   return (
651     <div
652       key={circle.pos}
653       className="circle-box bg-secondary"
654       // title="Click me to change my colour"
655       onClick={() => this.editCircle(circle)}
656     >
657       {circle.pos}
658       <div
659         className="circle"
660         style={{ backgroundColor: circle.colour.name }}
661       >></div>
662     </div>
663   );
664 }

```

```

665
666 /**
667  * Renderes an HTML pallete (for changing circle colours) on the screen
668  */
669 renderPalette() {
670   if (!this.state.editingCircle) return;
671   return (
672     <div
673       className={`palette ${
674         this.state.editingCircle.refreshing
675           ? "bg-secondary"
676           : "bg-secondary-strong"
677       }`}
678     >
679     <p>
680       <b>Palette : </b>Editing circle {this.state.editingCircle.pos}
681     </p>
682     <div className="colours">
683       {this.availableColours.map((colour) => (
684         <div
685           key={colour.id}
686           className="colour"
687           style={{
688             backgroundColor: colour.name,
689             border:
690               this.state.editingCircle.colour.id === colour.id
691               ? "solid 3px white"
692               : null,
693           }}
694           onClick={() => {
695             const editing = this.state.editingCircle;
696             editing.colour = colour;
697             this.setState({ editingCircle: editing });
698           }}
699         ></div>
700       )}}
701     </div>
702   </div>
703   );
704 }
705
706 /**
707  * Renders the HTML page for the create-password stage
708  */
709 renderForCreate() {
710   return (
711     <div id="password" className="container main">
712       <p>
713         This is your password for <b>{this.state.type}</b>, take some time to
714         remember it.
715       <br />
716       When you think you've got it, click on "I'm ready" to confirm the
717       password.
718       <br />
719       If you'd like a new password, click on "Refresh".
720     </p>
721     <div className="circles">
722       {this.state.circles.map(this.renderCircle)}
723     </div>
724     <div>
725       <button onClick={this.imReady} className="submitButton color-accent">

```

```

726         I'm ready
727     </button>
728     <button onClick={this.refreshPage} className="refreshButton">
729         Refresh
730     </button>
731 </div>
732 </div>
733 );
734 }
735
736 /**
737  * Renders the HTML page for the confirm-password stage
738  */
739 renderForConfirm() {
740     return (
741         <div id="password" className="container main confirm">
742             <p className="confirmText">
743                 {" "}
744                 Re-enter your <b>{this.state.type}</b> password then press the
745                 "Confirm Password" button
746             </p>
747             <div className="circles">
748                 {this.state.password.map(this.renderCircle)}
749             </div>
750             {this.renderPalette()}
751             <div className="submit">
752                 <button onClick={this.confirmPassword} className="confirmButton">
753                     Confirm Password
754                 </button>
755                 <a
756                     href={`\password?
action=create&type=${this.state.type}&progressId=${this.state.progressId}`}
757                     >
758                     <button onClick={this.refreshPage} className="refreshButton">
759                         Back to Learn
760                     </button>
761                 </a>
762             </div>
763         </div>
764     );
765 }
766
767 /**
768  * Renders the HTML page for the test-password stage
769  */
770 renderForTest() {
771     return (
772         <div id="password" className="container main confirm">
773             <p className="confirmText">
774                 {" "}
775                 Now that you've confirmed you know all your passwords, please enter
776                 your <b>{this.state.type}</b> password and press submit.
777             </p>
778             <div className="circles">
779                 {this.state.password.map(this.renderCircle)}
780             </div>
781             {this.renderPalette()}
782             <div className="submit">
783                 <button onClick={this.submitPassword} className="confirmButton">
784                     Submit
785                 </button>

```

```

786         </div>
787     </div>
788     );
789 }
790
791 /**
792  * Default render function. Actual result will depend on the state of the
793  * current progress.
794  */
795 render() {
796     return (
797         <div>
798             {this.state.action === "create" ? this.renderForCreate() : null}
799             {this.state.action === "confirm" ? this.renderForConfirm() : null}
800             {this.state.action === "test" ? this.renderForTest() : null}
801         </div>
802     );
803 }
804 }
805
806 /**
807  * Easter egg class. Not important.
808  */
809 class Report extends React.Component {
810     constructor(props) {
811         super(props);
812         this.state = {};
813         this.generate = this.generate.bind(this);
814     }
815
816     generate() {
817         alert("gotcha xD");
818     }
819
820     render() {
821         return (
822             <div id="learn" className="container main">
823                 <p>
824                     Generate report CSV. Can take some time, <b>BE PATIENT</b>
825                 </p>
826                 <button onClick={this.generate} className="bg-accent color-inverse">
827                     Click
828                 </button>
829             </div>
830         );
831     }
832 }
833

```