```
import React, { Component } from "react";
import { BrowserRouter as Router, Switch, Route, Link } from "react-router-dom";
import * as queryString from "query-string";

/**
 * ./stitch is our MongoDB Stitch/Atlas interface
 */
import { StitchService, LogItem, LogItemType } from "./stitch";

import "./styles.css";
import "./app.css";
import { ObjectId } from "bson";

const stitch = new StitchService();

/**
 * Root componenet, holds everything in the app
 */
export default class app extends Component {
  constructor(props) {
    super();
    this.state = {};
  }

  render() {
    return (
      <Router>
        <div>
          <div className="header">
            <h1>
              <span style={{ color: "red" }}>C</span>
              <span style={{ color: "green" }}>o</span>
              <span style={{ color: "blue" }}>l</span>
              <span style={{ color: "black" }}>o</span>
              <span style={{ color: "pink" }}>u</span>
              <span style={{ color: "brown" }}>r </span>
              Password
            </h1>
          </div>
          <Switch>
            <Route path="/learn" component={Learn} />
            <Route path="/password" component={Password} />
            <Route path="/report" component={Report} />
            <Route path="/" component={Homepage} />
          </Switch>
        </div>
      </Router>
    );
  }
}

/**
 * Homepage component, hold 2 buttons including "I am a participant"
 * This component leads people to the /learn page
 */
class Homepage extends React.Component {
  constructor(props) {
    super(props);
    this.state = {};
  }

```

```
  render() {
    return (
      <div className="container main">
        <h2>What would you like to do?</h2>
        <div className="buttons">
          <button className="bg-accent">
            <Link to="/learn" className="color-inverse">
              I am a participant
            </Link>
          </button>
          <button>
            <Link to="/report" className="color-primary">
              FOR PROJECT TEAM USE ONLY
            </Link>
          </button>
        </div>
      </div>
    );
  }
}

/**
 * Learn page. We don't really need this but we were making lots of changes
 * to the code base and in the interest of not breaking things. We kept it.
 * This page starts a new pathway for password scheme testing.
 */
class Learn extends React.Component {
  constructor(props) {
    super(props);
    this.state = {};
  }

  /**
   * Authenticates the user, assigns a participant number and begins
   * testing of the password scheme
   * @param {string} type what type of password should we start with?
   *                      === "email"
   */
  startPasswordScheme(type) {
    // alert when something goes wrong
    const failHandler = (err) => {
      console.error(err);
      alert("Something went wrong :(");
    };
    stitch
      // authenticate the user
      .login()
      .then((stitchUser) => {
        stitch
          // create a new progress document in Atlas
          .startProgress(stitchUser)
          .then((progress) => {
            stitch
              // log that a participant started
              .postLog(
                new LogItem(
                  new Date(),
                  new ObjectId(stitchUser.id),
                  "-",
                  LogItemType.CREATE_START,
                  navigator.userAgent,
```

```
                  progress._id
                )
              )
              .then(
                () =>
                  // navigate to the password scheme testing url
                  (window.location.href = `/password?
action=create&type=${type}&progressId=${progress._id}`)
              )
              .catch(failHandler);
          })
          .catch(failHandler);
      })
      .catch(failHandler);
  }

  render() {
    return (
      <div id="learn" className="container main">
        <p>Pick a type of password</p>
        <button
          onClick={() => this.startPasswordScheme("email")}
          className="border-accent"
        >
          Create for Email
        </button>
      </div>
    );
  }
}

/**
 * Password scheme testing page. Handles creating passwords, confirming
 * passwords and testing them. (The whole process)
 */
class Password extends React.Component {
  // define some constants (for the coloured circles)
  blankColour = { name: "", id: 9 };
  availableColours = [
    { name: "#263238", id: 0 }, // black
    { name: "#2196f3", id: 1 }, // blue
    { name: "#795548", id: 2 }, // brown
    { name: "#4caf50", id: 3 }, // green
    { name: "#f48fb1", id: 4 }, // pink
    { name: "#9c27b0", id: 5 }, // purple
    { name: "#d32f2f", id: 6 }, // red
    { name: "#ffeb3b", id: 7 }, // yellow
  ];

  constructor(props) {
    super(props);

    // get query options
    const query = queryString.parse(props.location.search);

    this.state = {
      userName: "",
      progressId: query.progressId,
      action: query.action,
      type: query.type,
      progress: null,
```

```
    circles: [],
    password: [],
    editingCircle: null,
    confirmedPassword: null,
    numTries: 0,
  };

  // bind functions
  this.initCircles = this.initCircles.bind(this);
  this.renderCircle = this.renderCircle.bind(this);
  this.refreshPage = this.refreshPage.bind(this);
  this.imReady = this.imReady.bind(this);
  this.confirmPassword = this.confirmPassword.bind(this);
  this.submitPassword = this.submitPassword.bind(this);

  stitch
    // get progress document from Atlas
    .getProgress(this.state.progressId)
    .then((progress) => {
      if (!progress) return alert("progress not found!");
      this.setState({ progress: progress }, () => {
        if (this.state.action !== "create") {
          // post some logs
          stitch
            .postLog(
              new LogItem(
                new Date(),
                this.state.progress.userId,
                this.state.type,
                this.state.action === "confirm"
                  ? LogItemType.CONFIRM_SHOW
                  : LogItemType.TEST_SHOW,
                "-",
                this.state.progress._id
              )
            )
            .catch((err) => {
              console.error(err);
              alert("Something went wrong when updating logs");
            });
        }
        // generate new password if we're in the create stage
        this.initCircles();
      });
    })
    .catch((err) => {
      console.error(err);
      alert("something went wrong");
    });
}

// build a password array (not neccessarily a real one, can be blank)
buildCircles(blank) {
  const circles = [];
  for (let i = 0; i < 7; ++i) {
    circles.push({
      pos: i + 1,
      colour: blank
        ? this.blankColour
        : this.availableColours[
            Math.floor(Math.random() * this.availableColours.length)
```

```
            ],
      });
    }
    return circles;
  }

  /**
   * Serialize a password for storage in Atlas.
   * Also used for verifying a password -- if 2 serialized passwords match,
   * then the passwords are the same
   * @returns {string}
   */
  serializeCircles(circles) {
    return circles.reduce((acc, cur) => (acc += cur.colour.id), "");
  }

  /**
   * Converts a serialized password back into an object array
   * @param {string} text
   */
  deserializeCircles(text) {
    const circles = [];
    for (let i = 0; i < 7; ++i) {
      const colourId = parseInt(text.charAt(i));
      circles.push({
        pos: i + 1,
        colour:
          this.availableColours.find((c) => c.id === colourId) ||
          this.blankColour,
      });
    }
    return circles;
  }

  // geenrate / retrieve password for display
  initCircles() {
    const failHandler = (err) => {
      console.error(err);
      alert("Something went wrong");
    };

    const progress = this.state.progress;
    const passwordName = this.state.type + "Password";
    const password = progress[passwordName];
    console.log(password);

    // if a password has been made, just retrieve it
    if (password) {
      this.setState({
        circles: this.deserializeCircles(password),
        password: this.buildCircles(true),
      });
    } else {
      // else create a new password, save it and add a log
      progress[passwordName] = this.serializeCircles(this.buildCircles());
      stitch
        .updateProgress(progress)
        .then(() => {
          stitch
            .postLog(
              new LogItem(
```

```
              new Date(),
              progress.userId,
              this.state.type,
              LogItemType.CREATE_PASSWORD,
              progress[passwordName],
              progress._id
            )
          )
          .then(() =>
            this.setState({
              progress: progress,
              circles: this.deserializeCircles(progress[passwordName]),
              password: this.buildCircles(true),
            })
          )
          .catch(failHandler);
      })
      .catch(failHandler);
  }
}

/**
 * Swap an existing password for a new one
 */
refreshPage() {
  const progress = this.state.progress;
  const circles = this.buildCircles();
  const newPassword = this.serializeCircles(circles);
  progress[this.state.type + "Password"] = newPassword;

  const failHandler = (err) => {
    console.error(err);
    alert("something went wrong");
  };

  stitch
    .updateProgress(progress)
    .then(() =>
      stitch
        // post a log that a password's been reset
        .postLog(
          new LogItem(
            new Date(),
            progress.userId,
            this.state.type,
            LogItemType.CREATE_RESET,
            newPassword,
            progress._id
          )
        )
        .then(() =>
          this.setState({
            circles,
            editingCircle: null,
          })
        )
        .catch(failHandler)
    )
    .catch(failHandler);
}
```

```
  /**
   * Move on to the password-confirmation stage where the user inputs the
   * password again to show they've memorised it
   */
  imReady() {
    stitch
      .postLog(
        new LogItem(
          new Date(),
          this.state.progress.userId,
          this.state.type,
          LogItemType.CREATE_READY,
          "-",
          this.state.progress._id
        )
      )
      .then(
        () =>
          (window.location.href = `/password?
action=confirm&type=${this.state.type}&progressId=${this.state.progressId}`)
      )
      .catch((err) => {
        console.error(err);
        alert("Something went wrong");
      });
  }

  /**
   * Function for changing the react state to editing a specific circle's
   * colour
   */
  editCircle(circle) {
    circle.refreshing = true;
    this.setState({ editingCircle: circle }, () =>
      setTimeout(() => {
        circle.refreshing = false;
        this.setState({ editingCircle: circle });
      }, 75)
    );
  }

  /**
   * Checks that the entered password matches the actual password during the
   * password confirmation stage.
   */
  confirmPassword() {
    // serialize the passwords for verification
    const enteredPassword = this.serializeCircles(this.state.password);
    const actualPassword = this.serializeCircles(this.state.circles);

    const failHandler = (err) => {
      console.error(err);
      alert("something went wrong");
    };

    if (enteredPassword !== actualPassword) {
      // if the entered password is incorrect, post a log and let them try
      // again
      return stitch
        .postLog(
          new LogItem(
```

```
            new Date(),
            this.state.progress.userId,
            this.state.type,
            LogItemType.CONFIRM_INCORRECT,
            enteredPassword,
            this.state.progress._id
          )
        )
        .then(() => alert("Sorry, the password you entered is incorrect"))
        .catch(failHandler);
    }

    // if the entered password is correct, post a log and allow them to
    // confirm the password again, return to the learn stage, or move on
    stitch
      .postLog(
        new LogItem(
          new Date(),
          this.state.progress.userId,
          this.state.type,
          LogItemType.CONFIRM_CORRECT,
          enteredPassword,
          this.state.progress._id
        )
      )
      .then(() => {
        alert("You correctly entered the password!");

        const action = prompt(
          "What would you like to do next?\n" +
            "Enter 'next' to skip to the next stage\n" +
            "Enter 'learn' to see the current password again\n" +
            "Enter 'confirm' to try entering the current password again"
        );

        console.log(action);

        if (!action) return;

        if (action === "learn") {
          return (window.location.href = `/password?
action=create&type=${this.state.type}&progressId=${this.state.progressId}`);
        } else if (action === "confirm") {
          return this.setState({
            password: this.buildCircles(true),
            editingCircle: null,
          });
        }

        // if they're moving on to the next stage, select the next password
        // to be confirmed; or if they've all been confirmed, randomly select
        // a type of password to being testing.
        let type;
        if (this.state.type === "email") {
          type = "banking";
        } else if (this.state.type === "banking") {
          type = "shopping";
        } else {
          type = ["email", "banking", "shopping"][
            Math.floor(Math.random() * 3)
          ];
```

```
          return (
            stitch
              // log that the confirm process is complete
              .postLog(
                new LogItem(
                  new Date(),
                  this.state.progress.userId,
                  "-",
                  LogItemType.CONFIRM_COMPLETE,
                  "-",
                  this.state.progress._id
                )
              )
              .then(
                () =>
                  // navigate to the testing stage
                  (window.location.href = `/password?
action=test&type=${type}&progressId=${this.state.progressId}`)
              )
              .catch(failHandler)
          );
        }
        // move on to the next password
        window.location.href = `/password?
action=create&type=${type}&progressId=${this.state.progressId}`;
      })
      .catch(failHandler);
  }

  /**
   * Submit a password entered during the testing stage
   */
  submitPassword() {
    const that = this;
    // serialize the passwords for verification
    const enteredPassword = this.serializeCircles(this.state.password);
    const actualPassword = this.serializeCircles(this.state.circles);

    const failHandler = (err) => {
      console.error(err);
      alert("something went wrong");
    };

    if (enteredPassword !== actualPassword) {
      // if the entered password is incorrect, check how many tries the user
      // has left
      if (this.state.numTries < 2) {
        // if we haven't used up our 3 tries, log the error and display the
        // password again
        stitch
          .postLog(
            new LogItem(
              new Date(),
              this.state.progress.userId,
              this.state.type,
              LogItemType.TEST_FAIL,
              enteredPassword,
              this.state.progress._id
            )
          )
          .then(() => {
```

```
            alert(
              `Sorry, the password you entered is incorrect\nYou have ${
                2 - this.state.numTries
              } tries left`
            );
            this.setState({ numTries: this.state.numTries + 1 });
          })
          .catch(failHandler);
      } else {
        // if we've used up our 3 tries, log the failure and move on to the
        // next password
        stitch
          .postLog(
            new LogItem(
              new Date(),
              this.state.progress.userId,
              this.state.type,
              LogItemType.TEST_FAIL,
              enteredPassword,
              this.state.progress._id
            )
          )
          .then(() =>
            alert(
              "sorry, the password you entered is incorrect\nYou've used up all your tries, taking you to the next password now."
            )
          )
          .then(continueToNext)
          .catch(failHandler);
      }
    } else {
      stitch
        .postLog(
          new LogItem(
            new Date(),
            this.state.progress.userId,
            this.state.type,
            LogItemType.TEST_PASS,
            enteredPassword,
            this.state.progress._id
          )
        )
        .then(() => alert("You correctly entered the password!"))
        .then(continueToNext)
        .catch(failHandler);
    }
    return;

    function continueToNext() {
      const buildParamName = (type) =>
        "tested" + type.charAt(0).toUpperCase() + type.substr(1);

      const progress = that.state.progress;

      // mark the password type (email/banking/shopping) as tested
      progress[buildParamName(that.state.type)] = true;

      stitch
        .updateProgress(progress)
        .then(() => {
```

```
          // filter out password types we've already tested
          const types = ["shopping", "email", "banking"].filter(
            (t) => !progress[buildParamName(t)]
          );

          // if we still have password types left to test, randomly pick one
          // and test it
          if (types.length > 0) {
            window.location.href = `/password?action=test&type=${
              types[Math.floor(Math.random() * types.length)]
            }&progressId=${that.state.progressId}`;
          } else {
            // else log that we're done the password-scheme process
            stitch
              .postLog(
                new LogItem(
                  new Date(),
                  that.state.progress.userId,
                  "-",
                  LogItemType.FINISH,
                  "-",
                  that.state.progress._id
                )
              )
              .then(() => {
                alert("Congrats, you've completed the process!");
                // navigate to the questionnaire
                window.location.href =
                  "https://hotsoft.carleton.ca/comp3008limesurvey/index.php/318265?
newtest=Y&lang=en";
              })
              .catch(failHandler);
          }
        })
        .catch((err) => {
          console.error(err);
          alert("something went wrong");
        });
    }
  }

  /**
   * Renders an HTML circle on the screen
   * @param {{id: number, name: string, pos: number}} circle
   */
  renderCircle(circle) {
    return (
      <div
        key={circle.pos}
        className="circle-box bg-secondary"
        // title="Click me to change my colour"
        onClick={() => this.editCircle(circle)}
      >
        {circle.pos}
        <div
          className="circle"
          style={{ backgroundColor: circle.colour.name }}
        ></div>
      </div>
    );
  }
```

```
665
666   /**
667    * Renderes an HTML pallete (for changing circle colours) on the screen
668    */
669   renderPallette() {
670     if (!this.state.editingCircle) return;
671     return (
672       <div
673         className={`pallette ${
674           this.state.editingCircle.refreshing
675             ? "bg-secondary"
676             : "bg-secondary-strong"
677         }`}
678       >
679         <p>
680           <b>Pallette : </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">
```

```
            I'm ready
          </button>
          <button onClick={this.refreshPage} className="refreshButton">
            Refresh
          </button>
        </div>
      </div>
    );
  }

  /**
   * Renders the HTML page for the confirm-password stage
   */
  renderForConfirm() {
    return (
      <div id="password" className="container main confirm">
        <p className="confirmText">
          {" "}
          Re-enter your <b>{this.state.type}</b> password then press the
          "Confirm Password" button
        </p>
        <div className="circles">
          {this.state.password.map(this.renderCircle)}
        </div>
        {this.renderPallette()}
        <div className="submit">
          <button onClick={this.confirmPassword} className="confirmButton">
            Confirm Password
          </button>
          <a
            href={`/password?
action=create&type=${this.state.type}&progressId=${this.state.progressId}`}
          >
            <button onClick={this.refreshPage} className="refreshButton">
              Back to Learn
            </button>
          </a>
        </div>
      </div>
    );
  }

  /**
   * Renders the HTML page for the test-password stage
   */
  renderForTest() {
    return (
      <div id="password" className="container main confirm">
        <p className="confirmText">
          {" "}
          Now that you've confirmed you know all your passwords, please enter
          your <b>{this.state.type}</b> password and press submit.
        </p>
        <div className="circles">
          {this.state.password.map(this.renderCircle)}
        </div>
        {this.renderPallette()}
        <div className="submit">
          <button onClick={this.submitPassword} className="confirmButton">
            Submit
          </button>
```

```
        </div>
      </div>
    );
  }

  /**
   * Default render function. Actual result will depend on the state of the
   * current progress.
   */
  render() {
    return (
      <div>
        {this.state.action === "create" ? this.renderForCreate() : null}
        {this.state.action === "confirm" ? this.renderForConfirm() : null}
        {this.state.action === "test" ? this.renderForTest() : null}
      </div>
    );
  }
}

/**
 * Easter egg class. Not important.
 */
class Report extends React.Component {
  constructor(props) {
    super(props);
    this.state = {};
    this.generate = this.generate.bind(this);
  }

  generate() {
    alert("gotcha xD");
  }

  render() {
    return (
      <div id="learn" className="container main">
        <p>
          Generate report CSV. Can take some time, <b>BE PATIENT</b>
        </p>
        <button onClick={this.generate} className="bg-accent color-inverse">
          Click
        </button>
      </div>
    );
  }
}

```