This project was bootstrapped with Create React App.

About

This is one half of the Revitalize Virtual Wellness System. This repository contains the website to be used by clients of the Revitalize program. The Revitalize API can be found here, written in Django.

Contributors

- Luke Schipper
- Alfred Uzokwe
- Kareem Abulez

Running the App

Initial Setup

1. Clone this repo and the Revitalize API repo.
2. Follow the setup instructions in the API repo.
3. Ensure Node.js and NPM are installed.
4. Open this repo in your IDE of choice (VSCode, Webstorm, etc.)
5. Open a command line with this project set as the current directory. Run `npm i`.

Starting the Project

Ensure the Revitalize API is running before starting this project. See this for instructions. In the project directory, you can run:

```
npm start
```

Runs the app in the development mode.
Open http://localhost:3000 to view it in the browser.

The page will reload if you make edits.
You will also see any lint errors in the console.

```
npm test
```

Launches the test runner in the interactive watch mode.
See the section about running tests for more information.

```
npm run build
```

Builds the app for production to the `build` folder.
It correctly bundles React in production mode and optimizes the build for the best performance.

The build is minified and the filenames include the hashes.

See the section about deployment for more information.

Third Party Dependencies

Name	Description
`@material-ui/core`	The app folows standard material theming using the various components from this library. For more information on material theming, see this.
`i18next`	A library that streamlines internationalization, including changing between different languages.
`lodash`	A popular JS library for simplifying common operations.
`moment`	A popular JS library for formatting and displaying dates in correct locale format.
`notistack`	A useful library for functionally creating and stacking material-ui snackbars.
`react-redux`	Provides standard component-level redux integration. See Redux Store.
`react-router-dom`	Component routing for react.
`recharts`	A graphing library for generating dynamically sized graphs. Used for showing goal progress.
`redux-form`	A useful library that ties all form states to the redux store.
`redux-logger`	Middleware for logging all redux store state updates.

Redux Store

This app has one global redux store. Included in this store is:

- authentication
- alert
- profile
- form

Authentication

```
export function authentication(state = initialState, action) {
    switch (action.type) {
        case userConstants.LOGIN_REQUEST:
            return {
                loggingIn: true,
                user: action.user,
            };
        case userConstants.REFRESH_REQUEST:
            return {
                loggingIn: true,
                user: state.user,
            }
```

```
        case userConstants.LOGIN_SUCCESS:
        case userConstants.REFRESH_SUCCESS:
            return {
                loggedIn: true,
                user: action.user,
            };
        case userConstants.LOGIN_FAILURE:
        case userConstants.REFRESH_FAILURE:
        case userConstants.LOGOUT:
            return {
                loggedIn: false,
                error: action.error,
            };
        default:
            return state;
    }
}
```

`authentication.reducer.js` describes the authentication state. Because the app uses the JWT standard, actions can include LOGIN_REQUEST, LOGIN_SUCCESS, or LOGIN_FAILURE for whenever the the user logs in with their username and password. When the user's access token expires, the authentication state must be refreshed, resulting in a REFRESH_REQUEST, then a REFRESH_SUCCESS or REFRESH_FAILURE.

Alert

```
export function alert(state = {}, action) {
    switch (action.type) {
        case alertConstants.SUCCESS:
            return {
                type: 'alert-success',
                message: action.message,
            };
        case alertConstants.ERROR:
            return {
                type: 'alert-danger',
                message: action.message,
            };
        case alertConstants.CLEAR:
            return {};
        default:
            return state;
    }
}
```

`alert.reducer.js` is used to propagate messages throughout the app.

Profile

```
export function profile(state = {}, action) {
    switch (action.type) {
        case profileConstants.PROFILE_REQUEST:
            return {
                loadingProfile: true,
            };
        case profileConstants.PROFILE_SUCCESS:
            return {
                profileLoaded: true,
                payload: action.profile,
            };
        case profileConstants.PROFILE_FAILURE:
            return {};
        default:
            return state;
    }
}
```

`profile.reducer.js` describes the the user's profile, which is retrieved upon initial log in.

Form

This state is handle entirely by redux form. All form states are within this state.

Survey Generation

A primary feature of Revitalize VWS, surveys can be completed by a user on a regular basis. Surveys have indicator values associated with them indicating aspects of mental and physical health. Such surveys are generated dynamically based on a JSON from the API. This section will give an overview of the component structure of survey generation.

Below is an example of a survey JSON. Continue reading afterwards for more information.

```
{
    "name" : "Self-Efficacy for Managing Chronic Disease 6-item Scale",
    "id" : 1,
    "description" : "This 6-item scale contains items taken from several SE scales
developed for the Chronic Disease SelfManagement study.",
    "number_of_elements" : 7,
    "elements" : [
        {
            "element_type" : "text",
            "text" : "We would like to know how confident you are in doing certain
activities. For each of the following questions, please choose the number that
corresponds to your confidence that you can do the tasks regularly at the present
time."
        },
        {
            "element_type" : "question_group",
            "number" : null,
```

```
            "question_group_type" : "integer_range",
            "question_group_type_data" : {
                "minimum" : 1,
                "maximum" : 10,
                "step" : 1,
                "initial" : 6,
                "labels" : [ "1", "2", "3", "4", "5", "6", "7", "8", "9", "10" ],
                "annotations" : {
                    "minimum" : "not at all confident",
                    "maximum" : "totally confident",
            },
            "text" : null,
            "number_of_questions" : 1,
            "questions" : [
                {
                    "number" : "1",
                    "text" : "How confident do you feel that you can keep the
fatigue caused by your disease from interfering with the things you want to do?",
                    "help_text" : null
                }
            ]
        },
        {
            "element_type" : "question_group",
            "number" : null,
            "question_group_type" : "integer_range",
            "question_group_type_data" : {
                "minimum" : 1,
                "maximum" : 10,
                "step" : 1,
                "initial" : 6,
                "labels" : [ "1", "2", "3", "4", "5", "6", "7", "8", "9", "10" ],
                "annotations" : {
                    "minimum" : "not at all confident",
                    "maximum" : "totally confident",
                }
            },
            "text" : null,
            "number_of_questions" : 1,
            "questions" : [
                {
                    "number" : "2",
                    "text" : "How confident do you feel that you can keep the
physical discomfort or pain of your disease from interfering with the things you
want to do?",
                    "help_text" : null
                }
            ]
        },
        {
            "element_type" : "question_group",
            "number" : null,
            "question_group_type" : "integer_range",
            "question_group_type_data" : {
```

```
                "minimum" : 1,
                "maximum" : 10,
                "step" : 1,
                "initial" : 6,
                "labels" : [ "1", "2", "3", "4", "5", "6", "7", "8", "9", "10" ],
                "annotations" : {
                    "minimum" : "not at all confident",
                    "maximum" : "totally confident",
                }
            },
            "text" : null,
            "number_of_questions" : 1,
            "questions" : [
                {
                    "number" : "3",
                    "text" : "How confident do you feel that you can keep the
emotional distress caused by your disease from interfering with the things you
want to do?",
                    "help_text" : null
                }
            ]
        },
        {
            "element_type" : "question_group",
            "number" : null,
            "question_group_type" : "integer_range",
            "question_group_type_data" : {
                "minimum" : 1,
                "maximum" : 10,
                "step" : 1,
                "initial" : 6,
                "labels" : [ "1", "2", "3", "4", "5", "6", "7", "8", "9", "10" ],
                "annotations" : {
                    "minimum" : "not at all confident",
                    "maximum" : "totally confident",
                }
            },
            "text" : null,
            "number_of_questions" : 1,
            "questions" : [
                {
                    "number" : "4",
                    "text" : "How confident do you feel that you can keep any
other symptoms or health problems you have from interfering with the things you
want to do?",
                    "help_text" : null
                }
            ]
        },
        {
            "element_type" : "question_group",
            "number" : null,
            "question_group_type" : "integer_range",
            "question_group_type_data" : {
```

```
                "minimum" : 1,
                "maximum" : 10,
                "step" : 1,
                "initial" : 6,
                "labels" : [ "1", "2", "3", "4", "5", "6", "7", "8", "9", "10" ],
                "annotations" : {
                    "minimum" : "not at all confident",
                    "maximum" : "totally confident",
                }
            },
            "text" : null,
            "number_of_questions" : 1,
                {
                    "number" : "5",
                    "text" : "How confident do you feel that you can the different
tasks and activities needed to manage your health condition so as to reduce your
need to see a doctor?",
                    "help_text" : null
                }
            ]
        },
        {
            "element_type" : "question_group",
            "number" : null,
            "question_group_type" : "integer_range",
            "question_group_type_data" : {
                "minimum" : 1,
                "maximum" : 10,
                "step" : 1,
                "initial" : 6,
                "labels" : [ "1", "2", "3", "4", "5", "6", "7", "8", "9", "10" ],
                "annotations" : {
                    "minimum" : "not at all confident",
                    "maximum" : "totally confident",
            },
            "text" : null,
            "number_of_questions" : 1,
            "questions" : [
                {
                    "number" : "6",
                    "text" : "How confident do you feel that you can do things
other than just taking medication to reduce how much your illness affects your
everyday life?",
                    "help_text" : null
                }
            ]
        }
    ]
}
```

A survey is described in this JSON as a list of elements. Elements represent sections of a survey (in the HTML, these sections are separated by horizontal dividers). An element can be a piece of text or a group of 1 or more

questions (as indicated by `element_type`). Text elements are separate from the name and description of a survey. They may, for example, represent a set of instructions between questions in the survey. Question elements can have type `boolean`, `exclusive_choice`, `text_area`, `integer_range`, etc.

Depending on the question type (as defined by `question_group_type`), question elements will include extra data (as defined by `question_group_type_data`). For example, a likert scale (`integer_range`) question will usually include a minimum or maximum annotation indicating the range of the likert scale. All question types that involve multiple choice will have a list of labels, and some can have an initial selection (as defined by `initial`). Elements and individual questions can have a number and text associated with them.

Below is the render function from `GenerateSurvey.js`.

```
// Survey component map.
const surveyMap = {
    'integer_range': ExclusiveChoices,
    'exclusive_choices': ExclusiveChoices,
    'boolean': ExclusiveChoices,
    'text_area': TextArea,
};

/**
 * Dynamically generate components based on type of elements in model.
 * @param {*} element - Element from list of elements in model.
 */
function renderElement(element, classes, isMobile) {
    const elementType = element.element_type;
    if (elementType === 'text') {
        return <p className={classes.questionPadding} dangerouslySetInnerHTML=
{{__html: element.text}}></p>;
    } else if (elementType === 'question_group') {
        const SurveyComponent = surveyMap[element.question_group_type];
        if (!element.number && !element.text){
            return <SurveyComponent
                        model={element}
                        isMobile={isMobile} />
        }
        return (
            <React.Fragment>
                <Typography
                    className={classes.questionPadding}
                    component="p"
                    variant="subtitle1"
                    dangerouslySetInnerHTML={{__html:
                        element.number ? `<b>${element.number}</b>:
${element.text}` : element.text
                    }}>
                </Typography>
                <SurveyComponent
                    model={element}
                    isMobile={isMobile} />
            </React.Fragment>
        );
```

```
    } else {
        return null;
    }
}
```

Given an element from a list of elements, this function will generate JSX depending on the element type. The key idea is an object is being used to map an element type to a corresponding component type. Using the `surveyMap`, the correct component is retrieved and passed its props.

Below is the component that handles booleans, exclusive choices, and integer ranges.

```
// Sets state and updates model with new selection.
handleChange = e => {
    this.setState({[e.target.name]: e.target.value});
    this.props.model.questions[e.target.name].response = e.target.value;
};

render() {
    const { classes, model, isMobile } = this.props;
    const questionData = model.question_group_type_data;
    return (
        <React.Fragment>
            {model.questions?.map((question, i) =>
                <div
                    className={`${classes.questionPadding}
${classes.colFlexContainer} ${classes.questionJustify}`}
                    key={i}>
                    <Typography
                        component="p"
                        variant="subtitle1"
                        dangerouslySetInnerHTML={{
                            __html: question.number ?
                                `<b>${question.number}</b>: ${question.text}` :
`${question.text}`
                        }}>
                    </Typography>
                    <FormControl
                        component="fieldset">
                        {isMobile ?
                        <Select
                            variant="outlined"
                            name={String(i)}
                            value={this.state[i]}
                            onChange={this.handleChange}>
                            {questionData.labels?.map((label, j) =>
                            <MenuItem
                                key={j}
                                value={String(j + 1)}>
                                {this.determineAnnotation(j, questionData)}
                            </MenuItem>
                            )}
```

```
                        </Select>
                        :
                        <RadioGroup
                            className={`${classes.rowFlexContainer}
${classes.questionJustify}`}
                            name={String(i)}
                            value={this.state[i]}
                            onChange={this.handleChange}>
                            <Typography
                                component="h6"
                                variant="subtitle1">
                                {questionData.annotations?.minimum}
                            </Typography>
                            {questionData.labels?.map((label, j) =>
                                <FormControlLabel
                                    className={classes.integerScaleMargin}
                                    key={j}
                                    label={label}
                                    value={String(j + 1)}
                                    control={<Radio color="primary" />}
                                    labelPlacement="bottom" />
                            )}
                            <Typography
                                component="h6"
                                variant="subtitle1">
                                {questionData.annotations?.maximum}
                            </Typography>
                        </RadioGroup>
                        }
                    </FormControl>
                </div>
            )}
        </React.Fragment>
    );
}
```

Here we see labels and radio buttons are generated for each question in the element. Minimum and maximum annotations are added to the left and right respectively to create a likert scale. When a radio button is selected, the `handleChange` function is called. The component updates its inner state as well as the `model` that was passed in as a prop.

It is important to keep in mind that the survey JSON retrieved from the API is parsed into a config object and passed directly to `GenerateSurvey` as a prop called `model`. Each component that represents a question (in this case, `ExclusiveChoices`) is then passed its corresponding element from the config object as a prop called `model`. Thus, all generated question components are manipulating the same config object (that is, adding a `response` property containing the user's selected value, as seen in `handleChange`.)

Below is from `DoSurvey.js`, where the `model` is retrieved from the API and passed to `GenerateSurvey`.

```
// When initially mounted, get the survey ID from the path and use it to get the
survey model (needed to generate survey).
```

```
componentDidMount() {
    const { surveyId  } = this.props.match.params;
    apiCall(`/surveys/${surveyId}/`, { method: 'GET'})
        .then(response => this.setState({model: response, spinner: false}));
}


render() {
    // Show spinner.
    if(this.state.spinner) {
        return <div className="progress-spinner-container"><CircularProgress size=
{100} /></div>
    }
    return (
        <React.Fragment>
            <NotifyDisplay
                header="Submission Error"
                items={this.generateErrorMessages(this.state.errors)}
                itemsRef={this.errorsRef} />
            <GenerateSurvey
                model={this.state.model}
                submit={this.handleSubmitSurvey} />
        </React.Fragment>
    );
}
```

When the user attempts to submit the survey, this manipulated model object, now with attached `response` properties to each question for each question element, is submitted to the API for validation.

```
/**
 * When user attempts to submit the survey, get the survey model, submit it to the
API.
 * If success, show snackBar and redirect back to surveys page. Otherwise, show
validation
 * messages generated by error from API.
 */
handleSubmitSurvey = () => {
    // Show spinner
    this.setState({spinner: true});
    const model = JSON.stringify(this.state.model);
    const { surveyId  } = this.props.match.params;
    apiCall(`/surveys/${surveyId}/submit/`, { method: 'POST', body: model },
false)
        .then(_ => {
            this.props.enqueueSnackbar('Survey submitted successfully!', {
                action: key => <Button style={{color: 'white'}} onClick={() =>
this.props.closeSnackbar(key)}>Dismiss</Button>,
            });
            this.props.history.push('/program/surveys')
        })
        // Scroll the NotifyDisplay into view.
```

```
          .catch(error => this.setState({errors: error.data.errors, spinner: false},
() => this.errorsRef.current.scrollIntoView()));
};
```

If there are errors, the API will return a list of error messages to display. Otherwise, the survey is submitted successfully.