SW Engineering CSC 648-848

OnTask

Section 4 Team 5

Rhoanna Perez (Team Lead), Christopher Su (Backend Lead), Jia Li (Frontend Lead)
Nelson Pang (Scrum Master), Seongjung Kim (Github Master)

Milestone 4

May 2, 2022

I. Product Summary

Our product is OnTask, a task manager application that is slightly different from the typical task manager applications like Jira, Notion, or Trello. Every task manager application has the goal of organizing tasks to ensure completion, but OnTask is unique in that our primary goal is to make sure you continue to stay productive and work towards completing tasks in a timely manner. OnTask aims to measure your productivity through the Pomodoro Technique to prevent hard workers from burnout, a form of exhaustion caused by constantly feeling drowned in tasks. Burnout is a result of excessive and prolonged stress, and one way to reduce burnout is to promote taking healthy breaks, which is one function that OnTask provides for users to ensure they take breaks after working.

One of the P1 functions that we feel is important is our **sign-up and log-in function**, which allows users to create an account and start creating their own to-do lists and tasks to complete. The second P1 function is **adding tasks or to-do lists**, which allows users to add tasks with a due date and a priority. Once the user completes a task, they can mark the task as complete and view it in a separate tab of tasks they've completed. The third P1 function is **adding users to a to-do list** which allows users to share an entire to-do list with other registered users. This is a vital function for those who would like to share to-do lists with their team, students, co-workers, or anyone who might need reminders about grocery lists or daily tasks with a set schedule. The fourth P1 function is our **work-study timer**, where users can become more productive and set intervals to work and remind themselves to take breaks. This allows users to stay on task and prevent distractions, while a timer will go off to remind them to take a break or to start working. Our last P1 function is our **deadline and upcoming tasks tab,** where users will be able to see what tasks they have yet to complete with an upcoming deadline. This allows users to see what they have left to do and can use our work-study timer to start working efficiently.

The important part of working is being able to manage our time wisely and being able to balance our life with work. Sites like Trello, Notion, and Jira focus on making sure we complete these tasks, but we want to be able to prioritize breaks, understanding that a huge factor of burnout is because of the lack of taking healthy breaks. According to Harvard Business Review, studies have shown that taking sporadic breaks replenish our energy, improve self-control and decision-making, and fuel productivity.

We are currently working on deploying our application on http://18.218.8.124:4000/

II. Unit Testing

I. Registration

There are two individual tests for the registration function to test the submit button with and without user inputted data. There are three other tests to check that the user-inputted data in the name, email, and password are valid.

```
// First test tests the submit button with no user data
test("Testing submit button with no user data", () => {

  const store = createStore(rootReducer, applyMiddleware(thunk));

  const onSubmit = jest.fn();


  render(<Provider store={store}> <Register onSubmit={onSubmit} /> </Provider>);
  const button = screen.getByRole('button');

  user.click(button);
  expect(onSubmit).toHaveBeenCalledTimes(0);
})
```

This test checks the submit button without any user data in which the registration would not go through.

```
test("Name should have correct value", () => {
  const store = createStore(rootReducer, applyMiddleware(thunk));
  const onSubmit = jest.fn();

  const { getByLabelText } = render(<Provider store={store}> <Register onSubmit={onSubmit} /> </Provider>);

  const nameField = getByLabelText(/name/i);

  expect(nameField.value).toBe("");
  fireEvent.change(nameField, { target: { value: "Nelson20" } });

  expect(nameField.value).toBe("Nelson20");


})
```

Testing the name field with “Nelson20” to check if the name field matches with what was entered.

```
test("Email should have correct value", () => {
  const store = createStore(rootReducer, applyMiddleware(thunk));
  const onSubmit = jest.fn();

  const { getByLabelText } = render(<Provider store={store}> <Register onSubmit={onSubmit} /> </Provider>);

  const emailField = getByLabelText(/email/i);

  expect(emailField.value).toBe("");
  fireEvent.change(emailField, { target: { value: "Nelson20@test.com" } });

  expect(emailField.value).toBe("Nelson20@test.com");


})
```

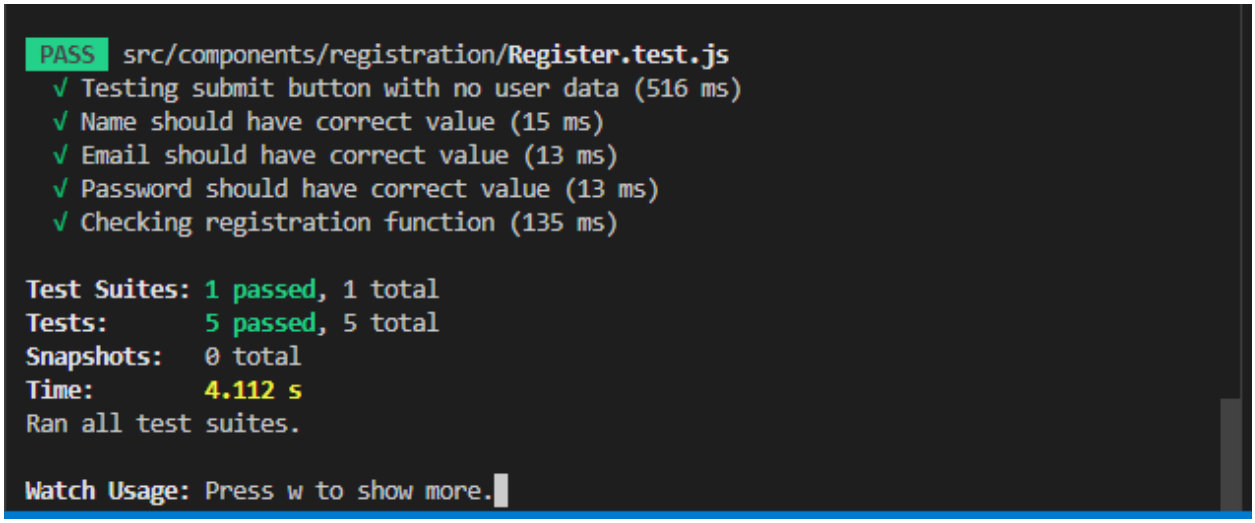
Testing email field with “Nelson20@test.com” to check if the email field matches with what was entered.

```
test("Password should have correct value", () => {
  const store = createStore(rootReducer, applyMiddleware(thunk));
  const onSubmit = jest.fn();

  const { getByLabelText } = render(<Provider store={store}> <Register onSubmit={onSubmit} /> </Provider>);

  const passwordField = getByLabelText(/password/i);

  expect(passwordField.value).toBe("");
  fireEvent.change(passwordField, { target: { value: "12345678" } });

  expect(passwordField.value).toBe("12345678");


})
```

Testing password with “12345678” to check if the password field matches with what was entered.

```
test("Checking registration function", async () => {
  const store = createStore(rootReducer, applyMiddleware(thunk));
  const onSubmit = jest.fn();
  const username = "Nelson20";
  const email = "Nelson20@test.com";
  const password = "12345678";

  render(<Provider store={store}> <Register onSubmit={onSubmit} /> </Provider>);

  const usernameInput = screen.getByLabelText(/name/i);
  const emailInput = screen.getByLabelText(/email/i);
  const passwordInput = screen.getByLabelText(/password/i);

  user.type(usernameInput, username);
  user.type(emailInput, email);
  user.type(passwordInput, password);
  const submitButton = screen.getByRole('button', { name: /^Submit$/i });
  user.click(submitButton);
  await expect(onSubmit).toHaveBeenCalledTimes(0);


})
```

Testing the registration with user-inputted data to see



All five tests passed without any errors.

II. Log-in

I made 4 individual tests for the log in function, which tests login buttons, username, password, and to see if each value gets rendered and changed. The beginning of the test sets up a store in order to implement redux during the test, and checks to see that each input variable is there.

```
describe('<Check Login form>', () => {
  it('renders Login', () => {
    const store = createStore(rootReducer, applyMiddleware(thunk));
    const login = {
      email: "",
      loggedIn: false,
      attempt: false
    };
    const { getByText } = render(
      <Provider store={store}>
        <Login login={login} />
      </Provider>
    );
    expect(getByText('Username')).toBeInTheDocument();
    expect(getByText('Password')).toBeInTheDocument();
  });
});
```

The second part of the test checks if the login button is properly working without any user data.

```
test('test login buttion does not work',()=>{
const onSubmit = jest.fn();
const store = createStore(rootReducer, applyMiddleware(thunk));
render(<Provider store={store}>
    <Login onSubmit={onSubmit} />
    </Provider>  );
const button = screen.getByRole('button');
user.click(button);
expect(onSubmit).toHaveBeenCalledTimes(0);

});
```

The next three and four tests check each input value and on every onChange event of the value.

```
test('set user name', ()=>{
  const store = createStore(rootReducer, applyMiddleware(thunk));
const username = "test6@mail.com";
render(<Provider store={store}>
  <Login />
    </Provider>  );
  const usernameInput = document.getElementById('user-email');
  fireEvent.change(usernameInput, {target: {value:username}});
  expect(document.getElementById("user-email").value).toEqual("test6@mail.com");

});
```

```
PASS src/components/login/login.test.js
  √ test login buttion does not work (269 ms)
  √ set user name (24 ms)
  √ set user password (15 ms)
  <Check Login form>
    √ renders Login (50 ms)

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        7.867 s
Ran all test suites matching /Login/i.
```

All four tests passed without any errors.

III. Adding tasks

```
describe('<List>', () => {
  it('renders without crashing', () => {
    const randomTask = "TEST: Add a Cat"
    const store = createStore(rootReducer, applyMiddleware(thunk));

    const { container, queryByText, getByDisplayValue } = render(
      <Provider store={store}>
        <List id={0} />
      </Provider>
    );
```

The beginning of the test sets up a store in order to implement redux during the test

```
22
23      // adding task to the input field
24      expect(queryByText(randomTask)).not.toBeInTheDocument();
25      fireEvent.change(container.querySelector('#input'), { target: { value: randomTask } });
26      expect(getByDisplayValue(randomTask)).toBeInTheDocument();
27
28      // clicking submit
29      axios.post.mockImplementation(() => Promise.resolve({data:"task completed"}));
30      fireEvent.click(container.querySelector('#submit'));
31
```

The second part of the test checks the usage of the input field by first checking if the desired input is present, then adding it to the input field, then checking if it is present again.

The next part tests the axios call of the submit button

```
    // reseting the page
    axios.get.mockImplementation(() => {
      return Promise.resolve({
        data: [
          {
            title: 'Walk The Dog.'
          },
          {
            title: 'Meet with group.'
          }
        ]
      });
    }
    );

    // check if the todolist was reset
    setTimeout(() => {
      expect(store.getState().todo.items.length).toBe(2);
    },2)
  });
```

The last part checks the reset function that occurs after the user submits a new task in order to reset the task list to include the latest one.

```
 PASS  src/components/test/addTodo.test.js
  <List>
    √ renders without crashing (69 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        1.343 s
Ran all test suites matching /addTodo.test.js/i.
```

As you can see, the test ran w/o crashing or errors.

IV. Adding a user to the to-do list

I performed four individual tests for adding a user to the to-do list.

```
//test if collab component renders
it("renders without crashing", () => {
  const div = document.createElement("div");
  ReactDOM.render(<Collab></Collab>, div);
});
```

This first test simply tests if the component renders correctly.

```
//test if button renders
it("renders the modal", async () => {
  const { getByTestId, queryByText } = render(<Collab />);
  const button = getByTestId("mainbuttonT");
  expect(button).toBeTruthy();
  fireEvent.click(button);
  await waitFor(() =>
    expect(queryByText("Share with people and groups")).toBeInTheDocument()
  );
});
```

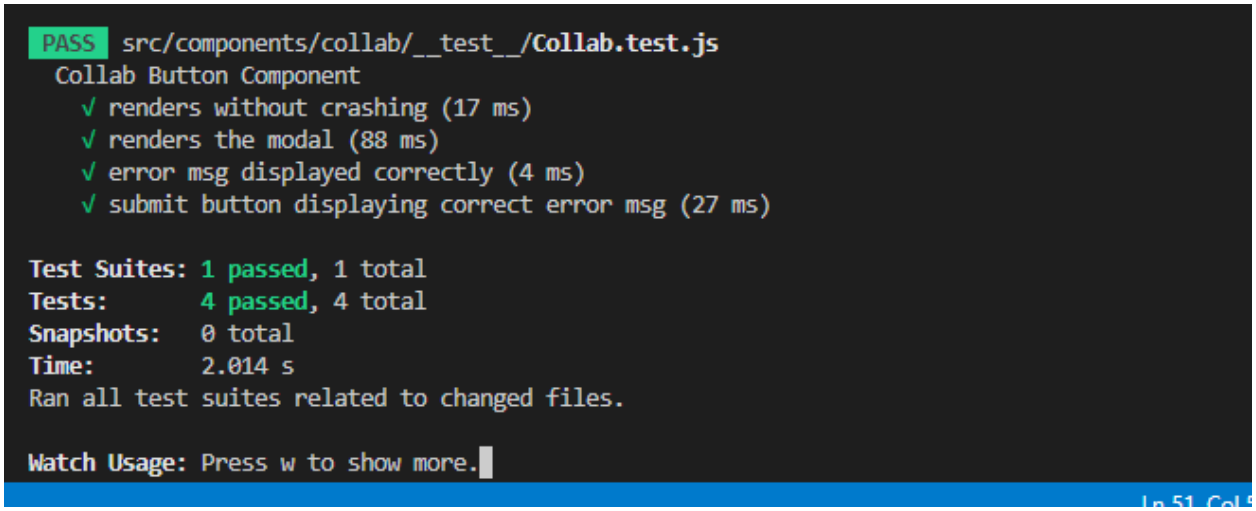
This second test tests if upon clicking the collaborative button, the Modal component which was imported from the react library renders correctly.

```
//test if error msg is hidden
it("error msg displayed correctly", () => {
  const { queryByTestId } = render(<Collab />);
  const errorMsg = queryByTestId("validEmailT");
  expect(errorMsg).toBeNull();
});
```

This third test tests if by default, the error message initially set as null.

```
//test if onsubmit displays correct error msg
it("submit button displaying correct error msg", async () => {
  const { getByTestId, queryByText } = render(<Collab />);
  const button = getByTestId("mainbuttonT");
  const submit = queryByText("Search", { selector: "button" });
  fireEvent.click(button);
  await waitFor(() => {
    const button2 = screen.queryByTestId("buttonT2");
    const errorMsg = screen.queryByTestId("validEmailT");
    fireEvent.click(button2);
    expect(errorMsg.textContent).toBe("User not found");
  });
});
```

This final test is to test whether or not, upon entering an email and clicking the search button, the correct error message will be displayed. Since we only have one log in user in the database, any input in the search function would return the same error message.



As you can see, all four tests pass without failing.

V. Work/Study Timer

I performed 5 individual tests for the work/study timer, which tests each individual input and checks to see if the value of each time gets rendered and changed. This tests each case, which allows both the work and break timer to start.

```
test('Render timer', () => {
    render(<SetTimers/>);
    const workMin = document.getElementById("work-min");
    const workSec = document.getElementById("work-sec");
    const breakMin = document.getElementById("break-min");
    const breakSec = document.getElementById("break-sec");
    expect(workMin).toBeInTheDocument();
    expect(workSec).toBeInTheDocument();
    expect(breakMin).toBeInTheDocument();
    expect(breakSec).toBeInTheDocument();
})
```

This first test checks to see if it renders the component, and checks to see that each input variable is there.

```
test('Set work minute to 60', () => {
    render(<SetTimers/>);
    const workMin = document.getElementById("work-min");
    fireEvent.change(workMin, {target: {value: '60'}})
    expect(document.getElementById("work-min").value).toEqual("60");
})
```

The next four tests checks each input value and on every onChange event of the value, we should expect it to equal what value we previously set it to equal.

```
 PASS  src/components/work-intervals/WorkOrBreakTime.test.js
  Set timers
    ✓ Render timer (123 ms)
    ✓ Set work minute to 60 (67 ms)
    ✓ Set work second to 30 (84 ms)
    ✓ Set break minute to 15 (149 ms)
    ✓ Set break second to 30 (166 ms)

Test Suites: 1 passed, 1 total
Tests:       5 passed, 5 total
Snapshots:   0 total
Time:        6.887 s, estimated 12 s
Ran all test suites matching /WorkOrBreakTime/i.

Watch Usage: Press w to show more.
```

III. Integration Testing

I. Registration

Test Case ID	1	Test Case Description	Test the Work-Study Timer Function		
Created By	Rhoanna	Reviewed By	Nelson	Version	1

QA Tester's Log

Tester's Name	Rhoanna	Date Tested	April 23, 2022	Test Case (Pass/Fail/Not Executed)	Pass

S #	Prerequisites:
1	Access to Chrome Browser
2	
3	
4	

S #	Test Data
1	Work Time = 30 minutes
2	Break Time = 15 minutes
3	
4	

Test Scenario Entering valid work time and break time to see if the timer will work

Step #	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Navigate to http://18.218.8.124:4000	Site should open	As Expected	Pass
2	Use navbar to open timer	Redirect to timer	As Expected	Pass
3	Add work and break time	Start work timer	As Expected	Pass
4	Press start break button	Start break timer	As Expected	Pass

The primary function of registration is to allow a user to create an account in order to access or web application. For this test, I entered user data to create a brand new account. The expected result was the user was created and redirected to the login page. I also tested the registration by clicking the submit button with no user data. The page was not able to submit because there is no user data entered.

II. Log-in

Test Case ID	3	Test Case Description	Test the login function		
Created By	Nelson	Reviewed By	Chris	Version	1.0

QA Tester's Log					
Tester's Name	Nelson	Date Tested	April 27, 2022	Test Case (Pass/Fail/Not Executed)	PASS

S #	Prerequisites:
1	Access to browsers
2	
3	
4	

S #	Test Data
1	Email: test20@test.com
2	Password: 12345678
3	
4	

Test Scenario	Check to see if login can redirect them to their account and see to do lists

Step #	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Navigate to http://18.218.	Site should load normally to homepage	As Expected	PASS
2	Click Log-in in top right of navigation bar	Login page should load	As Expected	PASS
3	Input non-existing email and password	User is not able to login	As Expected	PASS
4	Input existing email and password	User will login and see todo lists	As Expected	PASS

The primary function of this test is to ensure that the login function is working with existing user data. First, I entered a username and password that does not exist in the database. The login failed so the page refreshes. Next, I tested the login function with existing user data. The login was successful and redirects to the to-do list page.

III. Adding tasks

Test Case ID	4	Test Case Description	Test Adding Task		
Created By	Chris	Reviewed By	Jia	Version	1.0

QA Tester's Log

Tester's Name	Christopher Su	Date Tested	April 29, 2022	Test Case (Pass/Fail/Not Executed)	Pass

S #	Prerequisites:
1	
2	
3	
4	

S #	Test Data Requirement
1	Existing email and password
2	Input Any Task
3	
4	

Test Conditions

Step #	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Navigate to http://18.218.8.124:4000	Site should open	As Expected	Pass
2	Log into the site	Redirect to Todo Page	As Expected	Pass
3	Click add task button	Shows a input tab	As Expected	Pass
4	Input a task and click submit	Task is added to the todolist	As Expected	Pass

The primary function of adding a task is to allow a user to create an new task in their todo lists. For this test, I had the user first log in to the site, then add a task to a previously created todolist. The expected result was that a new task would be created and the todolist would be updated to include that task.

IV. Adding a user to the to-do list

Test Case ID	5	Test Case Description	Testing to see if users can be added to the to do list		
Created By	Jia	Reviewed By	Rhoanna	Version	1.0

QA Tester's Log

Tester's Name	Jia	Date Tested	April 29, 2022	Test Case (Pass/Fail/Not Executed)	PASS

S #	Prerequisites:
1	Access to Chrome Browser
2	
3	
4	

S #	Test Data Requirement
1	Login email = test6@mail.com
2	Login password = test6@mail.com
3	Email to add = rj@email.com
4	

Test Scenario Share a to do list with a valid email

Step #	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Go to http://18.218.8.124:4000	Site should open	As expected	PASS
2	Login	Should log in and lead to to do list page	As expected	PASS
3	Press "share" button	Open pop up to add email	As expected	PASS
4	Add email	Should add email to list	As expected	PASS
5	Add unregistered email	Should show error	As expected	PASS

The primary function for adding a user to the to-do list is adding a registered email to allow other users to view and edit a to-do list. For this test, I used an email that was already registered and located in the database and used the share button to add an email that was registered. I also tested to see what would happen if I attempted to add an email that wasn't created with OnTask, and it returned an error that the user is not registered.

V. Work/Study Timer

Test Case ID	1	Test Case Description	Test the Work-Study Timer Function		
Created By	Rhoanna	Reviewed By		Version	1
QA Tester's Log					
Tester's Name	Rhoanna	Date Tested	April 23, 2022	Test Case (Pass/Fail/Not Executed)	Pass

S #	Prerequisites:
1	Access to Chrome Browser
2	
3	
4	

S #	Test Data
1	Work Time = 30 minutes
2	Break Time = 15 minutes
3	
4	

Test Scenario: Entering valid work time and break time to see if the timer will work

Step #	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Navigate to http://18.218.8.124:4000	Site should open	As Expected	Pass
2	Use navbar to open timer	Redirect to timer	As Expected	Pass
3	Add work and break time	Start work timer	As Expected	Pass
4	Press start break button	Start break timer	As Expected	Pass

The primary function of the Work/Study Timer is setting the work and break time, starting the timer, and pausing the timer. For this test, I used the test data, 30 minutes for work time and 15 minutes for break time, as an input for the first page of the timer. After I start the timer, it should direct the user to use the work timer and after the work timer is over, it should redirect the user to start their break. After going through the integration test, it passes each step from navigating from the home page to finishing the break timer.

VI. Deadlines and upcoming due dates

Test Case ID	6	Test Case Description	View upcoming deadlines and due dates		
Created By	Jia	Reviewed By	Kim	Version	1.0

QA Tester's Log

Tester's Name	Jia Li	Date Tested	April 29, 2022	Test Case (Pass/Fail/Not Executed)	PASS

S #	Prerequisites:
1	Access to web browser
2	
3	
4	

S #	Test Data Requirement
1	N/A
2	
3	
4	

Test Conditions View upcoming deadlines and due dates given tasks with set deadline

Step #	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Go to http://18.218.8.124:4000	Site should open	As Expected	PASS
2	Log in to the site	Should redirect to user's home page	As Expected	PASS
3	View upcoming tasks and deadlines	Should show upcoming tasks at the top	As Expected	PASS

The primary function of the deadlines and upcoming due dates is to remind users and show users what tasks have deadlines coming up, and what they have left to do. For this test, as long as users have already added tasks using the add task function, it will show users at the top of the page once they log in to see and view the tasks with upcoming deadlines in chronological order. Our only test was to make sure that once they log in, the users can see the upcoming deadlines. If they do not have any, the section will be empty and navigate users to their current to-do lists. After going through each step, the results were as expected and passed each step.

IV. Code Reviews

For our code reviews, we used Github and made pull requests and commented on each of our code before we were able to merge pull requests.

https://github.com/CSC-648-SFSU/csc648-spring22-04-team05-new/pulls

V. Adherence to original non-functional specs

Compatibility: ON TRACK

- The application should be compatible with Chrome, Safari, Mozilla Firefox, Microsoft Edge (**DONE)**
- The application should be built with full mobile responsiveness in mind, employing a mobile-first mindset since most users are expected to be using the todo-lists "on the go"

Storage Space: DONE

- The application will be hosted on Amazon AWS
- The database will be hosted on Atlas using a cloud MongoDB database

Usability/Product Requirements: DONE

- Application should be straightforward and intuitive
- Usability studies should be used to make sure the application is easy to use to the general public

External Requirements: ON TRACK

- Security: The application will run under https at all times in order to keep connections secure **ISSUE (currently working to get https to work)**
    - Login information will be hashed and taken care of to maintain user confidentiality.
- Ethicality: User information and confidentiality will be protected and users will be notified on how their data is being used
    - The application will follow accessibility guidelines and be developed with inclusivity in mind
    - Application will leverage off-line first capabilities to help those without access to stable connection

Development Requirements: ON TRACK

- Github: Code will be well maintained and documented
    - Code should be developed in such a way that allows for future expansion

- Figma: Designs should be well documented and easy for other developers to understand designs **DONE**
    - A design system should be implemented to keep both developers and designers on the same page