



VAASAN AMMATTIKORKEAKOULU  
UNIVERSITY OF APPLIED SCIENCES

Nguyen Bui

INTEGRIFY'S  
ESR-APPLICANT

School of Technology  
2020

## ABSTRACT

Author	Nguyen Minh Bui
Title	Integrify's ESR-APPLICANT
Year	2020
Language	English
Pages	30 + 3 Appendices
Name of Supervisor	Timo Kankaanpää

---

This thesis was made to demonstrate the creation of a web application with ReactJS, based on creating a new web application for Integrify Oy. The idea of this thesis is provided by Integrify Oy.

As a powerful and rising programming language for building web application, React with TypeScript was chosen for building the frontend. For building the backend of the system, NodeJS was the chosen programming language with the help of NestJS framework. For deploying and CI/CD, Github Actions and AWS was chosen to be the tools and services used in this project.

The result of this thesis is a web application to allow applicants and students of Integrify to take test and submit answers to Integrify's system. The final software product will be a subset of Integrify's Student Management System (SMS). The project demonstrated in this thesis does not reflect the final product implementation.

All information in the thesis is not personal research but created by summarizing knowledge from official documentation of React, NestJS and its experts. The knowledge can be changed and updated in new versions in the future.

The project has fulfilled basic requirements in its first phase of development.

---

Keywords                      React, NestJS, SMS, Github Actions

# CONTENTS

## ABSTRACT

1. INTRODUCTION .....	8
1.1 About Integrify Oy .....	8
1.2 Integrify's Student Management System .....	9
1.3 Objectives .....	9
2 THEORETICAL BACKGROUND .....	11
2.1 React .....	11
2.2 Redux .....	11
2.3 Redux Saga .....	12
2.4 Node.js .....	13
2.5 NestJS .....	13
2.6 PostgreSQL Database .....	15
2.7 TypeScript .....	15
2.8 Authentication with Google Login .....	16
2.9 CI/CD .....	16
3 SOFTWARE SPECIFICATION AND ARCHITECTURE .....	18
3.1 Software Requirement Specification .....	18
3.2 Application Architecture .....	19
3.2.1 Database Architecture .....	19
3.2.2 ESR-Applicant Workflow .....	20
3.2.3 Styling, Color and Typography .....	21
4 IMPLEMENTATION .....	23
4.1 Google Login .....	23
4.2 Countdown Timer .....	26
4.3 Displaying Test .....	28
4.3.1 Test View .....	29
4.3.2 Question View Component .....	33
4.3.3 Custom hooks useAnswer .....	37
4.4 Feedback Page .....	40
4.5 Storybook .....	43
4.6 CI/CD with Github Actions .....	45

4.7 Version Controlling .....	47
4.8 Testing.....	49
5 CONCLUSIONS .....	53
REFERENCES.....	54

## APPENDICES

## LIST OF FIGURES AND TABLES

<b>Figure 1.</b> User stories of ESR-Applicant.....	10
<b>Figure 2.</b> Redux cycle /5/ .....	12
<b>Figure 3.</b> Node.js core .....	13
<b>Figure 4.</b> NestJS application’s architecture – folder structure. ....	14
<b>Figure 5.</b> NestJS application’s architecture – diagram.....	15
<b>Figure 6.</b> Authentication with Google login workflow /11/.....	16
<b>Figure 7.</b> CI/CD pipeline process .....	17
<b>Figure 8.</b> SMS-Backend database architecture. ....	20
<b>Figure 9.</b> ESR-Applicant workflow. ....	21
<b>Figure 10.</b> ESR-Applicant typography and color palette. ....	22
<b>Figure 11.</b> Google console .....	23
<b>Figure 12.</b> Login with Google page .....	26
<b>Figure 13.</b> Countdown Timer .....	26
<b>Figure 14.</b> Home page of ESR-Applicant .....	29
<b>Figure 15.</b> Test page .....	29
<b>Figure 16.</b> Multiple-choice question type .....	36
<b>Figure 17.</b> Checkbox question type.....	36
<b>Figure 18.</b> Feedback page of ESR-Applicant.....	40
<b>Figure 19.</b> Storybook console.....	45
<b>Figure 20.</b> Github Actions jobs finished .....	47
<b>Figure 21.</b> Pull Request created by the author of the thesis .....	48
<b>Table 1.</b> Functional requirements .....	18
<b>Table 2.</b> Smoke tests.....	49

## LIST OF CODE SNIPPETS

**Code Snippet 1.** React-Google-Login component for authentication with Google.24

**Code Snippet 2.** Handle log in and log out actions25

**Code Snippet 3.** Countdown Timer component28

**Code Snippet 4.** Test view component32

**Code Snippet 5.** Question view component34

**Code Snippet 6.** Question option component36

**Code Snippet 7.** Question script component37

**Code Snippet 8.** Custom Hooks useAnswer40

**Code Snippet 9.** Feedback page implementation43

**Code Snippet 10.** Storybook scripts44

**Code Snippet 11.** Custom button's storybook44

**Code Snippet 12.** Github Actions configuration46

## LIST OF ABBREVIATIONS

<b>API</b>	Application Programming Interface
<b>CSS</b>	Cascading Style Sheets
<b>DOM</b>	Document Object Model
<b>HTML</b>	Hypertext Markup Language
<b>JSON</b>	JavaScript Object Notation
<b>UI</b>	User Interface
<b>UX</b>	User Experience
<b>XML</b>	Extensible Markup Language
<b>JSX</b>	JavaScript XML

**I/O**

Input/Output

**OS**

Operating System

**CI/CD**

Continuous Integration and Continuous Delivery

## 1. INTRODUCTION

Nowadays, a web application has become a standard for software application, many companies have started to switch their applications to web-based applications. Thanks to the development of web technologies, web applications now can handle complicated services, such as compiling documents and taking tests. Compared to just a decade ago, a web application was only used to display simple static content.

Taking advantage of the development of web technology, Integrify decided to create a web-based system called Student Management System (SMS) to manage Integrify student and applicant data, such as test results, tests, personal information. This thesis documents a new version of ESR-Applicant application, a part of SMS, which is an online tool that is used to create and send the test to the applicants of Integrify's courses.

This thesis contains five chapters. The first chapter gives a clear view about Integrify, the project documented in the thesis. All the theoretical information about technologies used in the application are stated in chapter two. The next chapter describes application's requirements and its architecture. Chapter four documented the implementation of the application of the author. Conclusion, drawbacks, and future improvements are written in chapter 5.

### 1.1 About Integrify Oy

Founded in 2016, Integrify was founded during the refugee crisis to untap the potential of international talent in Finland, by training talents in programming as a fast track to jobs. After the training period, Integrify expects international talents to land jobs as developers [1].

In 2019, Integrify launched two new programs – full-stack and machine learning, marked a new milestone for Integrify to extends further. In 2020, Integrify have enrolled for over 200 people to join programs



## 1.2 Integrify's Student Management System

Integrify's Student Management System (SMS) is an Integrify internal, online tool which is used by Integrify to manage students, courses, and the online testing of applicants. SMS is a combination of SMS-Admin – an online tool for teachers, supervisors, and ESR-Applicant – an online tool for applicants and students of Integrify.

SMS-Admin is part of SMS tools and allows teacher and supervisors to manage students and courses; it acts as a central database for all information about applicants, students, courses, and alumni. Moreover, SMS-Admin support course application, applicant interviewing process as well as sales businesses.

ESR-Applicant is the tool built for students and applicants to take the test and send it to the Integrify's admins to evaluate for Integrify's courses or application. Admins creates the test with different questions with SMS-Admin tool and the test is sent to applicants to do it online. Applicants then can finish the test and submit results, feedbacks to SMS.

## 1.3 Objectives

The main objective of this thesis is to create a new version of ESR-Applicant to solve the existing problems of the current ESR application of Integrify's SMS. With the current version of ESR-applicant, the tool shows several drawbacks:

- It is not possible to create tests with different questions. All candidates receive the same set of questions in the same test.
- For a test, it is not possible to create different sections dynamically.
- The number of different question types is limited. On the current version, it only supports multi-choice question.
- There is no difficulty level of questions.
- Buggy UI, some questions may be invisible to the users.
- Users cannot navigate between questions or sections in a test.

The main objective of the application is to solve these drawbacks and extend its functionality:

- It should allow users to view questions with more details displayed on the screen.
- It should enable navigating between questions and save answered results.
- Save and retrieve the saved answers from the server in case of the internet crash.
- Prevent the user from cheating by disabling copy or open web inspector.

The user stories of ESR-Applicant are demonstrated in Figure 1.

User	Story
Applicant	As an applicant, I can log in with Google
	As an applicant, I can do the example test to get familiar with the test process
	As an applicant, I can go back & forth between questions when time is not up
	As an applicant, I can submit the test when time is up or when I've completed it
	As an applicant, I can submit the feedback (if any) after the test

**Figure 1.** User stories of ESR-Applicant

## 2 THEORETICAL BACKGROUND

### 2.1 React

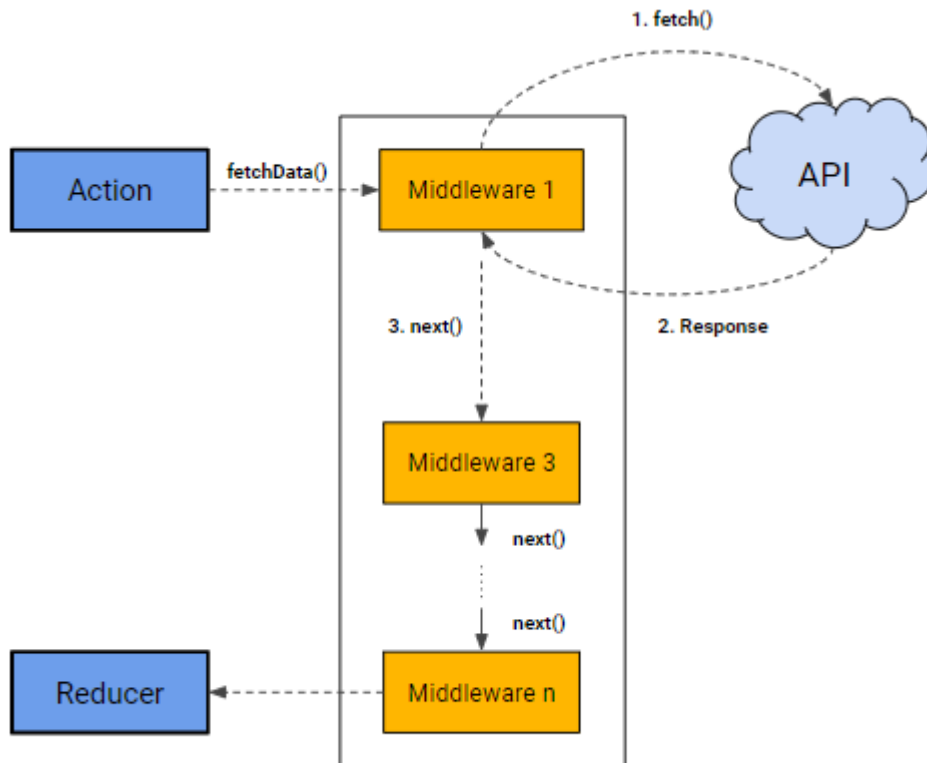
React is a JavaScript library, designed to simplify the progress of making interactive UIs, created by Facebook, and is maintained by individual developer community. The React application is considered a component-based application because it is made of entities called component, which can manage their own state to be composed to make a complex UI /2/.

React components can be divided into two types: functional component and class-based component. The functional components are declared as a function to return some JSX, while the class-based components are classes that extends React Component – a class built in React library. Before patch 16.8, most of the React components were class-based components since class-based components support life-cycle methods for the state management of the component. However, since patch 16.8, React introduced a new concept called Hooks, which offers a new way to use state and other React features in a functional component. With Hooks, a stateful logic of a component can be extracted to be tested independently and reused without changing component hierarchy. Furthermore, Hooks allows the developer to split a component into smaller functions based on relation, rather than splitting based on lifecycle methods /3/.

### 2.2 Redux

Redux is a JavaScript library for application state management, commonly used with JavaScript libraries, frameworks like Angular or React /4/.

With Redux implemented in the application, it is easier to manage the data flow in the application, because Redux provide a state store for every component in the application to access data without having to pass data as props or using Context, which might cause the component to re-render itself and affect badly the application performance. Moreover, Redux can work with multiple middleware to handle side effects or fetching data from an external server. A Redux workflow is illustrated in Figure 2.



**Figure 2.** Redux cycle /5/

In the Redux cycle shown above, once a component makes changes to the application state, it will dispatch an action to the Redux store. An action is a plain function that returns a JavaScript object. As a convention, that returned object should contain two properties: type and payload. Once an action has been dispatched, it goes through all the middleware in the system to execute side effects. The reducer is a function used for taking dispatched actions and based on the type and the payload of the action, it changes the application state.

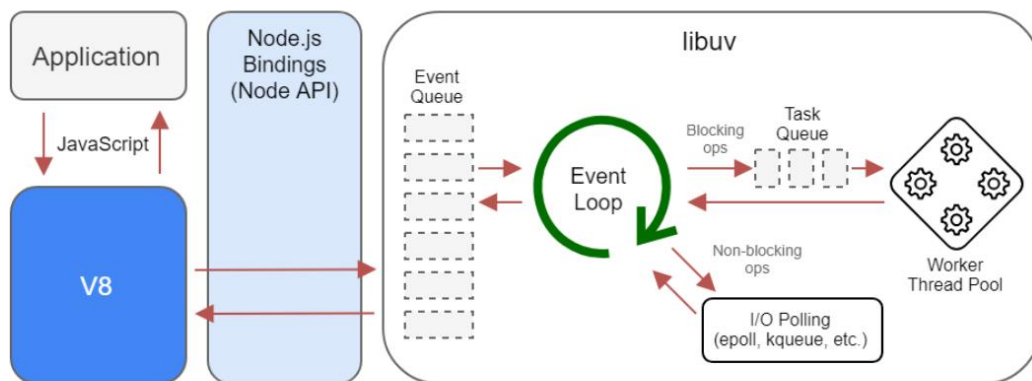
### 2.3 Redux Saga

Redux Saga is a library used to manage application side effects, execute more efficiently and handle failures better. Redux-Saga was made to be a Redux middleware, which make Saga a separate thread in the application which is responsible for side

effects. Redux-Saga can be started, paused, and cancelled by executing normal Redux actions while it also has access to the Redux state /6/.

## 2.4 Node.js

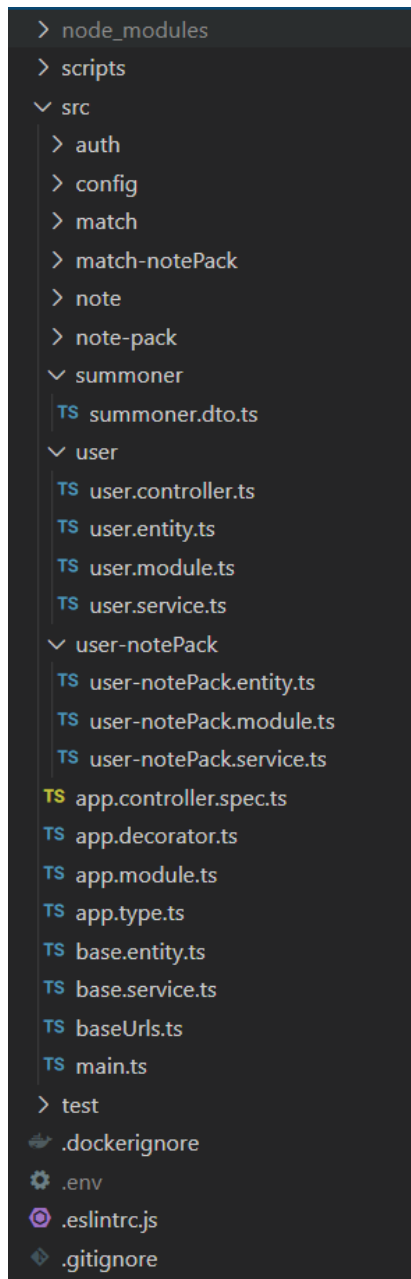
Node.js is a JavaScript runtime based on Chrome's V8 runtime engine, designed to build scalable applications. Node.js contains an event-driven architecture and operates on a single-thread event loop, which makes it be able for asynchronous I/O to optimize throughput and scalability in web applications. Node.js. Node.js core is shown in Figure 3 /7/.



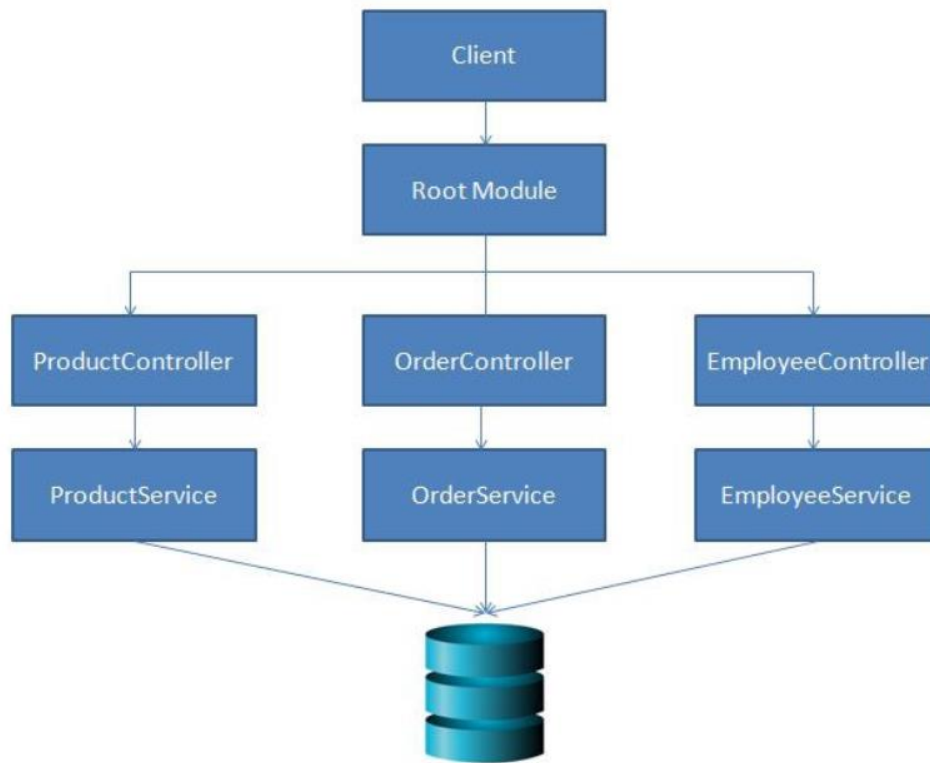
**Figure 3.** Node.js core

## 2.5 NestJS

Nest (NestJS) is a Node.js framework for building efficient, scalable server-side applications. NestJS makes use of HTTP Server frameworks, such as Express by providing a level of abstraction above these frameworks. Thanks to that, NestJS provides developers with flexibility with the application; developers are free to use other available third-party modules. NestJS provides an outstanding architecture, which is heavily inspired by Angular. A sample NestJS application architecture is illustrated in Figure 4 and in Figure 5 /8/.



**Figure 4.** NestJS application's architecture – folder structure.



**Figure 5.** NestJS application’s architecture – diagram.

## 2.6 PostgreSQL Database

PostgreSQL is an open source, powerful object relational database system, which runs on all major OS and has been ACID-compliant /9/.

PostgreSQL contains features, such as table inheritance, function overloading and protecting data integrity at transaction level. Owing to this, it makes PostgreSQL less vulnerable to data corruption.

## 2.7 TypeScript

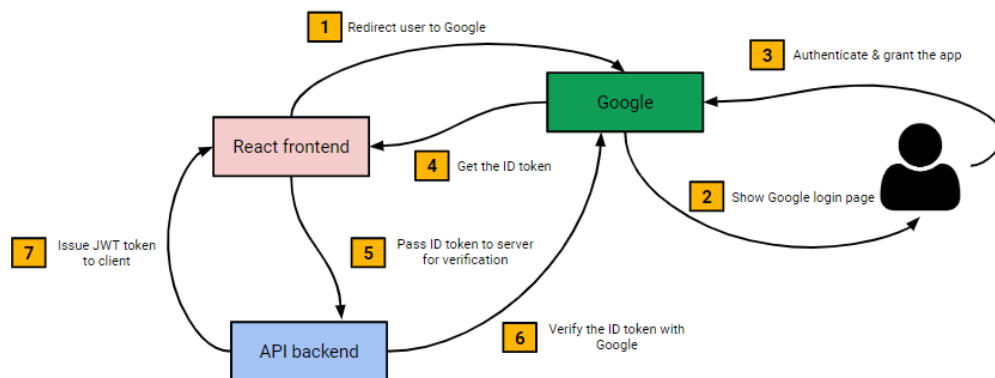
TypeScript is an open-source programming language developed and maintained by Microsoft, built on JavaScript by adding static type definitions. Thanks to this improvement, TypeScript makes it much easier to debug, develop and better security. However, there is a drawback with a compile time from TypeScript to JavaScript because the web browser can only understand JavaScript /10/.

TypeScript can be used to replace JavaScript in web applications for both client-side and server-side.

## 2.8 Authentication with Google Login

Nowadays, Google provides the application with an authentication solution without having to implement an own version while still ensuring the security of the application.

The workflow of authentication with Google Login is begun with client-side application redirecting users to the Google site and authenticating by logging in with their Google account, then Google sends back an ID token. If authentication is successful the client-side application will handle the ID token sent back from Google by sending it to the server-side application to authenticate and generate an JWT and send it back to the client for any further requests. This workflow is illustrated in Figure 6.



**Figure 6.** Authentication with Google login workflow /11/

## 2.9 CI/CD

CI/CD Continuous integration and continuous delivery(CI/CD) is a method to deliver the application to the customer by applying automation into the stages of application development. CI/CD also introduces continuous monitoring the lifecycle of the application, from integration and testing to delivery and deployment /12/. The CI/CD pipeline process is shown in Figure 7.





**Figure 7.** CI/CD pipeline process

### 3 SOFTWARE SPECIFICATION AND ARCHITECTURE

#### 3.1 Software Requirement Specification

In this chapter, the requirements of the application, which were requested by the customer (Integrify Oy), is documented. The functional requirements are listed in Table 1 and sorted by priority.

Priority levels:

1. Must have
2. Should have
3. Nice to have

**Table 1.** Functional requirements

Reference	Description	Priority
F1	Authentication with Google account	1
F2	Letting user do the test if user is assigned to do the test	1
F3	Displaying test by topics	1
F4	Displaying question detail	1
F5	Submitting answer	1
F6	Enabling navigating between questions	1
F7	User's process is saved in app state in case of bad internet	1
F8	Disabling toggle web inspector and copy text from tests	1

F9	Time countdown when user started / resumed the test	1
F7	Show process of test per topic	2
F11	Display test process of user	2
F12	User can submit feedback when the test is finish	2
F14	Bookmarking questions	2
F13	Display application tour to explain UI of the application	3

The nonfunctional requirements of the application are:

- Responsive with tablet and large screen monitor.
- Only authorized users can access a specific test case.
- The UI must be clean and user friendly.

## 3.2 Application Architecture

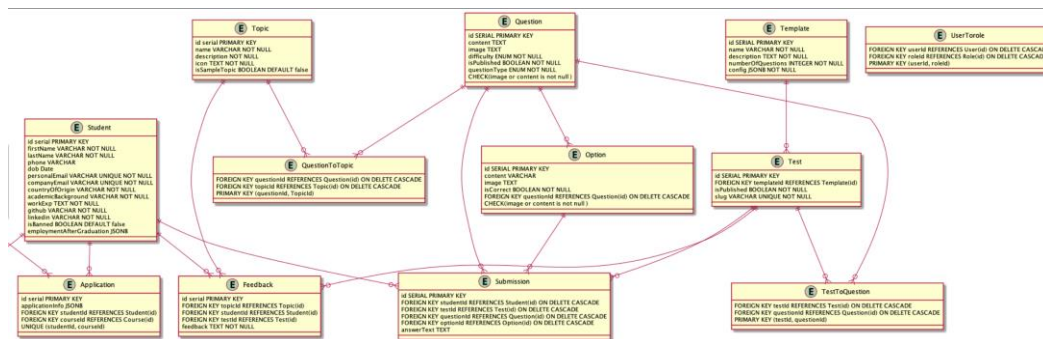
ESR-Applicant as a part of Integrify's SMS, so there is no need for a separated server implemented to handle requests and separated database storage for only the ESR-applicant application. All services and data of ESR-Applicant will be handled by SMS-Backend.

### 3.2.1 Database Architecture

In the SMS-Backend application, PostgreSQL is chosen to be the database storage. Since the thesis focuses on documenting the implementation of ESR-Applicant, this chapter focuses on the architecture of related tables used in ESR-applicant.

The main point in the database is the relation of Template, Test and Submission tables. Template is an entity for blueprint for a set of tests with properties, such as number of questions, difficulties of questions, so that different users can receive

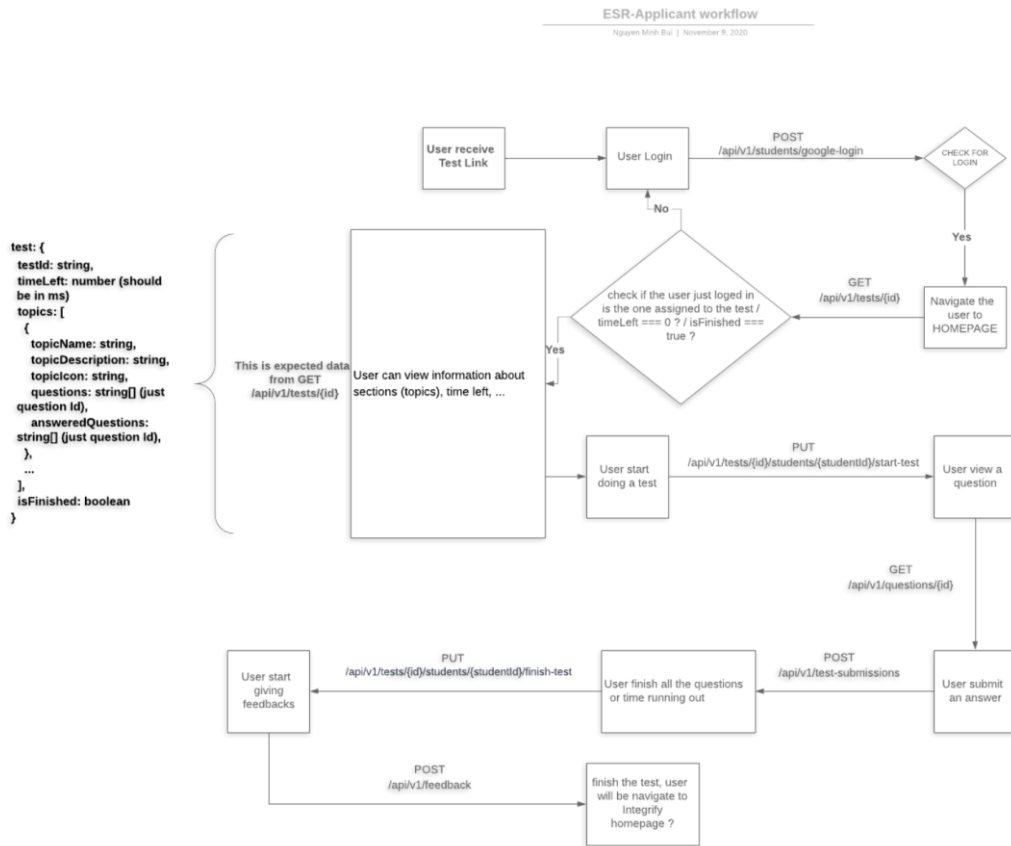
different questions with the difficulty of the test unchanged. The test entity is for a specific test set created for set of students, it can only be assigned for a specific set of students for a specific time, it contains properties, such as number of questions, answered questions, time left, topics of the test. Submission entity is for storing the student process of taking the test for individual question, such as the chosen answer or answer text. The architecture of the database is shown in Figure 8.



**Figure 8.** SMS-Backend database architecture.

### 3.2.2 ESR-Applicant Workflow

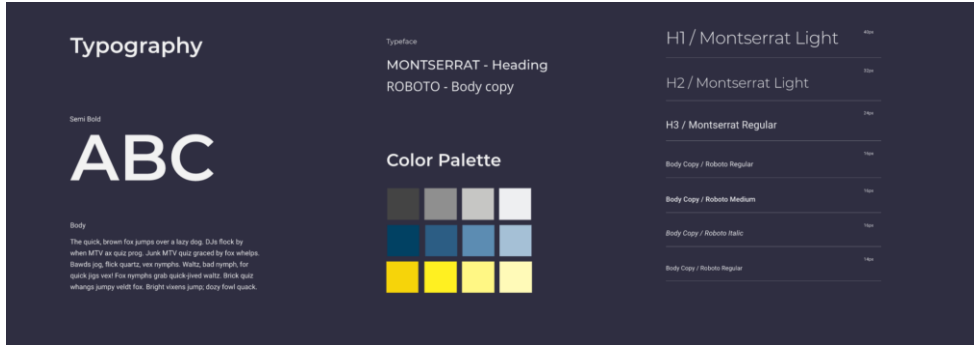
When visiting the ESR-applicant test link with test ID, if user is not authenticated, but the user must log in using Google accounts. If the user is unauthorized to do the test, the user will have to log in again with the right Google account. If the user is authorized, the user will be redirected to the main page with all information about the test displayed and the test will be divided further into topics. Once the user starts the test in a topic, the timer will trigger and start counting down, for every question in a test. If the user completes a question, the answer to that question will be saved immediately to the SMS-Backend. Once the user has finished all tests, a feedback page will be displayed for the user for experience feedback. If the user has finished the test or time has run out, the user will not be able to do the test anymore. The workflow of ESR-applicant is illustrated in Figure 9.



**Figure 9.** ESR-Applicant workflow.

### 3.2.3 Styling, Color and Typography

Since ESR-Applicant is a part of Integrify's SMS, its styling must follow the same color palette to match Integrify's branding color palette. The typography, color palette of ESR-Applicant application is shown in Figure 10.



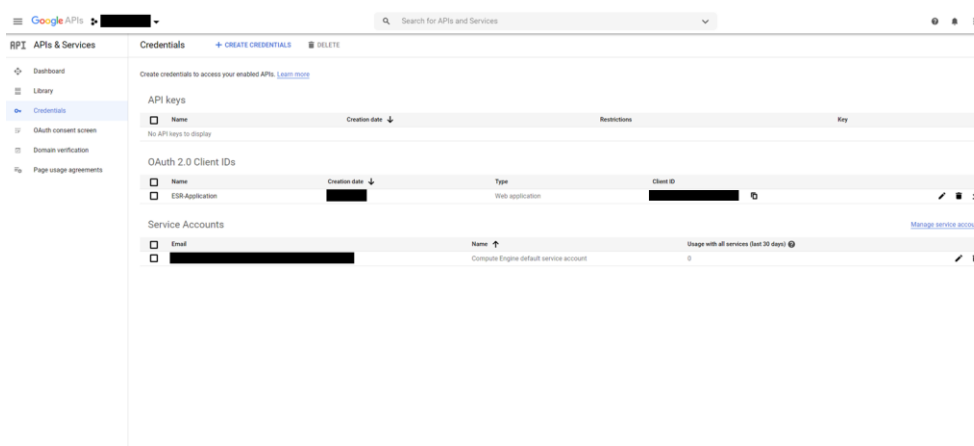
**Figure 10.** ESR-Applicant typography and color palette.

## 4 IMPLEMENTATION

The implementation of the ESR-Applicant is documented in this chapter.

### 4.1 Google Login

As one of the most popular account in the world, log in using Google account is very suitable for students or applicants of Integrify. First, to enable authentication with Google, the application must have been authorized with credential to Google's OAuth 2.0 server /13/. Figure 11 displays the Google console to create and manage the credentials of the application.



**Figure 11.** Google console

After the application has its authorization credentials, the implementation of authentication with a Google account is very convenient since there are multiple third-party libraries in the market to help implementing it, and React-Google-Login is used in this project for its convenience. This library can be found at <https://github.com/anthonyjgrove/react-google-login>. Code snippet 1 shows the implementation of React-Google-Login in the application.

```

import GoogleLogin from 'react-google-login'

import { login } from '../redux/actions'

const GoogleSignIn = () => {
  const dispatch = useDispatch()
  const history = useHistory()

  const responseGoogle = (response: any) => {
    dispatch(login(response.tokenObj.id_token, history))
  }

  return (
    <>
      <GoogleLogin
        clientId={process.env.REACT_APP_GOOGLE_CLIENT_ID!}
        buttonText="Login with Google"
        onSuccess={responseGoogle}
        onFailure={responseGoogle}
        onLoadFinished={responseGoogle}
        cookiePolicy={'single_host_origin'}
      />
    </>
  )
}

```

**Code Snippet 1.** React-Google-Login component for authentication with Google.

In this component, the `clientId` property of `GoogleLogin` component is the client id that have been authorized with the application. If the authenticating with Google is successful, an action will be dispatched to send the token ID to the SMS-Backend for validating if the signed-in Google account is authorized for the test. After the action is dispatched into the Redux store, Redux-Saga will receive the token ID and handle it by calling a request to the SMS-Backend server. If the login is successful, the server will send back a user object with an access token to be attached to request the header of any further requests. While logging out the user will remove the access token from the header. The Redux-saga code is documented in Code snippet 2.

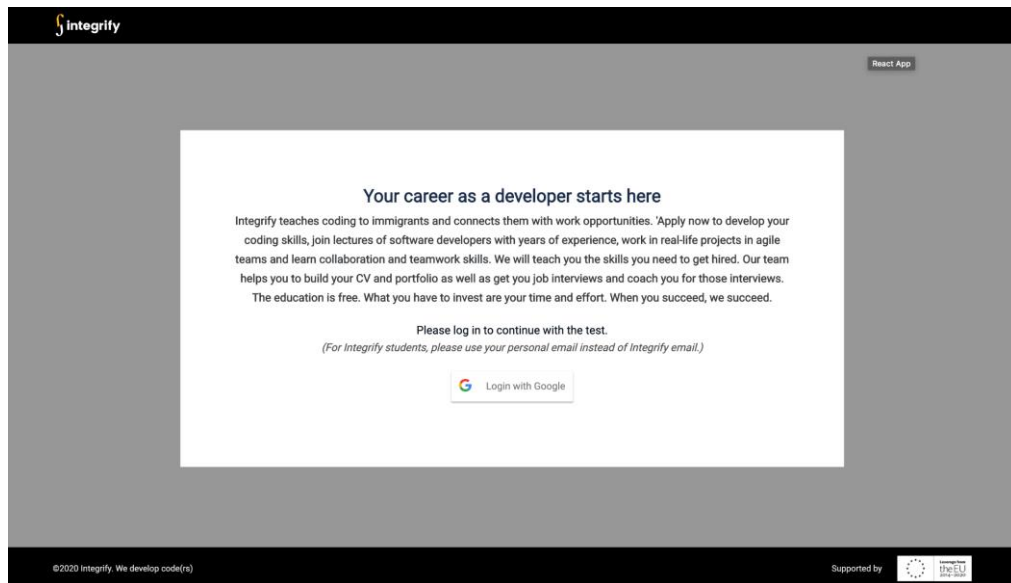


```
function* login() {
  yield takeEvery(LOG_IN, function*(action: LogInAction) {
    const { idToken, history } = action.payload
    try {
      const response = yield call(API.login, idToken)
      const user = response as AuthenticatedUser
      yield put(loginSuccess(user))
      axios.defaults.headers.common[
        'Authorization'
      ] = `Bearer ${user.accessToken}`
      history.push('/')
    } catch (error) {
      yield put(showNotification(error.data.message, 'er-
ror'))
    }
  })
}

function* logout() {
  yield takeEvery(LOG_OUT, function*() {
    yield localStorage.clear()
    delete axios.defaults.headers.common['Authorization']
  })
}
```

**Code Snippet 2.** Handle log in and log out actions

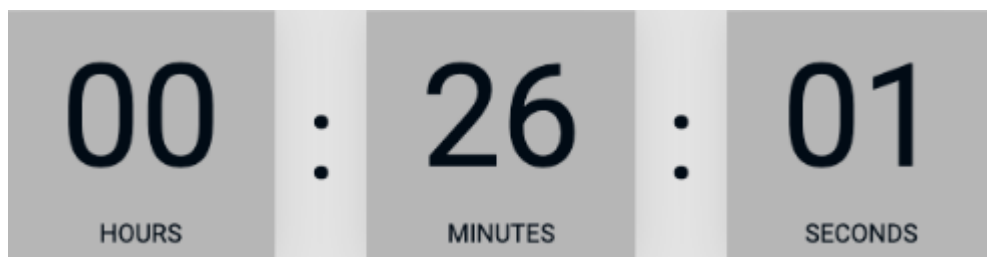
Figure 12 illustrates the login page with React-Google-Login button implemented.



**Figure 12.** Login with Google page

## 4.2 Countdown Timer

A test set of Integrify always has a time limit to finish, so there must be a countdown timer implemented in ESR-Applicant application. Figure 13 illustrates the timer of ESR-Application.



**Figure 13.** Countdown Timer

If the user accesses to a test set, the remaining time of the section will be fetched from the SMS-Backend. And if the user starts the test, the timer of application will start counting down and the remaining time will be saved to the local storage of the web browser in case of refreshing the page or interrupted internet connection. The testing time is updated by adding a starting time stamp in the server when the user starts the test and based on that time to calculate the remaining time of user. If the

time runs out when the user is doing the test, the current answer to the current question that the user is completing, will be submitted automatically to the SMS-Backend for validation. The implementation of the timer is described in Code snippet 3.

```

const { initialTime, startAt, isFinished } = useSelector(
  (state: RootState) => {
    return {
      initialTime: state.test.test?.timeLeft! * 1000 || 0,
      startAt: state.test.test?.startAt,
      isFinished: state.test.test?.isFinished,
    }
  }
)
const [duration, setDuration] = useState(initialTime)
const displayTime = () => {
  return moment.duration(duration).format(
    'hh:mm:ss', { trim: false }
  )
}
const delay = 1000
useEffect(() => {
  if (isFinished) {
    dispatch(showNotification(timeUpMessage, 'error'))
    history.push('/feedback')
  }
  const interval = setInterval(() => {
    if (startAt && duration > 0) {
      setDuration((value) => {
        localStorage.setItem('duration', (value - delay).toString())
        return value - delay
      })
    } else if (duration <= 0 && startAt && !isFinished) {
      dispatch(sendAnswer(false))
      dispatch(finishTest())
    }
  }, delay)
  return () => clearInterval(interval)
}

```

```

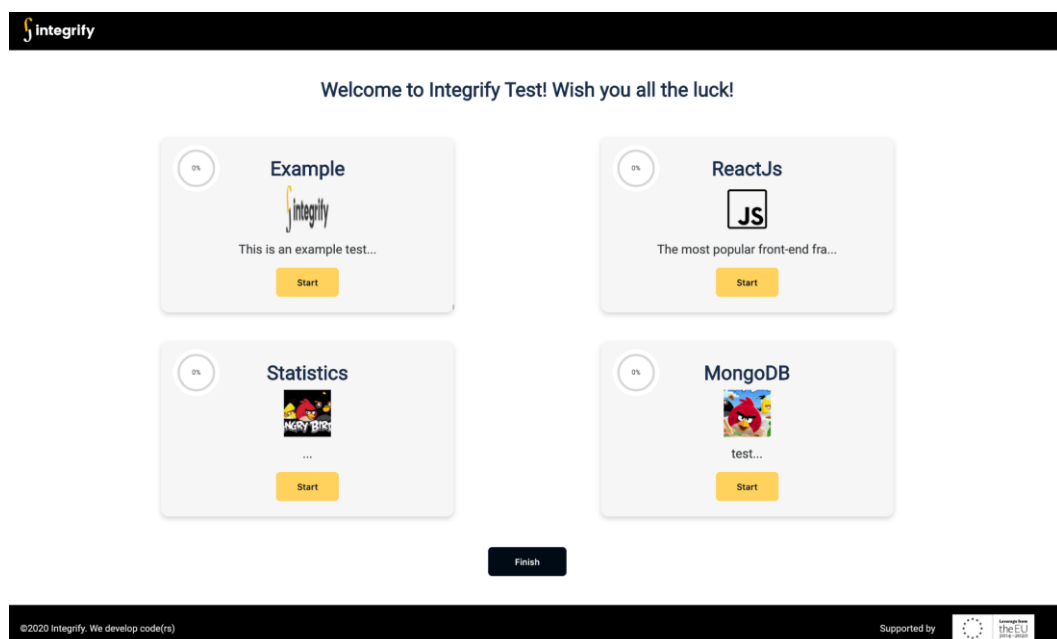
    }, [dispatch, duration, history, initialTime, isFinished, startAt])

    useEffect(() => {
      const savedDuration = +localStorage.getItem('duration')!
      if (initialTime <= savedDuration || savedDuration === 0) {
        localStorage.setItem('duration', initialTime.toString())
        setDuration(initialTime)
      } else {
        setDuration(+savedDuration)
      }
    }, [initialTime])
  
```

**Code Snippet 3.** Countdown Timer component

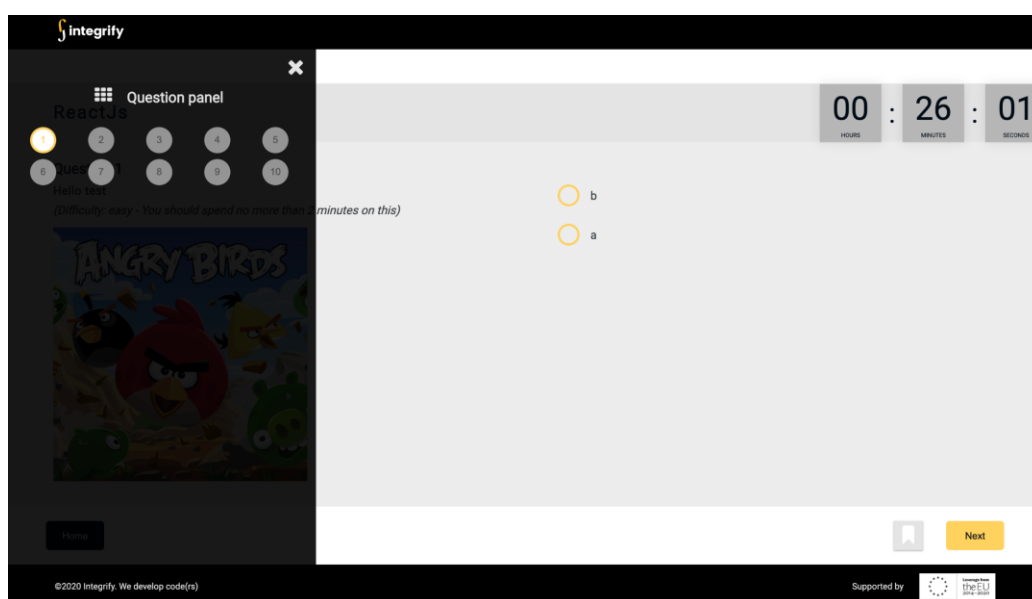
### 4.3 Displaying Test

After the user has logged in with Google successfully, the user will be navigated to the home page to view all the sections of the test set and choose a section to start the test. The home page of application is shown in Figure 14.



**Figure 14.** Home page of ESR-Applicant

Once the user starts a test, they will be navigated to the test page. Since ESR-Applicant is an application for the user to do online test, the most important feature of ESR-Applicant must be to display questions to users and record answers for validating answers. The final test page is displayed in Figure 15, with navigation buttons on a panel, question details, question options, timer, and bookmark button.



**Figure 15.** Test page

### 4.3.1 Test View

The test view component is the top component in displaying the question hierarchy. Its function is to get the current question ID from URL parameters, then to dispatch an action to fetch question details from the SMS-Backend. The reason for implementing the logic of fetching individual question is to prevent the user getting all the questions detail with a single request.

Another function of the test view is once question details have been fetched successfully, it will base on answered questions to handle navigating among questions.

If all questions are answered, the user will be redirected to the home page to continue other sections or finish the test set, or else it will navigate the user to remaining unanswered questions in the test.

The implementation of test view component is shown in Code snippet 4.

```
export default function TestPage() {
  const { questionId, topicName } = useParams()
  const dispatch = useDispatch()
  const history = useHistory()
  const { bookmarked, onClickHandler } = useBookmark(topicName, questionId)
  const { onSubmit, answeredQuestions, navigationReady } = useAnswer(questionId)
  const messageFinish = 'Congratulation !!!'
  const { questionNumber, questions, question, fetchFinished } = useQuestion(
    topicName,
    questionId
  )
  const isTourOpen = useProductTour()
  const [isSideBarOpen, setSideBarOpen] = useState(false)

  useEffect(() => {
    if (navigationReady) {
      if (answeredQuestions.length >= questions.length) {
        dispatch(showNotification(messageFinish, 'success'))
      } else if (fetchFinished) {
        const unansweredQuestions = questions.filter((q) => {
          return answeredQuestions.find((ans) => ans.questionId === q)
            ? false
            : true
        })
        const messageWarning = `You still have ${unansweredQuestions.length} questions left. Make sure you have checked these questions out`
      }
    }
  })
}
```

```

        dispatch(showNotification(messageWarning, 'warn-
ing'))
        history.push(`/topics/${topicName}/${unAnsweredQues-
tions[0]}`)
      } else history.push(`/topics/${topicName}/${ques-
tions[questionNumber]}`)
      dispatch(toggleNavigation())
    }
  }, [
    answeredQuestions,
    dispatch,
    fetchFinished,
    history,
    navigationReady,
    questionNumber,
    questions,
    topicName,
  ])

return (
  <div className="test--container">
    <ProductTour steps={testSteps} isOpen={isTourOpen} />
    <div
      className={`test-button test-button--${isSideBarO-
pen ? 'hide' : ''}`}
    >
      <Icon icon={th} size={24} onClick={() => setSide-
BarOpen(true)} />
    </div>
    <div className={`side-bar side-bar--${isSideBarO-
pen ? 'show' : ''}`}>
      <SideBar
        setSideBarOpen={setSideBarOpen}
        questionId={questionId!}
        topicName={topicName!}
        questions={questions}
      />
    </div>
    <div className="test">
      <div className="test_header">

```

```

        <Heading mainHeading={topicName} variant="second-
ary" />
    </div>
    {question && (
        <div className="test__body">
            <Question question={question} question-
Number={questionNumber} />
        </div>
    )}
</div>
<div className="test__footer">
    <Button variant="dark" text="Home" on-
Click={() => history.push('/')} />
    <button className="btn-bookmark" onClick={onClick-
Handler}>
        <Bookmark className={` ${bookmarked ? 'book-
marked' : 'bookmark'} `} />
    </button>
    <Button
        variant="yellow"
        text={
            answeredQuestions.length === ques-
tions.length ? 'Submit' : 'Next'
        }
        onClick={() => onSubmit(bookmarked)}
    />
</div>
</div>
)
}

```

**Code Snippet 4.** Test view component



### 4.3.2 Question View Component

The question details will be passed from the test view component to the question component to display its content. Besides the static state data such as question description, the question component will display different methods of answering question based on the question type. The content of question component is displayed in Code snippet 5.

```
const QuestionComponent = ({ question, question-
Number }: QuestionProps) => {
  const mess = `You should spend no more than ${
    question.difficulty === 'easy'
      ? 2
      : question.difficulty === 'medium'
        ? 4
        : 7
  } minutes on this`

  return (
    <>
      <div className="test-question__title">
        <Heading
          mainHeading={`Question ${questionNumber}`}
          variant="tertiary"
        />
      </div>
      <div className="test-question__body">
        <div className="test-question__content">
          <p className="question-content">{question.con-
tent}</p>
          <p className="question-difficulty">
            (Difficulty: {question.difficulty} - {mess})
          </p>
          {question.image ? <QuestionImage image={ques-
tion.image} /> : ''}
        </div>
        <div className="test-question__answer">
          {question.type === 'freetext' ? (
            <QuestionScript questionId={question.id} />

```

```

    ) : (
      <QuestionOptions
        questionId={question.id}
        type={question.type}
        options={question.options}
      />
    )}
  </div>
</div>
</>
)
}

```

**Code Snippet 5.** Question view component

If the question type is multiple-choice, the component will render the question options component, to display all options as a list with clickable boxes, or else the component will render the question script component. If an option is chosen or a text is typed in the text box, the component will call an action to save the current process to the local state of the application. The implementation of the question option component is shown in Code snippet 6 and the question script is shown in Code snippet 7. The interfaces of multiple-choice question options are shown in Figure 16 and the checkbox question options is illustrated in Figure 17.

```

const QuestionOptions = ({
  options,
  type,
  questionId,
}: QuestionOptionProps) => {
  const questionType = type === 'checkbox' ? 'checkbox' : 'radio'
  const { currentAnswer, setCurrentAnswer } = useAnswer(questionId)
  let answer: any = currentAnswer?.optionId

  const onChangeHandler = (event: React.ChangeEvent<HTMLInputElement>) => {
    const { id } = event.target
    if (questionType === 'checkbox') {

```

```

    if (answer === null || !answer) answer = []
    if (answer.includes(id)) {
      answer = answer.filter(
        (choice: any) => choice !== id
      )
    } else answer.push(id)
    setCurrentAnswer((value) => {
      return {
        ...value!,
        optionId: answer,
      }
    })
  } else {
    answer = id
    setCurrentAnswer((value) => {
      return {
        ...value!,
        optionId: answer,
      }
    })
  }
}

const checked = (id: string): boolean => {
  if (questionType === 'checkbox') {
    if (answer === null || !answer) answer = []
    return answer.includes(id)
  } else return answer === id
}

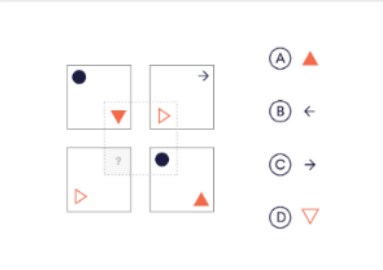
const listItems = options.map((option) => (
  <QuestionOption
    key={option.id}
    questionType={questionType}
    option={option}
    checked={checked}
    onChangeHandler={onChangeHandler}
  />
))

```

```
return <div className="test-question_options">{listItems}</div>
}
```

### Code Snippet 6. Question option component

**Question 3**  
How do you feel about this sample test? (There might be more than 1 correct answer)  
*(Difficulty: hard - You should spend no more than 7 minutes on this)*

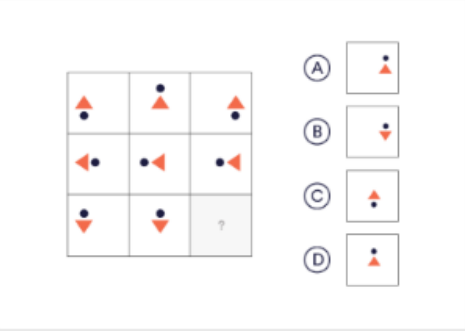






Great!  
 Super great!  
 Soooooooooo greattttttttttttt!  
 I am going to report this !!!

(A) ▲  
 (B) ←  
 (C) →  
 (D) ▼

**Figure 16.** Multiple-choice question type

**Question 1**  
Which of the given shapes would complete the sequence?  
*(Difficulty: easy - You should spend no more than 2 minutes on this)*



A   
 B   
 C   
 D 

**Figure 17.** Checkbox question type

```
const QuestionScript = ({ questionId }: QuestionScriptProps) => {
  const { currentAnswer, setCurrentAnswer } = useAnswer(questionId)
```

```

const initialValue = currentAnswer?.answerText ?? ''

const onChangeHandler = (event: React.ChangeEvent<HTMLText-
TextAreaElement>) => {
  const { value } = event.target
  setCurrentAnswer((answer) => {
    return {
      ...answer!,
      answerText: value,
    }
  })
}

return (
  <>
    <textarea
      className="test-question__script"
      name="textarea"
      placeholder="Type your answer here!"
      value={initialValue}
      onChange={(event) => onChangeHandler(event)}
    />
  </>
)
}

```

**Code Snippet 7.** Question script component

### 4.3.3 Custom hooks useAnswer

Custom Hooks is a method to extract reusable component logic. A custom Hook is a JavaScript function that should be named so that it starts with “use” that may call other Hook functions, such as useState, useSelector. In this application, custom Hook is used to manage the state of the logic and component , includes both the local state and the state from the Redux store /14/.

Custom Hook useAnswer is a function created to manage the answer to a question . This function has a question ID as a parameter; based on the question ID it will access the Redux store and the local storage to check for any record of the answer saved and make it become the initial value of the answer. Furthermore, there is a

validation to prevent the user to submit a blank answer to the system. The implementation of useAnswer Hooks is shown in Code snippet 8.

```
export default function useAnswer(questionId: string) {
  const dispatch = useDispatch()
  const [currentAnswer, setCurrentAnswer] = useState<Answer | null>(null)
  const messageError = 'Error !!! You can not leave answer blank'
  const {
    initialAnswer,
    answeredQuestions,
    savedAnswer,
    navigationReady,
  } = useSelector((state: RootState) => {
    let initialAnswer: Answer = {
      student: state.auth.user!.id!,
      test: state.test.testId!,
      questionId: questionId,
      optionId: null,
      answerText: null,
    }
    const { answeredQuestions, toggleNavigation } = state.topic
    const answeredQuestion = answeredQuestions.find(
      (e) => e.questionId === questionId
    )
    if (answeredQuestion) {
      const { optionId, answerText } = answeredQuestion
      initialAnswer.optionId = optionId
      initialAnswer.answerText = answerText
    }
    const savedAnswer = state.test.currentAnswer
    if (
      savedAnswer?.questionId === questionId &&
      initialAnswer !== savedAnswer
    ) {
      initialAnswer = savedAnswer!
    }
  })
}
```

```

    return {
      navigationReady: toggleNavigation,
      initialAnswer,
      answeredQuestions,
      savedAnswer,
    }
  })

  const onSubmit = (bookmarked: boolean) => {
    if (
      savedAnswer?.answerText !== null ||
      savedAnswer.optionId !== null ||
      bookmarked
    ) {
      if (savedAnswer?.answer-
Text !== null || savedAnswer.optionId !== null)
        dispatch(sendAnswer(false))
      else dispatch(sendAnswer(true))
    } else dispatch(showNotification(messageError, 'error'))
  }

  useEffect(() => {
    if (currentAnswer !== null) {
      dispatch(setTestAnswers(currentAnswer))
    }
  }, [currentAnswer, dispatch])

  useEffect(() => {
    if (currentAnswer === null || currentAnswer.question-
tionId !== questionId) {
      setCurrentAnswer(initialAnswer)
    }
  }, [currentAnswer, dispatch, initialAnswer, questionId])

  return {
    currentAnswer,
    setCurrentAnswer,
    onSubmit,
    answeredQuestions,
  }
}

```

```
navigationReady,
}
}
```

**Code Snippet 8.** Custom Hooks useAnswer

#### 4.4 Feedback Page

After the user has finished the test set, the user will be redirected to the feedback page to give feedback about the user's experience with the application, so that Integrify can improve the application for a perfect application. Figure 18 illustrates the feedback page of the ESR-Applicant application.

**Figure 18.** Feedback page of ESR-Applicant

On the feedback page, the user is asked about the individual experience of every sections of that test set. If the user clicks a reaction button, a sample text of that reaction represented for will be generated and merged into the feedback text of the user.

The implementation of feedback page is shown in Code snippet 9.

```
const FeedbackPage = () => {
  const dispatch = useDispatch()
```



```
    const [feedback, setFeedback] = useState<Feedback>({ feedback: [] })
    const emotionTexts = ['Disappointed', 'Sad', 'Unclear', 'Happy', 'Simple']

    const topics = useSelector((state: RootState) => {
      return state.test.test?.topics
    })

    const onSubmit = (event: React.FormEvent<HTMLFormElement>) => {
      dispatch(sendFeedback(feedback))
      event.preventDefault()
    }

    const handlerEmotionTextx = (index: number, topicName: string) => {
      let feedbacks = feedback.feedback
      let topic = feedbacks.find((tp) => tp.topicName === topicName)
      if (!topic) {
        setFeedback({
          feedback: [
            ...feedbacks,
            {
              topicName: topicName,
              feedback: emotionTexts[index] + '! ',
            },
          ],
        })
      } else {
        setFeedback({
          feedback: [
            {
              topicName: topicName,
              feedback: topic.feedback + (emotionTexts[index] + '! '),
            },
          ],
        })
      }
    }
  }
}
```

```

    }
  }

  const onChangeText = (topicName: string, value: string) => {
    let feedbacks = feedback.feedback
    let topic = feedbacks.find((tp) => tp.topicName === topicName)
    if (topic) {
      topic.feedback = value
      feedbacks = feedbacks.map((fb) =>
        fb.topicName === topicName ? topic! : fb
      )
      setFeedback((value) => {
        return {
          ...value,
          feedback: feedbacks,
        }
      })
    } else {
      topic = {
        topicName: topicName,
        feedback: value,
      }
      feedbacks = [...feedbacks, topic]
      setFeedback((value) => {
        return {
          ...value,
          feedback: feedbacks,
        }
      })
    }
  }
}

return (
  <div className="feedback">
    <Heading
      mainHeading="CONGRATULATION!"
      subHeading="You have reached the finish line"
      variant="primary"
    />
  </div>
)

```

```

    />
    <div className="feedback__description">
      <Description
        content="Thank you for participating in Integ-
rify test. We would like to ask some feed-
back about the test today. This would help us im-
prove our testing system."
        type="fullCenter"
        variant="light"
      />
    </div>
    <form onSubmit={onSubmit} className="feedback__form">
      {topics?.map((topic) => {
        return (
          <FeedbackInputField
            key={topic.name}
            topicName={topic.name}
            onChange={onChange}
            emotionTexts={emotionTexts}
            onClick={handlerEmotionTextx}
          />
        )
      })}
      <Button text="Send Feedback" variant="yel-
low" size="large" />
    </form>
  </div>
)
}

```

**Code Snippet 9.** Feedback page implementation

## 4.5 Storybook

Storybook is an open source tool used to help developers build and manage the edge cases of UI components in isolation for multiple frameworks, such as React, Vue, Angular and more. In this application, storybook was installed and used as a UI management tool for developers and product owner /15/.

Storybook has its own scripts to build and run a storybook at a separate port of the machine - port 6006. The scripts of storybook in “package.json” file is shown in Code snippet 10.

```
"storybook": "start-storybook -p 6006",
"build-storybook": "build-storybook -o build-storybook"
```

**Code Snippet 10.** Storybook scripts

An example of documenting all custom button style in a storybook file is documented in code snippet 11.

```
storiesOf('Common/Button', module).add('All Buttons', () => (
  <div>
    <Button variant="light" text="Button" />
    <Button variant="blue" text="Button" />
    <Button variant="yellow" text="Button" />
    <Button variant="blue" size="small" text="SM" />
    <Button variant="dark" text="Button" />
  </div>
))

storiesOf('Common/Button', module).add('Blue', () => (
  <Button variant="blue" text="Back to home" />
))

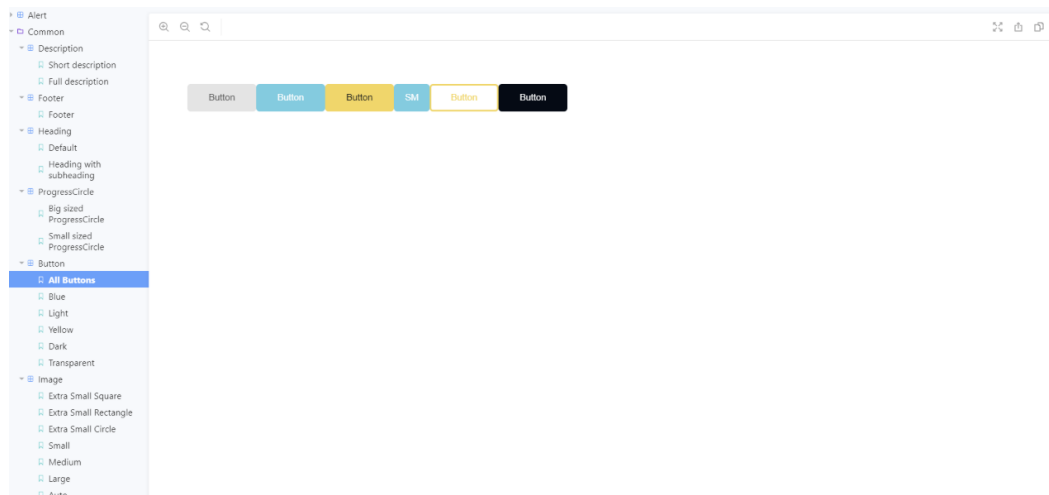
storiesOf('Common/Button', module).add('Light', () => (
  <Button variant="light" text="Topic" />
))

storiesOf('Common/Button', module).add('Yellow', () => (
  <Button variant="yellow" text="Home" size="small" />
))

storiesOf('Common/Button', module).add('Dark', () => (
  <Button variant="dark" text="Next" />
))
```

**Code Snippet 11.** Custom button’s storybook

Figure 19 shows the custom button in the storybook console.



**Figure 19.** Storybook console

#### 4.6 CI/CD with Github Actions

The CI/CD tool of ESR-Applicant is Github Actions because Github Actions is a built-in tool of Github, which is also the version control tool for this project. With Github Actions as a CI/CD tool, every pull request created to the develop branch gets a set of tests for code convention and the deploy status to the target machine. For Github Actions to run, the developer must put a “.yml” file at the path “~root\_folder/.github/workflows” of the application. The content of “.yml” file of ESR-Applicant is documented in Code snippet 12.

```

name: ESR Application workflow
on:
  push:
    branches:
      - develop
  pull_request:
    branches:
      - develop
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Setup node
  
```

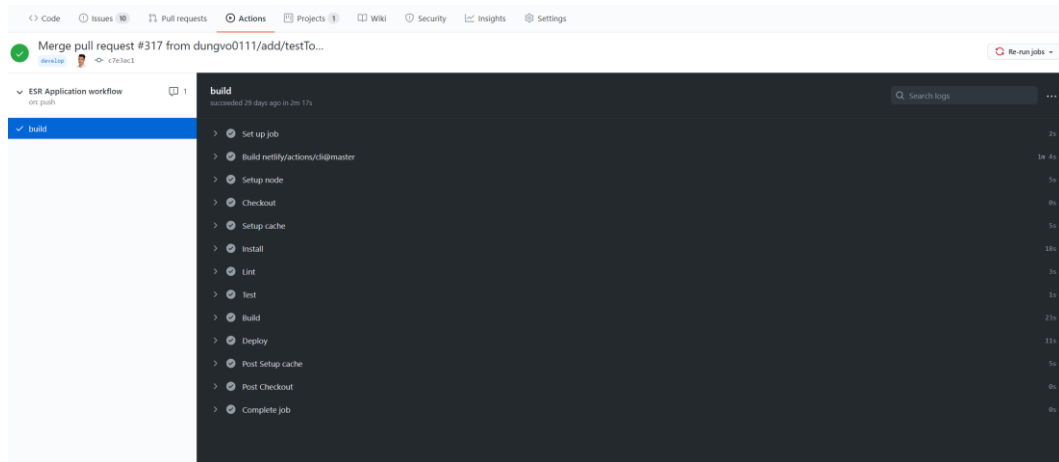
```

    uses: actions/setup-node@v1
    with:
      node-version: '12.x'
  - name: Checkout
    uses: actions/checkout@v2
  - name: Setup cache
    uses: actions/cache@v1
    with:
      path: ~/.npm
      key: ${{ runner.os }}-modules-${{ hash-
Files('**/package-lock.json')}}
      restore-keys: |
        ${{ runner.os }}-modules-
        ${{ runner.os }}-
  - name: Install
    run: npm ci
  - name: Lint
    run: npm run lint
  - name: Test
    run: echo "Test is under construction"
  - name: Build
    run: npm run build
  - name: Deploy
    if: github.event_name == 'push'
    uses: netlify/actions/cli@master
    env:
      NETLIFY_AUTH_TOKEN: ${{ secrets.NETLIFY_AUTH_TO-
KEN}}
      NETLIFY_SITE_ID: ${{ secrets.NETLIFY_SITE_ID}}
    with:
      args: deploy --dir=build --prod
      secrets: '["NETLIFY_AUTH_TOKEN", "NET-
LIFY_SITE_ID"]'

```

**Code Snippet 12.** Github Actions configuration

An example of Github Actions on Github desktop is illustrated in Figure 20.



**Figure 20.** Github Actions jobs finished

## 4.7 Version Controlling

GitHub was chosen to be the version control host of ESR-Applicant application. In this project, the author of the thesis works in a team of four developers so in this chapter, the working process using GitHub is documented.

First the working environment is initialized. After initializing the project with the folder structure, it is pushed on a GitHub repository by using “git init” and “git push”, a remote path is created to the upstream repository of the project by using “Git remote add upstream <path to repository>”.

For daily tasks, the latest update is fetched from the upstream repository with command “git fetch upstream”. After that, “git rebase upstream/develop” command is used to merge the updates to the local machine. When working with new features, another branch is created with command “git checkout -b <new feature name>”. When finishing the work, changes are made to the application by using “git commit -am “<message of commit>”. Then it was pushed to the upstream with command “git push upstream head”, .With this command, GitHub will automatically create another branch on the pushed repository and that branch will be deleted after the Pull Request has been approved and merge into the destination branch (develop

branch). An example of Pull Request created by the author is displayed in Figure 21.

## Bug test #163

**Merged** longro3000 merged 5 commits into `develop` from `bug-test` on 24 May

Conversation 0 Commits 5 Checks 1 Files changed 9

longro3000 commented on 23 May • edited

**Resolved issue**

What issue this PR has resolved? How is the issue resolved (implementation)?  
#158 #159

**Change log**

- removing Question Hooks in couple test components to prevent additional fetching
- prevent submitting blank answer by adding condition to useAnswer before submitting

**Affected Component**

What components do this change affect to ?

Please select an option below..

common

- timer

auth

- (this can be the list specific components that being affected)

home

- (this can be the list specific components that being affected)

landing

**Figure 21.** Pull Request created by the author of the thesis



## 4.8 Testing

Tests were done to ensure the quality of application. This project has ESLint and Husky installed to make sure the written codes follow the convention of Integrify.

No automated tests were implemented for this project, only smoke tests were performed as the main testing method. The smoke tests were made to ensure the functions of the application are executed correctly. The smoke tests performed is shown in Table 2.

**Table 2.** Smoke tests

Test ID	Scenarios	Description	Test step	Expected result
1	Validating login credentials	Test the login functionality of ESR-Applicant to make sure that only assigned users can login with their Google account	1. Open the web application without any test id in the URL	Application should throw an error to inform no valid test id found
			2. Open the application with test id in the URL	Application should forward users to the login page
			3. Click on the "Login with Google" button	Google dialog should be displayed and ask users to select or type in their Google account

			4. Allow the application to access Google data of assigned Google account	Login successfully. The application should redirect users to home page of the application
2	Taking a test in the test set	Test the taking test function to make sure that user can view question's detail, bookmark question, and submit answer.	1. Click on a test's topic	Application should navigate user to the test page and display the first question of the test or the current question that user is doing.
			2. Select an answer or type in answer in the text field	Application should save the current answer in the state and restore it if user refresh the page
			3. Click button "Next" to view next question	Application should submit the answer to the backend and fetch the next question
			4. Click button "Next" without giving any answer or click	The application should stop user from navigating to next question un-

			“Bookmark” button	less user give an answer or bookmark a question
			5. Click “Bookmark” button	The application should append the question to bookmarked questions list of the current topic.
			6. Answer all questions and finish the test	User will be navigated back to home page of the application to finish remaining topics
			7. Click “Finish” button to finish the test set	The application should navigate user to feedback page and prevent user to continue or re-do the test
3	Feedback function	Check feedback functionality of the application	1. Click reaction buttons	The application should generate text in the feedback area
			2. Type texts in the feedback text area	The application should append typed text to the feedback texts

			3. Click submit button to submit feedback	Application should submit feedbacks and log out user while delete all state data stored in the local storage
--	--	--	---	--

## 5 CONCLUSIONS

The implementation of the application documented in this thesis can be considered successful. All the functional requirements have been passed by the contribution of all developers in the team. Users can access the assigned tests with authorized Google accounts. If the login is successful, users can start the test and can resume the process anytime if the time does not run out. After users finish the test, users can give feedback to Integrify, but the test cannot be redone again.

The application was developed with React with TypeScript. The amount of external JavaScript libraries used in this project caused many challenges in the development process. Another challenge is that this project was developed during pandemic time with a team of four developers, so it created a problem with communicating between teammates which caused the delay of designing and developing the application. However, the problems caused by the crisis helped the author of the thesis gain more working experience with a team.

As for improvement, automation tests are needed to be implemented with JavaScript testing frameworks like Jest. The version of the application documented in this thesis is the first version of application in the development stage, so the application needs to be improved to fully responsive and support tablet users.

The thesis documented here is just an evidence of concept, so for the application to be ready for production stage, the application needs to be refactored for better user experience.

## REFERENCES

- /1/ Integrify - About us. Accessed 4 November 2020. <https://www.integrify.io/en/about-us>
- /2/ React (web framework). Accessed 5 November 2020. [https://en.wikipedia.org/wiki/React\\_\(web\\_framework\)](https://en.wikipedia.org/wiki/React_(web_framework))
- /3/ React - Introducing Hooks. Accessed 5 November 2020. <https://reactjs.org/docs/hooks-intro.html>
- /4/ Redux (JavaScript library). Accessed 5 November 2020. [https://en.wikipedia.org/wiki/Redux\\_\(JavaScript\\_library\)](https://en.wikipedia.org/wiki/Redux_(JavaScript_library))
- /5/ Ngoc, Phi. 2020. Redux. Internal PowerPoint. Integrify Oy. Accessed 5 November 2020.
- /6/ Redux-saga document. Accessed 5 November 2020. <https://redux-saga.js.org/>
- /7/ Node.js. Accessed 5 November 2020. <https://en.wikipedia.org/wiki/Node.js>
- /8/ NestJS Introduction. Accessed 7 November 2020. <https://docs.nestjs.com/>
- /9/ PostgreSQL - about. Accessed 7 November 2020. <https://www.postgresql.org/about/>
- /10/ TypeScript - What is TypeScript. Accessed 8 November 2020. <https://www.typescriptlang.org/>
- /11/ Ngoc, Phi. 2020. Security. Internal PowerPoint. Integrify Oy. Accessed 5 November 2020.
- /12/ Redhat - What is CI/CD ?. Accessed 9 November 2020. <https://www.redhat.com/en/topics/devops/what-is-ci-cd>
- /13/ integrating google sign-in into your web app. Accessed 10 November 2020. <https://developers.google.com/identity/sign-in/web/sign-in>
- /14/ React. Building Your Own Hooks. Accessed 10 November 2020. <https://reactjs.org/docs/hooks-custom.html>
- /15/ Storybook. Accessed 10 November 2020. <https://storybook.js.org/>