FeedHUB

A

Major Project Report

Submitted
In partial fulfilment
For the award of the Degree of

Bachelor of Technology (CSE- CT & IS)

In the Department of Computer Science & Engineering

Submitted By:	**Supervisor:**	**Submitted to:**
Sanchit Bhatnagar *(160101241)*	Ms Arushi Jindal	Dr Nitin Rakesh
Ashish Kr. Singh *(160101002)*	Asst. Professor	HOD (CSE)
Lalit Joshi *(160101148)*		
Amar Singh *(160101035)*		

Department of Computer Science and Engineering
Sharda University, Greater Noida

November 2019

Candidate's Declaration

I hereby declare that the work, which is being presented in this report, entitled **"FeedHUB"** in partial fulfilment for the award of Degree of "**Bachelor of Technology**" in the Department of Computer Science. **Sharda University** is a record of my own investigations carried under the Guidance of Ms Arushi Jindal, Department of Computer Science Engineering.

I have not submitted the matter presented in this report anywhere for the award of any other Degree.

Sanchit Bhatnagar

Computer Science Engineering (CT & IS),
Enrollment No.: 2016005152

Lalit Joshi

Computer Science Engineering (CT & IS),
Enrollment No.: 2016013397

Aashish Kumar Singh

Computer Science Engineering (CT & IS),
Enrollment No.: 2016005122

Amar Singh

Computer Science Engineering (CT & IS),
Enrollment No.: 2016008492

Counter Signed by

Ms Arushi Jindal

ACKNOWLEDGEMENT

We would like to pay our regards and heartfelt gratitude to Ms Arushi Jindal (Project Supervisor) and to my teachers especially Mr Nitish Patil for giving right suggestions that make the project fair and efficient. As my project guide, our teacher helps us through their suggestions. We shall be highly obliged for them.

We also pay our regards to Prof. Dr Nitin Rakesh (HOD - Computer Science) for his ongoing support and encouragement in every aspect and for giving us platform to develop our career out there. We are also thankful to our parents for giving us financial support and appreciating us at every step.

The successful completion of our project would not have been possible without the dedicated support from all our mentors, family and friends.

Table of Contents

Table of Figures

ABSTRACT

The Dynamic Feedback system is an application specifically built for budding entrepreneurs, product managers or product owners, who have listed their product in the market and have a considerable customer base. This application would allow owners to collect feedback from its users. This will help owners or managers to work on the flaws of the product and where it lacks. This will also help owners to have a robust relationship with its users as well. Users can post their reviews about the product, and can also inform the developer team if there's any bug in the product. These reviews would also help in analyzing the number of people actually using their product. Feedback from users would also help in the Quality improvement of the product and in its efficiency. The feedback mechanism helps in catering the weaknesses and further strengthening the strengths.

1. INTRODUCTION

Users are the backbone of one's product. So, in order to sustain in the market, the product must be as per the user need. Now, once it's done comes the major part, where product owners have to keep a track on product's growth and its downfall and the changes they need to adapt, in order to have a place in the market. For this, the owners and managers, require the reviews or feedbacks of its customers. So, we created an application where users can post their feedback about the product, and the Owners can collect these feedbacks about their product.

An Online Feedback System is a feedback generation system which gives proper feedback to teacher provides the proper feedback to the teachers about their teaching quality on basis of rating very poor, poor, average, good, very good. In the existing system students requires giving feedback manually. In existing system report generation by analysing all feedback form is very time-consuming. By online feedback system report generation is consumes very less time

.

2. REQUIREMENT ANALYSIS (SOFTWARE & HARDWARE)

2.1. HARDWARE REQUIREMENTS:

2.1.1 Computer Device:
- 2.1.1.1 RAM: Minimum – 4 GB Recommended – 6Gb
- 2.1.1.2 Graphic memory - 2Gb recommended
- 2.1.1.3 Windows 7 or higher Or Ubuntu or Mac OS
- 2.1.1.4 Storage Requirements – 10Gb

2.2 SOFTWARE REQUIREMENTS:

2.2.1 Code Editor Software – VSCode /Sublime Text /Atom etc.
2.2.2 Web Browser (Google Chrome – V69.0 or higher)
2.2.3 Node.js
2.2.4 MongoDB – V4.0.10 or higher

2.3 PREREQUISITES:

2.3.1 Basic ReactJS Knowledge
2.3.2 HTML, CSS, Bootstrap
2.3.3 Understanding of working of Databases
2.3.4 Basic Command-Line Interface usage
2.3.5 Basic Knowledge of working Web Apps

3. DESIGN DOCUMENT

3.1 Background

What is ReactJS?

ReactJS basically is an open-source JavaScript library which is used for building user interfaces specifically for single-page applications. It's used for handling view layer for web and mobile apps. React also allows us to create reusable UI components. React was first created by Jordan Walke, a software engineer working for Facebook.

Why ReactJS?

Whenever there is an update for apps written in Swift/Objective-C or Java, the whole app needs to be recompiled and a new version has to be distributed to the App Store again. All this can take a few weeks depending on the App Store review process.

What is MongoDB?

MongoDB is a *non-relational database* (often known as No-SQL). The term No-SQL is very popular but it contrasts to the fact that SQL has nothing to do with Relational Databases, (Tabular DB) other than the fact that it is just a querying language. They are a totally different thing so I think it's a bit weird that rather than using Non-Relational DB they use No-SQL.

What is ExpressJS?

Express.js, or simply Express, is a web application framework for Node.js, released as free and open-source software under the MIT License. It is designed for building web applications and APIs. It has been called the de facto standard server framework for Node.js.

Heroku for deployments

Heroku is a (Application Development) Platform-as-a-Service. What's that you say? It is a cloud-based platform that provides application developer an environment ready to code, test and run an app without needing to worry about the infrastructure details (Think where is my app running, where is the data stored, what are the networking configurations.)

3.2 Modules

3.2.1 Server Side

- **API Creation**: APIs are created which are called by client-side to interact with the back-end without direct interaction with the Database and other delicate areas.
- **Website Redirection**: It defines server-side routes for the WebApp for click events and another user interaction
- **Authentication**: It takes care of authentication and authorization of user via PassportJS implementing OAuth Login.
- **Payment Portal:** Payment for purchase of credits via Stripe API is done which is secure and no credit card details are saved during the process
- **Linking Front-End with Back-End**: The ExpressJS lib is used for Full Stack which takes care of linking front-end web app and back-end servers together to work in a flawless manner
- **Linking Back-End with Database:** The ExpressJS lib also links the Mongo DB Database via Mongoose lib which lets the client push and pull data from the database.

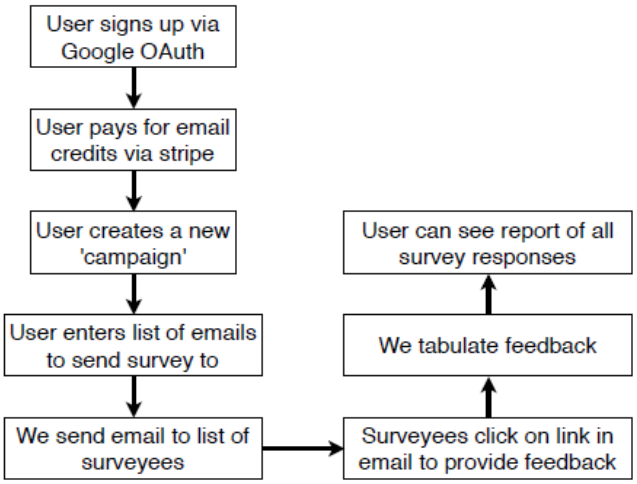
3.2.2 Database Management

- **Mongo DB Atlas:** In this WebApp, Mongo DB Atlas is used. Atlas is the cloud version of Mongo DB with is already deployed to the cloud.
- **Data Fetching:** The data is fetched and added for the client through backend server functionalities.
- **User Login:** The database is used to save user information when it first logs in and fetches data after every next login
- **User Data:** The database saves information that is created by the user in the WebApp like the number of credits, etc.
- **Form Data:** Every form created and sent by users are saved in the database
- **Response Data:** The responses obtained by clicks in the email is dynamically processed and saved in the database.

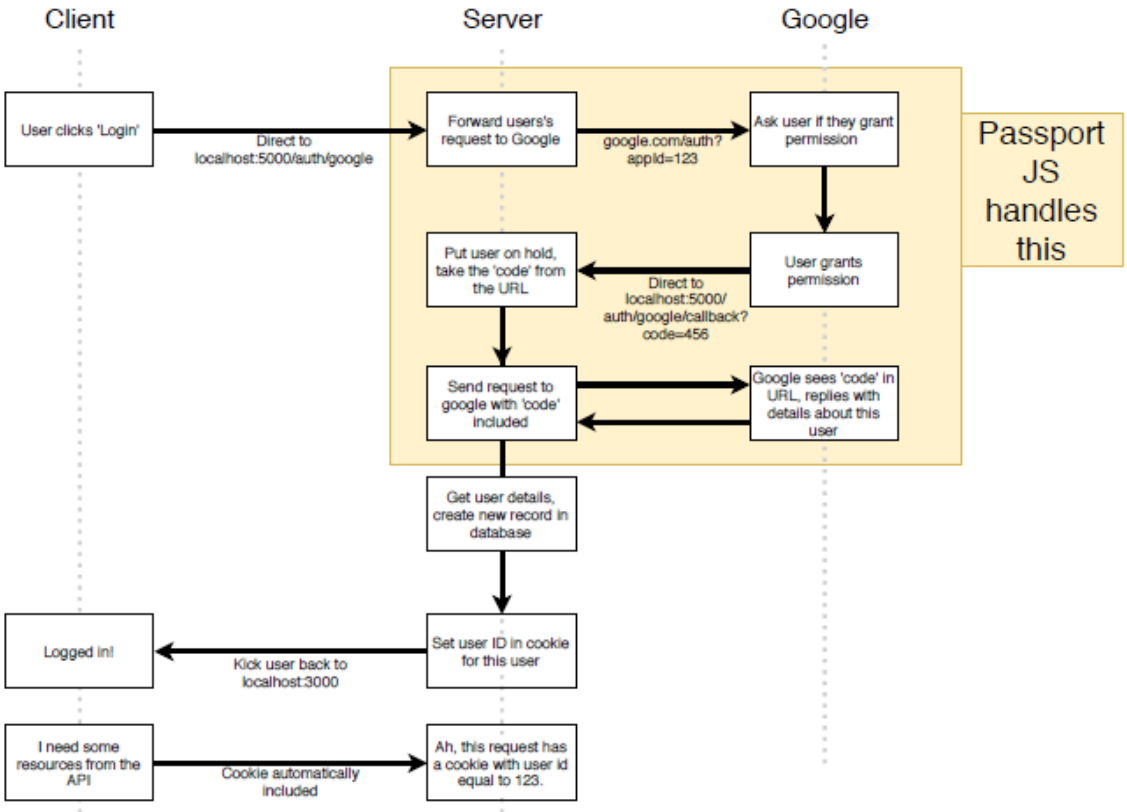
3.2.3 Front End WebApp

- **Homepage:** The landing page of WebApp, gives a glimpse of features and functionality about the app.
- **Header:** A persistent header which is fixed on top overall webpages, it shows link to homepage, gives user details such as credits and gives user the ability to logout or add credits
- **Dashboard:** It is the page that is shown after the user is successfully logged in, it shows the list of surveys created by that user and its responses if any, it also has a button to create new Surveys
- **Survey Add/Edit:** This page lets the user to create new surveys or edit previously created surveys and resend it if credits available
- **Payment Portal:** A payment portal by Stripe API which lets the user input his credit card information to obtain credits which are the currency in the WebApp to send Surveys

3.3 Flowchart and Designs



Flowchart 3.1.1



Flowchart 3.1.2

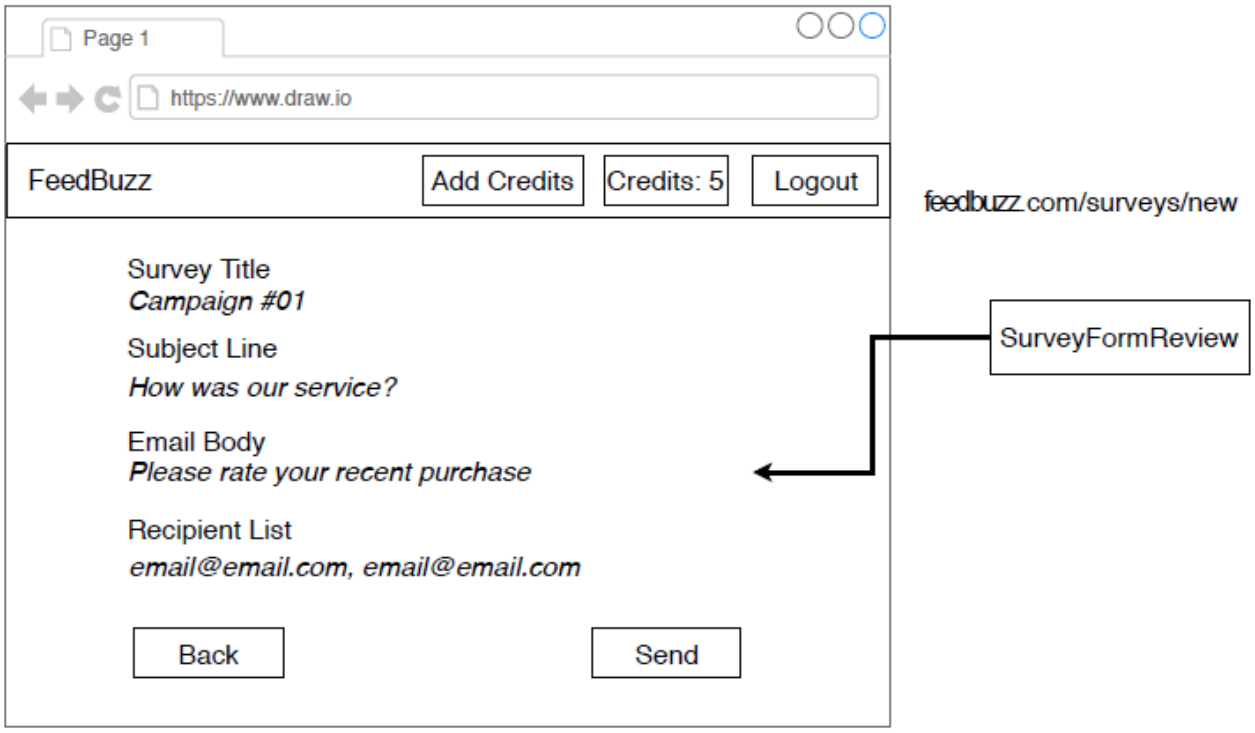


Figure 3.2.1

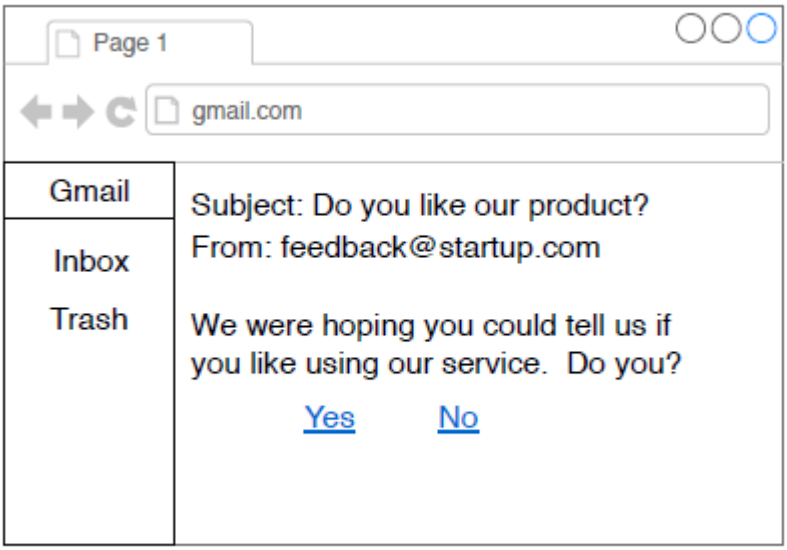


Figure 3.2.1

4. CODING IMPLEMENTATION

4.1 Server-Side Script Files

4.1.1 Index.js – Initialization of Mongo DB and other Services by ExpressJS

```
const express = require('express');
const mongoose = require('mongoose');
const cookieSession = require('cookie-session');
const passport = require('passport');
const bodyParser = require('body-parser');
const {cookieKey} = require("./config/keys");
const {mongoURI} = require("./config/keys");
require("./models/User"); //Loads the config
require('./models/Survey');
require("./services/passport");

mongoose.connect(mongoURI, {useNewUrlParser: true})
    .then(() => console.log("Mongo Connected"))
    .catch(e => console.log("Mongo Error", e));

const app = express();
app.use(bodyParser.json());
app.use(
    cookieSession({
      maxAge: 30 * 24 * 60 * 60 * 1000,
      keys: [cookieKey]
    })
);
app.use(passport.initialize());
app.use(passport.session());

require("./routes/authRoutes")(app); //??
require('./routes/billingRoutes')(app);
require('./routes/surveyRoutes')(app);

if (process.env.NODE_ENV === 'production') {
  app.use(express.static('client/build'));

  const path = require('path');
  app.get('*', (req, res) => {
    res.sendFile(path.resolve(__dirname, 'client', 'build', 'index.html'))
  })
}

const PORT = process.env.PORT || 5000;
app.listen(PORT);
```

4.1.2 Auth Routes - API Routes for Authentication

```
const passport = require('passport');
//Auth Routes
module.exports = (app) => {

  //info Initiates Google OAuth
  app.get(
      '/auth/google',
      passport.authenticate('google', {
        scope: ['profile', 'email']
      }));

  //info Callbacks for Google OAuth
  app.get('/auth/google/callback', passport.authenticate('google'), (req, res) =>
{
    res.redirect('/surveys');
  });

  //Logout (All)
  app.get('/api/logout', (req, res) => {
    req.logout();
    res.redirect('/')
  });

  // INFO Gets Info of Current user (in JSON)
  app.get('/api/current_user', (req, res) => {
    //temp res.send(req.session);
    res.send(req.user)
  });
}
```

4.1.3 Survey Routes - API Routes for Survey Creation/Edit

```
const mongoose = require('mongoose');
const requireLogin = require('../middlewares/requireLogin');
const requireCredits = require('../middlewares/requireCredits');
const Survey = mongoose.model('surveys');
const Mailer = require('../services/Mailer');
const surveyTemplate = require('../services/emailTemplate/surveyTemplate');

module.exports = (app) => {
  //idea Can be changed : Currently this is after the user responded function
  app.get('/api/surveys/thanks', (req, res) => {
    console.log(req);
    res.send('Thank you for your response')
  });
  app.post('/api/surveys', requireLogin, requireCredits, async (req, res) => {
//fixme require credits not working, number goes negative
    const {title, subject, body, recipients} = req.body;

    const survey = new Survey({
      title,
      subject,
```

```
      body,
      recipients: recipients.split(',').map(email => ({email: email.trim()})),
      _user: req.user.id,
      dateSent: Date.now() //idea for Creating draft this needs to be changes
    });

    const mailer = new Mailer(survey, surveyTemplate(survey));

    try {
      await mailer.send(); //sends mail
      await survey.save(); //saves survey to mongoDb
      req.user.credits -= 1; //deducts credits
      const user = await req.user.save(); //saves update done to user
      res.send(user)
    } catch (e) {
      res.status(422).send(e);
    }
  });
};
```

4.1.4 Billing Routes – API Routes for billing

```
const keys = require('../config/keys');
const stripe = require('stripe')(keys.StripeSecretKey);
const requireLogin = require('../middlewares/requireLogin');

module.exports = (app) => {

  app.post('/api/stripe', requireLogin, async (req, res) => {
    // console.log(req.body);

    const charge = await stripe.charges.create({
      amount: 100,
      currency: 'inr',
      description: '₹1 for Payment for 5 credits',
      source: req.body.id
    });

    // console.log(charge);
    req.user.credits += 5;
    const user = await req.user.save();
    res.send(user);
    res.redirect('/surveys');
  })
};
```

4.1.5 Passport.js – Authentication Service Configuration

```
const passport = require('passport');
const GoogleStrategy = require('passport-google-oauth20').Strategy;
const mongoose = require('mongoose');
const keys = require('../config/keys');
//Passport.js

const User = mongoose.model('users'); //1 args = Fetching (~Get)

passport.serializeUser((user, done) => {
  done(null, user.id)
});

passport.deserializeUser((id, done) => {
  User.findById(id)
      .then(user => {
        done(null, user)
      })
});

passport.use('google', new GoogleStrategy({
      clientID: keys.googleClientID,
      clientSecret: keys.googleClientSecret,
      callbackURL: '/auth/google/callback',
      proxy: true
    }, async (accessToken, refreshToken, profile, done) => {
      const existingUser = await User.findOne({googleId: profile.id});
      if (!existingUser) {
        const user = await new User({googleId: profile.id, createdAt:
Date.now()}).save();
        return done(null, user);
      }
      console.log("User Exists");
      done(null, existingUser)
    })
);
```

4.1.6 Mailer.js – Mail Sending Service Configuration

```
const sendGrid = require('sendgrid');
const helper = sendGrid.mail;
const keys = require('../config/keys');

class Mailer extends helper.Mail {
  constructor({subject, recipients}, content) {
    super();
    this.sendGridAPI = sendGrid(keys.sendGridKey);
    this.from_email = new helper.Email('no-reply@feedbuzz.com');
    this.subject = subject;
    this.body = new helper.Content('text/html', content);
    this.recipients = this.formatAddresses(recipients);
```

```
    this.addContent(this.body); //helper.Mail function
    this.addClickTracking();
    this.addRecipients();
  }

  formatAddresses(recipients) {
    return recipients.map(({email}) => {
      return new helper.Email(email);
    });
  }
  addClickTracking() {
    const trackingSettings = new helper.TrackingSettings();
    const clickTracking = new helper.ClickTracking(true, true);

    trackingSettings.setClickTracking(clickTracking);
    this.addTrackingSettings(trackingSettings)
  }

  addRecipients() {
    const personalize = new helper.Personalization();
    this.recipients.forEach(recipient => personalize.addTo(recipient));
    this.addPersonalization(personalize);
  }

  async send() {
    const request = this.sendGridAPI.emptyRequest({
      method: 'POST',
      path: '/v3/mail/send',
      body: this.toJSON()
    });
    return await this.sendGridAPI.API(request); //question
  }

}

module.exports = Mailer;
```

4.1.7 Login Middleware – Middleware for checking auth during API requests

```
module.exports = (req, res, next) => {
  if (!req.user) return res.status(401).send({error: 'You must log in'});
  next();
};
```

4.1.8 Require Credits Middleware – Middleware for checking credits before sending Email

```
module.exports = (req, res, next) => {
  if (req.user.credits < 1) return res.status(402).send({error: 'Not enough
credits!'});
  next();
};
```

4.1.9 Survey Model Schema – Survey Model for MongoDB

```
const mongoose = require('mongoose');
const {Schema} = mongoose;
const RecipientSchema = require('./Recipient');

const surveySchema = new Schema({
  title: String,
  body: String,
  subject: String,
  recipients: [RecipientSchema],
  yes: {type: Number, default: 0},
  no: {type: Number, default: 0},
  _user: {type: Schema.Types.ObjectId, ref: 'User'},
  dateSent: Date,
  lastResponse: Date
});

mongoose.model('surveys', surveySchema);
```

4.1.10 User Model Schema – User Model for MongoDB

```
const mongoose = require('mongoose');
const {Schema} = mongoose;

const userSchema = new Schema({
  googleId: {type: String},
  createdAt: {type: Date},
  credits: {type: Number, default: 1}
});

mongoose.model('users', userSchema);
```

4.2 Client-Side Code Files

4.2.1 Index.js

```
import React from "react";
import ReactDOM from 'react-dom';
import {applyMiddleware, createStore} from "redux";
import {Provider} from "react-redux";
import reduxThunk from "redux-thunk";

import 'materialize-css/dist/css/materialize.min.css'

import App from "./components/App";
import reducers from "./reducers";
//temp only for dev mode
import axios from "axios";

window.axios = axios;

const store = createStore(reducers, {}, applyMiddleware(reduxThunk));
ReactDOM.render(
    <Provider store={store}>
      <App/>
    </Provider>,
    document.querySelector('#root')
);
```

4.2.2 App.js – First rendering React-page

```
import React, {Component} from 'react';
import {connect} from 'react-redux';
import {BrowserRouter, Route} from "react-router-dom";
import Header from "./Header";
import {fetchUser} from "../actions";
import LandingPage from "./LandingPage";
import Dashboard from "./Dashboard";
import SurveyNew from "./surveys/SurveyNew";

// todo HTML File fix
class App extends Component {
  componentDidMount() {
    this.props.fetchUser();
  }

//idea Add user login check to redirect to landing or survey page
  render() {
    return (
        <BrowserRouter>
          <Header/>
          <div className='container'>
            <Route exact path="/" component={LandingPage.bind(this)}/>
            <Route exact path="/surveys" component={Dashboard}/>
```

```
            <Route exact path="/surveys/new" component={SurveyNew}/>
          </div>
        </BrowserRouter>
    );
  }
}

export default connect(
    '', {fetchUser}
)(App);
```

4.2.3 Header Component

```
import React, {Component} from 'react';
import {connect} from "react-redux";
import {Link} from "react-router-dom";
import StripeWrapper from "./StripeWrapper";


function mapStateToProps({auth}) {
  return {auth};
}

class Header extends Component {
// FIXED_fixme After Payment it shows as logged out - CWU in StripeWrapper
  renderContent() {
    switch (this.props.auth) {
      case null:
        return;
      case false:
        return <li><a href="/auth/google">Login With Google</a></li>;
      default:
        return [
          <li key="1"><StripeWrapper/></li>,
          <li key="3" style={{margin: '0 10px'}}>
            Credits: {this.props.auth.credits}
          </li>,
          <li key="2"><a href="/api/logout">Logout</a></li>
        ]
    }
  }

  render() {
    console.log(this.props);
    return (
        <nav>
          <div className="nav-wrapper">
            <Link to={this.props.auth ? '/surveys' : '/'} className="left brand-
logo">
              FeedBuzz</Link>
            <ul className="right">
              {/*<li><a>Login With Google</a></li>*/
                this.renderContent()
              }
```

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

export default connect(
    mapStateToProps
)(Header);
```

4.2.4 Stripe Payment Component

```
import React, {Component} from "react";
import StripeCheckout from "react-stripe-checkout";
import {connect} from "react-redux";
import {fetchUser, handleToken} from "../actions";

class StripeWrapper extends Component {
  componentWillUnmount() {
    // this.props.fetchUser();
  }

  render() {
    // todo Change this us dollar to Rupee
    return (
        <StripeCheckout
            name={"FeedBuzz"}
            description={"₹1 for 10 Credits"}
            amount={100}
            currency="INR"
            token={(token) =>
                this.props.handleToken(token)
            }
            stripeKey={process.env.REACT_APP_STRIPE_KEY}
        >
          <button className="btn">Add Credits</button>
        </StripeCheckout>
    )
  }
}

export default connect(null, {handleToken,fetchUser})(StripeWrapper);
```

4.2.5 Survey Form Page

```
import React, {Component} from 'react';
import {Field, reduxForm} from "redux-form";
import SurveyField from "./SurveyField";
import _ from "lodash";
import {Link} from "react-router-dom";
import validateEmails from "../../utils/validateEmails";
```

```
import formFields from "./formFields";


class SurveyForm extends Component {
  renderField() {
    return _.map(formFields, (field) => {
      return <Field
          key={field.name}
          component={SurveyField}
          type={"text"}
          label={field.label}
          name={field.name}
      />
    })
  }

  render() {
    return (
        <div>
          <form onSubmit={this.props.handleSubmit(this.props.onSurveySubmit)}>
            {this.renderField()}
            <Link    to={'/surveys'}    className='red    btn-flat    white-text
red'>Cancel</Link>
            <button type='submit' className='teal btn-flat right white-text'>
              <i className='material-icons left'>done</i>
              Submit
            </button>
          </form>
        </div>
    );
  }
}

function validate(values) {
  const errors = {};
  _.each(formFields, ({name, label}) => {
    errors.recipients = validateEmails(values.recipients || '');
    if (!values[name]) {
      // if (name === 'recipients') {
      //   errors[name] = `Please provide ',' separated correct Email IDs in correct
format`
      /*} else*/
      errors[name] = `Please provide a correct ${label}`
      // }
    }
  });
  return errors;
}

export default reduxForm({
  validate,
  form: 'surveyForm',
  destroyOnUnmount: false
})(SurveyForm);
```

4.2.6 Survey Form Review Page

```
import React, {Component} from 'react';
import {connect} from 'react-redux';
import formFields from "./formFields";
import _ from "lodash";

import {submitSurvey} from "../../actions";
import {withRouter} from "react-router-dom";


function mapStateToProps(state) {
  const {surveyForm: {values}} = state.form;
  return {values}
}

class SurveyFormReview extends Component {
  renderFormReviewFields = _.map(formFields, field => {
    return (
        <div key={field.name}>
          <label>{field.label}</label>
          <div>{this.props.values[field.name]}</div>
        </div>
    )
  });

  render() {
    const {onCancel, values, submitSurvey, history} = this.props;
    return (
        <div>
          <h4>Please Confirm your entries</h4>
          {this.renderFormReviewFields}
          <button
              onClick={onCancel}
              className="yellow darken-3 btn-flat">
            Go Back
          </button>
          <button
              type='submit'
              onClick={() => submitSurvey(values, history)}
              className="green right btn-flat">
            Save and Send
            <i className="material-icons right ">email</i>
          </button>
        </div>
    );
  }
}

export default connect(
    mapStateToProps, {submitSurvey}
)(withRouter(SurveyFormReview));
```

5. CONCLUSION

Easy maintenance of records of various Recruiters (Companies), job and job seekers has been maintained. To check for the detailed perspective, the jobseekers can do so through quick search provided in the portal. An attempt to prevent and reduce human error.
Reduced manual work. Classic functionality focuses on data storage. In addition, it also outlines the means to retrieve and analyze data to extract, transform, load and process data.

6. FUTURE SCOPE

This application can be easily implemented under various situations. We can add future updates as and when we require. Reusability is possible whenever required in the application. There is flexibility in all the modules.
There is a communication gap between organizations and their customers, this software, that we have built, will help in filling the gap that the above two, currently have. This software will let their organization know about their user's experience in the real time, using which the Organization can take the required actions or make changes, whatever needed.
For the software, we have used some of the most present-day tools and technologies, which assurances long term reliability, future updates, robust and secured system. This software, as of now, is specifically built up for Customer support and for the betterment of Client/ Organization relationship. In next few updates, we are planning to introduce even more features like, Educational Feedbacks and Employee Feedbacks.
This application has been build keeping in mind the future aspect of it.

7. REFERENCES

Toolkits:

[1] MongoDB - https://www.mongodb.com/
[2] ReactJS - https://reactjs.org/
[3] NodeJS - https://nodejs.org/en/
[4] VSCode Editor - https://code.visualstudio.com/

Websites for Reference:

[1] MongoDB - https://www.mongodb.com/
[2] ExpressJS – https://expressjs.com
[3] ReactJS - https://reactjs.org/
[4] Learn ReactJS - https://reactjs.org/tutorial/tutorial.html
[5] Learn MERN Stack Development - https://hashnode.com/post/react-tutorial-using-mern-stack-ciiyus9m700qqge53mer0isxz
[6] Video Tutorials for ReactJS - https://www.youtube.com/watch?v=MhkGQAoc7bc