Logo

Description automatically generated

GROUP 2



**Chongye Wang Wang**

**Pranjay Sagar**

**Dishant Shah**

**Apoorv Mehrotra**

**Supriya Sethi**

**Apoorv Mehrotra**

The leader cum driver of the group. He demonstrated to have a splendid match of specialized abilities and management characteristics. His errand tasks and following capacity astonished everybody. He was the main thrust of the everyday exercises. Aside from dealing with the group, facilitating SCRUM calls, he likewise dealt with muddled question rationale, performance testing, few front-end parts and many complicated back-end queries. He regarded everyone's assessment and settled on the best plausible choices to guarantee the fulfillment of the task.

**Dishant Shah**

The most carefree, cheerful, persevering individual of the group. He was probably the quickest performer of the group. He was consistently cheerful and prepared to acknowledge any errands. The expectations were constantly guaranteed before time with a solicitation to relegate him more. He took an interest in all the conversations and gave commendable proposals to deal with not many of the most convoluted scenarios. He kept the disposition light with this jaunty demeanor. He made magnificent commitments towards both front and backend advancement.

**Supriya Sethi**

The calmest member of the group. She was exceptionally calm during the whole undertaking and guaranteed to keep up the quiet climate during the whole time period of the task. She develop company persona frontend and backend with a sharp understanding with nothing to allude to. She guaranteed that the concealed worker UI supplements the current UI of Glassdoor site. She was a brisk student and a grand performed. She likewise guaranteed to keep up the uprightness of the project.

**Pranjay Sagar**

The mainstay of the group. From the earliest starting point till the last period of the undertaking, he guaranteed to finish his errands and help every other person in their appointed assignments. He was the Frontend champion. He made some astoundingly clear UIs propelling everybody to coordinate to the UI expectation in the group. He additionally made significant commitment in the administrator persona convoluted backend query handling. His can-do disposition drove this convoluted project to progress.

**Chongye Wang**

The silent contributor to the group. He took an interest in making databases. He likewise assumed the liability of actualizing redis-server to help improve the presentation of the framework. He made commitments towards looking for the trustworthiness of the task. He additionally contributed towards the finishing of documentation needed for the venture. He composed a few among the significant queries at backend and guaranteed the trustworthiness during the consolidation of those complex queries.

**Database Schema**

**Graphical user interface, application, table

Description automatically generated**

**Figure 1. Database Schema for the MySQL database.**

**Text

Description automatically generated**

**Figure 2. Database Schema for the MongoDB database.**

**System Architecture and Design**

**Diagram

Description automatically generated**

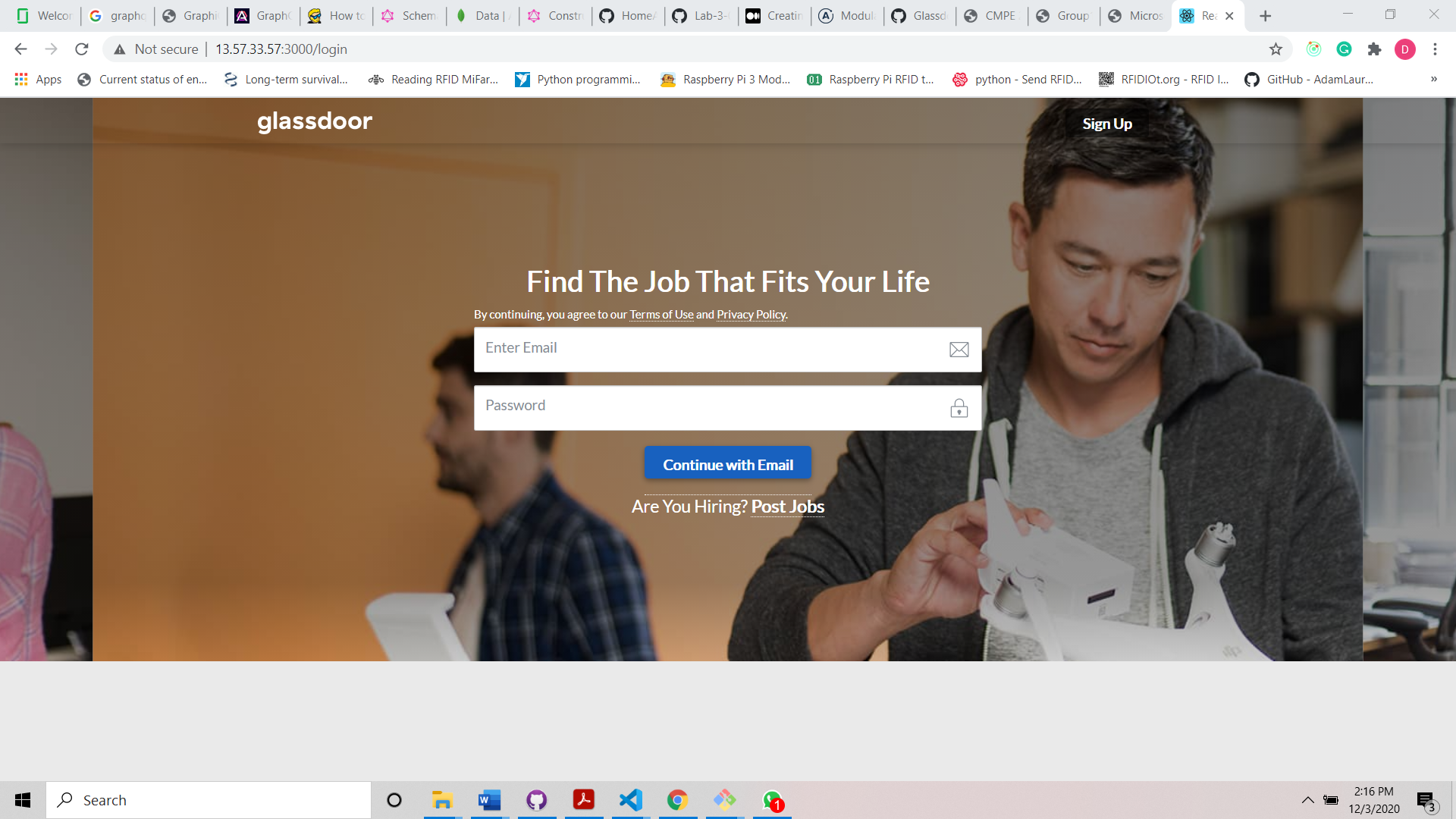
**Figure 3. Architecture Diagram of the application.**

**Object Management Policy**

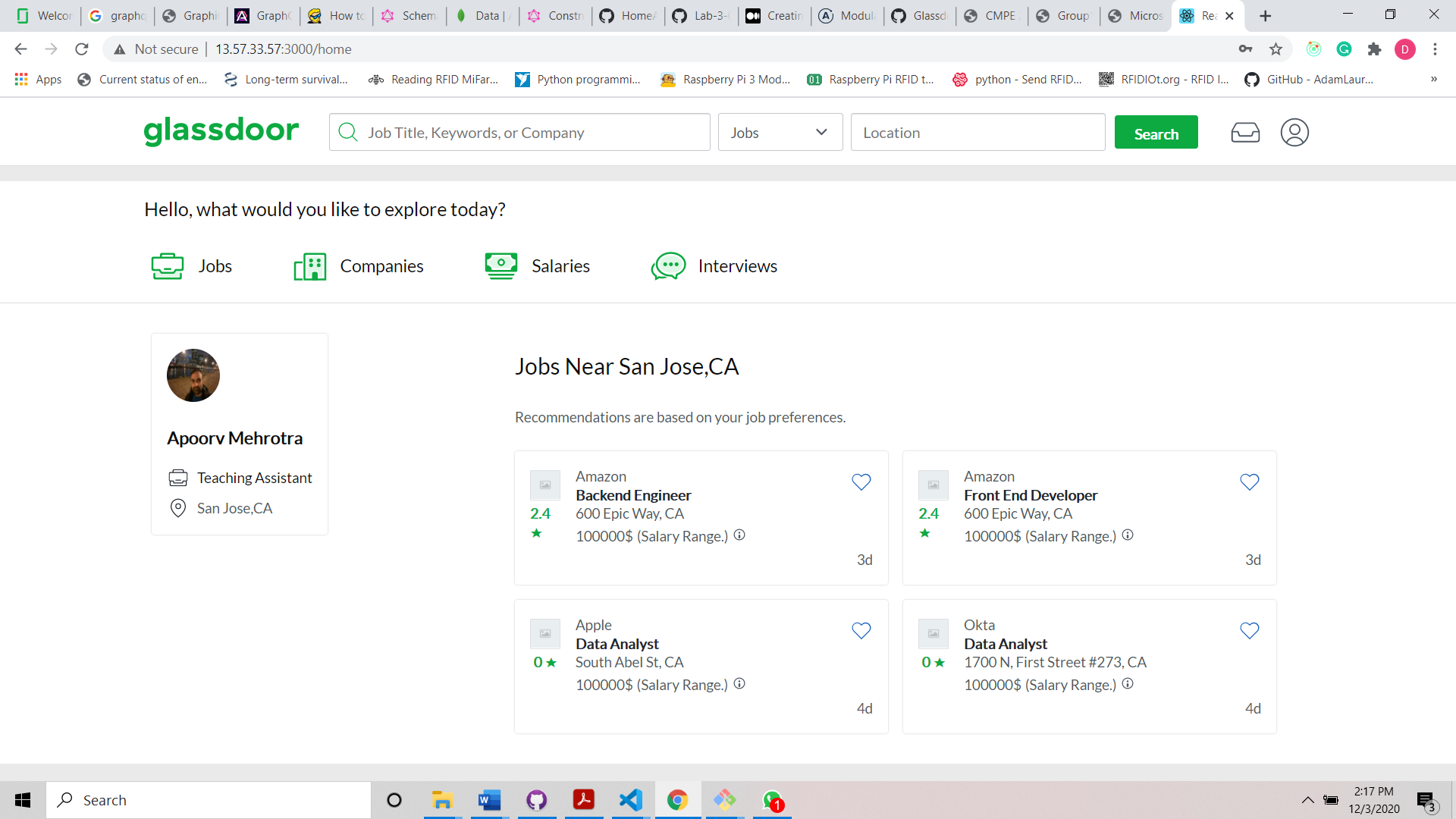
**Heavyweight Resource Handline**

**Database Decision Policy**

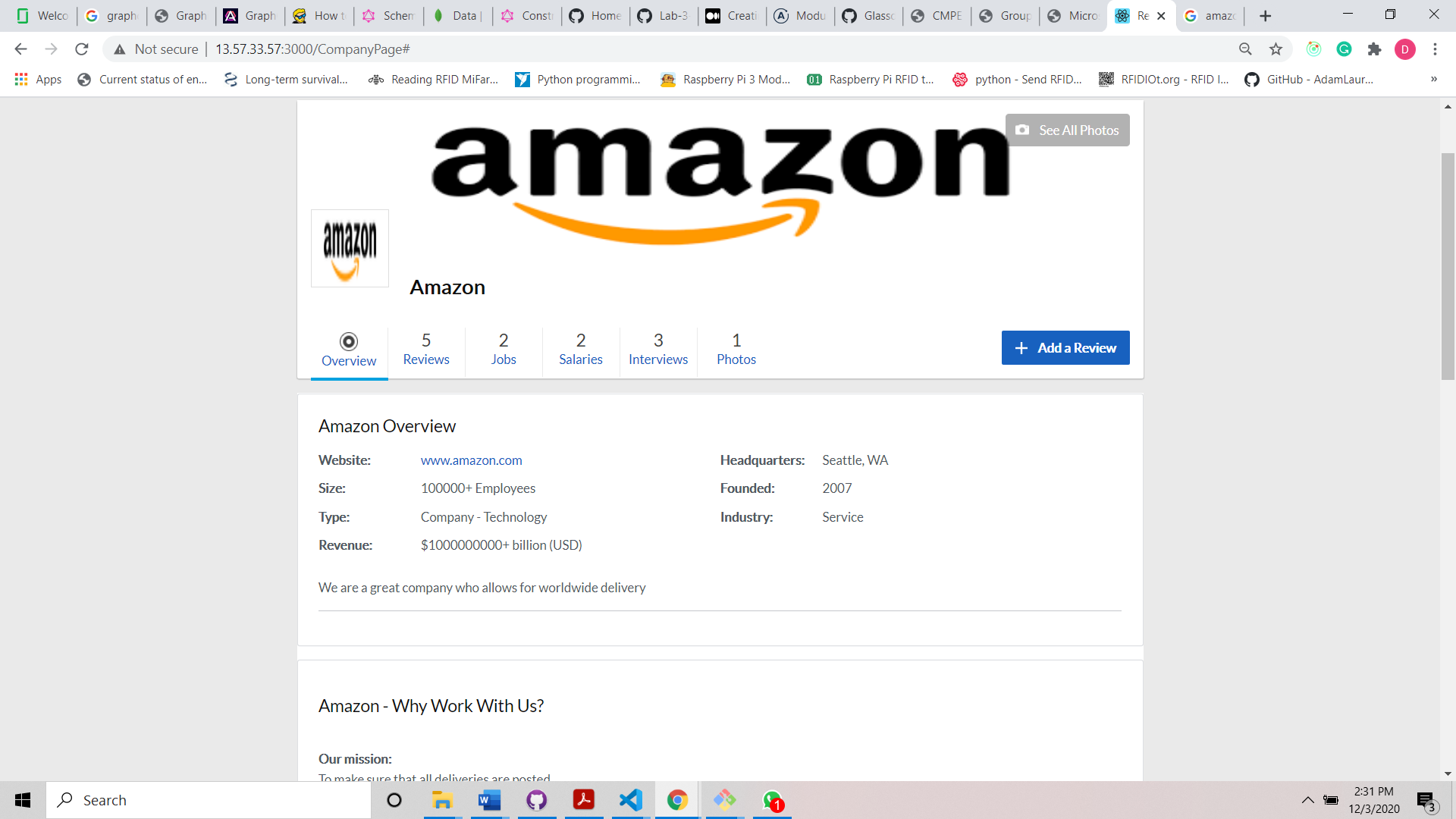
**Application Screen Captures**



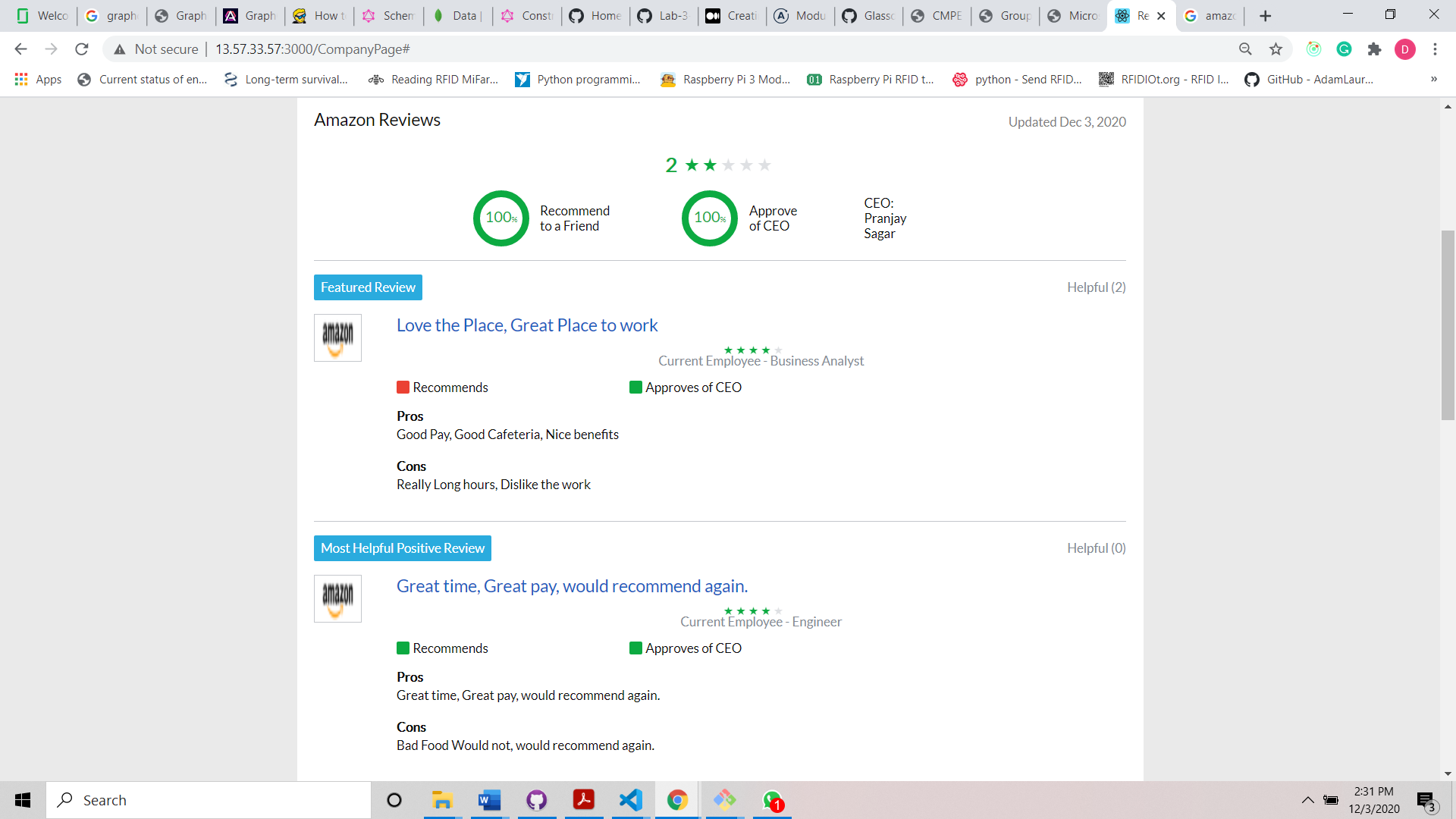
**Figure 4. Screen Capture of the login screen.**



**Figure 5. Screen Capture of the Main Screen After Student Login.**

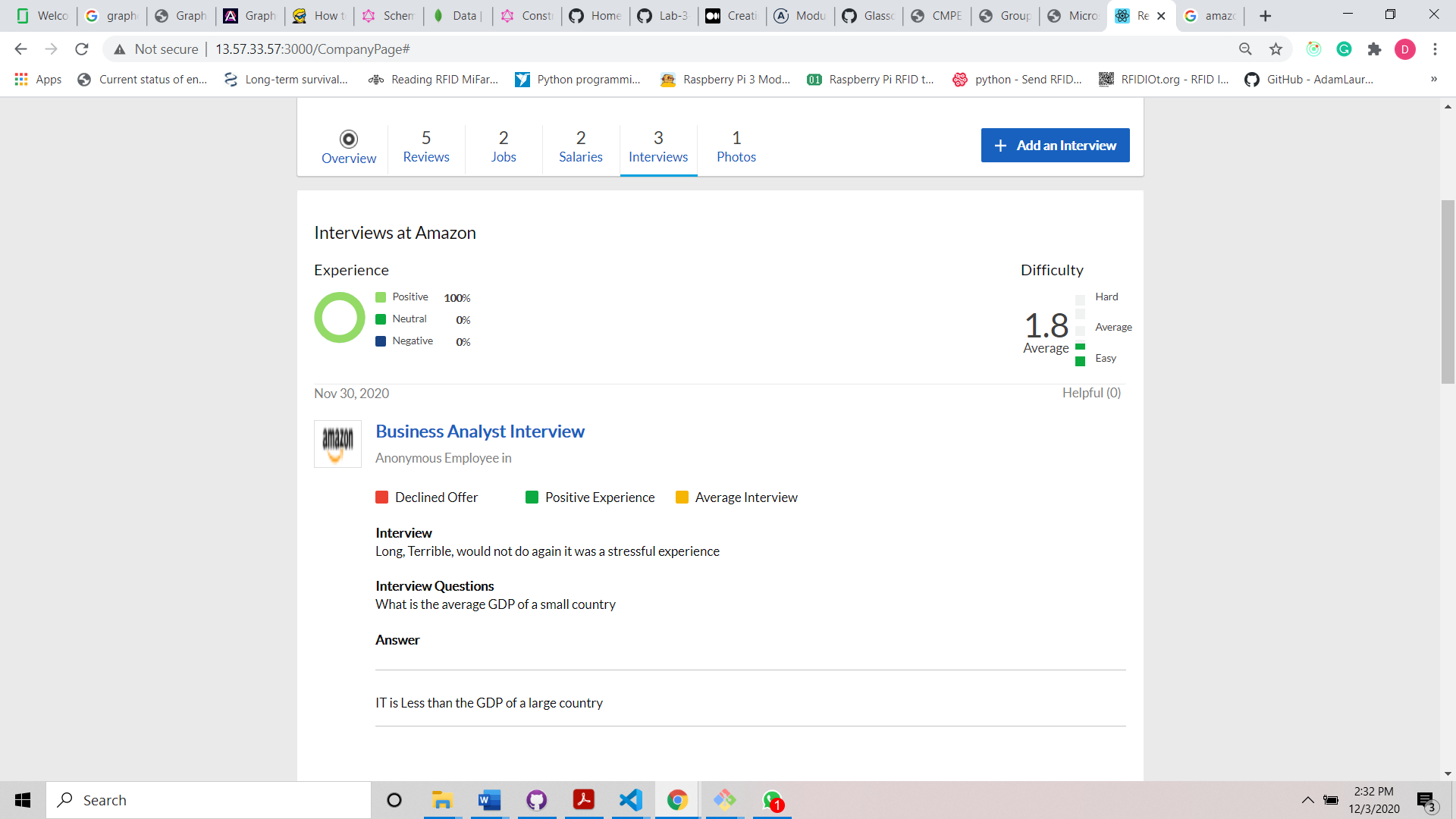


**Figure 6. Screen Capture of the Company Profile Page in the student persona.**

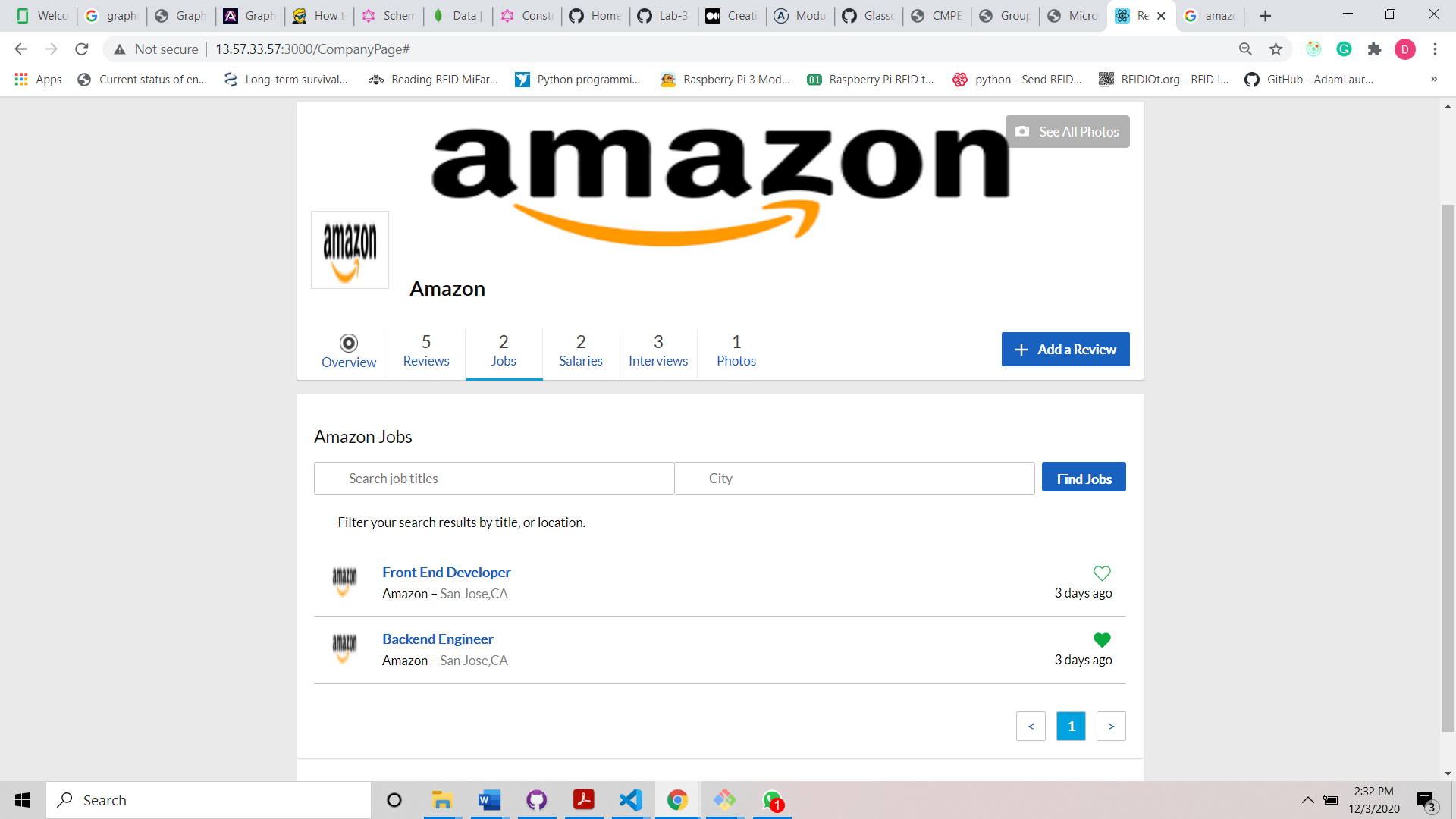


**Figure 7. Screen Capture of the Review Tab on the Company Profile Page**

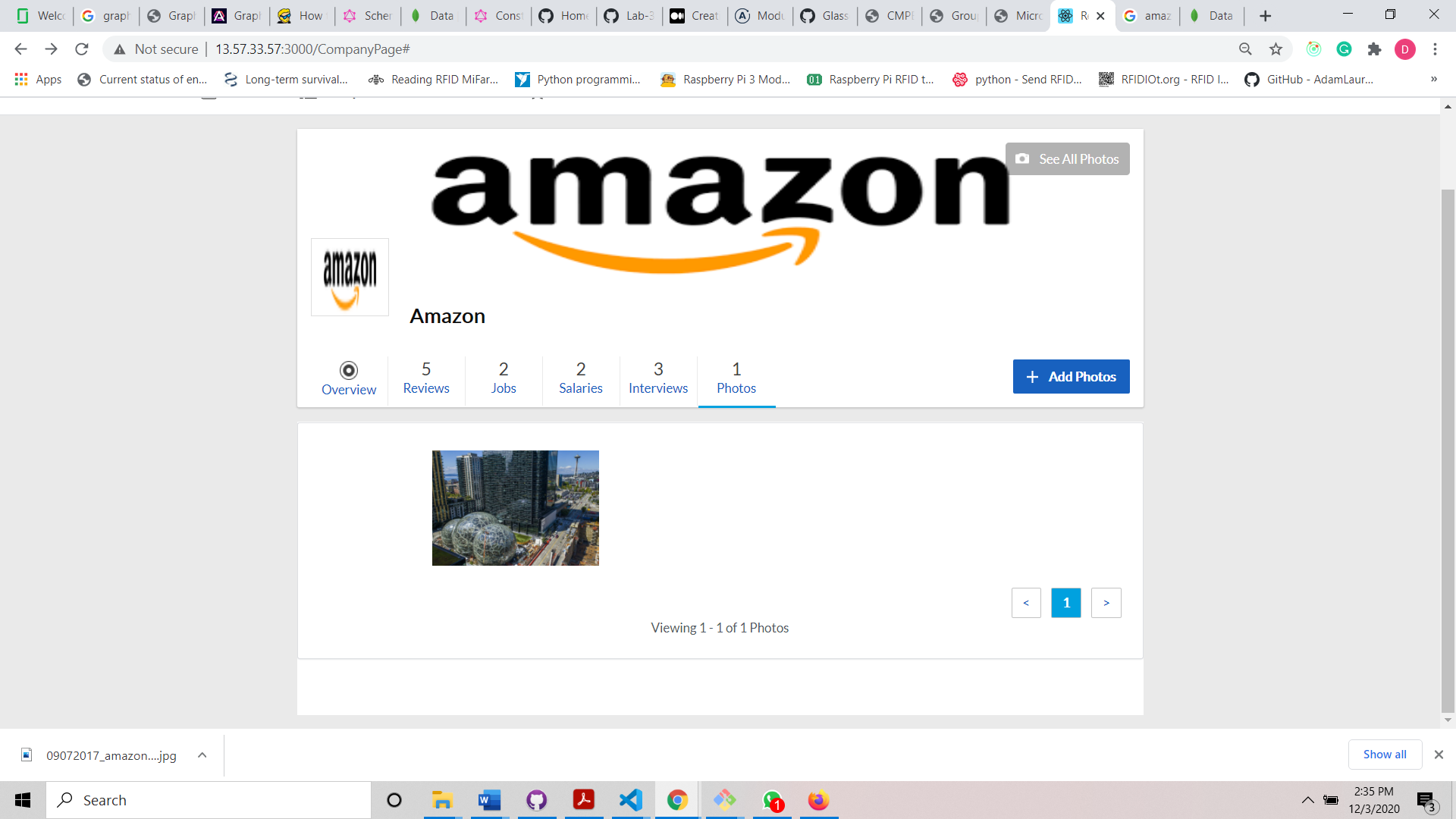
**Figure 8. Screen Capture of the Interview Tab on the Company Profile Page**

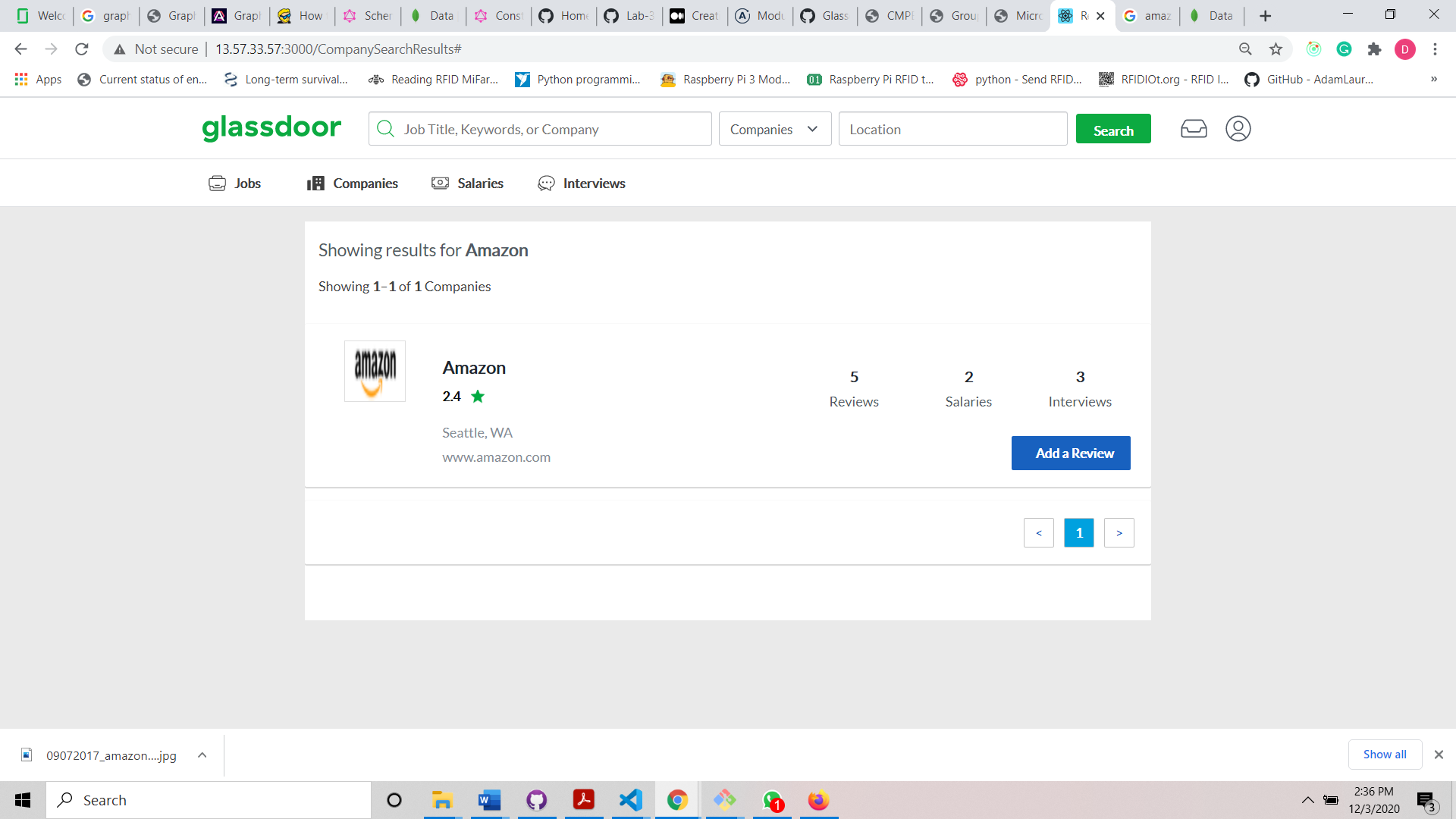


**Figure 9. Screen Capture of the Salary Tab on the Company Profile Page.**

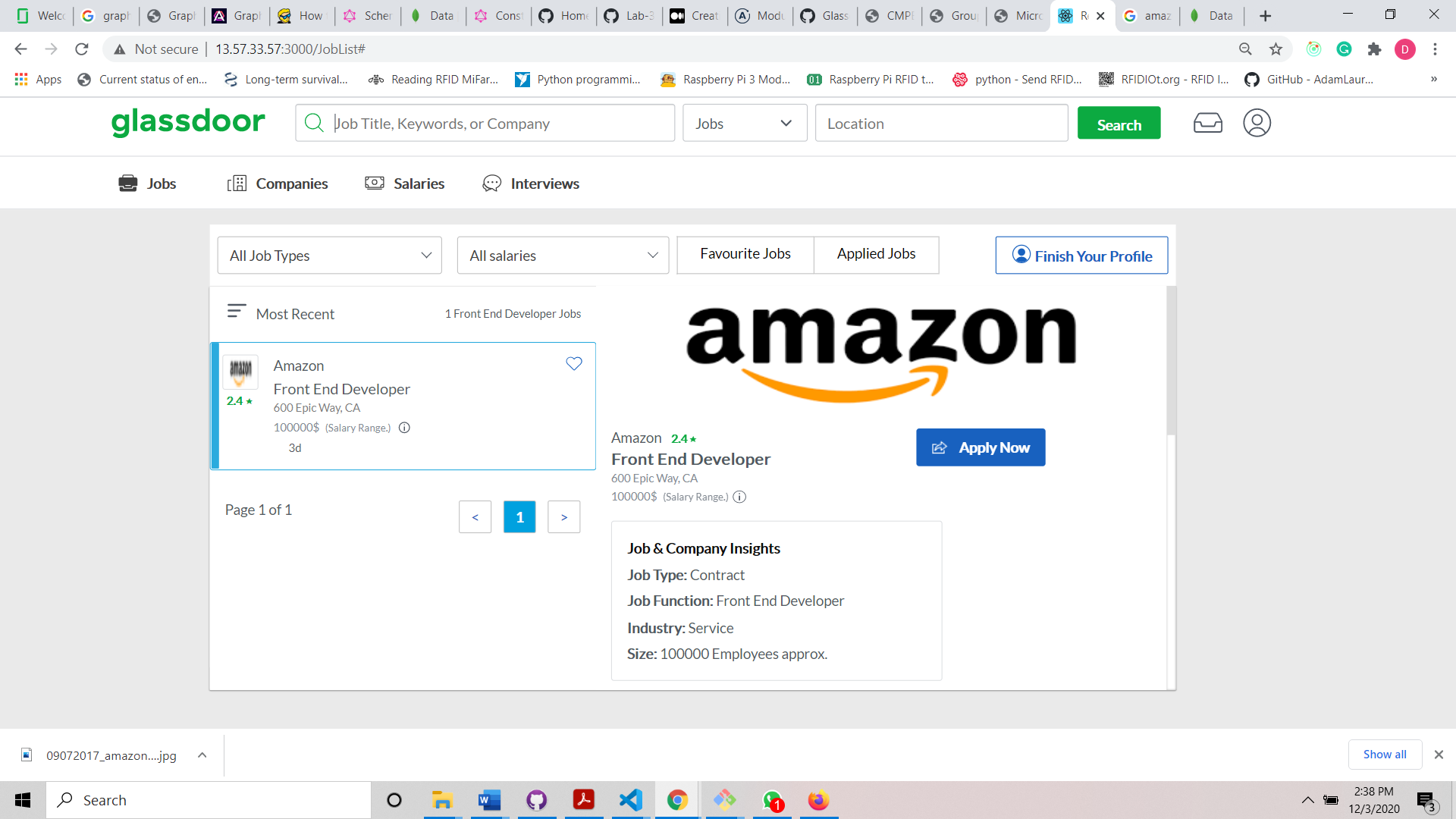


**Figure 10. Screen Capture of the Jobs Tab on the Company Profile**

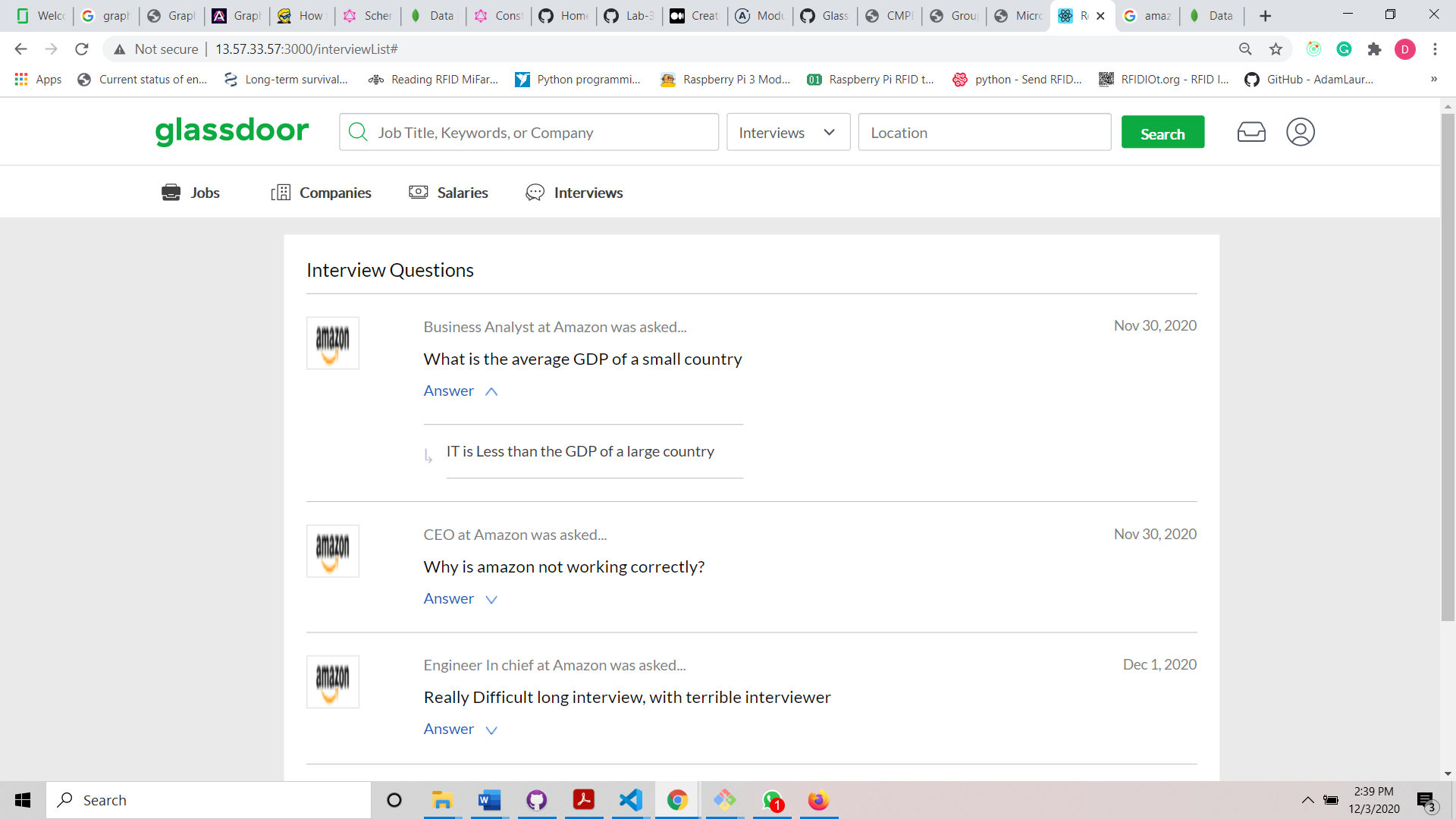
**Figure 11. Screen Capture of the Photos Tab on the Company Profile.**



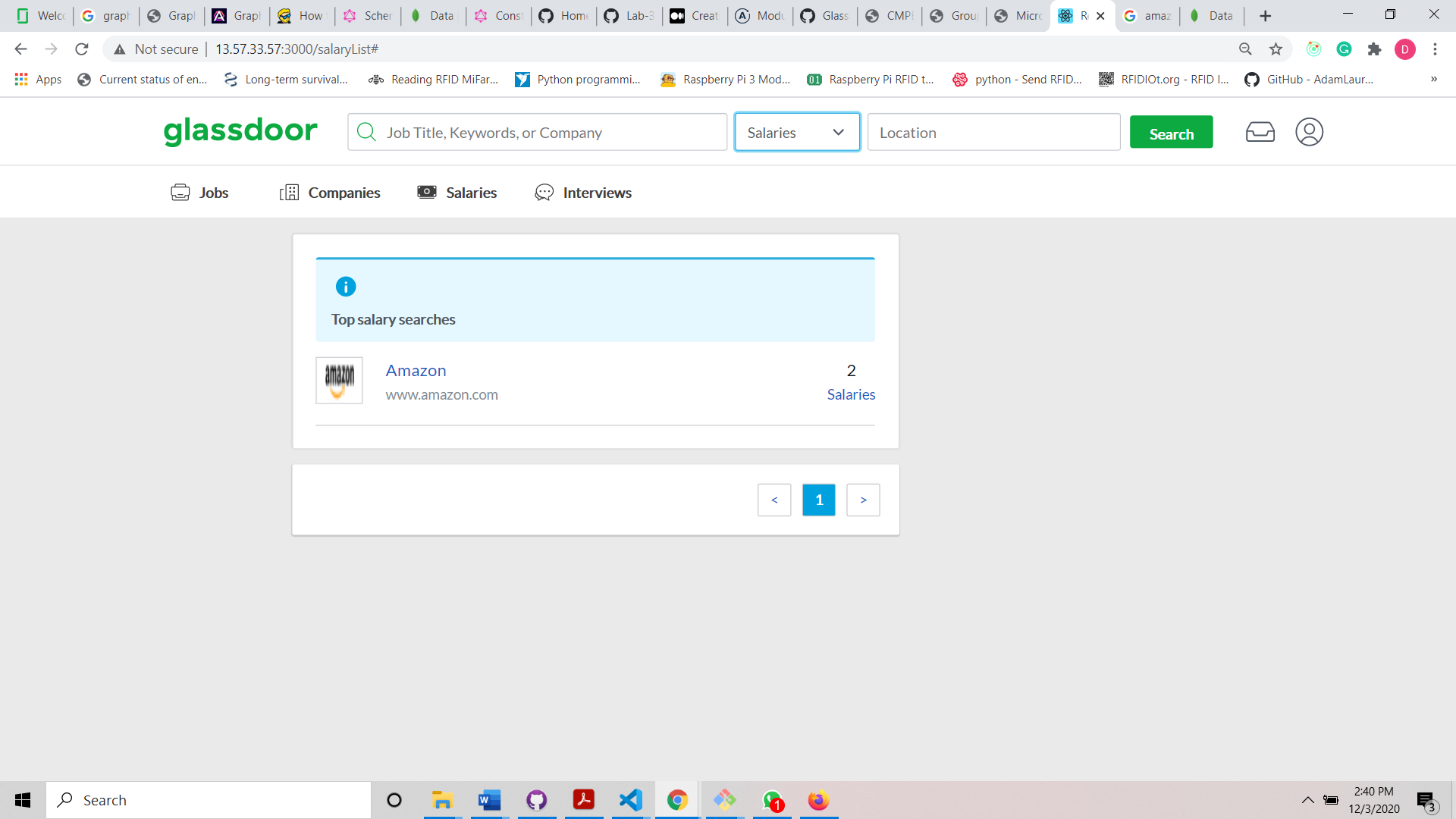
**Figure 13. Screen Capture of the Company Search Page.**



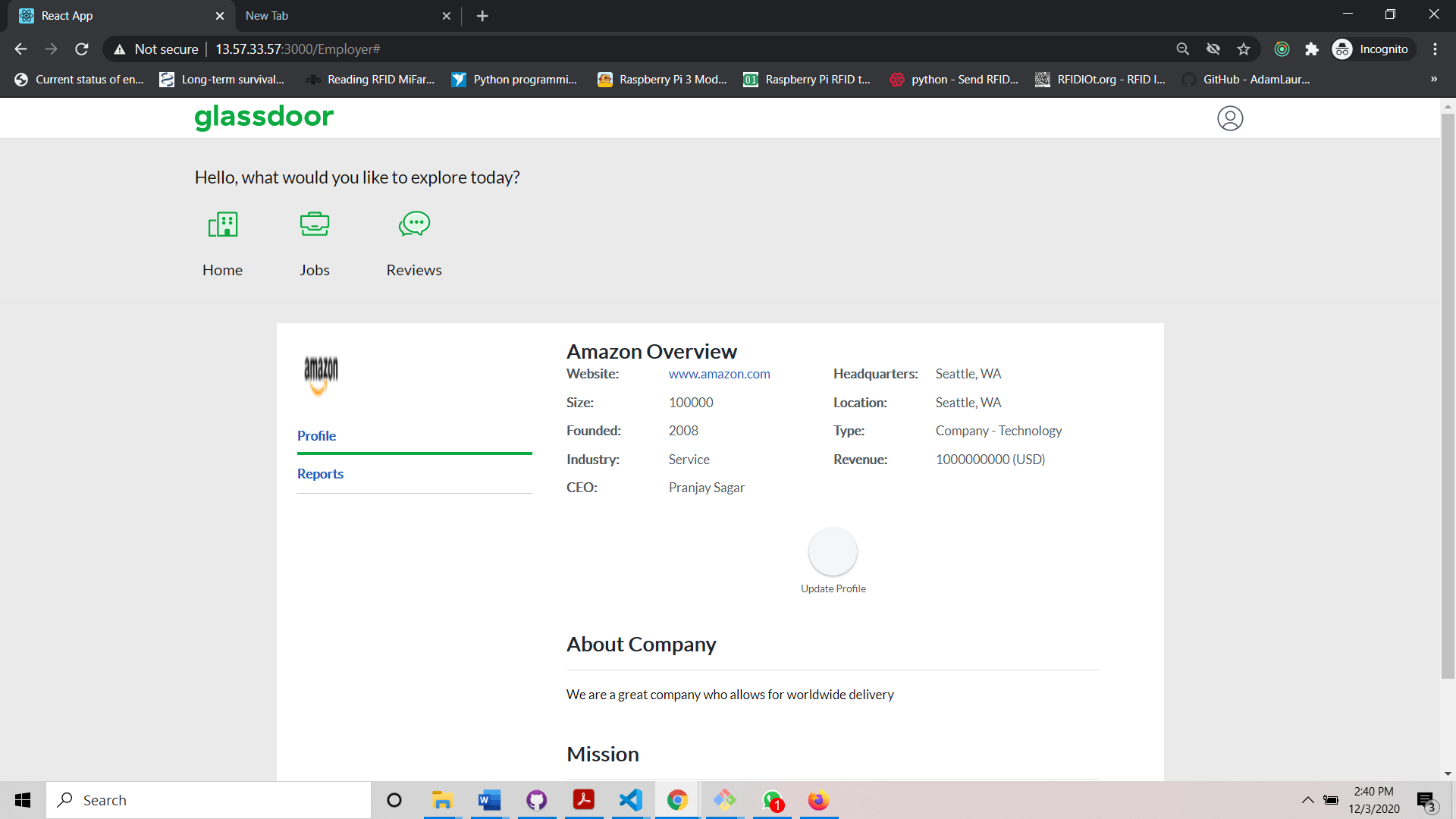
**Figure 14. Screen Capture of the Job Search Page.**



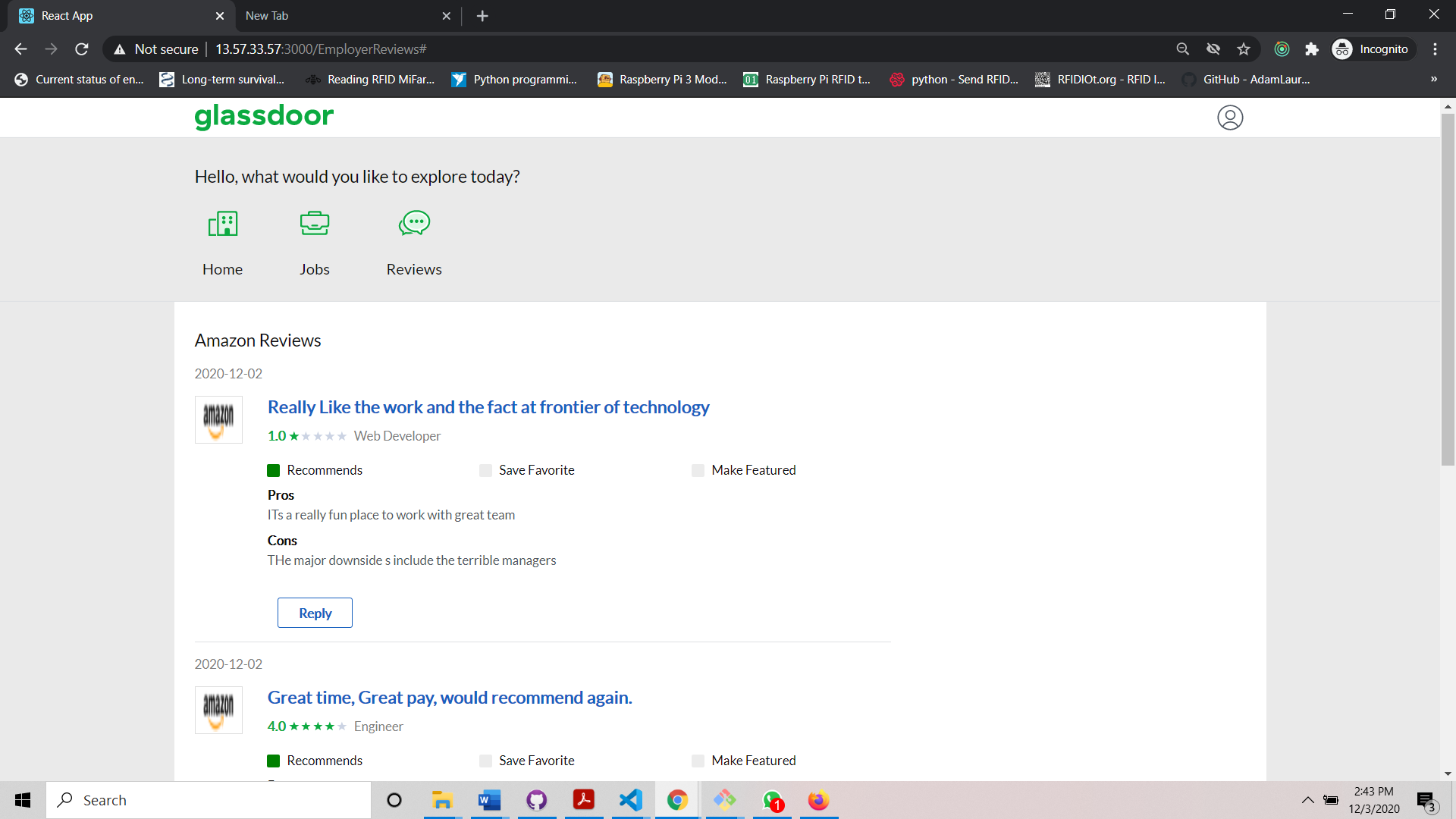
**Figure 15. Screen Capture of the Interview Search Page.**



**Figure 16. Screen Capture of the Salary Search Page.**



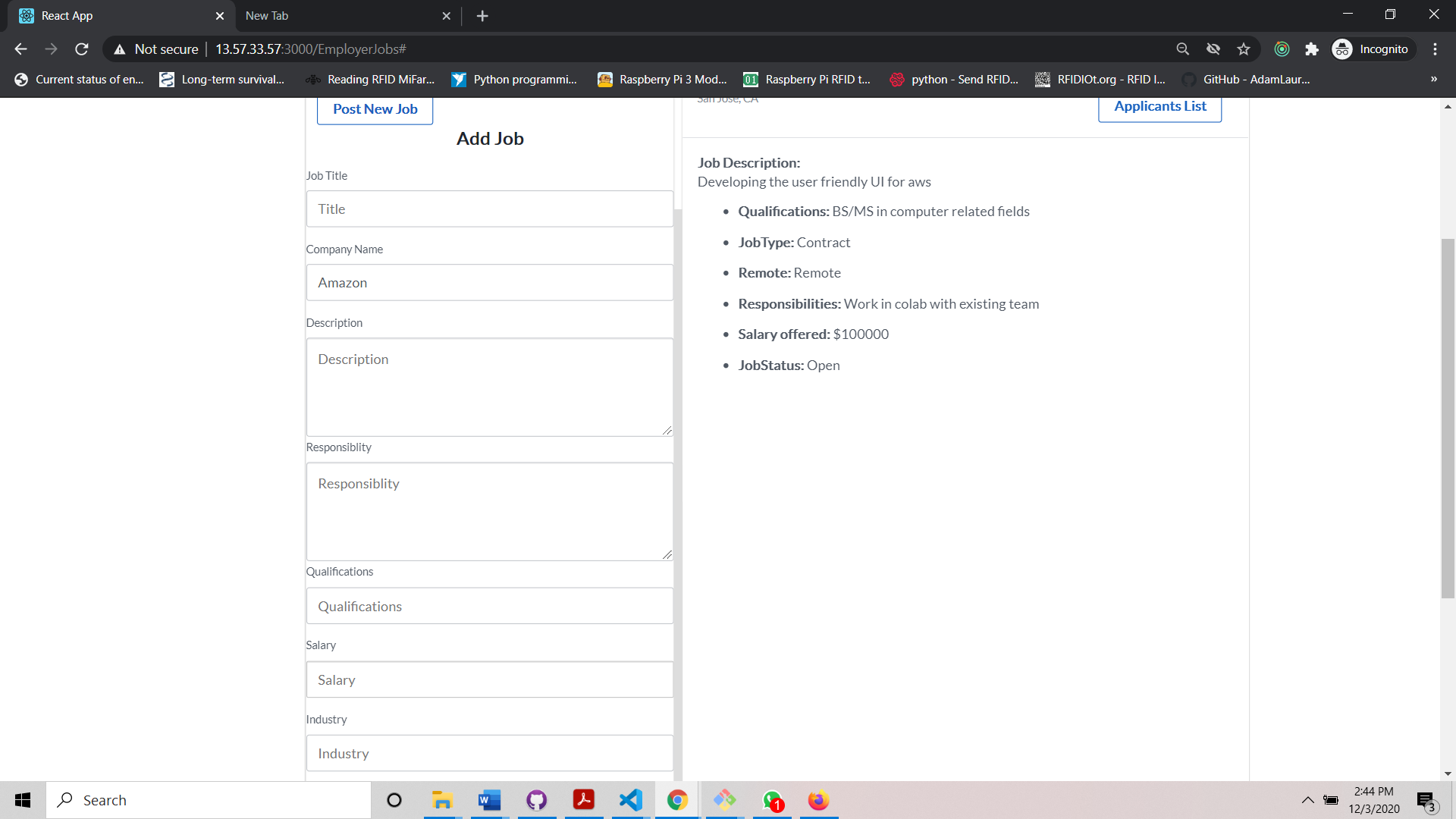
**Figure 17. Screen Capture of the Company Profile Page in the company persona.**



**Figure 18 Screen Capture of the Review Tab in the company persona.**



**Figure 19. Screen Capture of the Job Page in the company persona.**



**Figure 20. Screen Capture of the Job Posting modal in the company persona.**

**Figure 22. Screen Capture of the Admin Persona Home Page.**

**Figure 23. Screen Capture of the Approval Tab for the Review.**

**Figure 24 Screen Capture of the Approval of Photos in the admin persona.**

**Figure 25 Screen Capture of the Company Tab in the admin persona.**

**Code Listing of Entity Object**

const mongoose = require('mongoose');

const CompanySchema = new mongoose.Schema(

  {

    CompanyID: { type: String, required: true },

    CompanyName: { type: String },

    Website: { type: String },

    Size: { type: Number },

    ProfileImg: { type: String },

    Type: { type: String },

    Revenue: { type: String },

    Headquarter: { type: String },

    Industry: { type: String },

    Founded: { type: Date },

    CompanyDescription: { type: String },

    City: { type: String },

    State: { type: String },

    CompanyMission: { type: String },

    CEO: { type: String },

    ViewCount: { type: Number, default: 0 },

    approveCEOcount: { type: Number, default: 0 },

    recommendedcount: { type: Number, default: 0 },

    SalaryReviewCount: { type: Number, default: 0 },

    GeneralReviewCount: { type: Number, default: 0 },

    InterviewReviewCount: { type: Number, default: 0 },

    TotalGeneralReviewRating: { type: Number, default: 0 },

    JobCount: { type: Number, default: 0 },

    CoverPhoto: { type: String },

    PhotoCount: { type: Number, default: 0 },

    FeaturedReview: {

      ID: { type: Number },

      Status: {

        type: String,

      },

      CompanyName: { type: String },

      StudentID: { type: Number },

      Pros: { type: String },

      Cons: { type: String },

      Descriptions: { type: String },

      Rating: { type: Number },

      EmployeeStatus: {

        type: String,

        enum: ['Current', 'Former'],

      },

      Helpful: { type: Number },

      CEOApproval: { type: Boolean },

      JobType: {

        type: String,

      },

      Recommended: { type: Boolean },

      JobTitle: { type: String },

      Headline: { type: String },

      DatePosted: { type: Date },

      Response: { type: String },

      Favorite: { type: Boolean },

    },

    Photos: [String],

  },

  {

    versionKey: false,

  }

);

mongoose.set('useCreateIndex', true);

CompanySchema.index({ CompanyName: 1 }, { unique: true });

module.exports = mongoose.model('company', CompanySchema);

const mongoose = require('mongoose');

const StudentSchema = new mongoose.Schema({

  StudentID: { type: Number, required: true },

  Name: { type: String },

  ProfilePicURL: { type: String },

  Email: { type: String, required: true },

  PhoneNo: { type: Number },

  Website: { type: String },

  StreetAddress: { type: String },

  City: { type: String },

  State: { type: String },

  Country: { type: String },

  Zip: { type: Number, min: 10000, max: 99999 },

  AboutMe: { type: String },

  CurrentJobTitle: { type: String },

  Skills: [{ type: String }],

  ResumePrimary: { type: String },

  Resumes: [{ type: String }],

  FavouriteJobs: [{ type: String }],

  HelpfullGeneralReviews: [Number],

  HelpfullInterviewReviews: [Number],

  JobStatus: {

    type: String,

    enum: [

      'Select',

      'Not looking',

      'Not looking, but open',

      'Casually looking',

      'Actively looking',

    ],

    default: 'Select',

  },

  Ethnicity: {

    type: String,

    enum: [

      'Indigenous American or Alaska Native',

      'East Asian',

      'South Asian',

      'Southeast Asian',

      'Native Hawaiian or Other Pacific Islander',

      'Middle Eastern',

      'Black or African American',

      'Hispanic or Latinx',

      'White',

      'Prefer to Self Describe',

      'Prefer Not to Say',

    ],

    default: 'Prefer Not to Say',

  },

  JobType: [

    {

      type: String,

    },

  ],

  PreferredJobTitle: { type: String },

  TargetSalary: { type: Number },

  OpentoRelocation: { type: Boolean, default: false },

  WorkRemotely: { type: Boolean, default: false },

  Industry: { type: String },

  Race: [

    {

      type: String,

    },

  ],

  Gender: {

    type: String,

    enum: ['Male', 'Female', 'Non-Binary', 'Prefer Not to Say'],

    default: 'Prefer Not to Say',

  },

  AcceptedReviewCount: { type: Number },

  Disability: {

    type: String,

    enum: ['Yes', 'No', 'Prefer Not to Say'],

    default: 'Prefer Not to Say',

  },

  VeteranStatus: {

    type: String,

    enum: ['Yes', 'No', 'Prefer Not to Say'],

    default: 'Prefer Not to Say',

  },

  AppliedJobs: [String],

});

module.exports = mongoose.model('student', StudentSchema);

const mongoose = require('mongoose');

const StaticSchema = new mongoose.Schema({

  JobSearchDropDowns: [String],

  JobFilterInJobTab: [String],

  Ethnicity: [String],

  Gender: [String],

  State: [String],

  Status: [String],

  Country: [String],

  VeteranStatus: [String],

  Disability: [String],

  JobType: [String],

  reviews: [

    {

      Date: { type: Date },

      reviewcount: { type: Number },

    },

  ],

});

module.exports = mongoose.model('static\_data', StaticSchema);

const mongoose = require('mongoose');

const JobsSchema = new mongoose.Schema({

  JobID: { type: Number },

  Title: { type: String, required: true },

  CompanyID: { type: String, required: true },

  CompanyName: { type: String, required: true },

  CurrentStatus: {

    type: String,

    enum: ['Open', 'Close'],

  },

  Industry: { type: String },

  Remote: {

    type: String,

    enum: ['Remote', 'InPerson'],

  },

  StreetAddress: { type: String, required: true },

  City: { type: String, required: true },

  State: { type: String, required: true },

  Country: { type: String, required: true },

  Zip: {

    type: Number,

    min: 10000,

    max: 99999,

    required: true,

  },

  PostedDate: { type: Date, required: true },

  JobDescription: { type: String, required: true },

  Responsibilities: { type: String, required: true },

  Qualifications: { type: String, required: true },

  ExpectedSalary: { type: Number, required: true },

  Votes: { type: Number, required: true },

  JobType: {

    type: String,

    enum: ['Full-time', 'Part-time', 'Contract', 'Internship', 'Temporary'],

    required: true,

  },

});

module.exports = mongoose.model('jobs', JobsSchema);

const mongoose = require('mongoose');

const Photoschema = new mongoose.Schema({

  ID: { type: Number, required: true },

  CompanyID: { type: String, required: true },

  StudentID: { type: String, required: true },

  PhotoURL: { type: String, required: true },

  DateUploaded: { type: Date, required: true },

  CompanyName: { type: String, required: true },

  Status: {

    type: String,

    enum: ['Not Approved', 'Approved', 'Disapproved'],

    required: true,

  },

});

mongoose.set('useCreateIndex', true);

Photoschema.index({ ID: 1 }, { unique: true });

module.exports = mongoose.model('photo', Photoschema);

const mongoose = require('mongoose');

const Salaryschema = new mongoose.Schema({

  SalaryReviewID: { type: Number },

  CompanyID: { type: String, required: true },

  StudentID: { type: Number, required: true },

  CompanyName: { type: String, required: true },

  Status: {

    type: String,

    enum: ['Not Approved', 'Approved', 'Disapproved'],

    required: true,

  },

  DatePosted: { type: Date, required: true },

  BaseSalary: { type: Number, required: true },

  Bonuses: { type: Number, required: true },

  JobTitle: { type: String, required: true },

  Years: { type: Number, required: true },

  StreetAddress: { type: String, required: true },

  City: { type: String, required: true },

  State: { type: String, required: true },

  Zip: {

    type: Number, min: 10000, max: 99999, required: true,

  },

});

mongoose.set('useCreateIndex', true);

Salaryschema.index({ SalaryReviewID: 1 }, { unique: true });

module.exports = mongoose.model('salaryreview', Salaryschema);

const mongoose = require('mongoose');

const Interviewschema = new mongoose.Schema({

  InterviewReviewID: { type: Number },

  CompanyID: { type: String, required: true },

  StudentID: { type: Number, required: true },

  CompanyName: { type: String, required: true },

  Status: {

    type: String,

    enum: ['Not Approved', 'Approved', 'Disapproved'],

  },

  Helpful: { type: Number, default: 0, required: true },

  DatePosted: { type: Date, required: true },

  OverallExperience: {

    type: String,

    enum: ['Positive', 'Negative', 'Neutral'],

    required: true,

  },

  JobTitle: { type: String, required: true },

  Description: { type: String, required: true },

  Difficulty: {

    type: Number,

    required: true,

  },

  OfferStatus: {

    type: String,

    enum: ['No', 'Yes, but I declined', 'Yes, and I accepted'],

    required: true,

  },

  InterviewQuestions: { type: String, required: true },

  Answers: { type: String, required: true },

  StreetAddress: { type: String, required: true },

  City: { type: String, required: true },

  State: { type: String, required: true },

  Zip: {

    type: Number, min: 10000, max: 99999, required: true,

  },

});

mongoose.set('useCreateIndex', true);

Interviewschema.index({ InterviewReviewID: 1 }, { unique: true });

module.exports = mongoose.model('interviewreview', Interviewschema);

const mongoose = require('mongoose');

const ReviewSchema = new mongoose.Schema({

  ID: { type: Number, required: true },

  CompanyID: { type: String, required: true },

  StudentID: { type: String, required: true },

  CompanyName: { type: String, required: true },

  Pros: { type: String, required: true },

  Cons: { type: String, required: true },

  Description: { type: String, required: true },

  Rating: { type: Number, required: true },

  EmployeeStatus: {

    type: String,

    enum: ['Current', 'Former'],

    required: true,

  },

  Status: {

    type: String,

    enum: ['Not Approved', 'Approved', 'Disapproved'],

    required: true,

  },

  Helpful: { type: Number, required: true, default: 0 },

  CEOApproval: { type: Boolean, required: true },

  JobType: {

    type: String,

    enum: ['FullTime', 'PartTime', 'Contract', 'Intern', 'Freelance'],

    required: true,

  },

  Recommended: { type: Boolean, required: true },

  JobTitle: { type: String, required: true },

  Headline: { type: String, required: true },

  DatePosted: { type: Date, required: true },

  Response: { type: String },

  Favorite: { type: Number, required: true },

});

mongoose.set('useCreateIndex', true);

ReviewSchema.index({ ID: 1 }, { unique: true });

module.exports = mongoose.model('generalreview', ReviewSchema);

CREATE SCHEMA `glassdoor-prototype` ;

use `glassdoor-prototype`;

CREATE TABLE `SIGNUP` (

  `UserID` bigint NOT NULL AUTO\_INCREMENT,

  `UserName` varchar(60)  unique NOT NULL ,

  `Password` varchar(150) NOT NULL,

  `Role` enum('student','company','admin') NOT NULL,

  PRIMARY KEY (`UserID`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `APPLICATION\_JOB` (

  `JobID` bigint NOT NULL auto\_increment,

  `CompanyName` varchar(60) NOT NULL,

  `CompanyID` bigint NOT NULL,

  `PostedDate` date NOT NULL,

  `StreetAddress` varchar(45) NOT NULL,

  `City` varchar(45) NOT NULL,

  `State` varchar(45) NOT NULL,

  PRIMARY KEY (`JobID`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `APPLICATION\_RECEIVED` (

  `JobID` bigint NOT NULL ,

  `StudentID` bigint NOT NULL,

  `StudentName` varchar(45) NOT NULL,

  `ResumeURL` varchar(150) DEFAULT NULL,

  `CoverLetterURL` varchar(150) DEFAULT NULL,

  `Status` enum('Submitted','Reviewed','Initial Screening','Interviewing','Hired','Rejected') DEFAULT 'Submitted',

  `Withdrawn` tinyint DEFAULT NULL,

  PRIMARY KEY (`JobID`,`StudentID`),

  FOREIGN KEY (`JobID`) REFERENCES `APPLICATION\_JOB`(`JobID`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

**Code Listing of Security and Session Object**

**Code Listing of main server code**

**Frontend**

import React, { Component } from 'react';

import { Route, Switch } from 'react-router-dom';

import axios from 'axios';

import { connect } from 'react-redux';

import { updateMasterData } from '../constants/action-types';

import serverUrl from '../config';

// import Home from './Student/LandingPage/Home';

import Login from './Login/Login';

import CompanySearchResults from './Student/CompanySearchResults/CompanySearchResults';

import JobList from './Student/JobSearchResults/JobList';

import interviewList from './Student/InterviewSearchResults/interviewList';

import EmployerHome from './Employer/LandingPage/EmployerHome';

import ProfileUpdate from './Employer/ProfileUpdate/ProfileUpdate';

import EmployerReviews from './Employer/Reviews/ReviewsHome';

import EmployerJobs from './Employer/Jobs/JobsHome';

import EmployerReport from './Employer/Report/ReportHome';

import ApplicantProfile from './Employer/ApplicantProfile/ApplicantProfileHome';

import salaryList from './Student/SalarySearchResults/salaryList';

import Home from './Student/LandingPage/Home';

import Profile from './Student/PersonalInformation/Profile/Profile';

import ResumeUploadPage from './Student/PersonalInformation/ResumePage/ResumeUploadPage';

// import Navbar from './Student/Common/Navbar';

import CommonNavBar from './CommonNavBar';

import ContributionPage from './Student/Contributions/ContributionPage';

import CommonContribute from './Student/Contributions/CommonContribute';

import CompanyPage from './Student/CompanyProfile/CompanyPage';

import ReviewForm from './Student/CompanyProfile/CompanyReviews/ReviewForm';

import InterviewForm from './Student/CompanyProfile/CompanyInterviews/InterviewForm';

import PhotoUploadForm from './Student/CompanyProfile/CompanyPhotos/PhotoUploadForm';

import JobApplicationPage from './Student/CompanyProfile/CompanyJobs/JobApplicationPage';

import SalaryForm from './Student/CompanyProfile/CompanySalaries/SalaryForm';

import AdminHomePage from './Admin/LandingPage/AdminHomePage';

import CompanySearchResultsAdmin from './Admin/CompanySearchResultsAdmin/CompanySearchResultsAdmin';

import InterviewListAdmin from './Admin/InterviewSearchResultsAdmin/InterviewListAdmin';

import CompanyGeneralReviewsAdmin from './Admin/CompanyGeneralReviewsAdmin/CompanyGeneralReviewsAdmin';

import CompanyPhotosAdmin from './Admin/CompanySearchPhotosAdmin/CompanyPhotosAdmin';

import SalaryListAdmin from './Admin/CompanySalaryReviewsAdmin/SalaryListAdmin';

import CompanyPageAdminView from './Admin/CompanyProfile/CompanyPageAdminView';

import PostJob from './Employer/Jobs/PostJobModal';

class Main extends Component {

  componentDidMount() {

    axios.get(serverUrl + 'glassdoor/staticdata').then((response) => {

      let Countries = response.data[0].Country.map((country) => {

        return country;

      });

      let Gender = response.data[0].Gender.map((gender) => {

        return gender;

      });

      let VeteranStatus = response.data[0].VeteranStatus.map((veteranStatus) => {

        return veteranStatus;

      });

      let Disability = response.data[0].Disability.map((disability) => {

        return disability;

      });

      let States = response.data[0].State.map((state) => {

        return state;

      });

      let Status = response.data[0].Status.map((status) => {

        return status;

      });

      let JobType = response.data[0].JobType.map((jobType) => {

        return jobType;

      });

      let Ethnicity = response.data[0].Ethnicity.map((ethnicity) => {

        return ethnicity;

      });

      let payload = {

        Countries,

        Gender,

        VeteranStatus,

        Disability,

        States,

        Status,

        JobType,

        Ethnicity,

      };

      this.props.updateMasterData(payload);

    });

  }

  render() {

    return (

      <div>

        {/\* <Switch>\*/}

        <Route path="/" component={CommonNavBar} />

        <Route path="/ReviewForm" component={ReviewForm} />

        <Route path="/InterviewForm" component={InterviewForm} />

        <Route path="/PhotoUploadForm" component={PhotoUploadForm} />

        <Route path="/JobApplicationPage" component={JobApplicationPage} />

        <Route path="/SalaryForm" component={SalaryForm} />

        <Route path="/Login" component={Login} />

        <Route path="/CompanySearchResults" component={CompanySearchResults} />

        <Route path="/Employer" component={EmployerHome} />

        <Route path="/EmployerProfile" component={ProfileUpdate} />

        <Route path="/EmployerReviews" component={EmployerReviews} />

        <Route path="/EmployerJobs" component={EmployerJobs} />

        <Route path="/PostJob" component={PostJob} />

        <Route path="/EmployerReport" component={EmployerReport} />

        <Route path="/ApplicantProfile" component={ApplicantProfile} />

        <Route path="/Home" component={Home} />

        <Route path="/JobList" component={JobList} />

        <Route path="/salaryList" component={salaryList} />

        <Route path="/interviewList" component={interviewList} />

        <Route path="/Profile" component={Profile} />

        <Route path="/ResumeUploadPage" component={ResumeUploadPage} />

        <Route path="/ContributionPage" component={ContributionPage} />

        <Route path="/CompanyPage" component={CompanyPage} />

        <Route path="/AdminHomePage" component={AdminHomePage} />

        <Route path="/CompanySearchResultsAdmin" component={CompanySearchResultsAdmin} />

        <Route path="/InterviewListAdmin" component={InterviewListAdmin} />

        <Route path="/CompanyGeneralReviewsAdmin" component={CompanyGeneralReviewsAdmin} />

        <Route path="/CompanyPhotosAdmin" component={CompanyPhotosAdmin} />

        <Route path="/SalaryListAdmin" component={SalaryListAdmin} />

        <Route path="/CompanyPageAdminView" component={CompanyPageAdminView} />

        {/\*</Switch>\*/}

      </div>

    );

  }

}

// export default Main;

const mapDispatchToProps = (dispatch) => {

  return {

    updateMasterData: (payload) => {

      dispatch({

        type: updateMasterData,

        payload,

      });

    },

  };

};

export default connect(null, mapDispatchToProps)(Main);

**Backend**

if (process.env.NODE\_ENV !== 'production') {

  // eslint-disable-next-line global-require

  require('dotenv').config();

}

const express = require('express');

const session = require('express-session');

const cookieParser = require('cookie-parser');

const bodyParser = require('body-parser');

const cors = require('cors');

const { frontendURL } = require('./config');

const commonPart = require('./Routes/headingRoutes');

const companyRoute = require('./Routes/companyRoutes');

const studentRoute = require('./Routes/studentRoutes');

const adminRoute = require('./Routes/adminRoutes');

const Company = require('./model/Company');

const app = express();

app.use(cors({ origin: frontendURL, credentials: true }));

// use express session to maintain session data

app.use(

  session({

    secret: process.env.SESSION\_SECRET,

    resave: false,

    saveUninitialized: true,

    duration: 60 \* 60 \* 1000,

    activeDuration: 5 \* 60 \* 1000,

  })

);

app.use(cookieParser());

app.use(bodyParser.json());

app.use(bodyParser.urlencoded({ extended: true }));

// eslint-disable-next-line func-names

app.use(function (req, res, next) {

  res.setHeader('Access-Control-Allow-Origin', frontendURL);

  res.setHeader('Access-Control-Allow-Credentials', 'true');

  res.setHeader('Access-Control-Allow-Methods', 'GET,HEAD,POST,PUT,DELETE,OPTIONS');

  res.setHeader(

    'Access-Control-Allow-Headers',

    'Access-Control-Allow-Headers, Authorization, Origin,Accept, X-Requested-With, Content-Type, Access-Control-Request-Method, Access-Control-Request-Headers'

  );

  res.setHeader('Cache-Control', 'no-cache');

  next();

});

app.use('/glassdoor', commonPart);

app.use('/company', companyRoute);

app.use('/student', studentRoute);

app.use('/admin', adminRoute);

app.listen(3001);

**Kafka Backend**

/\* eslint-disable no-shadow \*/

/\* eslint-disable no-new-require \*/

/\* eslint-disable new-cap \*/

/\* eslint-disable no-console \*/

/\* eslint-disable camelcase \*/

const connection = new require('./kafka/Connection');

const mongoose = require('mongoose');

const { mongoDB } = require('./config');

const company = require('./functionality/companyHandling');

const student = require('./functionality/studenthandling');

const general = require('./functionality/generalHandling');

const admin = require('./functionality/adminHandling');

const Companys = require('./model/Company');

const options = {

  useNewUrlParser: true,

  useUnifiedTopology: true,

  poolSize: 500,

  bufferMaxEntries: 0,

  useFindAndModify: false,

};

// eslint-disable-next-line no-unused-vars

mongoose.connect(mongoDB, options, (err, res) => {

  if (err) {

    // eslint-disable-next-line no-console

    console.log('MongoDB connection Failed', err);

  } else {

    // eslint-disable-next-line no-console

    console.log('MongoDB Connected');

  }

});

cron.schedule('0 0 \* \* \*', async () => {

  try {

    const ViewCount = 0;

    await Companys.updateMany({}, { ViewCount });

    console.log('Updated');

  } catch (error) {

    console.log('did not update');

  }

});

function handleTopicRequest(topic\_name, fname) {

  // var topic\_name = 'root\_topic';

  const consumer = connection.getConsumer(topic\_name);

  const producer = connection.getProducer();

  console.log('server is running ');

  consumer.on('message', (message) => {

    console.log(`message received for ${topic\_name} `, fname);

    console.log(JSON.stringify(message.value));

    const data = JSON.parse(message.value);

    fname.handle\_request(data.data, (err, res) => {

      // console.log(`after handle${res}`);

      const payloads = [

        {

          topic: data.replyTo,

          messages: JSON.stringify({

            correlationId: data.correlationId,

            data: res,

          }),

          partition: 0,

        },

      ];

      producer.send(payloads, (err, out) => {

        // console.log(out);

      });

    });

  });

}

// Add your TOPICs here

// first argument is topic name

// second argument is a function that will handle this topic request

handleTopicRequest('company2', company);

handleTopicRequest('student2', student);

handleTopicRequest('general2', general);

handleTopicRequest('admin2', admin);

**Code Listing of database access.**

**MYSQL database access**

const mysql = require('mysql2/promise');

/\*\* RDS DB \*/

const host = process.env.MYSQL\_HOST;

const user = process.env.MYSQL\_USER;

const password = process.env.MYSQL\_PASSWORD;

const db = process.env.MYSQL\_DB;

const port = process.env.MYSQL\_PORT;

const mysqlConnection = async () => {

  return mysql.createConnection({

    host,

    user,

    password,

    database: db,

    multipleStatements: true,

    port,

  });

};

module.exports = mysqlConnection;

**MongoDB**

const { mongoDB } = require('./config');

const options = {

  useNewUrlParser: true,

  useUnifiedTopology: true,

  poolSize: 500,

  bufferMaxEntries: 0,

  useFindAndModify: false,

};

// eslint-disable-next-line no-unused-vars

mongoose.connect(mongoDB, options, (err, res) => {

  if (err) {

    // eslint-disable-next-line no-console

    console.log('MongoDB connection Failed', err);

  } else {

    // eslint-disable-next-line no-console

    console.log('MongoDB Connected');

  }

});

**Redis**

const redis = require('redis');

const { redisPort, redisHost, password } = require('./redis-config');

const redisClient = redis.createClient({ port: redisPort, host: redisHost, password });

redisClient.on('connect', (err) => {

  if (err) {

    console.log('Error while connecting to Redis server');

  } else {

    console.log('Redis Server Connected');

  }

});

module.exports = redisClient;

**Code listing for Mocha Testing and output results**

const chai = require('chai');

const chaiHttp = require('chai-http');

chai.use(chaiHttp);

const apiHost = 'http://localhost';

const apiPort = '3001';

const apiUrl = `${apiHost}:${apiPort}`;

const { expect } = chai;

const studentToken =

  'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2wiOiJzdHVkZW50IiwiTmFtZSI6InByYW5qYXlAZ21haWwuY29tIiwiSUQiOjEsImlhdCI6MTYwNTIxODMwMywiZXhwIjoxNjA2MjI2MzAzfQ.a4dbsIVdjGCBK6LHn2v5gqVb8CvFO8WZRnXjKAMTlF4';

const companyToken =

  'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2wiOiJjb21wYW55IiwiTmFtZSI6ImRpc2hhbnQuc2hhaEBnbWFpbC5jb20iLCJJRCI6NSwiaWF0IjoxNjA1MjI1NDk2LCJleHAiOjE2MDYyMzM0OTZ9.PtRVK9VIbDVNVgjKZ0oEVnCAepD6GMjM2bpOSzDoCYc';

// Fetching static data for drop-downs (Done)

it('Test Fetching data for all drop-downs', function (done) {

  chai

    .request(apiUrl)

    .get('/glassdoor/staticData')

    .send()

    .end(function (err, res) {

      expect(res).to.have.status(200);

      expect(res.text[3]).equal('J');

      done();

    });

});

// Signup for Student (Done)

it('Testing of student signup', function (done) {

  chai

    .request(apiUrl)

    .post('/glassdoor/signup')

    .send({

      UserName: 'MochaStudent@test.com',

      Password: 'test',

      Role: 'student',

    })

    .end(function (err, res) {

      expect(res).to.have.status(201);

      expect(res.text).to.equal(JSON.stringify('Profile Created'));

      done();

    });

});

// Valid login

it('Testing of valid student Login', function (done) {

  chai

    .request(apiUrl)

    .post('/glassdoor/login')

    .send({

      UserName: 'pranjay@gmail.com',

      Password: 'pranjay01',

    })

    .end(function (err, res) {

      expect(res).to.have.status(200);

      done();

    });

});

// Invalid login

it('Testing of invalid student Login', function (done) {

  chai

    .request(apiUrl)

    .post('/glassdoor/login')

    .send({

      UserName: 'pranjay@gmail.com',

      Password: 'pranjay02',

    })

    .end(function (err, res) {

      expect(res).to.have.status(400);

      expect(res.text).equal(JSON.stringify('Invalid Credentials'));

      done();

    });

});

// Fetching Nav Bar data and Student Profile

it('Test fetching navbar and student profile', function (done) {

  chai

    .request(apiUrl)

    .get('/student/navbar')

    .set({ Authorization: `JWT ${studentToken}` })

    .query({ StudentID: '1' })

    .send()

    .end(function (err, res) {

      expect(res).to.have.status(200);

      expect(res.body[0].Email).equal('pranjay@gmail.com');

      expect(res.body.length).equal(3);

      done();

    });

});

// Fetching results of company search

it('Test to fetch results of company search', function (done) {

  chai

    .request(apiUrl)

    .get('/student/searchCompany')

    .set({ Authorization: `JWT ${studentToken}` })

    .query({ SearchString: 'Amazon', State: 'CA', PageNo: 0 })

    .send()

    .end(function (err, res) {

      expect(res).to.have.status(200);

      expect(res.body[0][0].State).equal('CA');

      expect(res.body.length).equal(3);

      done();

    });

});

// Fetching results of company search

it('Test to fetch results of job search', function (done) {

  chai

    .request(apiUrl)

    .get('/student/searchJob')

    .set({ Authorization: `JWT ${studentToken}` })

    .query({

      SearchString: 'Blue',

      JobType: 'Full-time',

      State: 'CA',

      SalStart: 0,

      SalEnd: 20000000,

      PageNo: 0,

    })

    .send()

    .end(function (err, res) {

      expect(res).to.have.status(200);

      expect(res.body.jobs[0].State).equal('CA');

      done();

    });

});

// Test to fetch profile of a company

it('Test to fetch profile of a company', function (done) {

  chai

    .request(apiUrl)

    .get('/company/profile')

    .set({ Authorization: `JWT ${companyToken}` })

    .query({

      CompanyID: '5',

    })

    .send()

    .end(function (err, res) {

      expect(res).to.have.status(200);

      expect(res.text[0]).equal('{');

      done();

    });

});

// Updating company Profile

it('Test to update comapny profile', function (done) {

  chai

    .request(apiUrl)

    .post('/company/profileupdate')

    .set({ Authorization: `JWT ${companyToken}` })

    .send({

      CompanyID: '5',

      Website: 'www.apoorv.com',

      Size: 500,

      Type: 'E-commerce',

      Revenue: 500,

      Headquarter: 'San Jose, CA',

      Industry: 'E-commerce',

      Founded: 1946,

      CompanyMission: 'Grow amd let grow',

      CEO: 'Simon Shim',

      CompanyDescription: 'Cool Company',

      City: 'San Jose',

      State: 'CA',

    })

    .end(function (err, res) {

      expect(res).to.have.status(201);

      expect(res.text).equal(JSON.stringify('Profile Updated'));

      done();

    });

});

// Test to fetch reviews of a company

it('Test to fetch reviews of a company', function (done) {

  chai

    .request(apiUrl)

    .get('/company/review')

    .set({ Authorization: `JWT ${companyToken}` })

    .query({

      CompanyID: '5',

    })

    .send()

    .end(function (err, res) {

      expect(res).to.have.status(200);

      expect(res.text[1]).equal('[');

      done();

    });

});

// Test to mark a review as favorite

it('Test to mark a review as favorite', function (done) {

  chai

    .request(apiUrl)

    .post('/company/reviewFavorite')

    .set({ Authorization: `JWT ${companyToken}` })

    .send({

      ID: '5',

      Favorite: 1,

    })

    .end(function (err, res) {

      expect(res).to.have.status(201);

      expect(res.text).equal(JSON.stringify('response submitted'));

      done();

    });

});

// Updating response to a review

it('Test to update response to a review', function (done) {

  chai

    .request(apiUrl)

    .post('/company/reviewResponse')

    .set({ Authorization: `JWT ${companyToken}` })

    .send({

      ID: 2,

      Response: 'Thanks for the feedback',

    })

    .end(function (err, res) {

      expect(res).to.have.status(201);

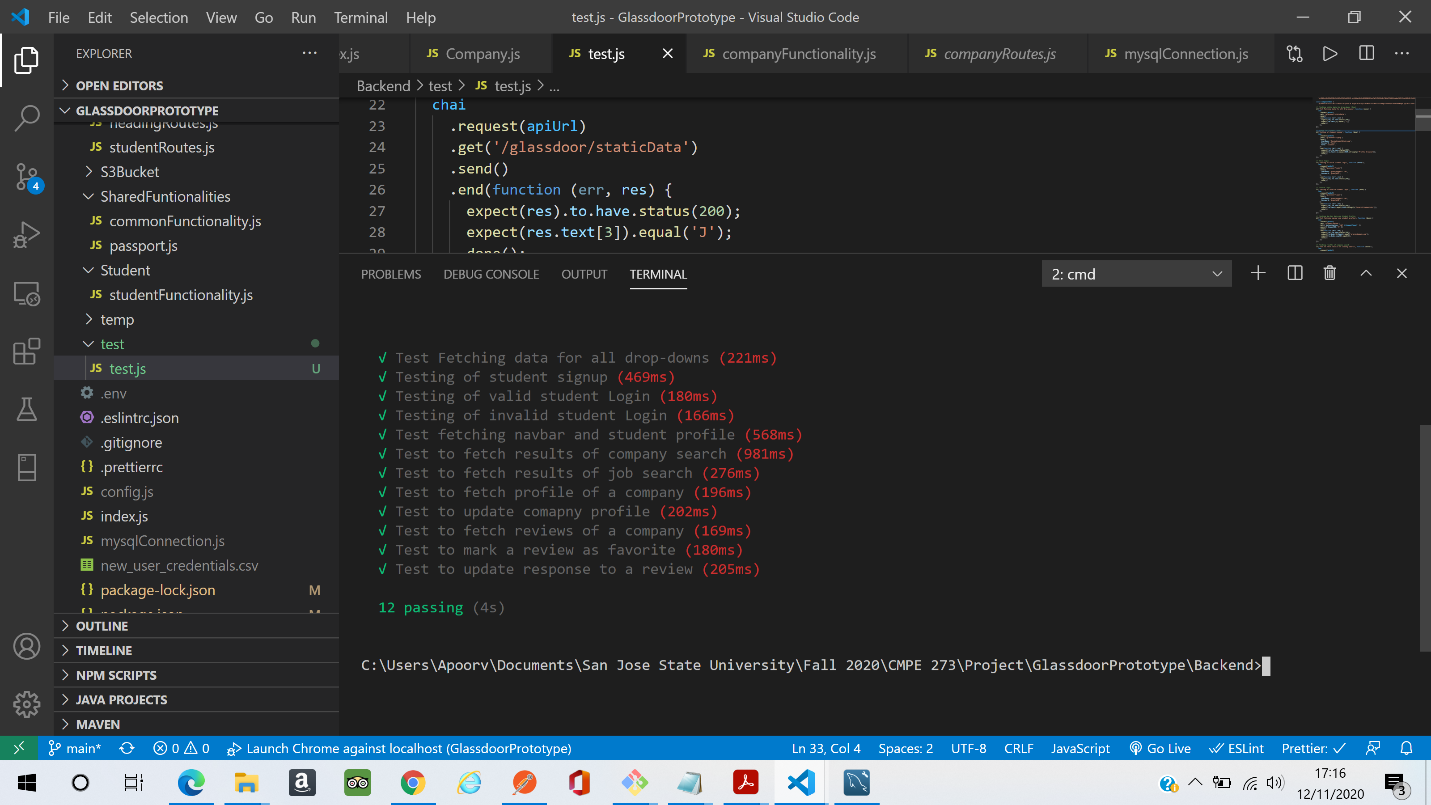
      expect(res.text).equal(JSON.stringify('response submitted'));

      done();

    });

});

**Output**

****