Tech Savvy

Austin Harvey

CST-451 Capstone Project Proposal

Grand Canyon University

Instructor: Professor Mark Reha

Revision: 1.0

Date: 11/29/2020

**ABSTRACT**

Tech Savvy is a simple and quick way to discover your future career in technology. When a user register, they are prompted to input a job title that will search for jobs in the field of technology. The application has a task of requesting and receiving data to display multiple jobs and their description on a table. The user has the option to view their history searches and save jobs to view in their profile. As an accomplishment, the application is made up of a .Net Core back-end and React front-end that continues to receive data using a REST API from the third-party GitHub Jobs API. Both project ends will transfer data between both ends to be displayed to the user. The secured application has a MYSQL database that hold registered users with security caches and is available on the Azure cloud.

|  |
| --- |
| History and Signoff Sheet |

**Change Record**

|  |  |  |
| --- | --- | --- |
| **Date** | **Author** | **Revision Notes** |
| 09/27/2020 | Austin Harvey | Project Proposal |
| 11/01/2020 | Austin Harvey | Project Requirements |
| 11/292020 | Austin Harvey | Architecture and Design |

|  |
| --- |
| **Overall Instructor Feedback/Comments** |

|  |
| --- |
| **Overall Instructor Feedback/Comments** |

**Integrated Instructor Feedback into Project Documentation**

Yes  No

**TABLE OF CONTENTS**

Design Overview 4

Detailed High-Level Solution Design 5

Detailed Technical Design 6

Appendix A – Technical Issue and Risk Log 7

Appendix B – References 8

Appendix C – External Resources 9

Design Introduction

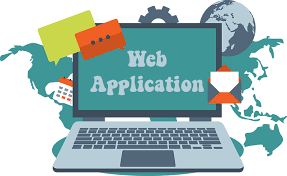
This document contains a design showcase and detailed presentation of the solution to the problem presented in the abstract. The proof of concepts listed support the solution with the technologies that are selected to perform in the Tech Savvy application. There are physical and logical diagrams listed that explain the process of the application running and the code logic through the back end. The detailed technical design section will go over the technical design decisions of the application process with the inclusion of class and database structure, Non-Functional Requirements, sitemap, wireframes, application monitoring and logging, security design, and occurring risks.

Detailed High-Level Solution Design

|  |  |  |
| --- | --- | --- |
| Proof of Concepts | |  |
| **Description** | **Rationale** | **Results** |
| 1 -- Use a cloud that can host a .Net Core back-end project and React font-end project. | This will allow React and .NET Core to connect and send requests together. | Reliable and efficient to use. Pages are rendered well. |
| 2 – Use a front-end client and a back-end project as their own project | REACT can be easy to understand that create user UI and move data around between another project using REST service URLs. .Net Core can handle REST service data requests and database calls. | A REACT Web App on Azure cloud that sends REST service requests to the .Net Core web app to handle API and database data requests. |
| 3 – A third party REST API to read data from | GitHub Job API is free to use and request data from without a key. | As a result, jobs from GitHub jobs are received as a JSON list when inserting correct parameters into the request HTTP URL. |

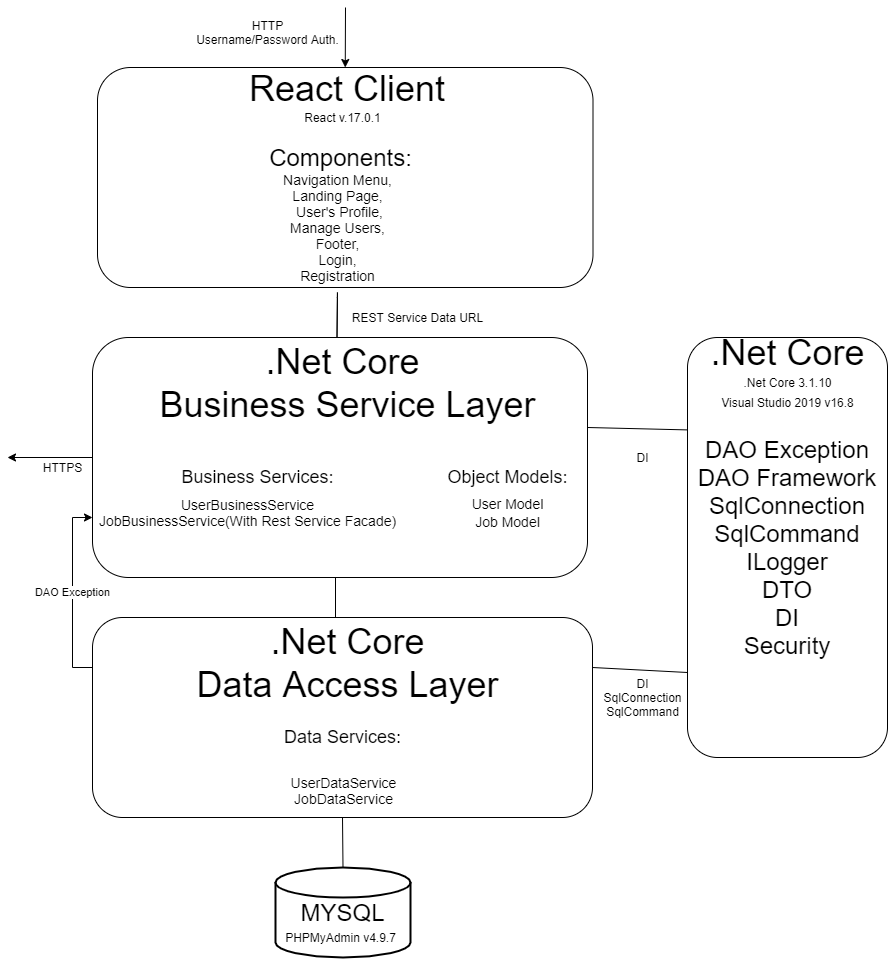
|  |
| --- |
| Hardware and Software Technologies |
| 1 -Visual Studio 2019 v16.8 |
| .Net Core v3.1 |
| 2 – REACT v17.0.1 |
| 3 – Azure Cloud (Latest Version Update on November 25) |
| 4 – GitHub Jobs API v1 |
| 5 -MYSQL Database v8.0 |
| 6. Windows 10 Personal Computer |

**Logical Solution Design:**

****

Web Application

Azure Cloud Service



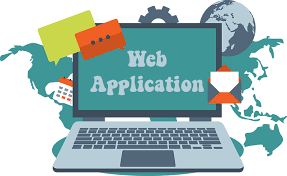
First Azure Cloud Web-App

****

GitHub Jobs API Interface

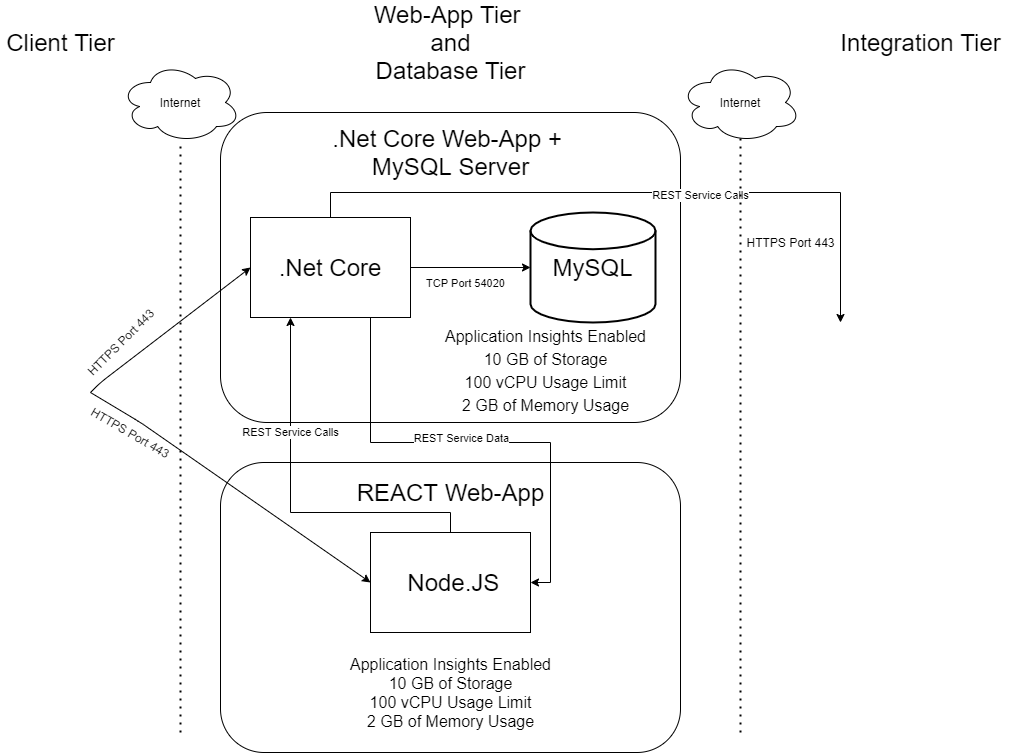
Second Azure Cloud Web-App

When a user either requests or send data in the Azure Cloud web application the front-end React v17.0.1 client will send class component data using REST service data URLs to the back end .Net Core v3.1 project in Visual Studio 2019 v.16.8. The .Net Core project will have several injections. Injections that are included are support a dependency injection software design pattern, database exceptions and framework, MYSQL database connection and commands, and logging to keep records of code pathing. After the user is authenticated the .Net Core business Service layer will receive the REST service data URLs and use the incoming data to access the Data Access layer as object models. The Data Access layer will connect to the MYSQL database using PhpMyAdmin v.4.9.7 to configure data with the incoming data object models. The .Net Core business service will use API data requests to receive data from the third-party API. The received data will be sent to the React client to be displayed to the user and start the process over again.

**Physical Solution Design:**

GitHub Jobs API Interface

Web Application



The React web application can be accessed by using the Azure Cloud web application URL. The React client is in a separate container in the Azure Cloud from the .Net Core back-end code. Each Azure Cloud web app has Application Insights enabled to measure data and log errors, 10 GB of storage, 100 vCPU usage limit, and 2 GB max of Memory Usage. With the internet, a HTTPS request is sent from the web application to the Azure cloud to deploy all the Node.js code to the React web app container. Users will send REST service calls to the .Net Core web app to handle data requests for the GitHub Jobs API or data to be configured in the MYSQL database. The web app will send the REST service data back to the React web app to be displayed to the user and start the process over again.

Detailed Technical Design

**General Technical Approach:**

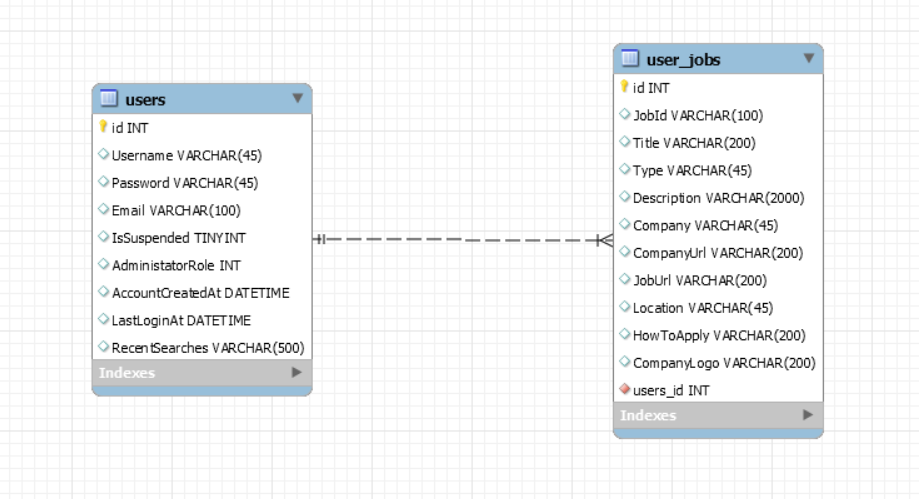
Tech Savvy’s back end will use the IDE Visual Studio with .NET Core framework to interact with the project’s MYSQL database and GitHub Jobs API as a third party. The back end will include a business layer which will contain services for the application and communicate with the database to request data to be retrieved or configured. The project’s front-end will use React and will send service calls to the back-end project as well as retrieve from it. The projects will be hosted on an Azure Cloud web-app with a MYSQL database to store all data into. The projects will implement security into our application through middleware and duplicate database checks.

**Key Technical Design Decisions:**

Much of the technologies throughout the project’s phases have changed due to restrictions or inconveniences. For the front end React JS will be used to implement the front end of the project. React will utilize Material UI to make the front end responsive as well as to have a unique design on multiple web browsers. The front-end JavaScript language will allow the project to perform accurate data and fast loading reactions. In the backend for the business service, .Net Core framework will be used because it is compatible with React and MYSQL. For the Data Access Layer, MYSQL will be the database service that will hold data for the user to interact with through-out the application. To deploy the application, Azure Cloud supports these technologies and is good for a new experience with cloud management as well as it has easy to implement configurations regarding the MYSQL Database. The database will be hosted on a web-app with the back-end .NET Core project. The front-end will be on a separate web-app that will interact with the other web-app to utilize the database and service requests.

**Database ER Diagram:**

Users will have the option to store multiple jobs.



**Database DDL Scripts:**

-- MySQL Workbench Synchronization

-- Generated: 2020-11-29 19:48

-- Model: New Model

-- Version: 1.0

-- Project: Name of the project

-- Author: Austin

SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0;

SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0;

SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='ONLY\_FULL\_GROUP\_BY,STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_ENGINE\_SUBSTITUTION';

CREATE SCHEMA IF NOT EXISTS `techsavvy` DEFAULT CHARACTER SET utf8 ;

CREATE TABLE IF NOT EXISTS `techsavvy`.`users` (

`id` INT(11) NOT NULL AUTO\_INCREMENT,

`Username` VARCHAR(45) NULL DEFAULT NULL,

`Password` VARCHAR(45) NULL DEFAULT NULL,

`Email` VARCHAR(100) NULL DEFAULT NULL,

`IsSuspended` TINYINT(4) NULL DEFAULT NULL,

`AdministatorRole` INT(11) NULL DEFAULT NULL,

`AccountCreatedAt` DATETIME NULL DEFAULT NULL,

`LastLoginAt` DATETIME NULL DEFAULT NULL,

`RecentSearches` VARCHAR(500) NULL DEFAULT NULL,

PRIMARY KEY (`id`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8;

CREATE TABLE IF NOT EXISTS `techsavvy`.`user\_jobs` (

`id` INT(11) NOT NULL AUTO\_INCREMENT,

`JobId` VARCHAR(100) NULL DEFAULT NULL,

`Title` VARCHAR(200) NULL DEFAULT NULL,

`Type` VARCHAR(45) NULL DEFAULT NULL,

`Description` VARCHAR(2000) NULL DEFAULT NULL,

`Company` VARCHAR(45) NULL DEFAULT NULL,

`CompanyUrl` VARCHAR(200) NULL DEFAULT NULL,

`JobUrl` VARCHAR(200) NULL DEFAULT NULL,

`Location` VARCHAR(45) NULL DEFAULT NULL,

`HowToApply` VARCHAR(200) NULL DEFAULT NULL,

`CompanyLogo` VARCHAR(200) NULL DEFAULT NULL,

`users\_id` INT(11) NOT NULL,

PRIMARY KEY (`id`),

INDEX `fk\_user\_jobs\_users\_idx` (`users\_id` ASC),

CONSTRAINT `fk\_user\_jobs\_users`

FOREIGN KEY (`users\_id`)

REFERENCES `techsavvy`.`users` (`id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8;

SET SQL\_MODE=@OLD\_SQL\_MODE;

SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS;

SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS;

**Data Dictionary:**

USERS: A table that will contain all the users of the application.

|  |  |  |
| --- | --- | --- |
| USERS | | |
| **Field Name** | **Type** | **Description** |
| Id | Int | The Identifier for a User. |
| Username | String | The Username of the User. |
| Password | String | The last Password of the User. |
| Email | String | The email of the User. |
| IsSuspended | String | An identifier to determine if the User is an Administrator. |
| AdministratorRole | Text | The role of the User. |
| AccountCreatedAt | String | An identifier to determine when the User created the account. |
| LastLoginAt | Int | An identifier to determine when the User logged into the application. |
| RecentSearches | String | A string array that will contain up to 50 job titles |

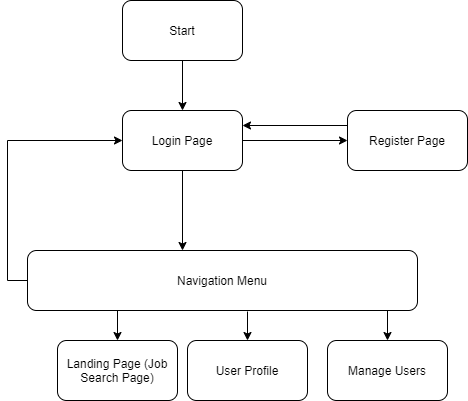
USER\_JOBS: A table that will contain all of the User’s jobs of the application.

|  |  |  |
| --- | --- | --- |
| USER\_JOBS | | |
| **Field Name** | **Type** | **Description** |
| ID | Int | The identifier for a User’s Job. |
| JobId | String | The identifier for a GitHub Jobs API Job. |
| Title | String | The title of the Job. |
| Type | String | The type of the Job. Either ‘Full-Time’ or ‘Part-Time’ |
| Description | String | The description of the Job. |
| Company | String | The company that created the Job. |
| CompanyUrl | String | The URL to the company website. |
| JobUrl | String | The URL to the Job website. |
| Location | String | The location of the Job |
| HowToApply | String | The URL to tell how to apply for the Job. |
| CompanyLogo | String | The company logo image URL. |
| Users\_id | int | The identifier for a User’s identification |

**Flow Charts/Process Flows:**

The sitemap below provides a navigation walkthrough through the application. No algorithms were used in this design.

**Sitemap Diagram:**



Footer included on every page

**User Interface Diagrams:**

**Navigation Menu:**

**No User Session:**

****

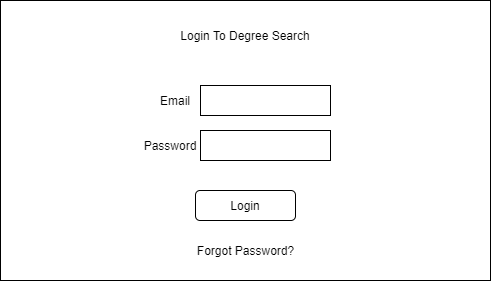
**User in Session:**

****

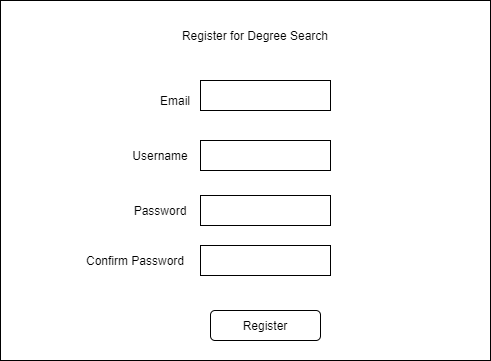
**Admin User in Session:**

****

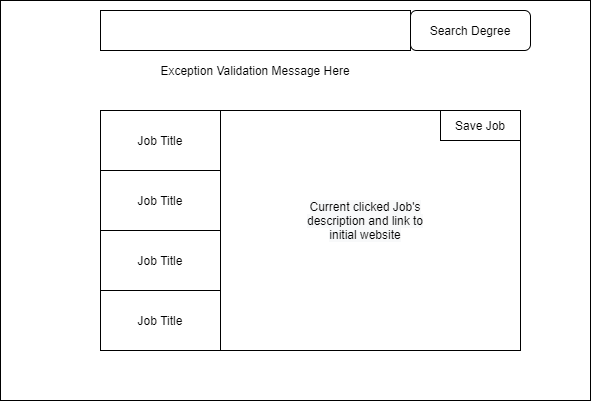
**Login:**



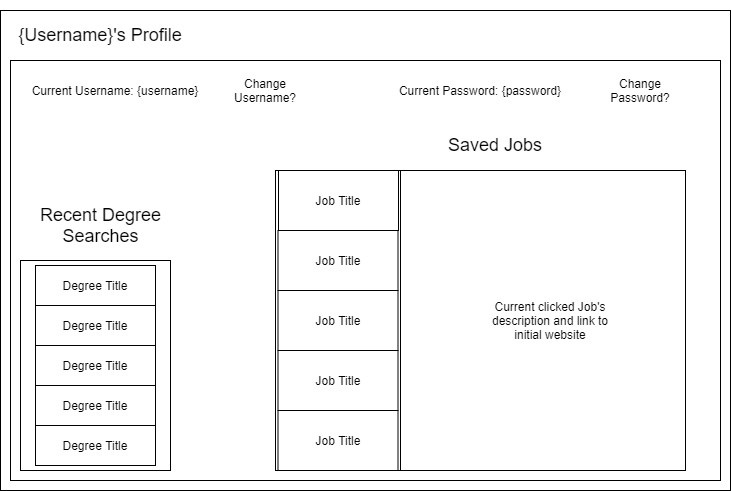
**Register:**

****

**Landing Page:**

****

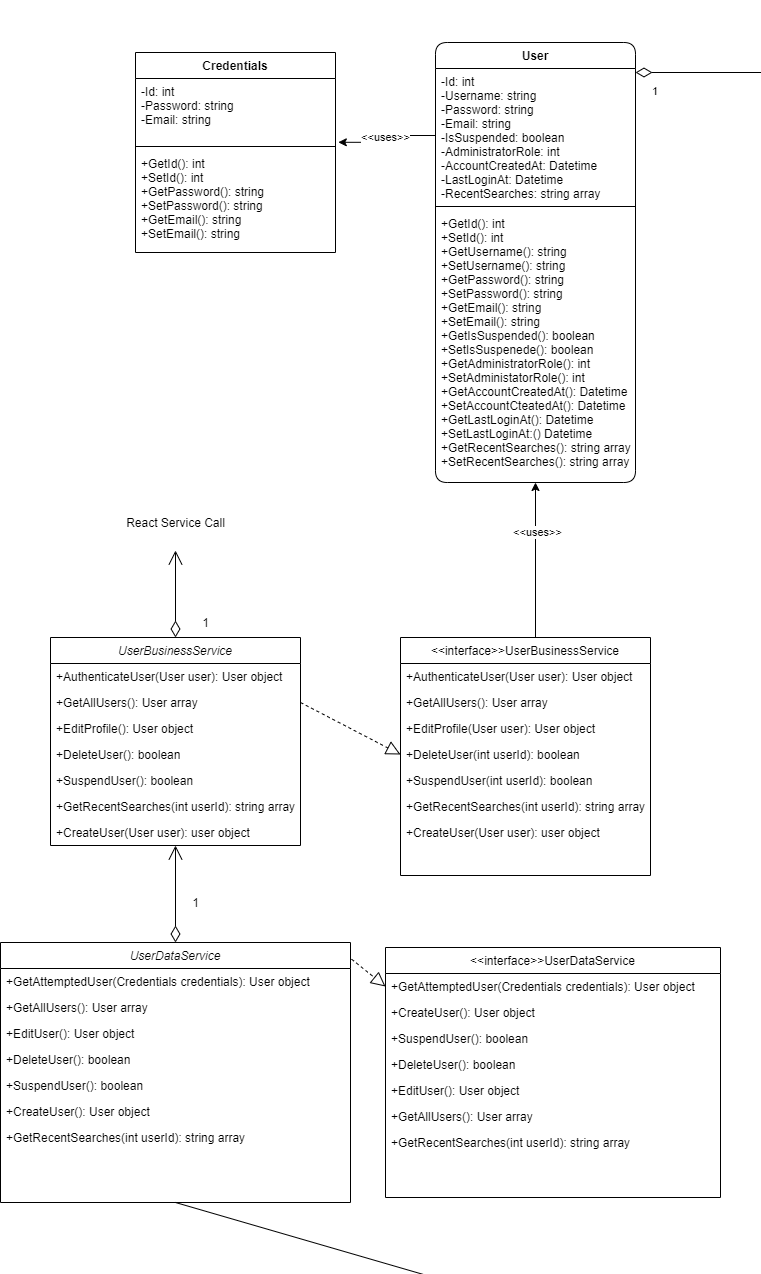
**User’s Profile:**

****

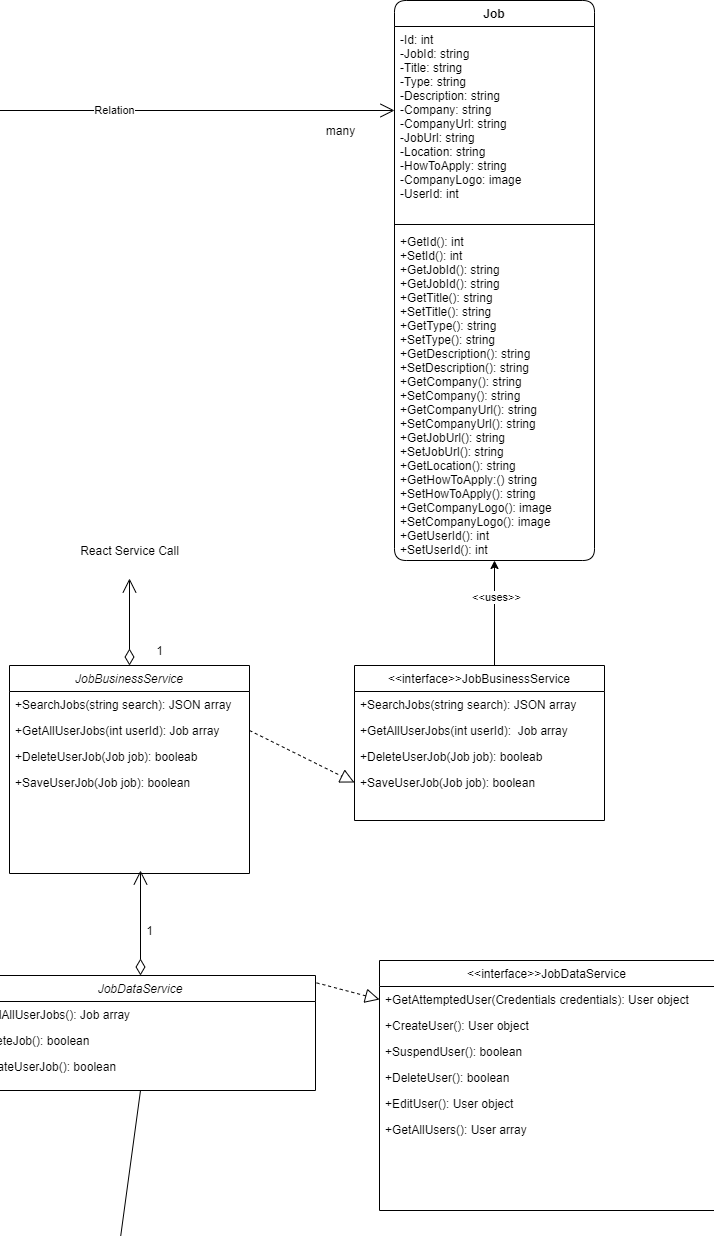
**UML Diagrams:**

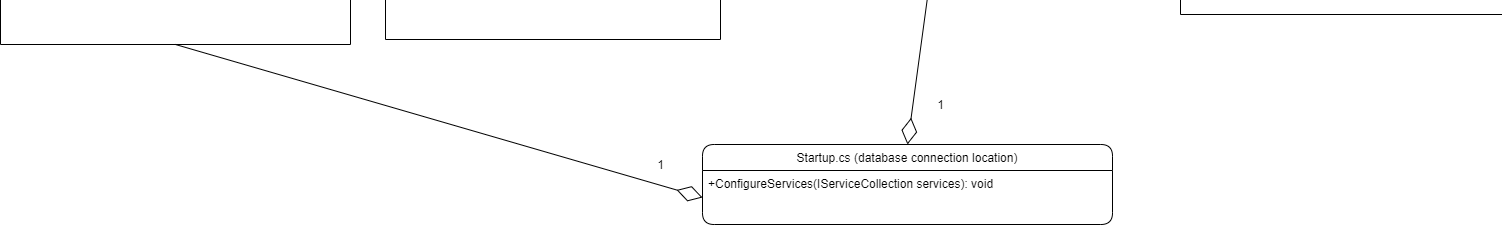
**Application UML Class Diagram:**

**User Class Diagram (Left Side)**

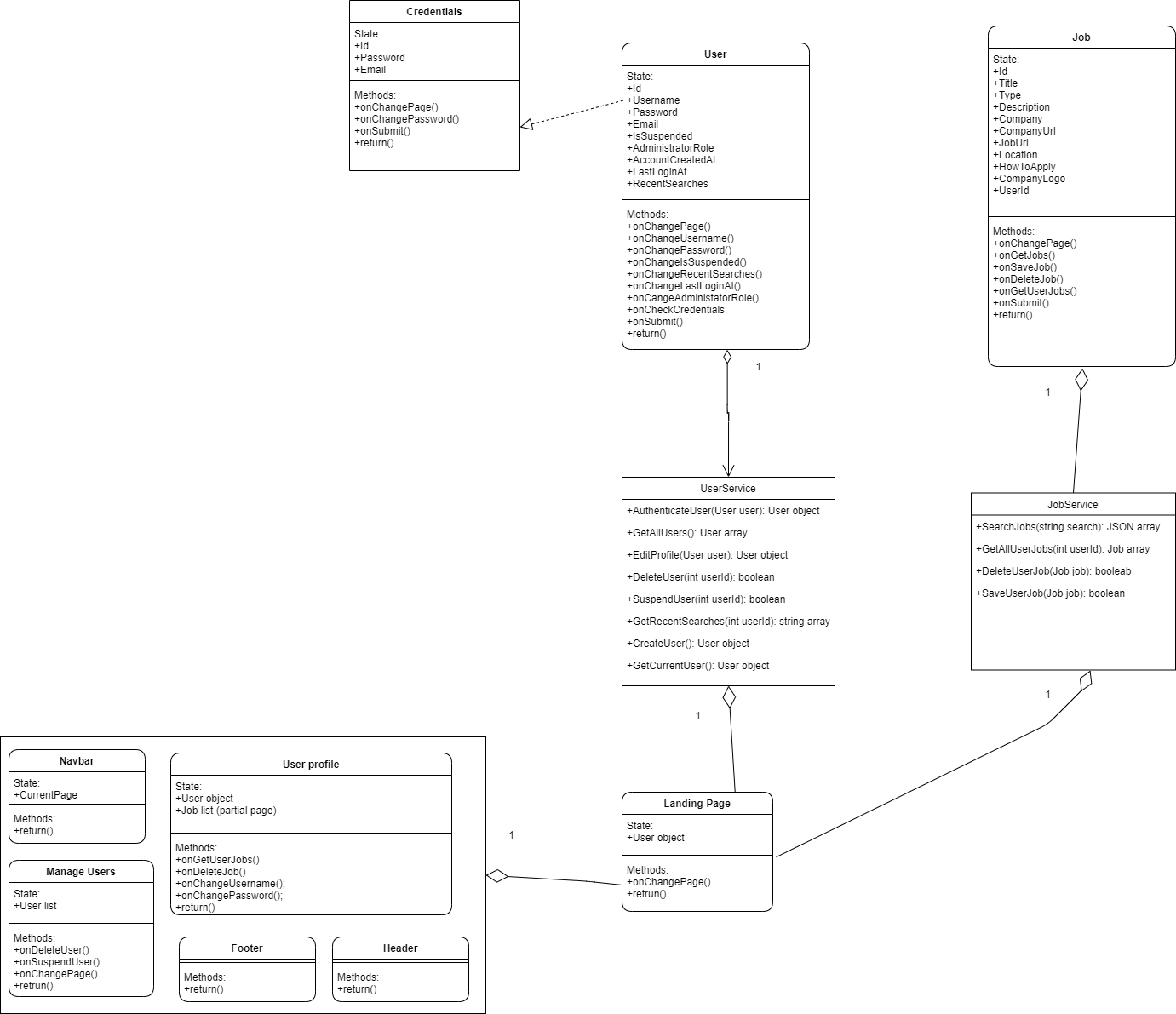


**Job Class Diagram (Right Side)**

****

**Both Data Services Connect to Startup.cs Class UML for Database Connection (Bottom)**

**React UML Class Diagram:**

****

**Service API Design:**

No REST Service API was designed for this application.

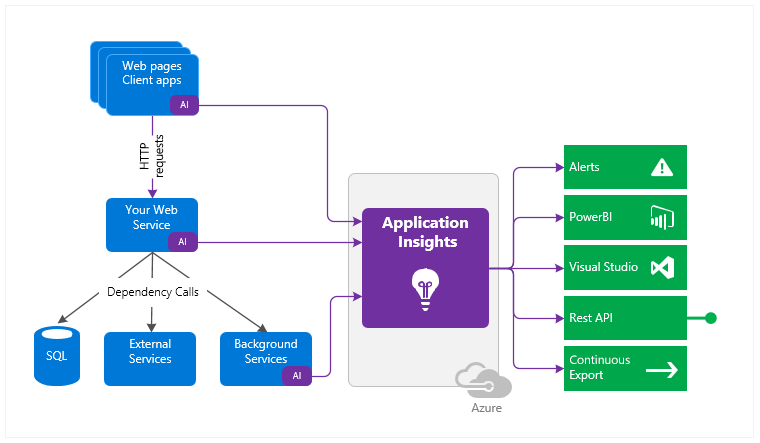
**NFR’s (Security Design, etc.):**

The React client can integrate a Material UI package that will enhance the responsiveness of the web application as well as make the web browser look clean. Node JS can include this library to provide customization of the React components by using tags similar to the HTML language. Material UI will allow the web browser to expand or collapse React components from the size of the screen. The application will be easy to navigate and look responsive to the user because of the Material UI library. Additionally, user passwords will be hashed to keep user’s information safe and secured. By using the node.js library, password-hash, generating a hashed password and verifying a plain-text password against a hashed password. For a bit of added strength, a random salt is generated when the password is hashed. The hashed password contains both the cryptographic algorithm that was used as well as the salt. All that is needed to verify a plain-text password is the hashed password itself.

**Operational Support Design:**

Locally, this project back-end will be using the Microsoft extension ILogger to monitor and log coding data pathing. It will log string messages, warnings, and errors to a log file on the project’s local computer. The project’s front-end will log string messages, warnings, and errors to the current browser the client is running on. In the Azure Cloud, the application insight feature is enabled to monitor and log exceptions, dependencies, and performances to the project’s Application Insights resource page.

**Other Documentation:**

**How Azure Cloud Application Insights work:**

**Security Design:**

**What Application Insights Will Monitor:**

* Request rates, response times, and failure rates - Find out which pages are most popular, at what times of day, and where your users are. See which pages perform best. If your response times and failure rates go high when there are more requests, then perhaps you have a resourcing problem.
* Dependency rates, response times, and failure rates - Find out whether external services are slowing you down.
* Exceptions - Analyze the aggregated statistics or pick specific instances and drill into the stack trace and related requests. Both server and browser exceptions are reported.
* Page views and load performance - reported by your users' browsers.
* AJAX calls from web pages - rates, response times, and failure rates.
* User and session counts.
* Performance counters from your Windows server machines, such as CPU, memory, and network usage.
* Host diagnostics from Azure.
* Diagnostic trace logs from your app - so that you can correlate trace events with requests.

**Authentication and Authorization:**

Along with Application insights, authentication and authorization were supported through users being stored into a Database while the React client called an authenticate process to see if desired user is stored within the database to be able to continue to use the application. Before creating a new user, attempted user email is searched in the database for an existing user. If there is a user by that email, no new user is created.

Appendix A – Technical Issue and Risk Log

1. Use the template to identify and monitor project issues and risks.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Issues and Risk Log | | | | | | | | |
| **Issue or Risk** | **Description** | **Project Impact** | **Action Plan/Resolution** | **Owner** | **Importance** | **Date Entered** | **Date to Review** | **Date Resolved** |
| I/R | What is the issue or risk? | How will this impact scope, schedule, and cost? | How do you intend to deal with this issue? | Who manages this issue? |  |  |  |  |
| 1 | Rest service calls from React Client to back-end in another web-app on the Azure Cloud | This impacts the entire project | Research and practice projects deployed to the Azure cloud can subdue the risk. | Austin Harvey | High | 11/22/2020 | 11/30/2020 | N/A |
|  |  |  |  |  |  |  |  |  |

Appendix B – References

What is Azure Application Insights? - Azure Monitor (2019, June 3). *Microsoft Docs*. <https://docs.microsoft.com/en-us/azure/azure-monitor/app/app-insights-overview>

Appendix C – External Resources

|  |  |
| --- | --- |
| **GIT URL:** | *N/A* |
| **Hosting Azure Cloud URLs:** | *React Web-App:* [*https://reacttechsavvy.azurewebsites.net*](https://reacttechsavvy.azurewebsites.net)  *.NET Core Web-App:* [*https://gcutechsavvy.azurewebsites.net/*](https://gcutechsavvy.azurewebsites.net/) |