**Developer Documentation**

**Team 9 – ResumYay**

Bracken Brinkel, Tarek Fouad, Tommy Pham, and Marina Alsip

**Project setup and installation (only for windows)**

**Frontend Installation**

To develop Angular applications, you need to install Node.js and npm (Node Package Manager). Follow these steps to install Node.js, npm, and Angular CLI.

**Install Node.js and npm**

**On Windows**

1. Download Node.js:
   * Go to the [Node.js download page](https://nodejs.org/) and download the Windows Installer (.msi) for the LTS (Long Term Support) version.
2. Run the Installer:
   * Double-click the downloaded .msi file and follow the prompts. Ensure that the option to install npm is selected.
3. Verify Installation:
   * Open Command Prompt and run:
   * “ node -v ”
   * “ npm -v ”
   * Confirm that Node.js and npm are installed correctly by seeing a version upon running those commands.
4. If you get an error saying "cannot be loaded because running scripts is disabled on this system": - Type env in the windows bar. - Select the Environment Variables in the bottom corner. - Click on "Path" - Find the path for npm. For example "C:\Users\SomeUser\AppData\Roaming\npm" - Close env.

**Angular Installation**

With Node.js and npm installed, you can now install Angular CLI.

1. Install Angular CLI:
   * Open a terminal or Command Prompt and run:

“ npm install -g @angular/cli ”

1. Verify Angular CLI Installation:
   * Check the version of Angular CLI:

“ ng version ”

1. If you get this error "Could not find the implementation for builder @angular-devkit/build-angular:dev-server on ng serve command [duplicate]":
   * In VS code go to the ResumeBuilderUI.2 folder
   * Run this command in VS Code:

“ npm install ”

1. Reset Visual Studio Code

**Database Installation**

We will be using SQL Server 2022 and to view the data in the database the application being used SQL Server Management Studio 20.

1. Download SQL Server 2022:
   * Go to the [SQL Server download page](https://www.microsoft.com/en-us/sql-server/sql-server-downloads) and download the SQL Server 2022 Developer.
   * After installation a page will prompt the users to connect the server using the connect now button.
   * A command prompt window will pop up after hitting the connect button. Then you can close the installer.
2. Download SQL Server Management Studio 20:
   * Go to the [SQL Server Managment Studio 20 download page](https://learn.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16) and download the SQL Server 2022 Developer. You can download SMSS from the SQL Server 2022 Installer.
3. Using SQL Server:
   * Open up SQL Server Management Studio 20.
   * Uses the icon with an outlet.
   * Look at Server name and copy that name. (for later use)
   * Open the C# project in visual studio 2022, it will be located in the ResumeBuilder(Team9) folder.
   * In the solution explorer on the right side of the project, find appsettings.json and look for this code on line 10. (The line bellow is the line that you are modifying)

"Data Source=DESKTOP-HB3HD5S;TrustServerCertificate=True;Initial Catalog = ResumeBuilderDb; Integrated Security = true;"

* + Replace Data Source with the Server name you copied from the previous step. ("Source=DESKTOP-HB3HD5S;")
  + In Package Manager Console, update the database with the command bellow.

“ update-database ”

* If you can not find the Package manager Console, use

tools->Nu Get Package Manager->Package Manager Consol.

* + Now the database is set up and ready to be used.
  + Now you can go back to SQL server management studio, make sure to press the check box next to the trust server certificate, then press connect at the bottom left of the connect to server window.

**Run the Angular Application**

1. Start the Development Server:
   * Open the ResumeBuilderUI.2 folder in VS code.
   * Run the following command in the terminal to start the Angular development server: (Make sure the terminal path is the ResumeBuilderUI.2)

“ ng serve -o ”

* If you get an error that states you can not run scripts on the current system. You will have to give your computer permission to runs scripts by running this command in your PowerShell in administrator mode.

“ Set-ExecutionPolicy -ExecutionPolicy Unrestricted ”

* If you get this error :"Error: Could not find the '@angular-devkit/build-angular:dev-server' builder's node package."   
  look at the solution in part 2 step 3.

1. Access the Application:
   * Open your browser and navigate to http://localhost:4200 to view your Angular application.

**Run the Angular Application**

To start running the database make sure to run RsumeBuilder(Team 9) project by pressing the HTTPS button near the top where you would normally see your debug compiler button.

Once the back end is running a swagerUI page with open in your default browser. (make sure that the settings on that browser are allowing localhost pages to be launched).

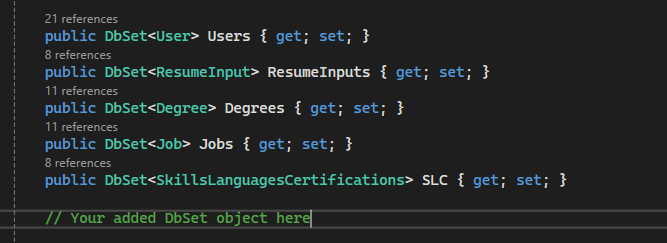
The front end will start running when you type in the terminal "ng serve -o" (we did this in part 4 step 1).

**ResumeBuilder(Team9).sln (The backend)**

Our Models:

* Using the Entity Core Framework, we used object relational mapping to interact with our database in our backend code. Each object created from the class represents an entity to be added to its associated data table inside of the database.
* A screen shot of a computer

  Description automatically generatedMore models, which in turn create new tables, can be added by creating a new class in the model folder, and adding a DbSet object to the AppDbContext.cs file where the other DbSets are. This DbSet object is for accessing the table and has commands for querying, adding, editing, or removing.



1. AuthenticateRequest  
   Does not represent an entity that gets entered into the database, rather an easy placeholder for seeing if the entered information when logging in is a valid user or not.
   * **Username: Property for returning or setting the Username to be tested for authentication.**
   * **Password: Property for returning or setting the Password to be tested for authentication.**
2. Degree

A type of entity that gets entered into the Degree table of the database, which holds certain information about a specific degree.

* + **Id:** Primary key for the record, ensures it is unique and cannot cause conflicting data issues. Property for getting or setting the ID of the degree.
  + **UserId:** Foreign key tied to User in User table, to ensure the degree is linked to a certain User. Property for getting or setting the associated User’s ID.
  + **College:** Property for getting or setting the college field of the entity.
  + **CityAndState:** Property for getting or setting the cityAndState field of the entity.
  + **DegreeType:** Property for getting or setting the degreeType field of the entity.
  + **DegreeName:** Property for getting or setting the degreeName field of the entity.
  + **YearGraduated:** Property for getting or setting the yearGraduated field of the entity.

1. Job

A type of entity that gets entered into the Job table of the database, which holds certain information about a specific job.

* + **Id:** Primary key for the record, ensures it is unique and cannot cause conflicting data issues. Property for getting or setting the ID of the degree.
  + **UserId:** Foreign key tied to User in User table, to ensure the job is linked to a certain User. Property for getting or setting the associated User’s ID.
  + **CompanyName:** Property for getting or setting the companyName field of the entity.
  + **Position:** Property for getting or setting the position field of the entity.
  + **StartDate:** Property for getting or setting the startDate field of the entity.
  + **EndDate:** Property for getting or setting the endDate field of the entity.
  + **JobResponsibilities:** Property for getting or setting the jobResponsibilities field of the entity.

1. ResumeInput

A type of entity that gets entered into the ResumeInput table of the database, which holds the User’s personal Information.

* + **Id:** Primary key for the record, ensures it is unique and cannot cause conflicting data issues. Property for getting or setting the ID of the degree.
  + **UserId:** Foreign key tied to User in User table, to ensure the degree is linked to a certain User. Property for getting or setting the associated User’s ID.
  + **FirstName:** Property for getting or setting the firstName field of the entity.
  + **LastName:** Property for getting or setting the lastName field of the entity.
  + **Email:** Property for getting or setting the email field of the entity.
  + **PhoneNumber:** Property for getting or setting the phoneNumber field of the entity.
  + **Website:** Property for getting or setting the website field of the entity.
  + **Summary:** Property for getting or setting the summary field of the entity.

1. SkillsLanguagesCertifications

A type of entity that gets entered into the SkillsLanguagesCertifications table of the database, which holds the Users skills, languages, and certifications.

* + **Id:** Primary key for the record, ensures it is unique and cannot cause conflicting data issues. Property for getting or setting the ID of the degree.
  + **UserId:** Foreign key tied to User in User table, to ensure the degree is linked to a certain User. Property for getting or setting the associated User’s ID.
  + **LanguageName:** Property for getting or setting the languageName field of the entity.
  + **CertificationName:** Property for getting or setting the certificationName field of the entity.
  + **Skills:** Property for getting or setting the skills field of the entity.
  + **Projects:** Property for getting or setting the projects field of the entity.

1. User

A type of entity that gets entered into the User table, holds the Users information as well as well as collections of other model types, for ensuring that it is a primary key to those types.

* + **Id:** Primary key for the record, ensures it is unique and cannot cause conflicting data issues. Property for getting or setting the ID of the degree. Is User ID in other tables it is associated to.
  + **FirstName:** Property for getting or setting the firstName field of the entity.
  + **LastName:** Property for getting or setting the lastName field of the entity.
  + **Username:** Property for getting or setting the username field of the entity.
  + **Email:** Property for getting or setting the email field of the entity.
  + **Password:** Property for getting or setting the password field of the entity.
  + **ResumeInput:** Property for getting or setting the ResumeInput associated entity of the field. Is Nullable, as in our code doesn’t automatically input this field on registration, dynamically does when User enters the specific Information.
  + **SkillsLanguagesCertifications:** Property for getting or setting the SkillsLanguagesCerticifcations associated entity of the field. Is Nullable, as in our code doesn’t automatically input this field on registration, dynamically does when User enters specific the Information.
  + **Degrees:** Property for getting or setting the list of Degree entities associated to the User. Is Nullable, as in our code doesn’t automatically input this field on registration, dynamically does when User enters specific the Information.
  + **Jobs:** Property for getting or setting the list of Degree entities associated to the User. Is Nullable, as in our code doesn’t automatically input this field on registration, dynamically does when User enters specific the Information.

Our Controllers:

* Our backend comes with a variety of different controllers. These controllers hold functions that receive Http requests and based on those requests, interact with the database in a certain way. These functions then return Http responses to the front end, holding certain information, like confirmation messages, error messages, or fields of data. Each controller is based off a specific Entity type/ table type for easy readability and less confusion. These controllers can be added to if you would like to add more functionality between the front and back end.

1. DegreesController  
   Handles operations related to user degrees within the system.
   * **SubmitDegree (POST)**: Allows adding a new degree for a user. Limits users to a maximum of three degrees.
   * **RemoveDegreeFromList (DELETE)**: Deletes a specific degree by its own ID, not the User’s.
   * **GetDegreesFromList (GET)**: Fetches all degrees associated with a given user, by the User’s ID.
   * **EditDegreeInfo (PUT)**: Updates an existing degree's information based on its own ID.
2. JobsController  
   Manages job-related user data within the database.
   * **SubmitJob (POST)**: Adds a new job for a user. Users can save up to three job entries.
   * **RemoveJobFromList (DELETE)**: Deletes a job by its own ID.
   * **GetJobsFromList (GET)**: Retrieves all jobs for a specific user, by the User’s ID.
   * **EditJobInfo (PUT):** Updates details of an existing job, given the job’s ID.
3. PersonalInformationController  
   Facilitates CRUD operations for user personal information.
   * **SubmitPersonalInfo (POST)**: Saves personal information for a user.
   * **EditPersonalInfo (PUT)**: Edits an existing personal information entry by User ID.
   * **GetPersonalInfo (GET)**: Fetches a user's personal information by User ID.
   * **DeletePersonalInfo (DELETE)**: Deletes a user's personal information by User ID.
4. ResumeCreatingController  
   Responsible for generating and retrieving resumes in various templates.
   * **SubmitResumeCreating (GET)**: Generates a resume in a specified template using user data (jobs, degrees, skills, etc.) and saves it as a PDF.
   * **GetResume (GET)**: Fetches a previously generated resume file by name.
5. SkillsLanguagesCertificationsController
   * **SubmitSLCInfo (POST)**: Adds new skills, languages, or certifications for a user.
   * **EditSLCInfo (PUT)**: Updates existing skills, languages, and certifications entry by User ID.
   * **GetPersonalInfo (GET)**: Retrieves a user's skills, languages, and certifications by User ID.
   * **DeleteSLCInfo (DELETE)**: Deletes specific skills, languages, or certifications by User ID.
6. UserController
   * **Authenticate (POST)**: Verifies a user's login credentials.
   * **CreateTempUser (GET)**: Creates a temporary user entry in the system and returns their ID.
   * **GetUserIdFromUsername (GET)**: Retrieves the user ID for a given username.
   * **DeleteAllUserInputs (DELETE)**: Deletes all user-related data, including personal information, jobs, degrees, and SLC entries. Optionally deletes the user as well.
   * **RegisterUser (POST)**: Registers a new user with validation for username, email, and password strength.

PDF Export and QuestPDF

QuestPDF is a library that we use to deal with PDF exports and designs.

* The templates used for the project are implemented in classes that would be called by the program and passed the specific user inputs to create the resume.
* QuestPDF used a basic header, body, and footer design. This means that all PDF designs can utilize all 3 of those. A note is that the header and footer would be produced on every page of the PDF, thust for this project to make resume templates that do not have the users personal info on the top of each page you would need to design the header in the body. We created a mix of both resumes to demonstrate this.
* QuestPDF requires an exact path to export PDFs. Most modern web browsers do not allow any website requests for exact paths on user devices, thus the PDF’s are created on the back end of the project. Once the PDF is created, it is sent to the front end of the project to be downloaded by the user.
* The quest PDF only requests data from the database and does not send/input any data to the database.

**The front end**

* The front end consists of webpages, each page has 4 files linked to it: an HTML, CSS, TypeScript, and a spec TypeScript file.
  + HTML: Consists of the HTML and the layout of the page for all the pages.
  + CSS: Deals with the design of the HTML components
  + TypeScript: deals with the functionality of the pages, what the HTML components are doing once interacted with or when the page is loaded.
  + Spec TypeScript: This is used for testing purposes.

All page components are located in the corresponding folder within the components read folder.

A screen shot of a computer

Description automatically generated

* The front end deals with the user inputs and functunality of the program from a visual and navigational perspective, then it interacts with the back end to retrieve data from the database and create the resumes and downloads.

**There are 2 different types of web pages in this project:**

* Information pages: These pages do not interact with the database and are just meant to be visual and informational pages. These pages have minimal type script and may only have a few buttons and consist of mostly HTML and CSS.   
  Examples of these would be the tips and tricks, about, and UCCS keywords.

A screenshot of a computer screen

Description automatically generated

* Input pages: These pages have a lot more functionality than the information pages. Consisting of many inputs, buttons, and functionality. These pages usually have heavy typescript functionality and would interact with the back end and database to deal with the user information.  
  Examples of these pages would include the pages education, PDF download, and login page.

A screenshot of a login screen

Description automatically generated

* Throught the project almost all input pages get or post information to the database (currently the only exception is the both keyword suggestion pages for computer science and bussnies).
  + To learn more on how to get and post to the database check the backend section on posting and getting data from the database.

**QuestPDF functionality**

QuestPDF is a C# library that deals with creating PDF using C#. The templates are created using the QuestPDF functions and calls that are used to implement out user data that is grabbed from the database.

QuestPDF requirements

* Exact paths for PDF creation and export
* Does not export to any other format other than PDF
* Headers and footers are placed on all pages and there is no way to remove them on later pages. To do this you need to build the header layout in the body of the templates.
* Has a PDF preview, however, has to be imported separately.
* QuestPDF prioritizes open source licensing and knows the importance of it to the progress of their project and community  
  (Place this at the top of you project.cs:   
  “ QuestPDF.Settings.License = LicenseType.Community; ”

Bugs and Future additions:

Known Bugs:

* As a temporary User, if you attempt to navigate to any pages by typing them on the search bar, you will get to that page, but it’s supposed to take you back home. Once you refresh or go back, it will take you home, however it doesn’t until then.
* If the database gets too many requests from the same user it would cause a database stall that would result in time outs to the requests sent. In extreme cases would disconnect the database.

Future Additions:

* As a temporary User at the moment, refreshing the page deletes the User’s data and sends them back to the home page. We would like this to be different, however in the scope of time, it was the only solution we had for the problem of deleting the Temporary User when they exit the application (as the same event for leaving is the same event for refreshing the page).
* User fields are not validated, and the User can realistically type whatever they want into the fields. It has a lot of freedoms, but we would like to add checks in the future to make sure that the User enters information that would ensure a strong resume.
* Add functionality to the forgot password button. This would be setting up a system to send a token to the users email that would be valid for x amount of time where they can use to reset the password.
* Be able to add PDF preview functionality so the user can preview their resume before downloading it.