

# 1. Introduction

This document aims to discuss ideas of design on interface, application and database for the Gender Pay Gap Web APP mentioned in coursework 1. Based on our problem statement, user personas, and prepared dataset, we have decided to develop a multi-page app includes:

## 1. A REST API that allows:

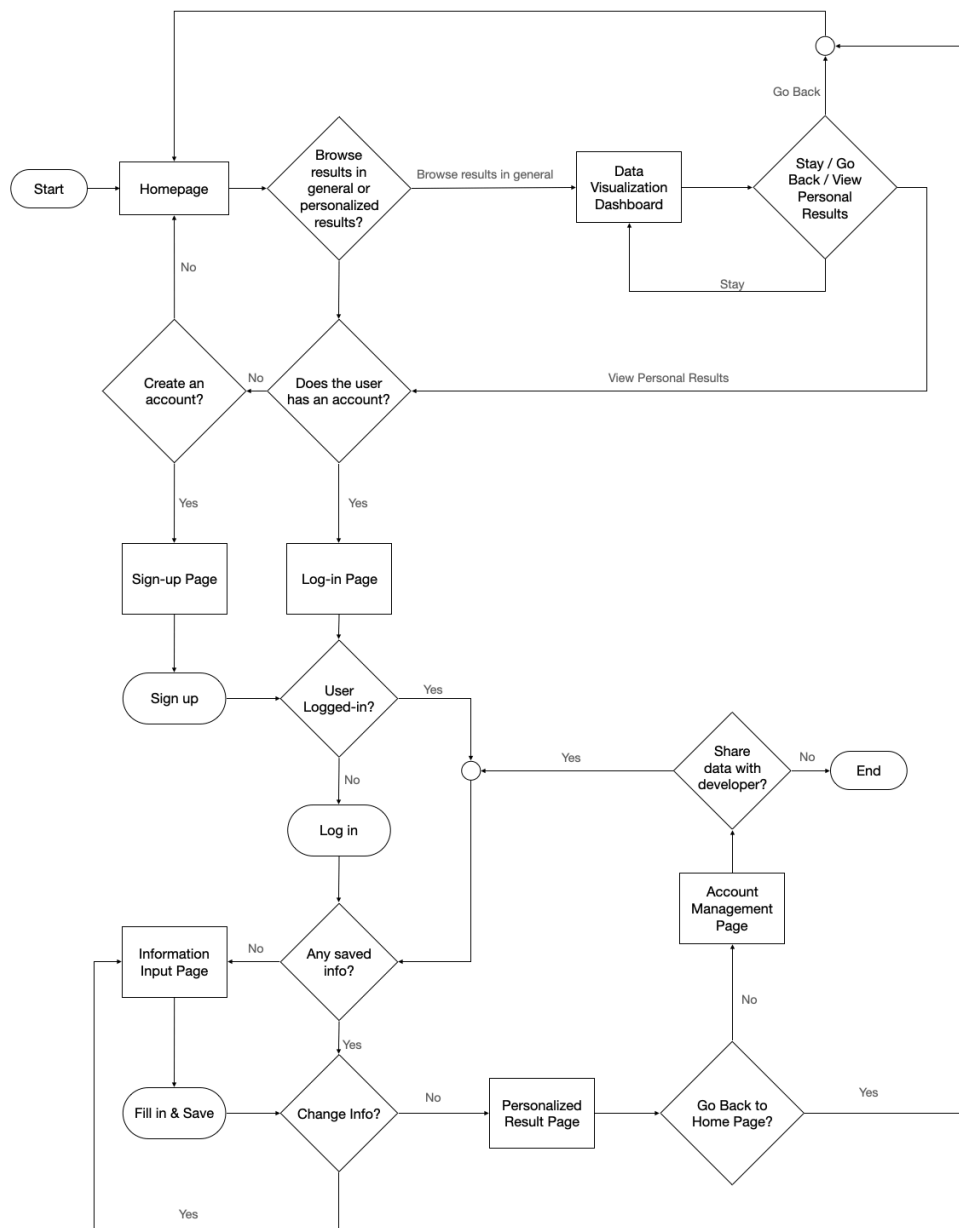
- Employers to request their companies' gender pay gap situations including salary difference, bonus difference, and percentage of highly paid women compared with other companies in the same industry / region / company size;
- Employees to request the gender pay gap information including salary difference, bonus difference, and percentage of highly paid women of a certain industry or region;
- Job applicants to request gender pay gap information including salary difference, bonus difference, and percentage of highly paid women of a certain industry;
- Users to share their own gender pay gap data to help improve the data quality;

## 2. A Data Visualization Dashboard that allows every viewer to gain data visualization of gender pay gap situations across UK in general based on data posted by GOV.UK;

to meet those requirements elicited before which were listed as Must and Should Haves. For those listed as Could and Won't Haves, due to time limit for this project, we decide to ignore them at the stage.

## 2. Interface Design

The following flow chart shows the routes of a user using the web app:



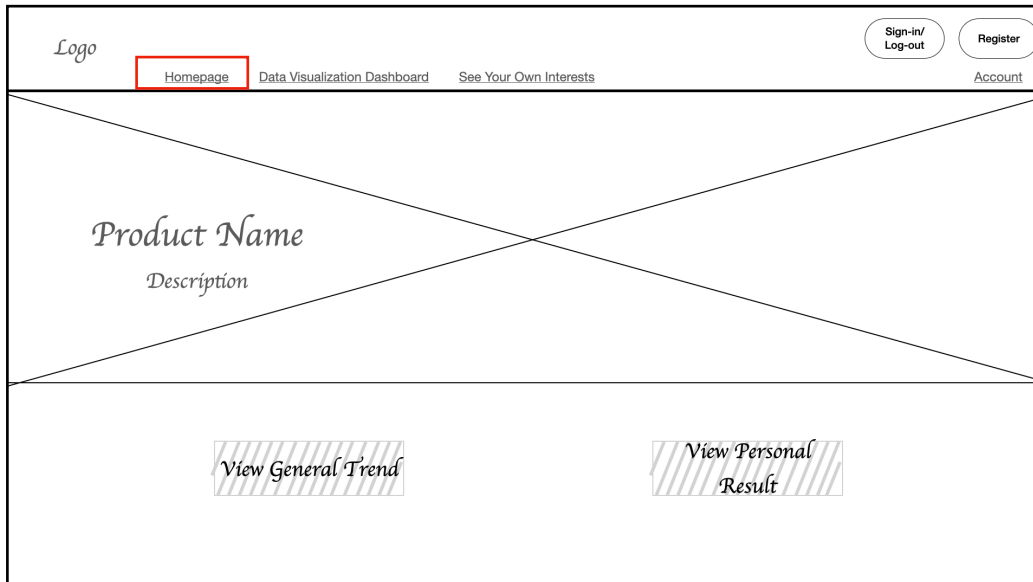
As shown in the flowchart, there are expected to be 7 pages in the Web APP:

1. **Homepage** - the first page that user see when clicking into the Web APP;
2. **Data Visualization Dashboard** - for every user to see the general gender pay gap trend as stated in the introduction section;
3. **Sign-up Page** - allows users to sign up an account;
4. **Log-in Page** - allows users to log in their own account;
5. **Information Input Page** - let users to select which industry / region / company size they are interested in and type their own information to generate comparison;
6. **Personalized Result Page** - show results based on input information;

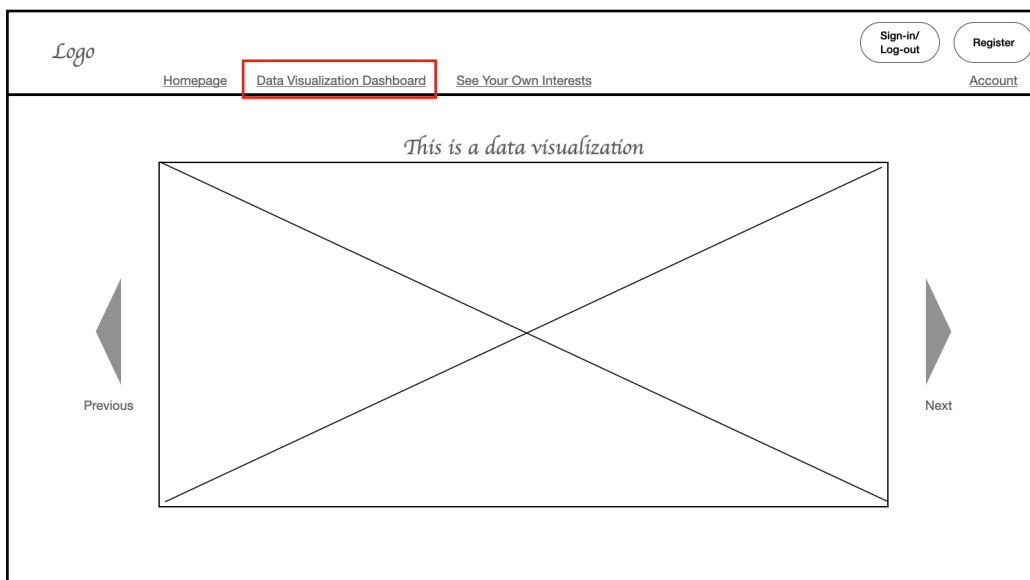
7. **Account Management Page** - let users to look at and manage information they saved, delete account, choose data sharing preference or change security information (password, registered email, etc.).

Wireframes for each page are created to help structure how the web app will look like.

### WIREFAME 1: HOMEPAGE



### WIREFRAME 2: DATA VISUALIZATION DASHBOARD



### WIREFRAME 3: SIGN-UP PAGE

Logo

Sign-in/  
Log-outRegister

HomepageData Visualization DashboardSee Your Own InterestsAccount

XXX

XXX

XXX

XXX

Save

Next

### WIREFRAME 4: SIGN-IN PAGE

Logo


Sign-in/  
Log-outRegister

HomepageData Visualization DashboardSee Your Own InterestsAccount

Username

Password

Verification Code



Sign-in

### WIREFRAME 5: INFORMATION INPUT PAGE

Logo

Sign-in/  
Log-outRegister

HomepageData Visualization DashboardSee Your Own InterestsAccount

Choose Your Role: ☐ Employer ☐ Employee ☐ Job Applicant

Select Your Industry: 

This is a dropdown

Select Your Region: 

This is a dropdown

Select Your Company Size: 

This is a dropdown

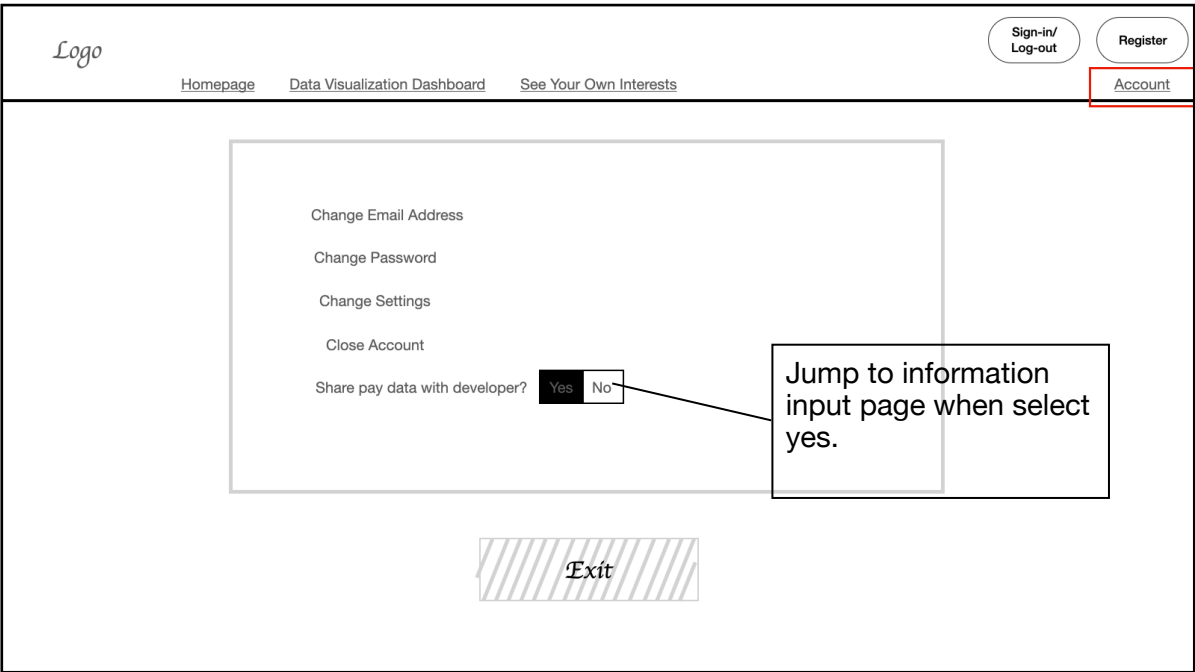
Save

Show Results

WIREFRAME 6: PERSONALIZED RESULT PAGE



WIREFRAME 7: ACCOUNT MANAGEMENT PAGE

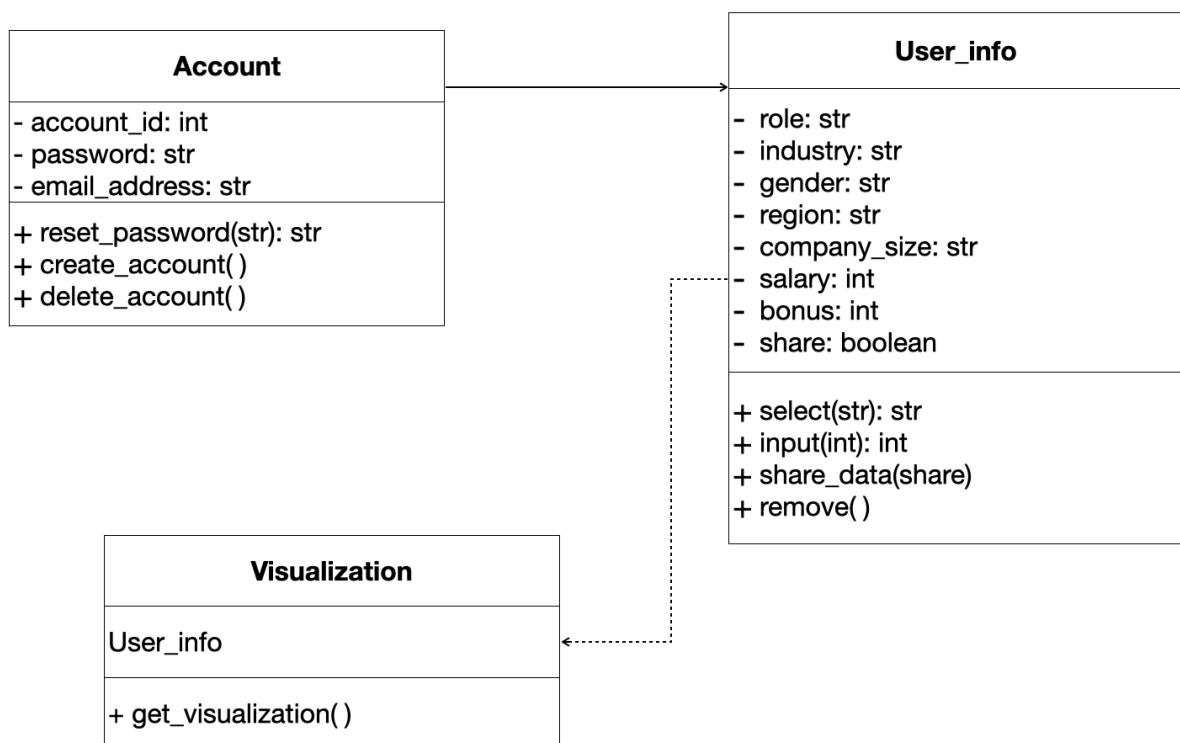


### 3. Application Design

A class diagram is drawn in order to help better understand what components are required in this system and their relationships. Meanwhile, since the project is about to be developed in Python, the class diagram can be really useful in the later coding stage as these classes can directly be used in software classes and objects. (IBM, 2021)

As shown in the interface design, the web app is expected to contain three major classes: **Account**, **User\_info**, and **Create\_result**.

- The Account class should include methods to set up and delete an account, which contains attributes such as user name and password;
- The User\_info class is used to set up personal information including the user role, industry, region, gender, and company size as shown in Wireframe 5. These attributes are used to generate personalized result page. Meanwhile, the class should also include method which allows users to share (or not to share) these data with the developer to help improve the database;
- The Create\_result class is expected to contain methods of drawing graphs for personalized result page. The attributes should come from data provided by User-info class.



As shown in the diagram, the Visualization class is dependent on the User\_info class and the User\_info class is associated with the Account.

Meanwhile, since wireframes are already created, it becomes clear and understandable to associate those diagrams when designing the application. In this case, Model-View-Controller (MVC) design pattern is chosen as all team members are new to software development, it is easier to breakthrough from those views. In addition, the method is easy to follow with 3 separated components, which allow team members to focus on one aspect at a time. The Model (Class) is

represented in the class diagram, Controllers (Routes) are listed in the table below with indication of corresponding View (Wireframes). Additionally, explanations for controller functions are shown.

Route	Wireframe View	Controller Function
'/'	Wireframe 1: Homepage	homepage() - direct to homepage
'/result/general'	Wireframe 2: Data Visualization Dashboard	go_general() - direct to data visualization dashboard;
'/user/signup'	Wireframe 3: Sign-up page	user_signup() - direct to the sign-up page
'/user/signin'	Wireframe 4: Sign-in page	user_signin() - direct to view of log in page
'/user/info'	Wireframe 5: Information input page	user_input() - direct to the information input page
'/result/personal'	Wireframe 6: Personalized result page	go_personal() - direct to personalized result page
'/result/personal/<account_id>'	Wireframe 6: Personalized result page	display_result(account_id) - show result based on user setting
'/user'	Wireframe 7: Account Management Page	account() - direct to account management page
'/user/<account_id>'	Wireframe 7: Account Management Page	account(account_id) - direct to account management page of a certain user when logged in

To describe the endpoints that the application is about to contain, HTTP methods are listed below.

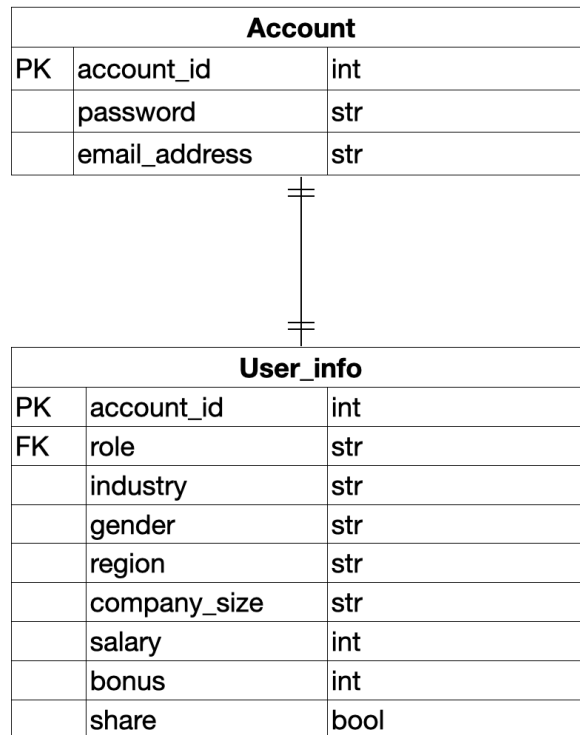
- **'GET /'** - retrieves home page;
- **'GET /result/general'** - retrieves data visualization results of general gender pay gap trend (the data visualization dashboard)
- **'POST /user/signup'** - create a new account
- **'Connect /user/signin/<account\_id>:password'** - identify whether account\_id and password match when user logs in
- **'GET /user/<account\_id>'** - displays the account management page of a certain user
- **'PUT /user/password/<account\_id>'** - reset the former set password
- **'DELETE /user/delete/<account\_id>'** - delete a registered account
- **'POST or PUT /user/info'** - create or update personal settings including role, region, industry, salary, etc.
- **'GET /result/personal/<account\_id>'** - retrieves data visualization results based on personal settings of a certain user account

## 4. Database Design

An Entity Relationship Diagram (ERD) is drawn to present the database of this application (only include database for REST API and exclude data prepared in coursework 1 for the visualization dashboard). The database should contain two entities: Account and User\_info and these two should be **one and only one** relationship between each other, a shared primary key for both entity can be <account\_id>. The Account entity saves basic registration information while the User\_info

entity saves personal features set by the user. A data dictionary is also linked below to provide descriptions and constrains of attributes in more detail.

#### ERD:



#### Data Dictionary:

Attribute	Data Type	Constraint	Description
<b>account_id</b>	Integer	Not Null Unique Must be 10 digits	A unique identifier generated automatically when a user register a new account
<b>password</b>	String	Not Null Minimum 8 characters Maximum 16 characters Combine at least two elements from numbers, capital letters, small letters and special characters	Password set by the user
<b>email_address</b>	String	Not Null Must end with @domain	Users' email address
<b>role</b>	String	Not Null Choose one and only one from: Employer, Employee, Job Applicant	Users' role



Attribute	Data Type	Constraint	Description
<b>industry</b>	String	Not Null Choose from a dictionary provided by developer (See coursework 1 data preparation)	Industry of users
<b>gender</b>	String	Not Null Choose from: Male, Female, Others and Prefer Not to Say	User gender
<b>region</b>	String	Not Null Choose from regions in the UK	Geographical location specified to region of user
<b>company_size</b>	String	Not Null Choose from a dictionary provided by developer (See coursework 1 data preparation)	Company size of the user's working company
<b>salary</b>	Integer	Not Null Type individual salary if choose employee as role Type average salaries for both male and female employees if choose employer as role Not available for users choose job applicant as role	Salary of employee type user; For employer user, they need to input both average salary of male employees and average salary of female employees
<b>bonus</b>	Integer	Not Null Type individual bonus if choose employee as role Type average bonus for both male and female employees if choose employer as role Not available for users choose job applicant as role	Bonus of employee type user; For employer user, they need to input both average bonus of male employees and average bonus of female employees
<b>share</b>	Boolean	Not Null	Identifies whether a user wants to share their User_info to the developing team and put into the database for general data visualization dashboard

## References:

IBM. (Mar 2021) *Class Diagrams* [Online] IBM. Available at: <https://www.ibm.com/docs/en/rsm/7.5.0?topic=structure-class-diagrams> [Accessed: Dec 2022]