Busybee

Project Design

Mason Krause, Mason Luna, Jacqueline Justice, Jason Moss

# Contents

* [Revised Requirements](#_k6xn1tt3exi7)
* [Design Description](#_uo15uvt403kn)
* [Appendix A: Block Diagram](#_nvi1k1sjkoar)
* [Appendix B: Component Diagram](#_aovusjjjsogg)
* [Appendix C: User Interface Storyboard](#_58t84rmnlxau)
* [Appendix D: Message Documentation](#_rnbufiuba7vm)
* [Appendix E: Storage Documentation](#_7bii06e5v9ed)
* [Appendix F: Data Flow Diagrams](#_rzbdol2qhzn6)
* [Appendix G: Data Model Diagram](#_e0swy5yv1hei)
* [Appendix H: CI/CD Pipeline Diagram](#_nn9ajcpoqkko)

# 

# Requirements

## Login Form

### Header text for ‘login’/Login Section with Name of application/logo

### Text box for email

### Text box for password

#### Text in the password field will be masked using bullets.

### Invalid Email/Password Text Label

#### If the user was denied access to the dashboard due to an invalid email or password, a text label will appear and display (Email or Password invalid) in red letters (See [Account Validation](#_gvbr0jnvohp7)).

### Login Button

#### Input Validation: Email

* + - 1. The Email text box must NOT be empty
      2. HTML attributes will be used to ensure a valid email is entered
      3. If user clicks Login button and email is invalid, a text label will appear and display “Invalid email” in red letters under the email input box.

#### Input Validation: Password

* + - 1. The Password text box must NOT be empty
      2. Passwords must contain 8-18 characters, at least one number, and at least one capital letter.
      3. Password character limit is 18 characters
      4. If user clicks Login button and password entry is too short or doesn’t contain a required character, a text label will appear and display “Invalid password” in red letters under the password input box.

#### Account Validation

* + - 1. The user’s entered email and password will be validated as a user account using Supabase Auth.
      2. If the user is not authenticated, a text label will appear and display “Invalid login credentials. Please try again, or Sign Up instead” in red letters at the top of

#### Post-login Navigation

* + - 1. Once the user’s account information is validated, they will be navigated to the Dashboard’s homepage   
         (see [Dashboard - Homepage](#_zc9heofpemth)).

### Forgotten Password Link

#### A link with the text “Forgot Password?” will be present and navigate the user to the account recovery page

#### The account recovery page will have a text field for users to enter their email and reset their password.

### Sign Up Now Link

#### A link labeled “Sign up Now” will navigate to the registration form (See [Registration Form](#_tn91yjjsxeos))

## Registration Form

### Header text ‘Register for BusyBee’

### Text input box for first name

### Text input box for last name

### Text input box for email

### Text input box for password

### Text input box for confirming password

### Submit button

* + 1. On click, validate/sanitize input (see [1.5.1](#_eh14hcyggu5) and [1.5.2](#_ohpks9xxeuzr))
       1. If input is valid, send data to the Supabase server.
          1. If the user is already registered, redirect to the login form with an alert that the account already exists with that email
          2. If the user is not already registered, add them into the database, redirect them to the login form with an alert that the account was created successfully
       2. If input is invalid, clear the invalid field and alert the user that a valid input is required for that field. The alert should be about the invalid field and appear in red text.
       3. If multiple fields are invalid, alert the user to the uppermost invalid field.

### Already have an account? Link

* + 1. Link labeled ‘Already have an account? login instead’ will navigate to the Login Form (See [Login Form](#_y3jd60jerrs5))

## Dashboard

### Header

* + 1. A Busybee logo that navigates to the [Dashboard - Homepage](#_zc9heofpemth) on user click
    2. A label with the text “Welcome,” followed by the user’s first name
    3. Sign Out button that signs the user out and navigates to the [Login Form](#_y3jd60jerrs5) on user click

### Footer

* + 1. A Busybee logo that navigates to the Dashboard - Homepage on user click
    2. Resource Section
       1. Static Contact Us Page link
       2. Static FAQ Page link
       3. Static About the Team Page link

### Easy Navigation Section (Left-Hand side)

* + 1. Contains a vertical list of navigation buttons
    2. All buttons will change the Page View into the selected dashboard page on user click.
    3. Buttons, in order
       1. [Jobs Page](https://docs.google.com/document/d/1lmBX_6J8kQaLSKshkKaCTw0_jkONi-QCqLXGEos6NgY/edit?tab=t.0#heading=h.y99h5j5v61nf)
       2. [Statistics Page](https://docs.google.com/document/d/1lmBX_6J8kQaLSKshkKaCTw0_jkONi-QCqLXGEos6NgY/edit?tab=t.0#heading=h.o56q4rapzosq)
       3. [Documents Page](https://docs.google.com/document/d/1lmBX_6J8kQaLSKshkKaCTw0_jkONi-QCqLXGEos6NgY/edit?tab=t.0#heading=h.vesbg9bz4fgl)
       4. [Groups Page](https://docs.google.com/document/d/1lmBX_6J8kQaLSKshkKaCTw0_jkONi-QCqLXGEos6NgY/edit?tab=t.0#heading=h.c26df1sxs9fn)
    4. Page View
       1. The page view will be rendered according to the currently selected dashboard page (Jobs, Statistics, Documents, or Groups)

## Dashboard - Home page

### Statistics Quick View

* + 1. Contains a preview of the statistics based on the most recently added data.
    2. On user click, Statistics Page will open   
       (See [Dashboard - Statistics](https://docs.google.com/document/d/1lmBX_6J8kQaLSKshkKaCTw0_jkONi-QCqLXGEos6NgY/edit?tab=t.0#heading=h.o56q4rapzosq))

### Jobs Quick View

* + 1. Contains a preview of the most recent jobs that the user has added
    2. On user click, Jobs Page will open   
       (See [Dashboard - Jobs](https://docs.google.com/document/d/1lmBX_6J8kQaLSKshkKaCTw0_jkONi-QCqLXGEos6NgY/edit?tab=t.0#heading=h.y99h5j5v61nf))

### Documents Quick View

* + 1. Contains a preview of the user’s most recently uploaded documents
    2. On user click, Documents Page will open   
       (See [Dashboard - Documents](https://docs.google.com/document/d/1lmBX_6J8kQaLSKshkKaCTw0_jkONi-QCqLXGEos6NgY/edit?tab=t.0#heading=h.vesbg9bz4fgl))

### Groups Quick View

* + 1. Contains a preview of the user’s Groups at the bottom of the page
    2. On user click, Groups Page will open   
       (See [Dashboard - Groups](https://docs.google.com/document/d/1lmBX_6J8kQaLSKshkKaCTw0_jkONi-QCqLXGEos6NgY/edit?tab=t.0#heading=h.c26df1sxs9fn))

## Dashboard - Jobs Page

### Jobs Information Section

* + 1. Contains a count of all jobs added displayed in the top right of the Jobs Page
    2. Add job button
       1. Contains a plus sign and label with “Add a job…”
       2. On click, a form will pop-up for the user to input all information related to a job entry (See [Entities: Job](https://docs.google.com/document/d/1lmBX_6J8kQaLSKshkKaCTw0_jkONi-QCqLXGEos6NgY/edit?tab=t.0#heading=h.rafjblkoa4n6) 9.1.1.3.2 - 9.1.1.3.12)
          1. Company Name (text box)
          2. Job Title (text box)
          3. Remote or not (check box)
          4. Country (drop-down list)
          5. State (drop-down list)
          6. City (text box)
          7. Date Posted (date picker)
          8. Platform (text box)
          9. Estimated Salary (text box)
       3. Save Button that, on user click, will save the information as a Job Entry and return to the Jobs Page
          1. If a required field (company\_name, job\_title, remote) is left blank, a text label will appear and display “Must enter a valid <field\_name>” above the required field in red text.

### Jobs List

* + 1. Displays all recently added or edited jobs that meet the user’s filter requirements with the following information
       1. Job Title
       2. Date Applied
       3. Current application status
    2. Each job in the job list is selectable, and will display the Job Popup once selected

### Job Popup

* + 1. Displays all information stored in the selected job’s entry (Company Name, Job Title, remote, city, state, country, date posted, date applied, platform, estimated salary, application status, company notes, interview notes)
    2. Edit Button
       1. Users will be able to edit all information in the selected job’s entry
       2. The edit button will be replaced by a save button that the users must press to save any changes made
    3. Delete Button
       1. on click, displays a confirmation popup, asking if the user is sure with a button labeled ‘Yes’ and a button labeled ‘No’
          1. When ‘Yes’ is clicked, the job entry is deleted permanently and the popup closes.
          2. When ‘No’ is clicked, the confirmation popup closes.

### Filter and Sort

* + 1. Users will be able to filter the jobs list based on month applied, application status or salary range, where all job entries not matching the criteria will not be shown
    2. Users will be able to change how the job entries are sorted, based on either date applied or alphabetically.

## Dashboard - Statistics Page

### Statistics Graphics

* + 1. Displays the following statistics in square graphics in the body of the page
       1. Total number of jobs applied to
       2. Applications Submitted this month
       3. Average Applications submitted per month
       4. Application-to-Interview Ratio
       5. Application-to-Offer Ratio
       6. Interview-to-Offer Ratio

## Dashboard - Documents Page

### Add Documents Button

* + 1. Contains a plus sign and label with “Add documents…”
       1. On user click, a filepicker will popup and allow user to upload PDF or DOCX files and add it to their documents

### Documents List

* + 1. All documents the user has added will be listed ordered by how recently they were uploaded.
    2. A user can select a document to download it.
    3. All documents will have an ‘X’ symbol on the far right of the listing for users to delete documents
       1. On click, a confirmation popup will appear and ask if the user is sure with a ‘Yes’ button and a ‘No’ button.
          1. If the user clicks ‘Yes’, the document is deleted and the popup closes.
          2. If the user clicks ‘No’, the popup closes

## Dashboard - Groups Page

#### Tab-based component for Groups

* + 1. Displays selectable name labels of all the user’s created groups
    2. The selected group will be highlighted similar to Chrome tabs or Microsoft Excel tabs
    3. A button with a plus symbol will be present at the end of the list of tabs to add a new group
       1. On click, a popup will appear for adding a new group with the following information
          1. Group Name (text box)
          2. Group Start Date (text box)
          3. Group End Date (text box) (optional)
       2. When the user is finished inputting the required information, they can click a button labeled ‘Add’ at the bottom of the popup to add the newly created group to their groups.
    4. Any job that has been previously added can be added to any group.

## Database

### Structured Database

* + 1. Entities
       1. User
          1. user\_id : uuid (Primary Key)
          2. first\_name : string
          3. last\_name : string
          4. email: string
          5. password: string
       2. Group
          1. group\_id : int (Primary Key)
          2. group\_name : string
          3. group\_start\_date : date
          4. group\_end\_date : date (NULLABLE)
          5. jobs : list[Job]
       3. Job
          1. job\_id : int (Primary Key)
          2. company\_name : string
          3. job\_title : string
          4. remote: boolean
          5. job\_city: string
          6. job\_state: string
          7. job\_country: string
          8. date\_posted : date
          9. date\_applied : date
          10. platform : string
          11. estimated\_annual\_salary : integer (NULL)
          12. status : string (categorical)
          13. notes : string

## Storage

### Uploaded .PDF and .DOCX files will be stored in Supabase Storage.

* + 1. Add document buttons will be available in the Dashboard Home Page in the “Documents” section and on each Job entry
       1. On click, a file selection window will pop up so the file can be uploaded
    2. Replace document buttons will be available in the dashboard section “Documents” and on each Job entry if a file was previously uploaded
       1. The newly uploaded document will replace the old one in storage, the old one will be deleted

# 

# Design Description

The Busybee system is designed to streamline the job search process for job seekers by providing an organized dashboard where users can manage job applications, documents, and related statistics. The system follows a client-server model, utilizing React for the front end to ensure a responsive user experience, and Supabase as the backend for authentication, database management, and storage solutions. The system’s architecture is built to provide scalability, reusability, and maintainability while enabling rapid development to accommodate the project’s time constraint.

The system is made up of the following key elements: the frontend, backend, and database. These elements work together to provide users with a structured job management experience. As shown in the [Block Diagram (Appendix A)](#_nvi1k1sjkoar) and the [Message Documentation (Appendix D)](#_rnbufiuba7vm), the frontend communicates with the backend via the Supabase JavaScript client by sending queries, and Supabase manages authentication, database operations, and file storage, and returns JSON response objects.

## Frontend

The frontend is developed using React and TypeScript to ensure a dynamic and maintainable interface. It employs a modular component structure that allows for easy scalability and reusability. The user interface (UI) is designed with an intuitive dashboard that provides easy navigation between the various functionalities offered by the app. User authentication and session management are handled by Supabase Authentication.

## Login/Registration

Not only are user authentication and session management handled by Supabase Authentication, it also offers the ability for the user to request a change to their user email or their password. BusyBee takes advantage of these services and the user can request changes to these fields via links in the login form.

## User Navigation

### React Components

As shown in the [Component Diagram (Appendix B)](#_aovusjjjsogg), App.js serves as the user’s entry point into the application, and it will be in charge of routing the user to and from all pages within the application. App.js will use the routing functionality from React Router to navigate the user through the login process and to the dashboard. Once the user is at the dashboard, the dashboard will render itself and its child components, which are the easy navigation section, header, footer, and dashboard view components. There are five possible options for the dashboard view component: jobs, groups, stats, documents, or main. The view currently selected by the user will be the component rendered.

### Static Pages

The user can interact with BusyBee outside of the primary React Components in the form of three static pages found in the footer of any dashboard page. These forms can be seen in the [Component Diagram (Appendix B)](#_aovusjjjsogg), and App.js will route the user to either the Contact Us, FAQ, or About the Team pages on user click. BusyBee is built to be mostly a single-page system, with all navigation occurring within the dashboard, so these static pages represent the only form of navigation outside of the dashboard available to the user.

## Backend

On the backend, Supabase also serves as the database cloud host, providing PostgreSQL as the database solution for structured data storage. The database consists of multiple tables that store information about Users, Jobs, Groups, and Documents. The [Entity Relationship Diagram (Appendix E)](#_7bii06e5v9ed) outlines the relationships between the entities. The User table maintains account details such as first name, last name, email, and password. The Job table stores data about the job applied to by the user with fields like company name, job title, location, estimated salary, etc. The Group table stores information about a user created Group, which has fields like name, start date, and end date. And the Document table has information about the users uploaded documents such as a document name, the path to the bucket where the document is stored, and the date the document was uploaded.

### Data Model

In the [Data Model Diagram (Appendix G)](#_e0swy5yv1hei), how the user’s job, group, and document data will be represented from the database on the backend can be seen. When the user logs into BusyBee, the system will query the database to fetch all data related to the user’s id. This data includes all jobs the user has added to BusyBee, any documents that the user has uploaded, and all groups the user has created. These sets of data will create collections of object instances, defined as JobModel, DocumentModel, and GroupModel respectively, that BusyBee will manage as the user navigates the application. As the user makes decisions that change the state of these models the associated model will adjust its state locally and on the database. For example, if a user creates a new job entry for a recently filled application, BusyBee will call createJob from the JobModel object, create the job instance, add that job instance to the local model, and update the database. The update to the local state will be seen by the JobsPage react component, which will update its UI accordingly.

### Data Flow

How the user navigates through the front end of the application can be understood by examining how data flows through the back end of BusyBee. As shown in DFD0 of the [Data Flow Diagram (Appendix F)](#_rzbdol2qhzn6), the user has six ways of interacting with BusyBee available to them. They can login to their account, register for a new account, manage their jobs, manage their groups, manage their documents, or view their statistics. For job management and group management, the user is able to create, view, update or delete any entries in the system. Updating jobs can include any publicly viewable information related to the job, which can be seen in [Appendix D: Message Documentation (Update Job Message)](#_zhji5hfqr63x). Groups can also be updated, but the only changeable field is the group’s name.

The nature of storing a user’s documents means it is not feasible for the user to edit their documents inside of BusyBee. Because of this constraint, the user will not be able to update their documents inside of BusyBee, but they will still be able to upload, view or delete their documents from Supabase Storage by navigating through the Documents component of the dashboard. Functionally, the creation, reading, and deletion of documents is similar to the processes seen in the jobs and groups systems.

Of the four dashboard related modes of interaction, the user’s statistics is the most unique. On login, the statistics component of BusyBee will compute all statistics from the user’s job entries and formulate a report, assigning values to the associated widget within the statistics page React Component. The user may view these statistics at any point by navigating to the statistics page. Also, if the user is interacting with the job entry management system, the statistics component will listen for any changes to the JobModel so that it may reformulate its report.

## System Functionality

### Dashboard – Main

One of the core functionalities of the Busybee system is the dashboard, which serves as the primary interface for users. The dashboard contains various sections, including a navigation panel on the left that provides quick access to job management, statistics, documents, and groups. From the dashboard, users can see quick views of their jobs, groups, docs, and stats. They can also click directly on the quick view to pull up that respective page, similar to if they use the navigation panel.

### Dashboard – Jobs

The job management features enable a user to add, edit, filter, and sort job applications, providing an organized job application tracking process. All jobs will be fetched from the database on login and generated as a collection of Job objects, known as JobModel. These job objects can be viewed here on the Jobs page of the dashboard, where they can be sorted, edited, or deleted. Jobs can also be assigned to groups by editing them, and users will be able to view job entries that only belong to specific groups (i.e. only show the user jobs that belong to the Clarksville, TN group).

### Dashboard – Groups

The groups system is essential for users to effectively organize their job applications. Groups are identified by the system by their id, but users will identify groups by their name. On user login, all groups associated with the user id will be fetched from the database, and built into a GroupModel, a collection of groups used by BusyBee to manage them. Once the GroupModel is created, the job management system uses it to generate lists of jobs associated with the group id so that the user can view all jobs specific to that group when it is selected on the jobs page.

### Dashboard – Documents

The document management system allows users to upload, view, and delete documents, which are stored in Supabase Storage and can be linked to a job, or independent of a job. For example, if a user wants to keep a basic copy of their resume which they can download and edit for new job applications, they can keep that base copy without associating it with a job. If they make a resume for a specific job application, they can store that resume with that job, so that when they view the job, they can see their documents related to that job. In the database, this simply means that the Document table holds a foreign key reference to the Job table, which is nullable. So if a Document instance has a null job\_id, that means it is an independent document, and if it has a job\_id then it is associated with the corresponding job.

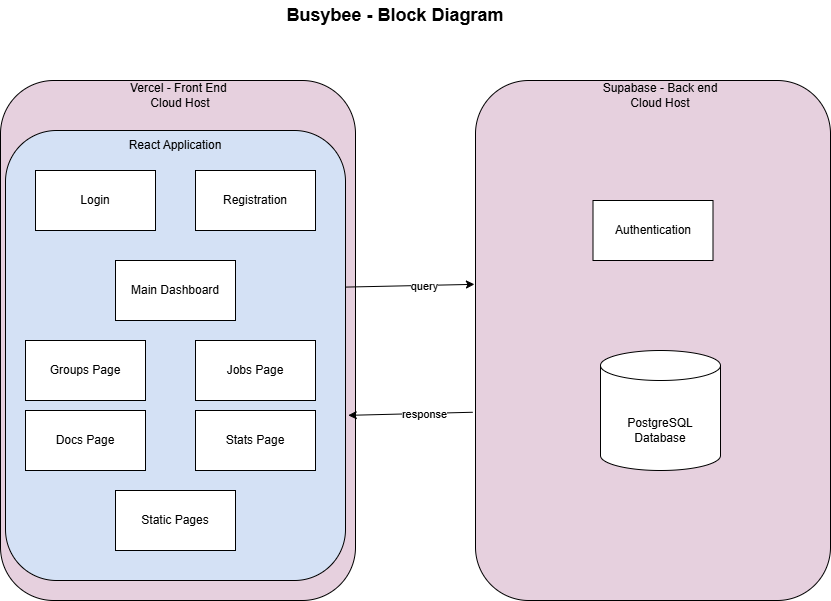
### Statistics

The management of jobs, groups, and documents use create, read, update and destroy (CRUD) functionality for each entity. This structure and organization is useful to the user, but ultimately it is their data being displayed back to them. The true value of the Busybee system comes from the statistics processing, which provides users with valuable feedback to inform their job search. For example, we provide the user with an application to interview ratio, and an interview to offer ratio. If a user has a very low application to interview ratio, it will help them understand that they need to fix up their resume, as that is keeping them from attaining interviews. If their application to interview ratio is high, but the interview to offer ratio is low, then they know that they need to work on interviewing. Additionally, the statistics calculated will include the total number of jobs applied to (and tracked through the app), number of jobs applied to this month, the ratios mentioned above, and an application to offer ratio.

When the user logs into BusyBee and their job data is being fetched, BusyBee will calculate all stats needed for the statistics page. Choosing to calculate stats on login means there will ensure a responsive application because there is no need to run any business logic when the user chooses to view the statistics page. Also, if the user creates a new job entry, updates the status of a job, or deletes a job, BusyBee will recalculate all statistics needed for the statistics page. This way, the statistics page will always be guaranteed to show accurate values to the user.

Overall, the Busybee system is designed to empower users through their job search and provide them with the tools they need to be organized and efficient.

# Appendix A: Block Diagram



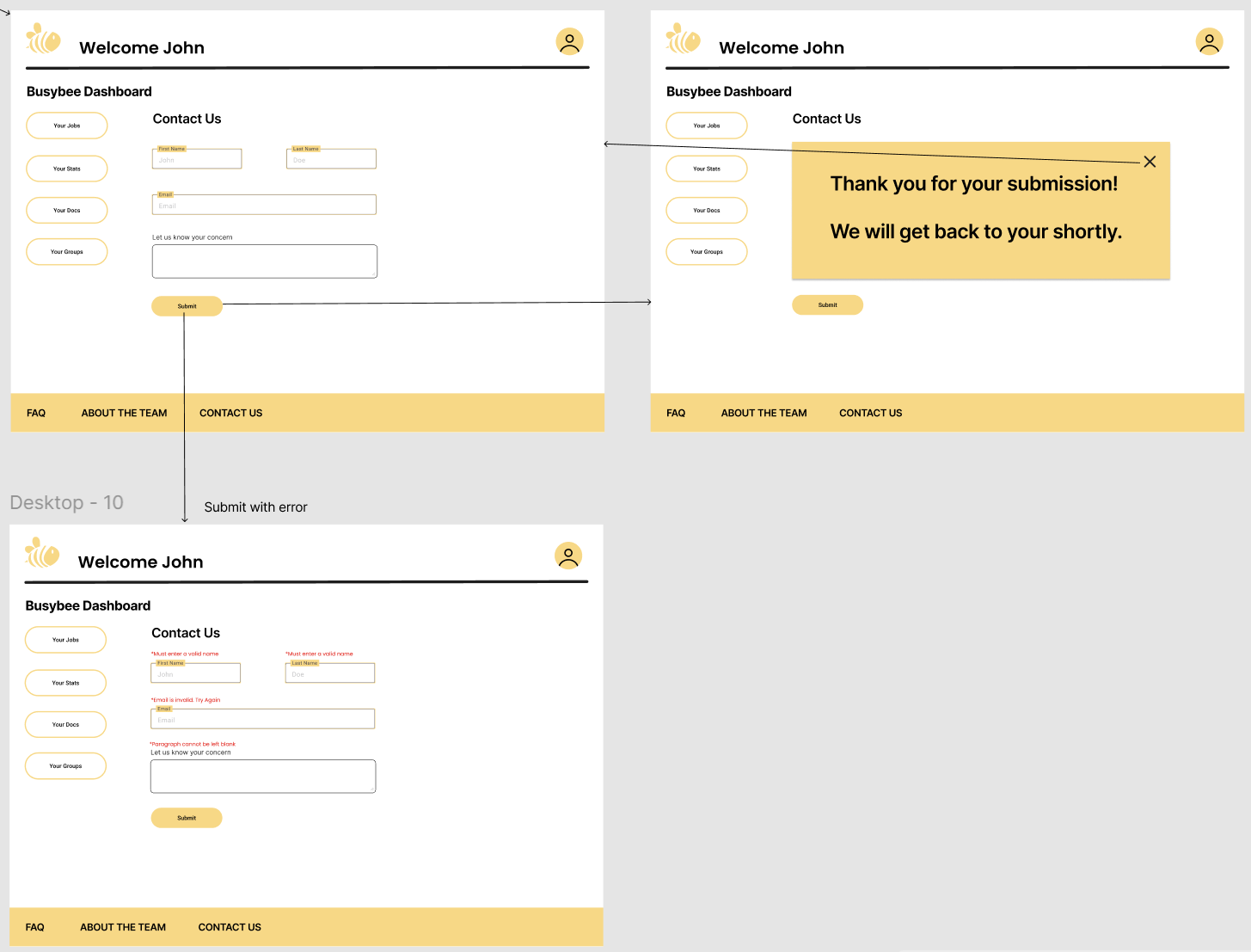
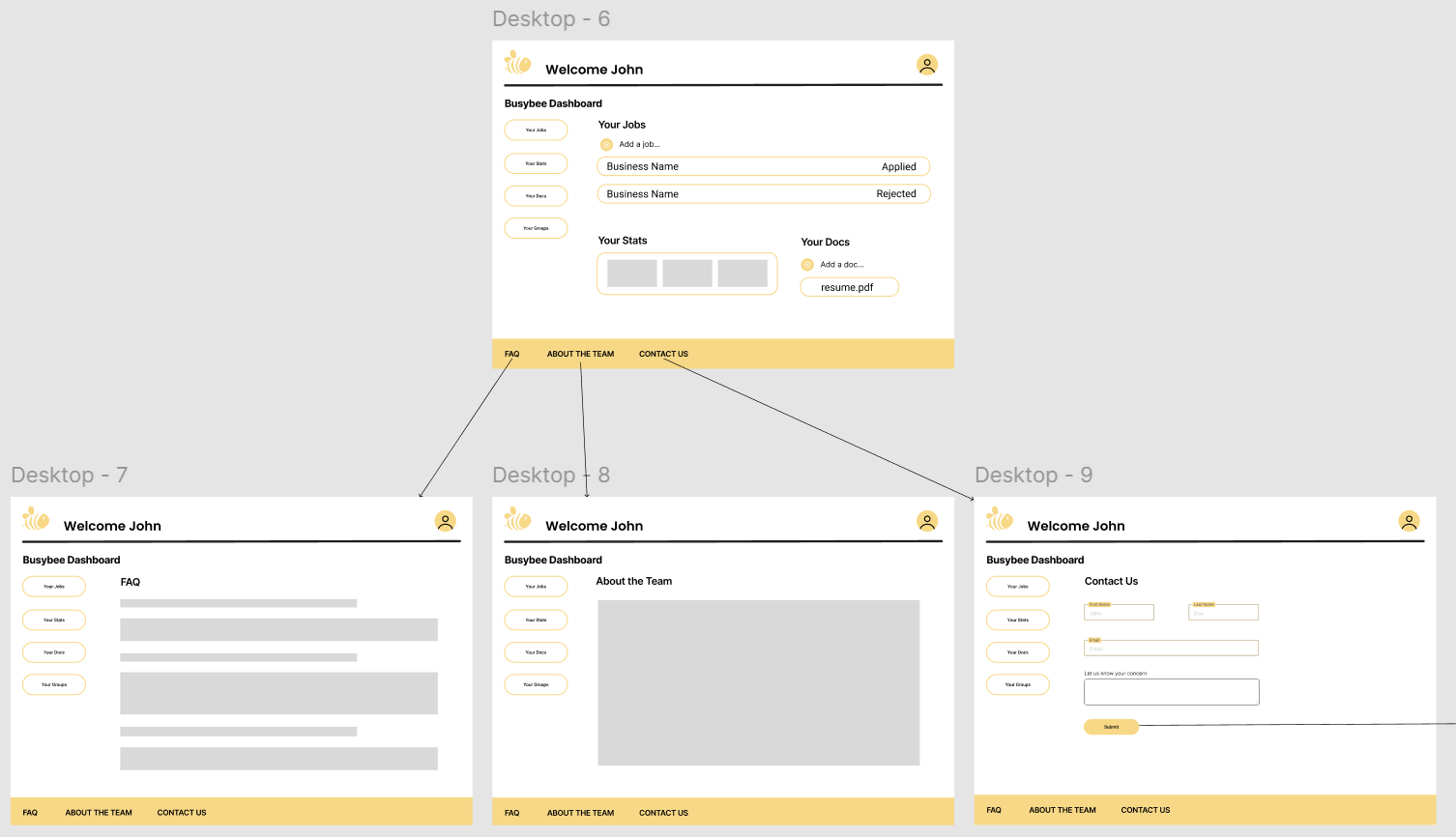
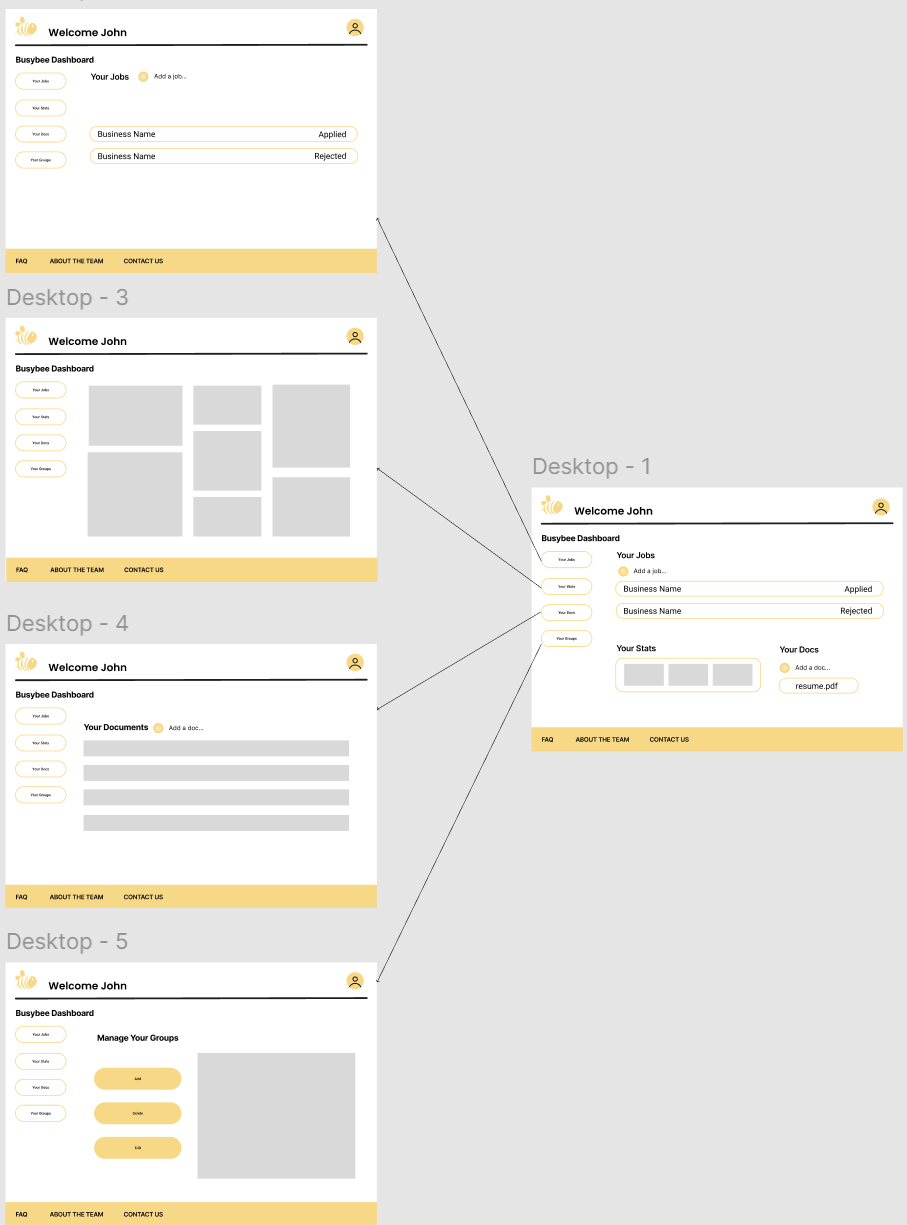
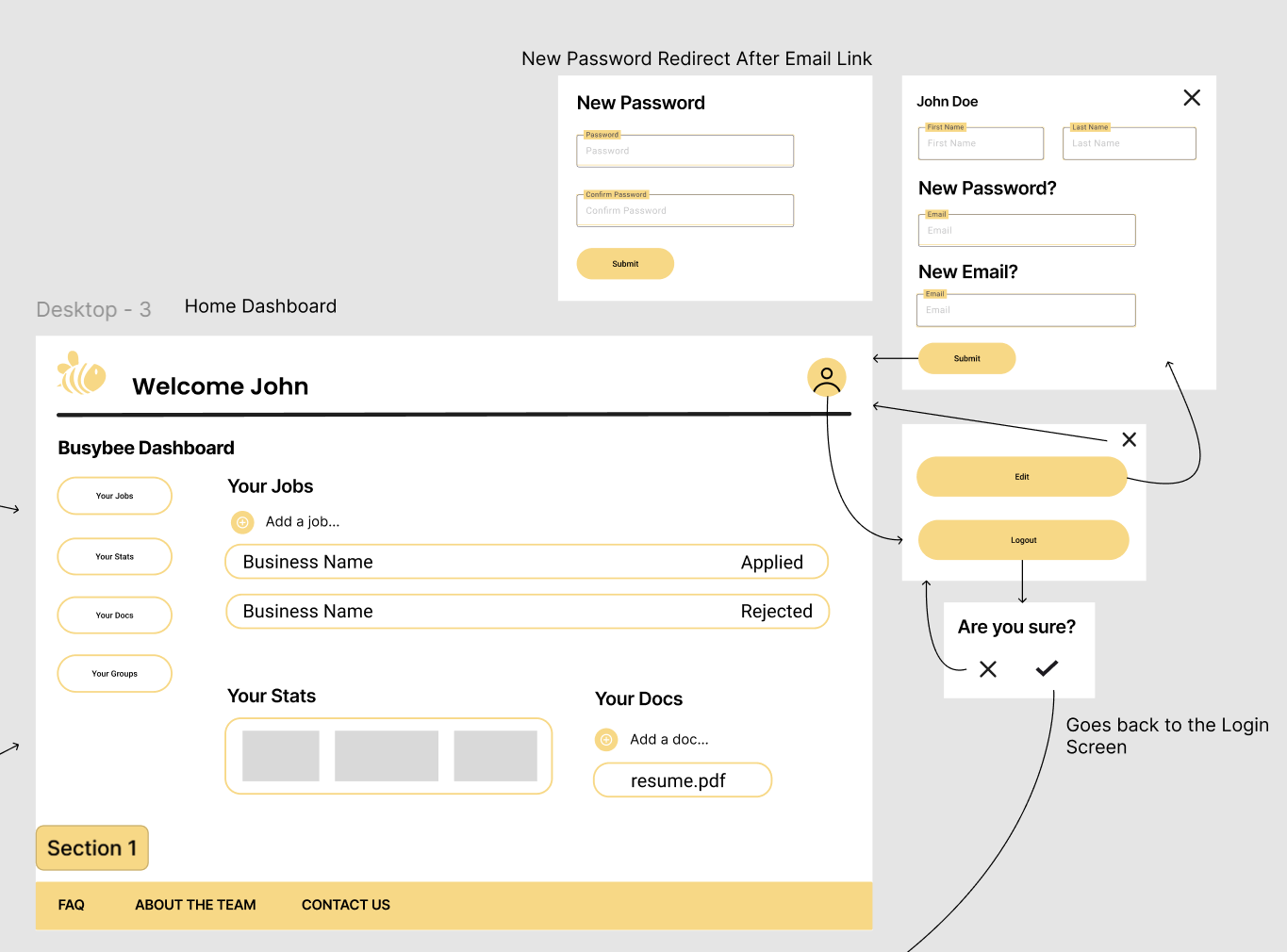
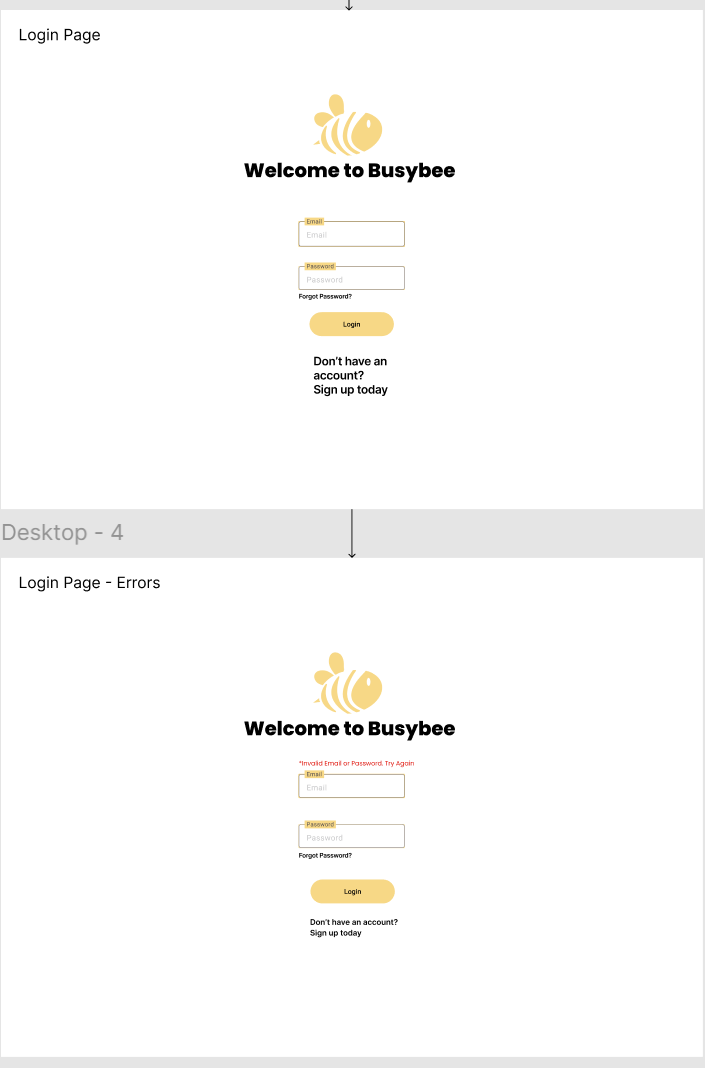
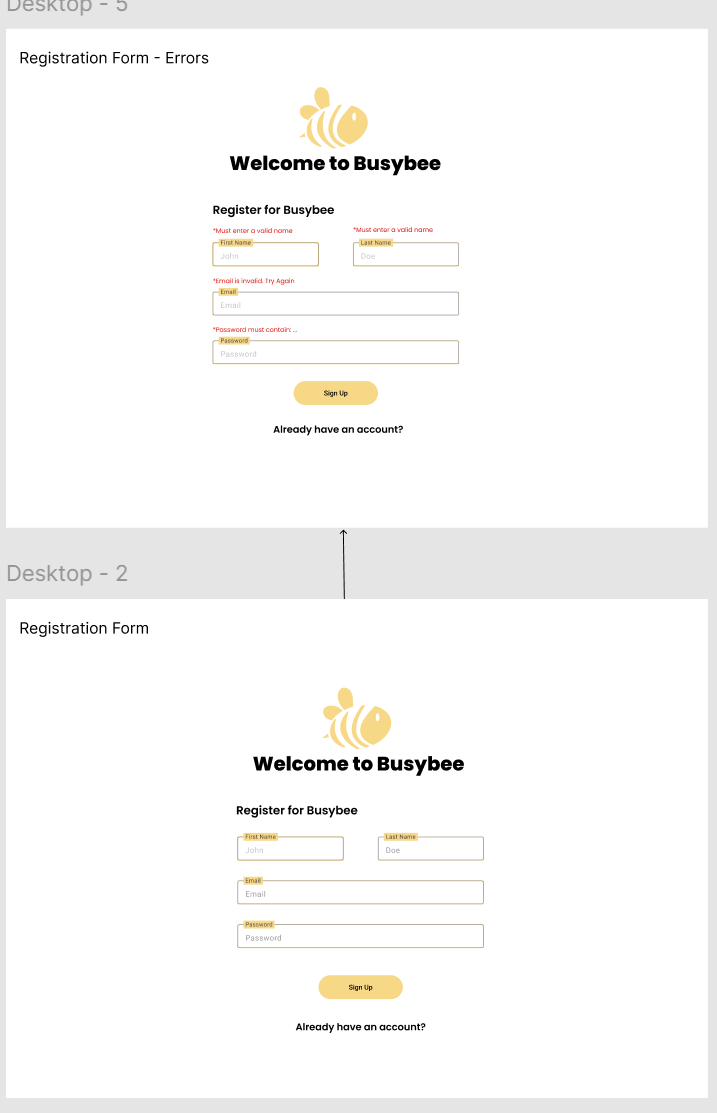
# Appendix B: Component Diagram

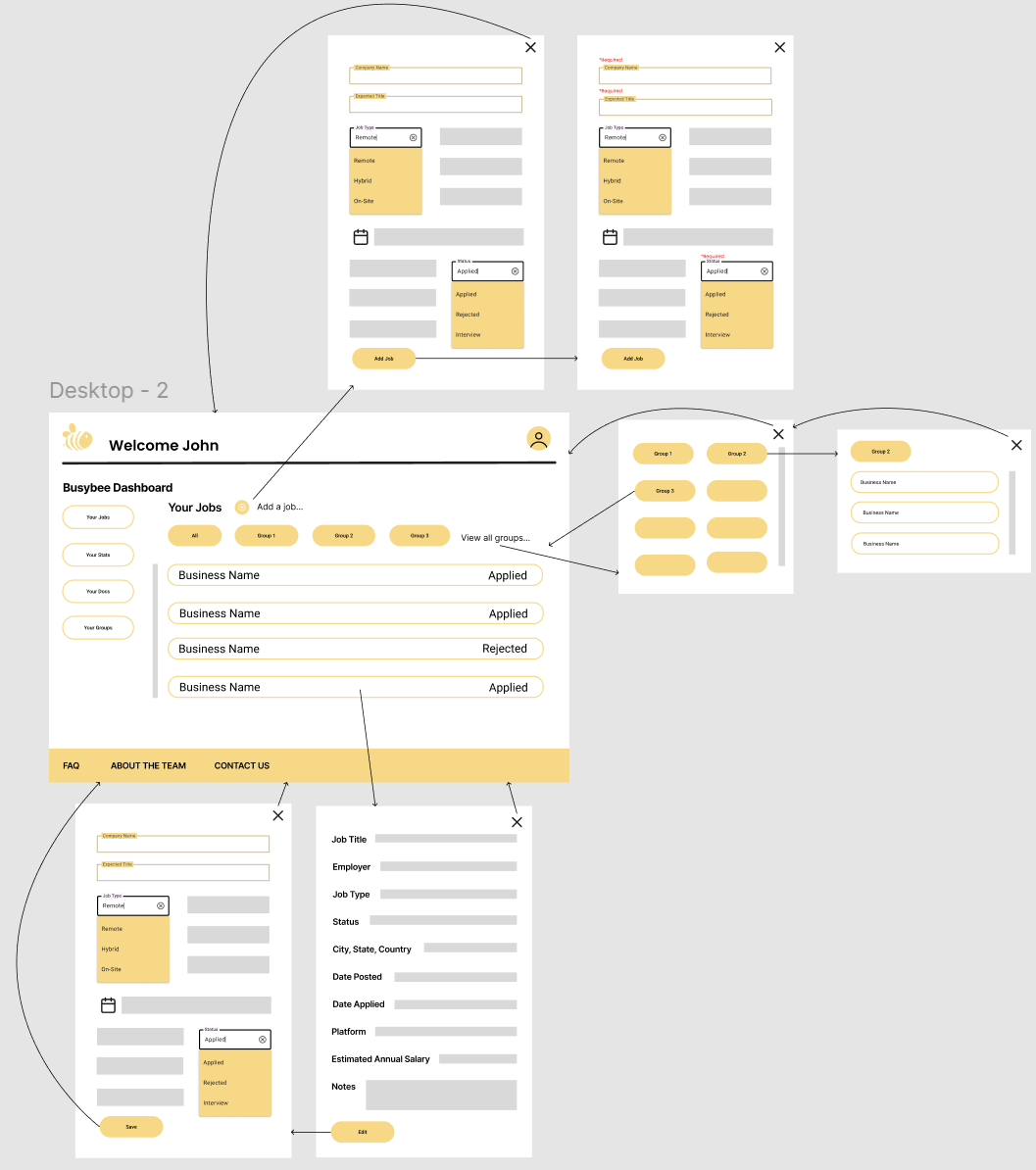
## Component Diagram

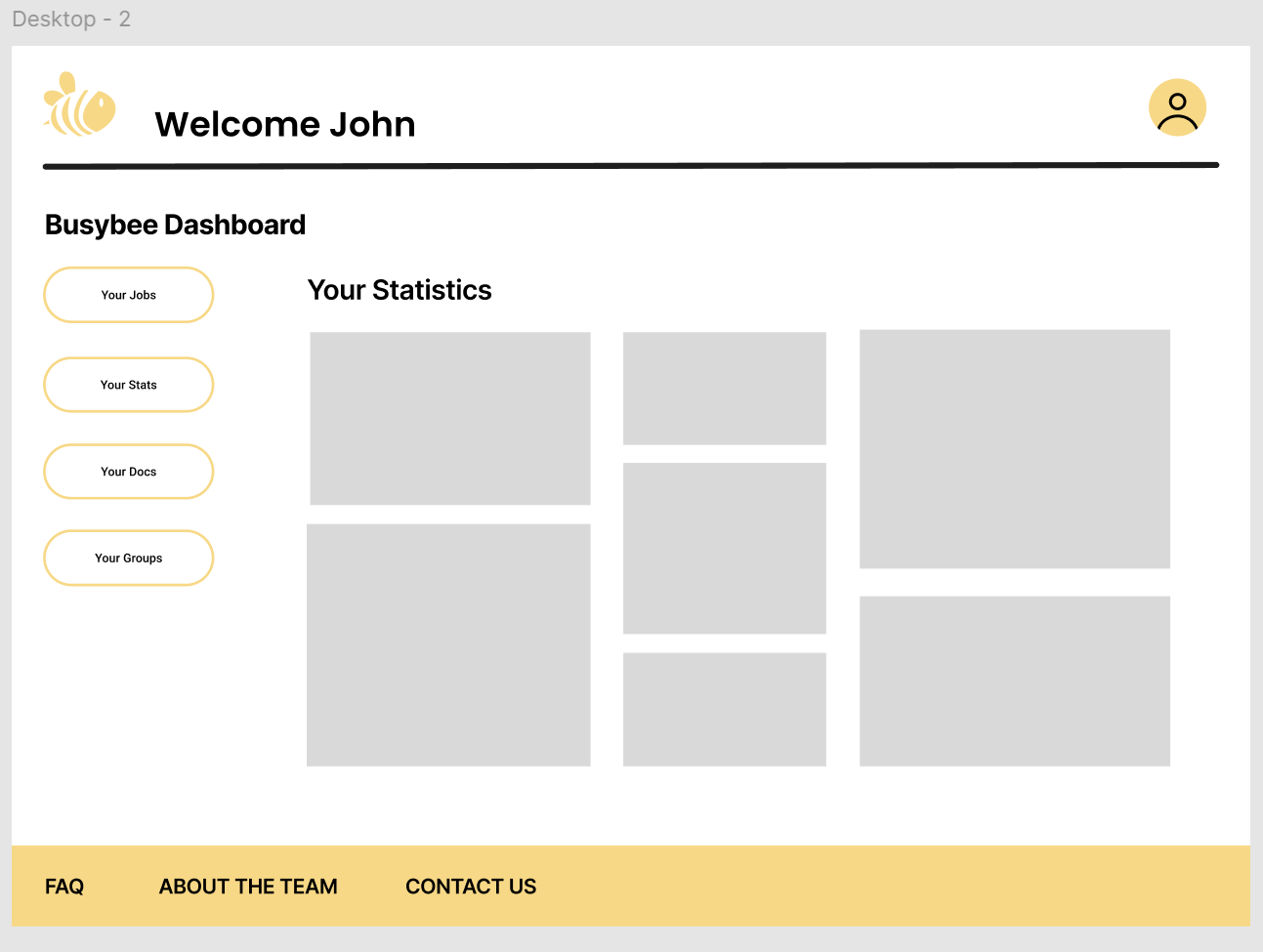
Refer to **Documentation/diagrams/component/component-diagram.drawio.png** for the component diagram

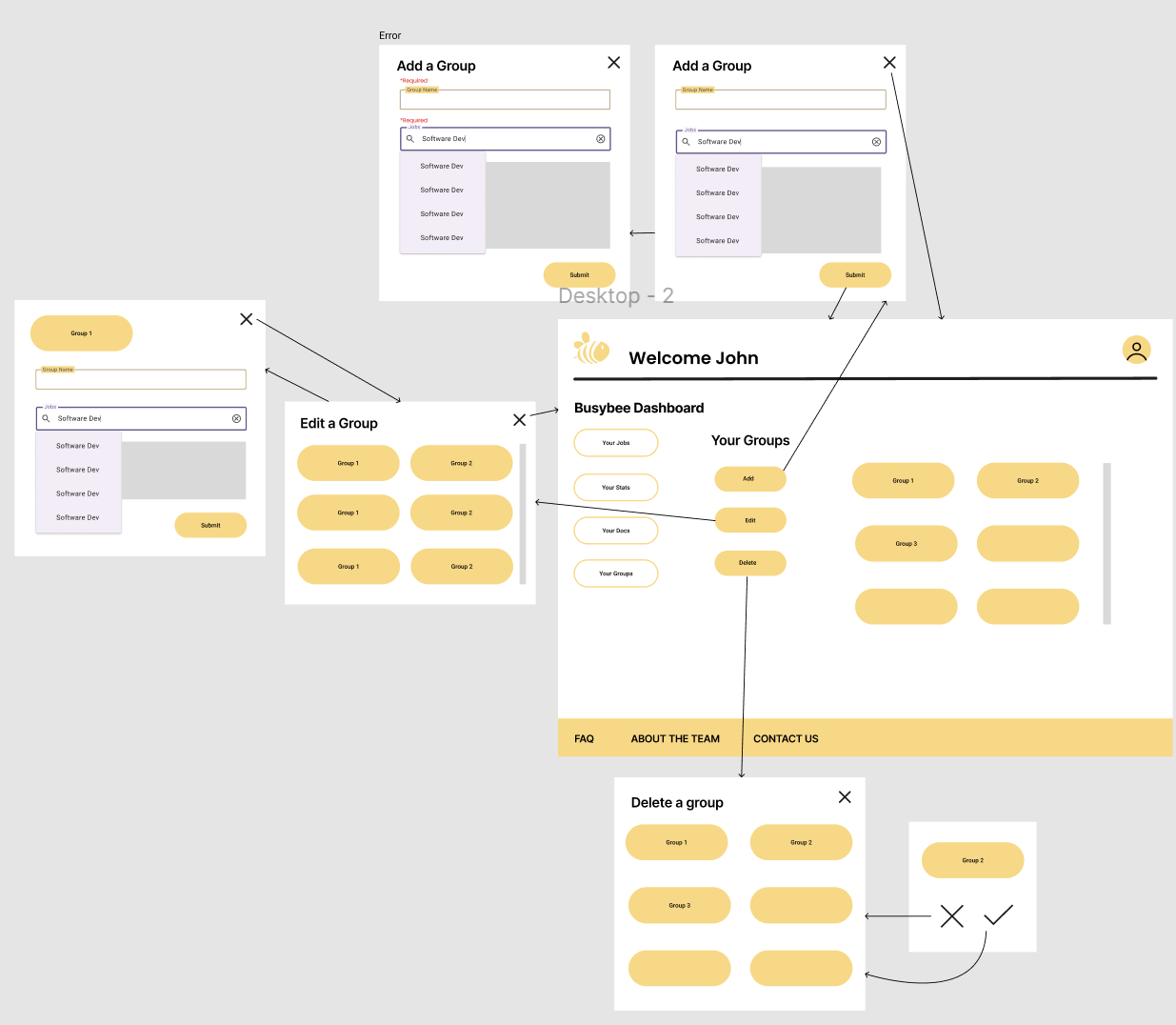
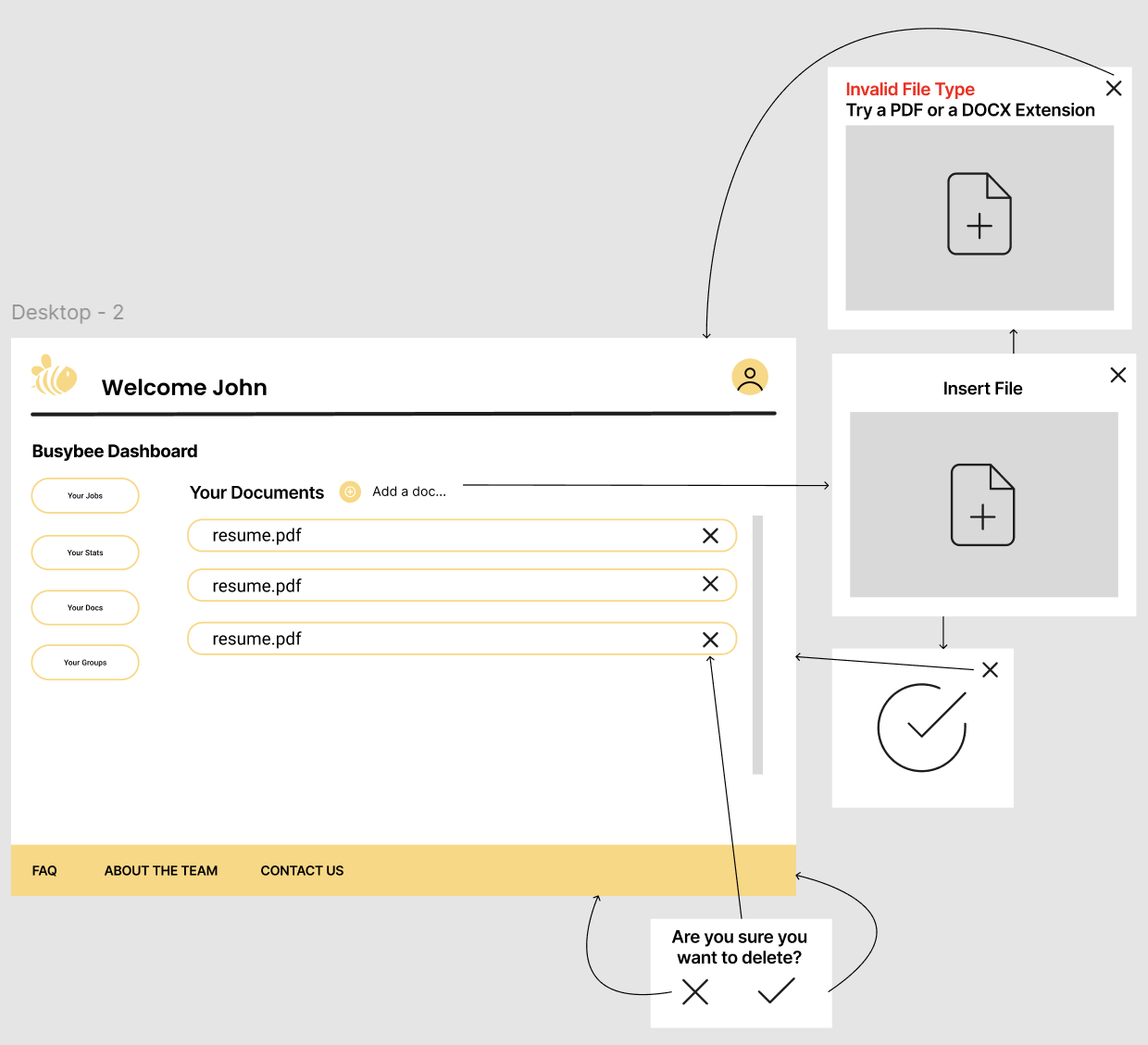
# Appendix C: User Interface Storyboard

# 









# 

# Appendix D: Message Documentation

## Login Message

**Client**: supabase.login(email: string, password: string)  
**Server**: { data, error }  
 data = {  
 User: {  
 user\_id: uuid,  
 firstName: string,  
 lastName: string,  
 email: string  
 },  
 message: string,  
 status: integer  
 },  
 error = {  
 message: string,  
 status: integer  
 }

## Forgot Password Message

**Client**: supabase.forgotPassword(email: string)  
**Server**: { data, error }  
 data = {  
 message: string,  
 status: integer  
 },  
 error = {  
 message: string,  
 status: integer  
 }

## Sign Up Message

**Client**: supabase.signUp(email: string, password: string, firstName: string, lastName: string)  
**Server**: { data, error }   
 data = {  
 User: {  
 user\_id: uuid,  
 firstName: string,  
 lastName: string,  
 email: string  
 }  
 },  
 error = {  
 message: string,  
 status: integer  
 }

## Select Users Jobs Message

**Client**: supabase.select(\*).from('Job').where(auth.user\_id === Job.user\_id)

**Server**: { data, error }

data = {

jobs: [

{

job\_id: number,

company\_name: string,

job\_title: string,

remote: Boolean,

job\_city: string,

job\_state: string,

job\_country: string,

date\_posted: date,

date\_applied: date,

platform: string,

estimated\_annual\_salary: integer,

status: string,

notes: string

},

]

},

error = {

message: string,

status: integer

}

## Create Job Message

**Client**: supabase.insert(

company\_name: string,

job\_title: string,

remote: Boolean,

job\_city: string,

job\_state: string,

job\_country: string,

date\_posted: date,

date\_applied: date,

platform: string,

estimated\_annual\_salary: integer,

status: string,

notes: string

).into('Job').values(

companyName,

jobTitle,

remote,

jobCity,

jobState,

jobCountry,

datePosted,

dateApplied,

platform,

estimatedAnnualSalary,

status,

notes

)

**Server**: { data, error}

data = {

Job: {

company\_name: string,

job\_title: string,

remote: Boolean,

job\_city: string,

job\_state: string,

job\_country: string,

date\_posted: date,

date\_applied: date,

platform: string,

estimated\_annual\_salary: integer,

status: string,

notes: string

}

},

error = {

message: string,

status: integer

}

## Update Job Message

**Client**: supabase.update(<fields to be updated>).from("Job").where(Job.job\_id === jobId).values(<new values>)

**Server**: { data, error }

data = {

Job: {

company\_name: string,

job\_title: string,

remote: Boolean,

job\_city: string,

job\_state: string,

job\_country: string,

date\_posted: date,

date\_applied: date,

platform: string,

estimated\_annual\_salary: integer,

status: string,

notes: string

}

},

error = {

message: string,

status: integer

}

## Delete Job Message

**Client**: supabase.delete().from("Job").where(Job.job\_id === jobId)

**Server**: { data, error }

data = {

message: string,

status: integer

},

error = {

message: string,

status: integer

}

## Select Users Groups Messages

**Client**: supabase.select(\*).from('Group').where(auth.user\_id === Group.user\_id)

**Server**: { data, error }

data = {

groups: [

1: {

group\_id: integer,

group\_name: string,

group\_start\_date: date,

group\_end\_date: date

},

],

message: string,

status: integer

},

error = {

message: string,

status: integer

}

## Create Group Message

**Client**: supabase.insert(

group\_name: string,

group\_start\_date: date,

group\_end\_date: date

).into('Group').values(

groupName,

groupStartDate,

groupEndDate

)

**Server**: { data, error}

data = {

Group: {

group\_name: string,

group\_start\_date: date,

group\_end\_date: date

}

},

error = {

message: string,

status: integer

}

## Select Users Documents

**Client**: supabase.select(\*).from('Document').where(Document.user\_id === auth.user\_id)

**Server**: { data, error}

data = {

documents: [

{

doc\_id: integer,

doc\_name: string,

path: string,

date\_uploaded: date,

job\_id: int

}

]

},

error = {

message: string,

status: integer

}

## Create Document Message

**Client**: //store image in bucket and get path back

supabase.insert(doc\_name, path, date\_uploaded,user\_id).

into('Document').values(docName, path, dateUploaded, auth.user\_id)

**Server**: { data, error}

data = {

doc\_id: integer,

doc\_name: string,

path: text,

date\_uploaded: date

},

error = {

message: string,

status: integer

}

## Delete Document Message

Client: //handle deletion from storage bucket

supabase.delete().from('Document').where(Document.doc\_id === docId)

Server: { data, error}

data = {

message: string,

status: integer

},

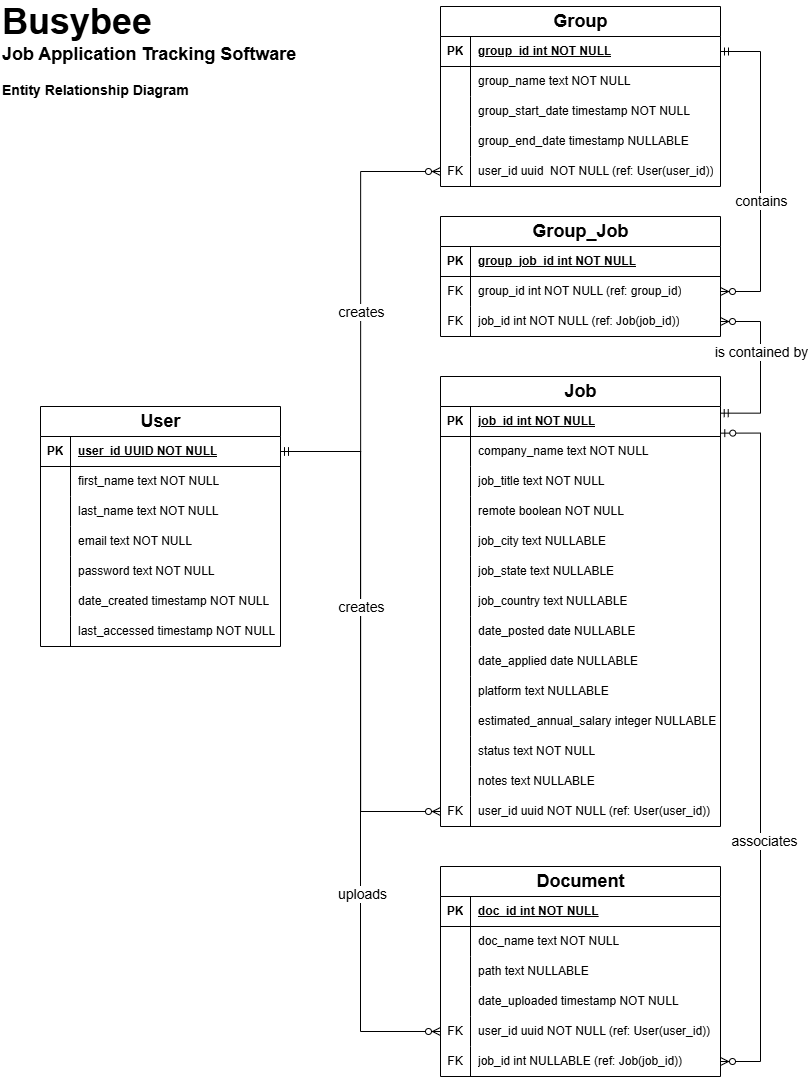
error = {

message: string,

status: integer

}

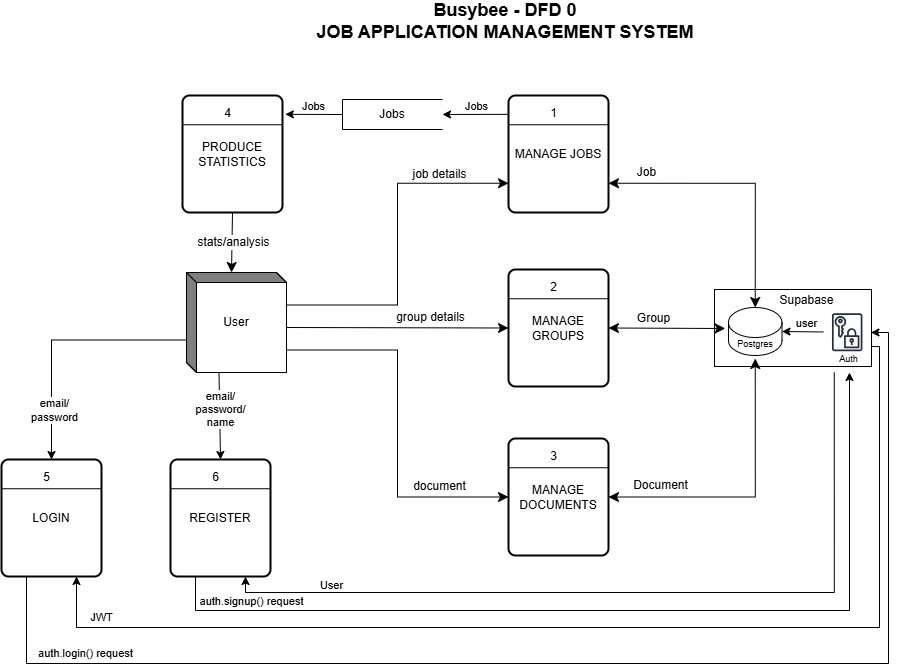
# Appendix E: Storage Documentation



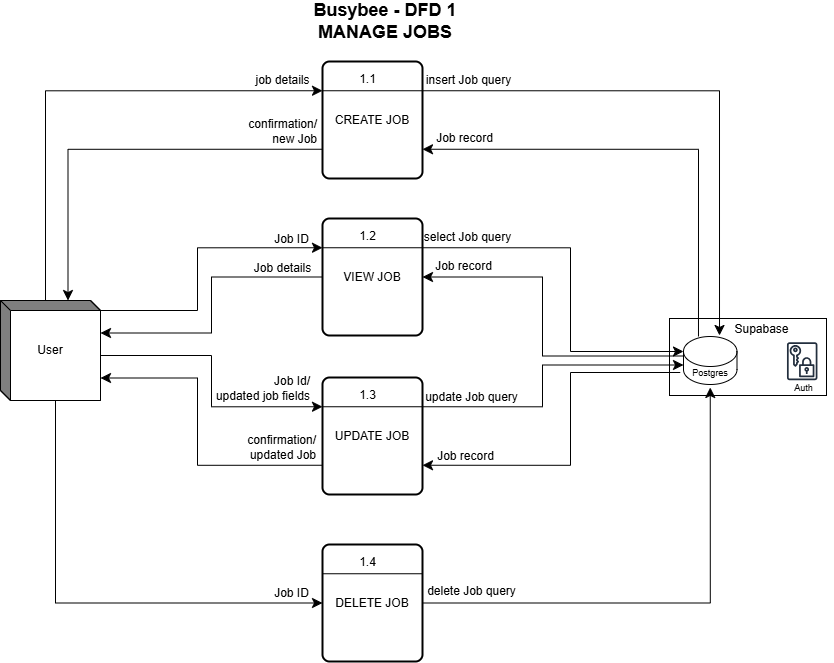
# Appendix F: Data Flow Diagrams

## Context Diagram

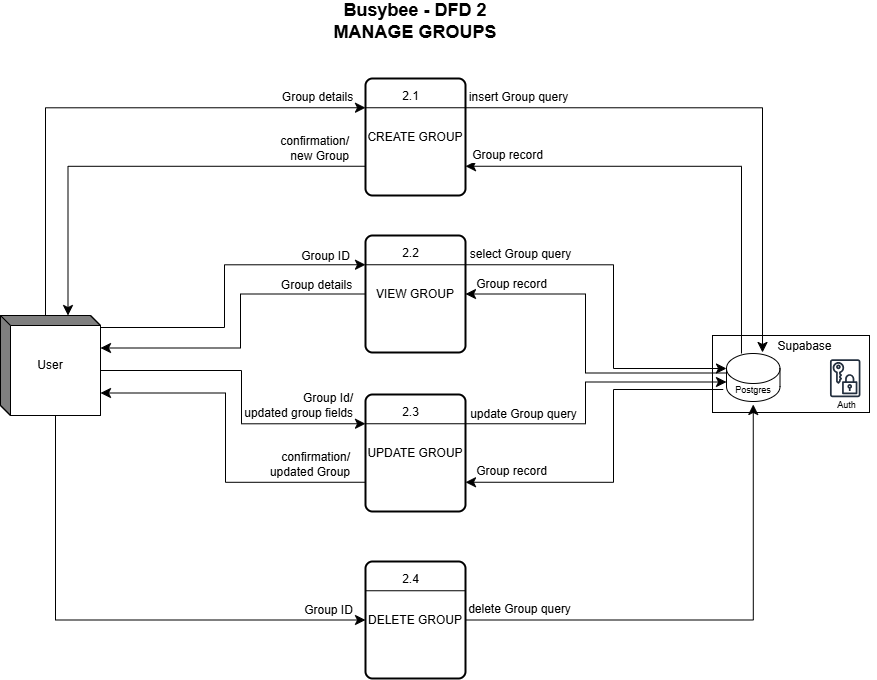
## DFD 0



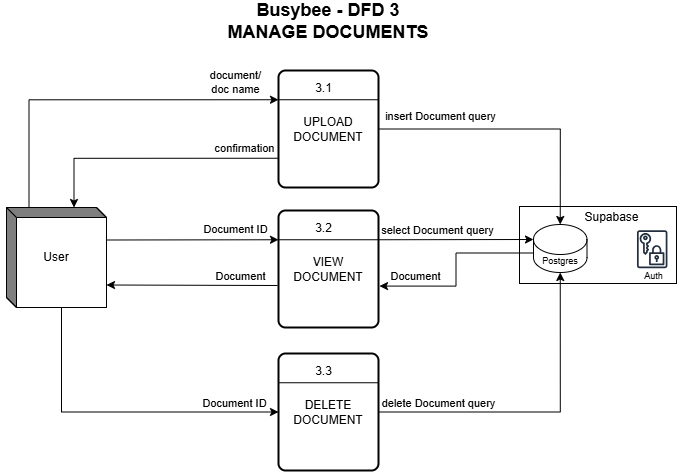
## DFD 1



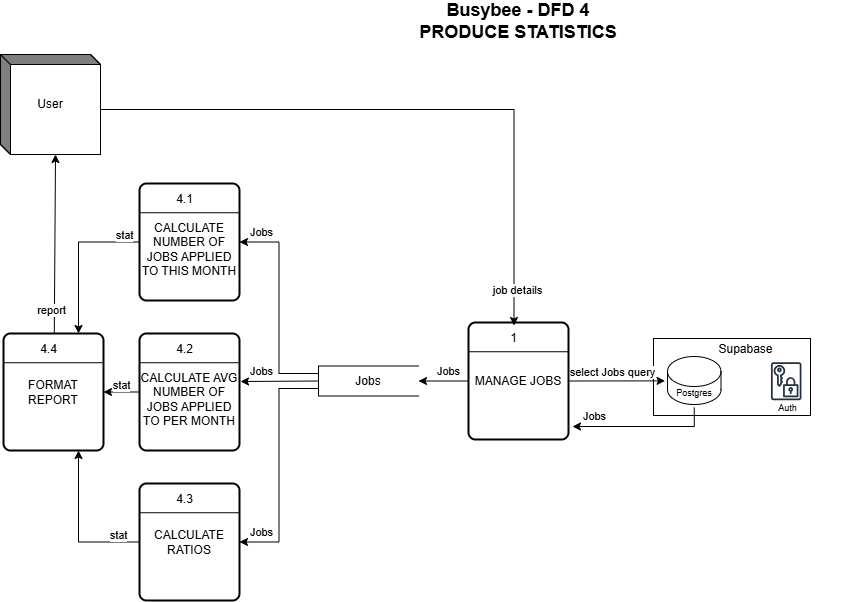
## DFD 2



## DFD 3



## DFD 4



# Appendix G: Data Model Diagram

See **Documentation/diagrams/component/component-data-model.drawio.png** for the data model diagram

# Appendix H: Continuous Integration/Continuous Delivery Pipeline

