**Implify HR**

Table of Contents

[**Implify HR** 1](#_Toc173163806)

[**1. Introduction** 2](#_Toc173163807)

[**2. Installation** 2](#_Toc173163808)

[**Backend Installation** 2](#_Toc173163809)

[**Frontend Installation** 2](#_Toc173163810)

[**Additional Commands** 2](#_Toc173163811)

[**3. Code Overview** 2](#_Toc173163812)

[**3.1 BackEnd** 3](#_Toc173163813)

[**3.2 FrontEnd** 4](#_Toc173163814)

[**3.3 Database** 6](#_Toc173163815)

[**4. How to Run** 7](#_Toc173163816)

[**4.1 BackEnd** 7](#_Toc173163817)

[**4.2 FrontEnd** 7](#_Toc173163818)

[**5. Results** 7](#_Toc173163819)

**1. Introduction**

This project involves developing a web-based application for managing employee data, featuring a responsive and user-friendly interface. It utilizes HTML, CSS, and TypeScript for the frontend, and Python with MySQL for the backend. The application will include functionalities such as viewing, filtering, adding, editing, and deleting employee records. The project aims to demonstrate proficiency in web development technologies and will be documented with design specifications, implementation details, and test cases to ensure a comprehensive and functional application.

**2. Installation**

**Backend Installation**

1. **Install Python:**
   * Ensure Python 3.9 is installed. Download from python.org.
2. **Install Required Python Packages:**
   * pip install Flask Flask-RESTful Flask-SQLAlchemy Flask-Cors pymysql
3. **Setup Database:**
   * Install MySQL Server if not already installed (refer to MySQL's documentation).
   * Create a database named Info using MySQL Workbench or command line.
   * Ensure that the MySQL server is running.
4. **Configure SQLAlchemy:**
   * Update the app.config['SQLALCHEMY\_DATABASE\_URI'] line in your script with the correct MySQL connection string if needed.

**Frontend Installation**

1. **In Terminal Navigate to the Frontend Directory:** cd frontEnd
2. **Install Node.js and npm:**
   * Ensure Node.js and npm are installed. Download from nodejs.org.
3. **Install Project Dependencies (once):** npm install (install all required dependencies specified in package.json)
4. **Install TypeScript Globally:** npm install -g typescript

**Additional Commands**

* **Check for Processes Using Port 5000:** netstat -ano | findstr :5000
* **Kill a Process Using Port 5000:** taskkill /PID <PID> /F

Replace <PID> with the actual process ID obtained from the previous command.

**3. Code Overview**

**3.1 Backend**

**Technologies Used**

* **Flask:** A lightweight web framework for Python used to build the RESTful API.
* **Flask-RESTful:** An extension for Flask that simplifies the creation of REST APIs.
* **Flask-SQLAlchemy:** An extension for Flask that integrates SQLAlchemy, a SQL toolkit and Object-Relational Mapping (ORM) library.
* **Flask-Cors:** An extension to handle Cross-Origin Resource Sharing (CORS), allowing the frontend to interact with the backend from different origins.
* **PyMySQL:** A pure-Python MySQL client for database connections.

**Setup and Configuration**

1. **Database Configuration:**
   * Utilizes MySQL for data storage.
   * Connection string: mysql+pymysql://root@localhost/info
   * SQLAlchemy is set to avoid tracking modifications with SQLALCHEMY\_TRACK\_MODIFICATIONS = False.
2. **Logging:**
   * Configured with TimedRotatingFileHandler to rotate logs daily.
   * Logs are stored in log within the project directory.
3. **File Uploads:**
   * Files are uploads to /Users/macbook/Desktop/int/frontEnd/dist/img.
   * Managed by the Upload resource, which handles file saving and renaming. **Ensure to update the upload path in the configuration according to your specific directory structure.**

**API Resources**

1. **Employee Resource (/e):**
   * GET /e - Retrieves a list of all employees.
   * GET /e/<int:employee\_id> - Retrieves details of a specific employee by ID.
   * POST /e - Creates a new employee entry.
   * PUT /e/<int:employee\_id> - Updates an existing employee's details. If the employee ID is 0, a new employee is created with a random password.
   * DELETE /e/<int:employee\_id> - Deletes an employee by ID.
2. **Company Color Resource (/c):**
   * GET /c - Retrieves a list of all company colors.
   * POST /c - Adds a new company color.
   * PUT /c/<int:color\_id> - Updates an existing company color by ID.
   * DELETE /c/<int:color\_id> - Deletes a company color by ID.
3. **Login Resource (/login):**
   * GET /login - Authenticates a user with a username and password, returning success status and access rights.
4. **Upload Resource (/upload):**
   * POST /upload - Handles file uploads. Files are saved with an optional specified name or their original filename.

**Models**

* **Employee:**
  + Represents an employee with personal information, login credentials, and company details.
  + Includes methods to find employees by username and verify passwords.
* **CompanyColor:**
  + Represents a company color with a name and color code.

**Running the Application**

* The application is configured to run in debug mode with app.run(debug=True), which facilitates easy debugging and automatic reloading during development.

**3.2 Frontend**

**Technologies Used**

* **HTML:** The basic markup language for creating the structure of web pages.
* **CSS3:** Used for styling the webpage with modern techniques, including animations and responsive design.
* **TypeScript**: Provides enhanced interactivity and dynamic content with static typing and modern features.
* **Bootstrap:** A front-end framework for building responsive and mobile-first web pages.
* **Google Fonts:** Specifically using the Mulish font for consistent typography.

**File Structure**

* **node\_modules:** Contains all the installed Node.js modules and dependencies.
* **dist:** Contains compiled JavaScript files that are used in the production build.
* **images:** Stores image files for icons, logos, top bars, and data images.
* **home.css:** Custom CSS for styling the home page.
* **home.html:** HTML structure for the home page.
* **loading.css:** CSS for styling loading animations or screens.
* **loading.html:** HTML structure for loading screens.
* **login.css:** CSS for styling login page.
* **login.html:** HTML structure for login page.
* **src:** Source code directory.
  + **home.ts:** TypeScript code for the home page.
  + **index.ts:** TypeScript entry point, possibly for general application logic.
  + **login.ts:** TypeScript code for handling login functionality.

**Design and Implementation**

1. **Styling and Layout:**
   * The interface is styled using CSS3, with an emphasis on modern design and animations.
   * Bootstrap is used for responsive design to ensure the layout adapts to various screen sizes and devices.
   * Google Fonts (Mulish) is integrated for a clean and professional typographic style.
2. **Interactivity:**

* TypeScript is employed for smooth animations and dynamic content updates. Animations and effects are managed directly in TypeScript, replacing the need for jQuery.

1. **Communication with Backend:**
   * Initially, TypeScript handles rest API requests to interact with JSON files for data retrieval and updates, simulating backend interactions before a real database is implemented.
2. **Responsiveness:**
   * The interface uses Bootstrap's grid system and components to ensure that the application is fully responsive and provides a consistent experience across different devices.

**Development Workflow**

1. **Initial Development:**
   * Start with HTML and CSS for basic layout and styling.
   * Integrate TypeScript for animations and interactive features.
   * Utilize Bootstrap for responsive design and layout management.
2. **TypeScript:**
   * Write business logic and interactivity using TypeScript.
   * Compile TypeScript files to JavaScript and place them in the dist folder for production use.
3. **Testing and Refinement:**
   * Test the interface across various devices and screen sizes to ensure responsiveness and functionality.
   * Refine animations and interactions based on user feedback and testing results.
4. **Integration:**
   * Transition from JSON-based data to backend integration once the backend is ready, adapting rest API requests to communicate with the API.

**3.3 Database**

**Database Name:** Info

**Description:** The Info database is designed for an HR management application, facilitating the storage and retrieval of employee and company color information. It is built using MariaDB, a popular open-source relational database management system, and is configured to handle Unicode data with UTF-8 encoding.

**Tables:**

1. **company Table:**
   * **Purpose:** Stores information about various company colors used in the application.
   * **Columns:**
     + **ID:** Unique identifier for each company color (integer, auto-increment).
     + **Company:** Name of the company associated with the color (varchar, 250 characters).
     + **Color:** Hexadecimal color code (varchar, 7 characters).
   * **Indexes:**
     + Primary key on ID for unique identification.
2. **employee Table:**
   * **Purpose:** Stores detailed information about employees, including their personal details, contact information, and associated company color.
   * **Columns:**
     + **employee\_id:** Unique identifier for each employee (integer, auto-increment).
     + **fname:** Employee's first name (varchar, 250 characters).
     + **lname:** Employee's last name (varchar, 250 characters).
     + **address:** Employee's address (text).
     + **city:** City where the employee resides (varchar, 250 characters).
     + **country:** Country where the employee resides (varchar, 250 characters).
     + **photo:** Filename of the employee's photo (text).
     + **company\_id:** Foreign key linking to the company\_color table (integer).
     + **email:** Unique email address for the employee (varchar, 510 characters).
     + **password:** Employee's password (varchar, 510 characters).
     + **Access\_right:** Specifies the access rights of the employee (varchar, 250 characters).
   * **Indexes:**
     + Primary key on employee\_id.
     + Foreign key constraint linking company\_color\_id to the ID in the company\_color table.

**Relationships:**

* The employee table is linked to the company\_color table via the company\_color\_id column. This foreign key establishes a relationship where each employee is associated with a specific company color.

**Indexes and Constraints:**

* Primary keys are set for unique identification in both tables.
* Foreign key constraints ensure referential integrity between the employee and company\_color tables.

**4. How to Run**

**4.1 BackEnd**

**Run the Flask App:** python app.py

or through VS Code run button. This will start the server on http://localhost:5000.

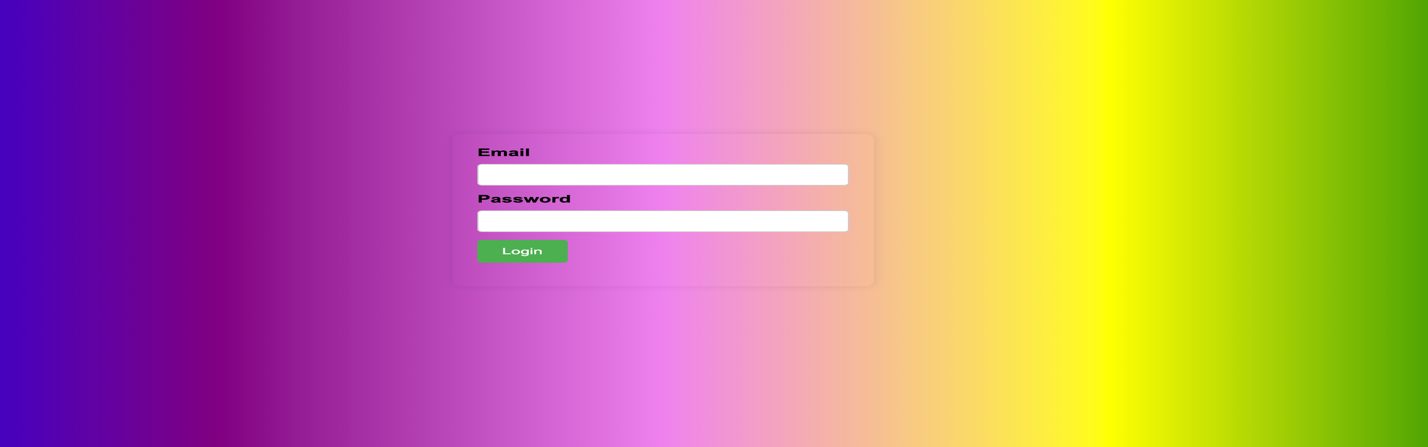
**4.2 FrontEnd**

These steps are required at first step and after every modification in the TypeScript code:

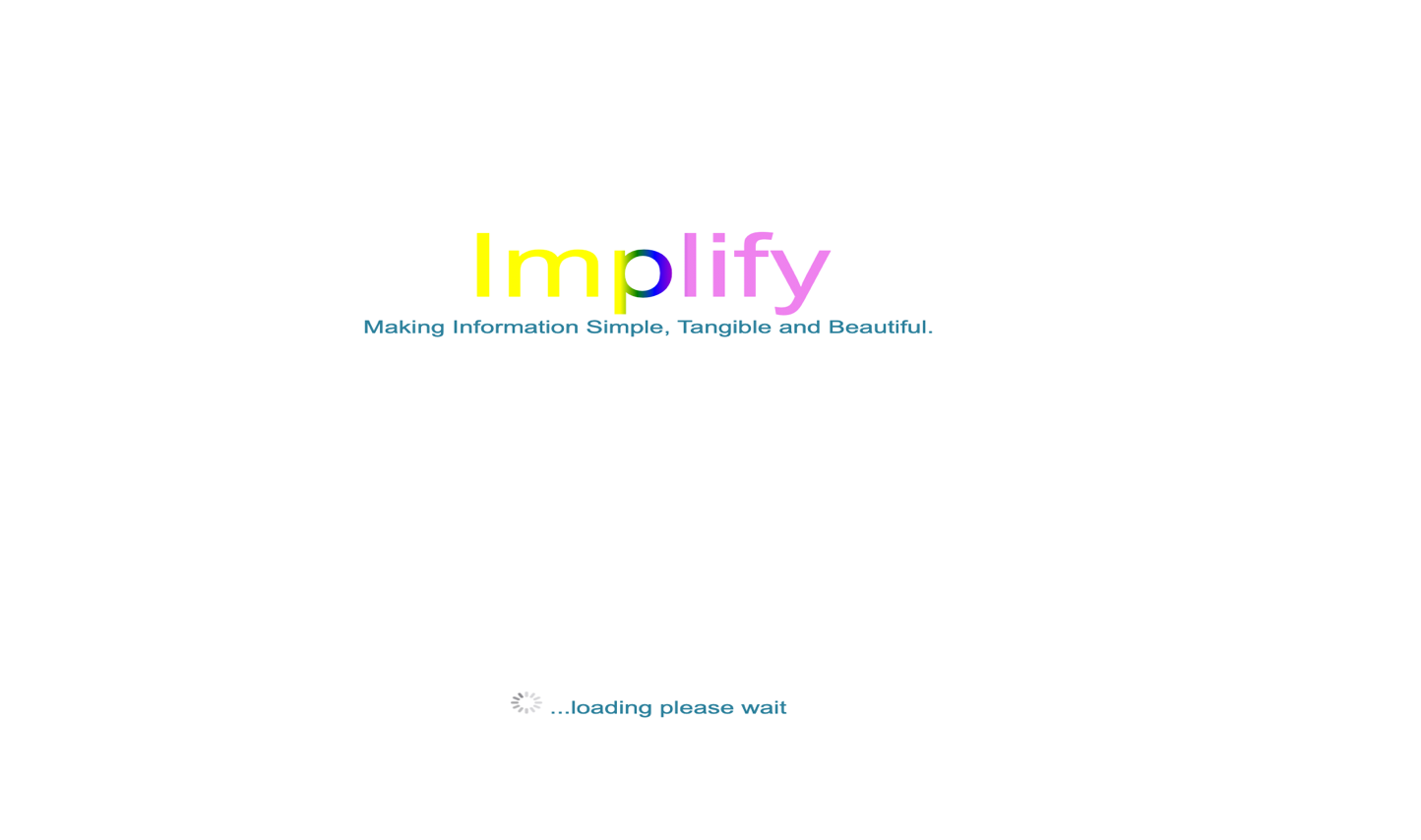
1. **Build the Project:** npm run build
2. **Start the Development Server:** npm start

You can access the frontend on http://localhost:3001 by default.

**5.** **Results**

**Login Page**

**Loading Page**

****

**Home Page**

