# 🚀 FULL PROJECT SETUP GUIDE: REACT + DIRECTUS + POSTGRESQL (DOCKERIZED)

This document provides a comprehensive, step-by-step guide for setting up your development environment and getting the project up and running from a completely fresh state. Whether you're new to React, Node.js, or Docker, this guide will walk you through all the necessary installations and configurations to ensure a smooth setup process. By the end of this guide, you will have a fully functional development environment for this project, comprising a React frontend, a Directus backend, a PostgreSQL database, all orchestrated with Docker Compose, and a convenient Mailhog for email testing.

## 1. INTRODUCTION

Welcome to the setup guide for our project! This application leverages a modern and robust technology stack designed for scalability, flexibility, and efficient development. Understanding the core components will help you navigate the project structure and contribute effectively.

### OVERVIEW OF THE TECH STACK:

- Frontend: React.js
  A declarative, component-based JavaScript library for building user interfaces. React allows us to create dynamic and interactive single-page applications efficiently.

- Backend: Directus (as a BaaS)
  Directus is an open-source Headless CMS that wraps your SQL database with a powerful API and a beautiful admin app. It transforms your raw SQL database into a dynamic, real-time API (REST & GraphQL) and a no-code data studio, allowing us to manage content and data effortlessly without writing custom backend code.

- Database: PostgreSQL
  A powerful, open-source object-relational database system known for its strong reliability, feature robustness, and performance. It serves as the

primary data store for our Directus instance.

- **Containerization: Docker and Docker Compose**
  Docker is a platform that uses OS-level virtualization to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries, and configuration files; they can communicate with each other through well-defined channels. Docker Compose is a tool for defining and running multi-container Docker applications. It allows us to define our entire application's services (Directus, PostgreSQL, Mailhog) in a single YAML file and launch them with a single command, ensuring consistency across development environments.

- **Mail testing: Mailhog**
  Mailhog is a simple, web-based SMTP server for email testing. It catches emails sent from your application during development so you can inspect them without actually sending them to external mailboxes. This is incredibly useful for testing features like user registration, password resets, and notifications.

## 2. PREREQUISITES

Before you can set up the project, you need to ensure that your development machine has the necessary tools installed. Follow the instructions below to get Node.js, npm, Docker, and Docker Compose ready.

### INSTALL NODE.JS AND NPM

Node.js is a JavaScript runtime environment that allows you to run JavaScript code outside of a web browser. npm (Node Package Manager) is the default package manager for Node.js, used to install and manage project dependencies for our React application. We recommend installing Node.js via your system's package manager for ease of updates and management.

For Ubuntu/Debian-based systems, you can install Node.js and npm using the following commands in your terminal:

```
sudo apt update
sudo apt install nodejs npm -y
```

**Note:** The `-y` flag automatically answers 'yes' to prompts, which can be convenient but be mindful of what is being installed. After installation, you can verify that Node.js and npm are correctly installed by checking their versions:

```
node -v
npm -v
```

You should see version numbers displayed.

## INSTALL DOCKER

Docker is essential for running our backend services (Directus, PostgreSQL, Mailhog) in isolated containers. This ensures that your local environment remains clean and that the services run consistently regardless of your operating system's specifics.

To install Docker on Ubuntu/Debian, use:

```
sudo apt install docker.io -y
```

**Important:** After installing Docker, you might need to add your user to the `docker` group to run Docker commands without `sudo`. This is a common practice for development. Run the following command and then log out and log back in (or restart your terminal session):

```
sudo usermod -aG docker $USER
```

You can verify your Docker installation by running a simple test container:

```
docker run hello-world
```

If successful, you will see a "Hello from Docker!" message.

## INSTALL DOCKER COMPOSE

Docker Compose simplifies the management of multi-container Docker applications. It allows you to define your entire service stack in a single YAML file ( `docker-compose.yml` ) and manage its lifecycle with simple commands.

Install Docker Compose on Ubuntu/Debian using:

```
sudo apt install docker-compose -y
```

**Verification:** Confirm Docker Compose installation by checking its version:

```
docker-compose -v
```

You should see the version number. Ensure it's version 1.29.2 or higher for best compatibility, although generally, any recent version should work.

## 3. CLONE THE REPOSITORY

With all prerequisites in place, the next step is to obtain the project source code from its GitHub repository. This involves using Git, a version control system, to clone the entire project onto your local machine.

Open your terminal or command prompt and execute the following commands. Replace `your-username/your-repo.git` with the actual repository URL provided for this project.

```
git clone https://github.com/your-username/your-repo.git
cd your-repo
Project Github repo: https://github.com/dyanesh-100/arch-
survey
```

The first command downloads the repository. The second command navigates you into the newly created project directory. All subsequent commands in this guide will assume you are executing them from the root of this project directory unless specified otherwise.

**Tip for Git:** If you don't have Git installed, you can typically install it via `sudo apt install git -y` on Debian/Ubuntu systems. Git is a fundamental tool for any developer working with source code repositories.

## 4. START BACKEND (DOCKER COMPOSE)

This project utilizes Docker Compose to manage its backend services efficiently. The `docker-compose.yml` file in the project root defines the Directus instance, the PostgreSQL database, and Mailhog. Starting these services is a single command away.

From the project root directory (where your `docker-compose.yml` file is located), run the following command:

```
docker-compose up -d
```

- `docker-compose up`: This command builds, (re)creates, starts, and attaches to containers for all services defined in your `docker-compose.yml` file.
- `-d` (detached mode): This flag runs the containers in the background, freeing up your terminal for other commands. If you omit `-d`, the logs from all services will be streamed directly to your terminal.

Upon successful execution, Docker Compose will download the necessary images (if not already present), create the containers, networks, and volumes, and start the services. You should see output indicating that the services are being created and started.

### SERVICES STARTED:

Once the Docker containers are running, you can access the backend services via your web browser:

- **Directus:** The Directus admin interface and API will be accessible at `http://localhost:8055`. This is where you will manage your database collections, content, and user permissions.
- **Mailhog:** The Mailhog web interface, used for catching and inspecting outgoing emails, will be available at `http://localhost:8025`.

**Stopping Backend Services:** To stop all running services and remove their containers, networks, and volumes defined in the `docker-compose.yml` file, navigate to the project root and run:

```
docker-compose down
```

This is useful when you want to reset your backend environment or free up resources.

## 5. FRONTEND SETUP

Now that your backend services are robustly running inside Docker containers, it's time to set up the React frontend. The frontend is a standard React application managed by npm.

First, navigate into the frontend folder of your cloned project. The exact name of this folder might vary, but it's typically named something like `frontend`, `client`, or similar. If you're unsure, check your project's directory structure.

```
cd frontend-folder
```

Replace **`frontend-folder`**: Make sure to replace `frontend-folder` with the actual name of your React application's directory within the main project repository. For instance, if your repository structure is `your-repo/client/`, you would use `cd client`.

Once inside the frontend directory, you need to install all the project's JavaScript dependencies. These dependencies are listed in the `package.json` file and include React itself, various libraries, and development tools.

```
npm install
```

This command will read the `package.json` file, download all the required packages from the npm registry, and place them in a `node_modules` directory within your frontend folder. This process might take a few moments, depending on your internet connection and the number of dependencies.

**Common Issues:** If you encounter issues during `npm install` (e.g., permission errors, corrupted cache), try running `npm cache clean --force` and then retrying the installation. For permission errors, ensure your user has appropriate write permissions to the project directory, or try running the command with `sudo` if on Linux, though this is generally not recommended for npm packages installed globally.

## 6. AXIOS BASE URL CONFIGURATION

Our React frontend communicates with the Directus backend using Axios, a popular promise-based HTTP client for the browser and Node.js. To ensure the frontend correctly connects to your Directus instance, you need to configure the base URL for these API requests.

The configuration for the Directus API base URL is typically managed in a dedicated file, which centralizes the endpoint for easier management. In this project, that file is:

```
src/axiosInstanceDirectus.js
```

Open this file in your code editor. You will find a line similar to the one below. You need to verify that the `baseURL` matches the address where your Directus instance is running. Since we are running Directus via Docker Compose on your local machine, the default port is `8055`.

### EXAMPLE CONFIGURATION:

Ensure the `baseURL` is set as follows:

```
// src/axiosInstanceDirectus.js

import axios from 'axios';

const axiosInstanceDirectus = axios.create({
  baseURL: 'http://localhost:8055', // Update if hosted
elsewhere
});

export default axiosInstanceDirectus;
```

**Important:** If your Directus instance were hosted on a different server or a different port (e.g., in a production environment), you would update this `baseURL` accordingly. For local development with Docker Compose as set up in this guide, `http://localhost:8055` is the correct value. Do not include a trailing slash in the `baseURL`.

This centralized configuration ensures that all API calls made by the React application to Directus will automatically prepend this base URL, making your API interactions clean and maintainable.

## 7. RUN THE REACT APP

With all dependencies installed and the Axios base URL configured, you are now ready to launch the React development server. This will compile your React application and serve it in your browser, typically with hot-reloading features for a smooth development experience.

Ensure you are still in the frontend directory (e.g., `frontend-folder`). If not, navigate back to it using `cd frontend-folder`.

To start the React development server, execute the following command:

```
npm start
```

This command typically initiates a script defined in your `package.json` file (often `react-scripts start` for projects created with Create React App). It will compile your application, open a new tab in your default web browser, and navigate to the application's URL.

### APP RUNS ON:

Your React application will be accessible through the URL displayed in the terminal once after the project successfully started.

**Stopping the App:** To stop the React development server, simply go back to the terminal where `npm start` is running and press `Ctrl + C`.

Congratulations! You should now have both your backend services (Directus, PostgreSQL, Mailhog) and your React frontend running successfully. The next step is to familiarize yourself with the frontend's code structure.

## 8. CODE STRUCTURE OVERVIEW (FRONTEND)

Understanding the project's file structure is crucial for navigating the codebase, identifying where to make changes, and contributing effectively.

The React frontend follows a common, logical organization designed for maintainability and scalability.

Here's a breakdown of the key directories and files within the `src/` folder of your React application:

```
src/
│
├── axiosInstanceDirectus.js     # Directus Axios base
URL configuration
│                                # This file exports a
pre-configured Axios instance
│                                # specifically for
interacting with the Directus API.
│                                # It ensures all API
calls use the correct base URL.
│
├── pages/                       # UI pages (e.g., Home,
Login, Dashboard, Profile, etc.)
│   │                            # Each file in this
directory represents a major
│   │                            # view or page of the
application. They typically
│   │                            # compose smaller
components to form a complete UI.
│   ├── Home.js
│   ├── Login.js
│   ├── Dashboard.js
│   └── ...
│
├── components/                  # Reusable UI components
│   │                            # Contains smaller,
reusable React components that can
│   │                            # be used across
different pages or within other components.
│   │                            # Examples include
buttons, input fields, navigation bars,
│   │                            # cards, modals, etc.
│   ├── Button.js
│   ├── Navbar.js
│   ├── Card.js
│   └── ...
│
├── services/                    # API call functions and
authentication logic
│   │                            # This directory
abstracts API interactions and business logic
│   │                            # related to
authentication and data fetching.
│   ├── apiService.js            # Centralized functions
for making various API calls
│   │                            # to Directus. It
typically uses `axiosInstanceDirectus`.
│   └── authService.js           # Functions for user
authentication (login, logout,
│                                # registration), token
management (storing/retrieving
│                                # JWTs), and session
handling.
│
├── context/                     # Global state
management using React Context API
│   │                            # This directory holds
files related to global state.
│   └── GlobalContext.js         # Stores data fetched
from the API and application-wide
│                                # state that needs to be
shared across multiple components
│                                # without prop-drilling.
It provides a central source of truth.
│
└── App.js / index.js            # Entry point and route
handlers
    │                            # `index.js` is
typically the very first file executed,
    │                            # rendering the root
`App` component. `App.js` defines
    │                            # the main structure of
the application, including
    │                            # routing (e.g., using
React Router) to switch between pages.
    │
    └── ... (other root-level files like CSS, assets,
etc.)
```

**Modular Design:** This structure promotes a modular and organized codebase, making it easier to locate specific functionalities, develop new features, and maintain the application over time. Components are designed to be self-contained and reusable.

## 9. NOTES

Keep the following important points in mind while working with this project. These notes highlight key architectural decisions and best practices implemented within the application.

- **Centralized State Management with GlobalContext.js:**

All crucial API data and the majority of the application's global state are centrally managed through `GlobalContext.js`. This file leverages React's Context API to provide data and functions that can be accessed by any component within the component tree, avoiding the need to pass props down manually through multiple levels (prop-drilling).

When you need to fetch data from Directus and share it across different parts of your application (e.g., user profile information, global settings, frequently accessed lists), you will typically interact with the context defined in `GlobalContext.js`. This approach ensures data consistency and simplifies state management for larger applications.

- **Directus Base URL Configuration:**

As mentioned in Section 6, the Directus base URL is configured in one single, dedicated place: `axiosInstanceDirectus.js`. This is a critical design choice to ensure maintainability.

**Always remember:** If you ever need to change the address of your Directus backend (e.g., moving from `localhost` to a staging server, or updating the port), you only need to modify this one file. Do not hardcode the Directus API URL in multiple places throughout your components or services. Doing so would lead to inconsistencies and make future updates significantly more complex and error-prone.

## 10. WHAT’S NEXT?

Congratulations on successfully setting up the entire development environment for the project! You now have the React frontend communicating with the Directus backend and PostgreSQL database, all containerized with Docker Compose, and Mailhog ready for email testing. This is a significant milestone!

Your next crucial step is to configure the Directus instance itself. While Directus is running, it needs to be initialized with an admin account, and then you'll define the data model for your application by creating collections, fields, and setting up permissions and relations. This process effectively transforms your raw database into a functional Headless CMS.

Please proceed to the dedicated documentation file for Directus configuration.

Thank you for following this guide. You are now well-equipped to dive into developing the application's features! Happy coding!