Project Plan: Placement Portal Frontend (React - Single Department)

This document outlines the development plan for the four key sections of the Placement Portal frontend, specifically for the Computer Science department. We will use React (with Vite) and Tailwind CSS.

Part 1: The Foundation (For Everyone) – done by Durva. She will send the folder to the group. Edit in the same folder, do not create a new project.

This shared foundation is critical for ensuring all modules integrate smoothly.

1. Project Setup (Team Lead):

- **Create a new React project using Vite:**

Bash

npm create vite@latest placement-portal-frontend -- --template react

cd placement-portal-frontend

- **Install necessary libraries:**

Bash

npm install

npm install -D tailwindcss postcss autoprefixer

npx tailwindcss init -p

- **Configure Tailwind CSS: Set up your tailwind.config.js and index.css files according to the official Tailwind guide for Vite.**
- **Install React Router for navigation and Chart.js for visualizations:**

Bash

npm install react-router-dom chart.js react-chartjs-2

2. Folder Structure: Use this structure in your src folder:

src/

|-- assets/ # Logos, etc.

|-- components/ # Reusable UI (Button, Modal, StatCard)

|-- pages/ # Main page components (LoginPage, DashboardPage, etc.)

|-- data/ # Mock data files (mockStudents.js, mockDrives.js)

|-- App.jsx # Main app component with routing logic

|-- index.css # Global styles and Tailwind imports

|-- main.jsx # App entry point

3. Shared Design System (UI/UX Guidelines): Everyone must adhere to these styles for a consistent look and feel.

- **Color Palette:**
    - **Background: bg-gray-50 (#F9FAFB)**
    - **Primary Brand Color: A professional Teal. Use bg-teal-600 for main buttons and text-teal-700 for links.**
    - **Text: text-gray-800 (main), text-gray-500 (subtitles).**
    - **Borders & Cards: border-gray-200 on bg-white cards.**
- **Elements:**
    - **Buttons: py-2 px-4 rounded-lg shadow-sm font-semibold with hover effects (hover:bg-teal-700).**
    - **Cards/Containers: bg-white, p-6, rounded-xl, border.**

Part 2: Individual Developer Tasks

Here are the specific instructions for each team member.

Task for Aradhya: Authentication Pages

Goal: Create clean and functional Login and Registration pages.

1. Create Page Components:

- **In src/pages/, create LoginPage.jsx and RegisterPage.jsx.**

2. Build LoginPage.jsx UI:

- **Use a centered layout (flex items-center justify-center min-h-screen bg-gray-50).**
- **Create a main container card (bg-white p-8 rounded-lg shadow-md w-full max-w-md).**
- **Add a title: "Admin Login" (text-2xl font-bold text-center).**
- **Add labeled input fields for "Email" and "Password".**
- **Add a full-width "Login" button (bg-teal-600 text-white).**
- **Add a link below: "Don't have an account? Register here". This should navigate to the register page.**

3. Build RegisterPage.jsx UI:

- **Duplicate the layout from the Login page.**
- **Change the title to "Create Admin Account".**
- **Add a "Full Name" input field.**

- **Change the button text to "Register".**
- **Change the bottom link to: "Already have an account? Login here".**

4. Functionality:

- **Use the useState hook to manage the form inputs.**
- **On form submission, console.log the input data. Backend integration will be handled later.**

5. Routing:

- **In App.jsx, configure the routes: /login for <LoginPage /> and /register for <RegisterPage />.**

Task for Purvika: Dashboard Page

Goal: Build an insightful dashboard providing a quick overview of placement activities.

1. Create Page Component:

- **In src/pages/, create DashboardPage.jsx.**

2. Mock Data:

- **Create src/data/dashboardData.js. Export mock data for the stats and the chart:**

JavaScript

```
export const dashboardStats = {
  totalStudents: 120, // Total CS students
  activeDrives: 3,
  totalSelected: 28,
  overallPlacementRate: 23.3 // (totalSelected / totalStudents) * 100
};
export const placementStatusData = {
   labels: ['Selected', 'Not Selected', 'In Progress'],
   data: [28, 45, 47] // Represents the 120 total students
};
```

3. Build DashboardPage.jsx UI:

- **This page should be the main view after login.**
- **Add a header: "Dashboard" (text-3xl font-bold).**
- **Statistics Section:**
    - **Create a grid (grid-cols-1 md:grid-cols-4 gap-6).**

    - **Use a reusable StatCard.jsx component to display: "Total Students", "Active Drives", "Total Selected", and "Placement Rate".**
- **Visualizations Section (Grid Layout):**
    - **Placement Status Chart:**
        - **Create a card titled "Overall Placement Status".**
        - **Use react-chartjs-2 to implement a Doughnut chart.**
        - **The chart should visualize the data from placementStatusData to show the proportion of students who are selected, not selected, or still in progress.**
    - **Upcoming Drives List:**
        - **Create another card titled "Upcoming Drives".**
        - **Inside, display a clean list of the next few companies and their recruitment dates.**

4. Routing:

- **In App.jsx, the main route / (or /dashboard) should render <DashboardPage />.**

Task for Aryan: Students Section

Goal: Build an interactive section for managing the master list of all Computer Science students.

1. Create Page Component:

- **In src/pages/, create StudentsPage.jsx.**

2. Mock Data:

- **Create src/data/mockStudents.js. Export an array of student objects (computer branch by default). Each object should have: id, rollNumber, name, moodleID, cgpa, email, phone.**

3. Build StudentsPage.jsx UI:

- **Add a header: "Manage Students" (text-3xl font-bold).**
- **Controls Section:**
    - **A search input field (for searching by name or roll number).**
    - **An "Upload Master List" button.**
    - **Note: The "Filter by Branch" dropdown is no longer needed.**
- **Students Table:**
    - **Create a table with columns: Roll Number, Name, Moodle ID, CGPA, Actions.**
    - **Render the student data from your mock file.**
    - **The "Actions" column must have a "View Details" button for each student.**

- **Student Detail Modal:**
    - **Clicking "View Details" should open a modal.**
    - **The modal must display all student details (Name, Roll No, CGPA, etc.).**
    - **It must also contain a "Placement Activity" section. This will be a table showing every company the student applied to and their final status for that drive (e.g., "Tech Solutions Inc. - Selected", "Innovate Global - Not Selected"). You can add package, etc., whichever extra columns you feel like adding accordingly.**

4. Functionality:

- **Use useState to manage the list of students and the search term.**
- **Implement the search logic to filter the students displayed in the table in real-time.**
- **Use useState to control the visibility of the detail modal and to pass the selected student's data into it.**

5. Routing:

- **In App.jsx, create a route /students that renders <StudentsPage />.**

Task for Durva: Drives & Companies Section

Goal: Create the complete workflow for managing company drives and viewing company details.

1. Create Page Components:

- **In src/pages/, create DrivesListPage.jsx, CompanyDetailPage.jsx, and ManageDrivePage.jsx.**

2. Mock Data:

- **Create src/data/mockDrives.js. Export an array of drive objects, including companyName, driveDate, status ('Upcoming', 'Completed'), and an array of rounds (e.g., ['Online Assessment', 'Technical Interview', 'HR Interview']).**

3. Build DrivesListPage.jsx UI:

- **This page lists all recruitment events.**
- **Add a header: "Recruitment Drives".**
- **Create a table listing all drives with columns: Company Name, Drive Date, Status, Actions.**
- **The "Actions" column should have two buttons: "View Company" and "Manage Drive".**

4. Build CompanyDetailPage.jsx UI:

- **This page gives an overview of one company.**
- **Display the company's name and logo at the top.**
- **Create a section for "Company Details" (e.g., Industry, Website, a brief description).**

- **Add a section for "Drives Conducted," listing all recruitment events by this specific company.**

5. Build ManageDrivePage.jsx UI (The Main Task):

- **This is the detailed page for tracking a single drive.**
- **Header: Show the company name and drive date.**
- **Controls: Include "Upload Registered Students" and "Bulk Update" buttons.**
- **Progress Tracking Table:**
    - **This table lists all students registered for this specific drive.**
    - **Columns: A Checkbox for bulk actions, Student Name, Roll No, one column for each round status (e.g., "Technical Interview Status"), and a Final Status column.**
    - **Each status cell for a round should be a dropdown with options: Appeared, Cleared, Not Cleared.**
    - **The Final Status column is the most important. It must be a dropdown with two distinct options: Selected and Not Selected. This is the final verdict for the student in that drive.**

6. Functionality & Routing:

- **Set up routes in App.jsx:**
    - **/drives renders <DrivesListPage />.**
    - **/company/:companyId renders <CompanyDetailPage />.**
    - **/drives/manage/:driveId renders <ManageDrivePage />.**
- **Use useState to manage the student progress data on the ManageDrivePage. Changes in the status dropdowns should update the component's state, ready for a future backend call.**