Architectural Documentation: Job Portal Platform

Executive Summary

This document provides a comprehensive technical blueprint for the development of a multi-sided job portal platform. The system is designed to serve three primary user groups: Employees (job seekers), Employers (companies posting jobs), and an internal Admin team. The core business model is driven by tiered subscription plans for both employees and employers, a detailed commission system for sales staff, and a content management system for site administration.

The architectural approach is based on a modern, decoupled structure comprising a Next.js frontend, a robust RESTful API backend, and a MySQL database. This design ensures optimal performance, scalability, and search engine discoverability, a key business requirement [1]. The report details the data models for all key entities, specifies every required API endpoint with its request and response payloads, and outlines the functional UI/UX for all user journeys. It also provides a detailed analysis of the underlying business logic, including complex workflows such as the WhatsApp-based notification system and the sales commission models. This documentation serves as the single source of truth for the development team, enabling them to build the entire platform with clarity and precision.

1. Architectural Overview

The proposed architecture for the job portal platform is a tiered, microservice-oriented design that separates core functionalities into distinct layers. This approach promotes modularity, independent scalability, and a clear separation of concerns, which is critical for a platform supporting multiple user types and complex business logic.

1.1. High-Level Component Diagram

The system is composed of four primary layers:

- **Frontend Layer:** This layer is a Next.js application, chosen specifically for its server-side rendering capabilities that significantly enhance search engine optimization (SEO) [1]. The frontend serves two distinct user interfaces: one for employees and one for employers. A separate Admin Panel, built with React.js, will provide a secure, role-based interface for administrators. Both frontend applications will consume data and interact with the backend via the RESTful API.
- **Backend API Layer:** A centralized, stateless RESTful API, built with Laravel, serves as the core business logic hub. It is responsible for handling all user authentication, managing database operations, and orchestrating interactions with third-party services. The API design adheres to REST principles, ensuring that resources are clearly identifiable and that communication is standardized.
- **Database Layer:** A MySQL database is recommended for its reliability, transactional integrity, and strong support for structured and semi-structured data [1]. The relational model is well-suited for the highly structured nature of the data, including users, jobs, plans, and commissions, and allows for efficient querying and data integrity constraints.
- **Third-Party Services Layer:** This layer encapsulates all external integrations. It includes a dedicated service for the WhatsApp messaging functionality [1] and another for the paid professional CV creation service [1]. These integrations are handled asynchronously by the backend to prevent blocking API requests and to ensure system resilience.

A significant architectural decision is the handling of the WhatsApp messaging feature. A direct API call to a third-party messaging service during a synchronous request (e.g., a user applying for a job) can be unreliable and slow. Therefore, a decoupled, asynchronous approach is mandated. When a trigger event occurs (e.g., a job application is submitted or a job status is updated), the backend will not wait for the message to be sent. Instead, it will immediately store the event details in a dedicated messaging queue. A separate, non-blocking worker process or serverless function will then pick up the message from the queue and send it to the WhatsApp service. This design ensures that the user experience remains fast and responsive while providing a resilient mechanism for handling external communications, capable of retrying failed messages without impacting the main application flow [1].

1.2. Technology Stack Recommendations

- **Frontend:** Next.js with React.js for the primary framework.
- **Backend:** Laravel for its robust and scalable MVC architecture.

- **Admin Panel:** React.js for building the internal Admin Panel with a focus on enhanced user experience.
- **Database:** MySQL for data persistence.
- **Authentication:** JSON Web Tokens (JWTs) for secure, stateless authentication. JWTs will be stored in HttpOnly cookies to mitigate cross-site scripting (XSS) attacks.
- **Asynchronous Processing:** RabbitMQ or AWS SQS for message queuing to handle background tasks like WhatsApp messaging and email notifications.
- **Hosting:** A scalable cloud platform such as AWS, Google Cloud, or Azure is recommended, utilizing services like serverless functions for background jobs and managed database services for high availability.

2. Data Models and Schema Definitions

The following tables define the database schemas for the core entities within the system. These schemas are foundational for the entire application and provide the data structure for all backend services and API payloads.

2.1. User and Account Schemas

The system defines distinct schemas for employees, employers, and administrators, reflecting their unique roles and data requirements [1].

Employee Schema

Field Name	Data Type	Description	Constraints
id	UUID	Unique identifier for the employee.	Primary Key
email	VARCHAR(255)	Employee's email address.	Unique, Not Null

mobile	VARCHAR(20)	Employee's mobile number.	Unique, Not Null
password_hash	VARCHAR(255)	Hashed and salted password.	Not Null
name	VARCHAR(255)	Employee's full name.	Not Null
gender	ENUM('M', 'F', 'O')	Employee's gender.	Not Null
dob	DATE	Date of birth.	
address	JSONB	Employee's full address details.	
education_details	JSONB	Array of educational qualifications.	
experience_details	JSONB	Array of professional experiences.	
skills_details	JSONB	Array of skills.	
cv_url	VARCHAR(255)	URL to the uploaded CV file.	
plan_id	UUID	Foreign key to the Plans table.	Foreign Key
created_at	TIMESTAMP	Record creation timestamp.	Not Null
updated_at	TIMESTAMP	Last updated timestamp.	Not Null

Employer Schema

Field Name	Data Type	Description	Constraints
id	UUID	Unique identifier for the employer.	Primary Key
company_name	VARCHAR(255)	Name of the company.	Not Null
email	VARCHAR(255)	Company's primary email.	Unique, Not Null
contact	VARCHAR(20)	Company's contact number.	
address	JSONB	Company's address details.	
industry_type	UUID	Foreign key to the Industries catalog.	Foreign Key
password_hash	VARCHAR(255)	Hashed and salted password.	Not Null
plan_id	UUID	Foreign key to the Plans table.	Foreign Key
created_at	TIMESTAMP	Record creation timestamp.	Not Null
updated_at	TIMESTAMP	Last updated timestamp.	Not Null

2.2. Job and Application Schemas

The job model is the central entity that connects employees and employers.

Job Schema

Field Name	Data Type	Description	Constraints
id	UUID	Unique identifier for the job post.	Primary Key
employer_id	UUID	Foreign key to the Employers table.	Foreign Key, Not Null
title	VARCHAR(255)	Job title.	Not Null
description	TEXT	Detailed job description.	Not Null
salary	VARCHAR(255)	Salary range or amount.	
location_id	UUID	Foreign key to the Locations catalog.	Foreign Key
category_id	UUID	Foreign key to the JobCategories catalog.	Foreign Key
is_featured	BOOLEAN	Indicates if the job is a featured listing.	Default: false

featured_end_date	TIMESTAMP	Expiry date for featured listing.	
created_at	TIMESTAMP	Job post creation timestamp.	Not Null
updated_at	TIMESTAMP	Last updated timestamp.	Not Null

Job Application Schema

Field Name	Data Type	Description	Constraints
id	UUID	Unique identifier for the application.	Primary Key
job_id	UUID	Foreign key to the Jobs table.	Foreign Key, Not Null
employee_id	UUID	Foreign key to the Employees table.	Foreign Key, Not Null
application_status	ENUM	The current status of the application.	Not Null, Default: applied
applied_at	TIMESTAMP	Timestamp of the application submission.	Not Null

The application_status field is critical as it drives the real-time communication workflow [1]. Any update to this field by an employer will trigger an event to send a WhatsApp notification to the corresponding employee.

2.3. Plans and Subscriptions Schemas

The business model relies on tiered plans for both user types [1].

Schema	Field Name	Data Type	Description
Plans	id	UUID	Unique identifier for the plan.
	name	VARCHAR(255)	Name of the plan (e.g., "Basic Employee").
	description	TEXT	Detailed plan description.
	type	ENUM('employee', 'employer')	The user type the plan is for.
	price	NUMERIC(10,2)	The price of the plan.
	validity_days	INTEGER	The duration of the plan in days.
Plan Features	id	UUID	Unique identifier for the feature.
	plan_id	UUID	Foreign key to the Plans table.
	feature_name	VARCHAR(255)	The name of the feature (e.g., num_job_applies, whatsapp_messages).
	feature_value	VARCHAR(255)	The value of the

			feature (e.g., '50', 'true').

2.4. Coupons and Commissions Schemas

A robust system is required to track commissions for sales staff [1].

Schema	Field Name	Data Type	Description
Coupons	id	UUID	Unique identifier.
	code	VARCHAR(255)	The unique coupon code.
	discount_percentage	NUMERIC(5,2)	The percentage discount.
	expiry_date	DATE	The date the coupon expires.
	staff_id	UUID	Foreign key to the Admin table.
Commission Transactions	id	UUID	Unique identifier for the transaction.
	staff_id	UUID	Foreign key to the Admin table.
	payment_id	UUID	Foreign key to a hypothetical Payments table.
	amount_earned	NUMERIC(10,2)	Commission amount in currency.

	type	ENUM('coupon_based', 'manual')	The type of commission.
	created_at	TIMESTAMP	Timestamp of the commission event.

3. API Specification and Endpoints

The following section outlines the complete API contract for the backend. The endpoints, methods, and payloads have been meticulously defined to provide a clear, unambiguous guide for the development team. All API versions will be prefixed with /api/v1/.

3.1. Authentication & User Management APIs

These APIs handle the multi-step registration and login processes for all user types [1].

Endpoint	Method	Purpose	Auth Required	Request Payload (JSON)	Response Payload (JSON)
/auth/register/employee-step1	POST	Start employee registration.	None	{ "email": "...", "mobile": "...", "name": "...", "password": "...", "gender": "..." }	{ "message": "Step 1 complete.", "tempToken": "..." }
/auth/register/employe	POST	Submit basic	tempToken	{ "dob": "...", "address":	{ "message":

e-step2		details.		{... } }	"Step 2 complete." }
/auth/register/employee-final	POST	Complete registration.	tempToken	{ "education": [...], "experience": [...], "skills": [...] }	{ "message": "Registration complete.", "token": "..." }
/auth/register/employer	POST	Register a new employer.	None	{ "company_name": "...", "email": "...", "contact": "...", "address": "...", "industry_type_id": "..." }	{ "message": "Registration complete.", "token": "..." }
/auth/login	POST	User login.	None	{ "identifier": "...", "password": "..." }	{ "token": "...", "user_type": "employee/employer/admin" }

3.2. Employee Module APIs

These endpoints enable employees to manage their profiles, search for jobs, and interact with job posts [1].

Endpoint	Method	Purpose	Auth Required	Request Payload (JSON)	Response Payload (JSON)
/employee/profile	GET	Retrieve employee profile.	Employee	None	{ "user": {... }, "plan": {... } }
/employee/profile/update	PUT	Update specific profile details.	Employee	{ "field": "address", "value": {... } }	{ "message": "Profile updated." }
/jobs/search	GET	Search jobs with filters.	Employee	Query Params: q, location_id, category_id	{ "jobs": [{... }] }
/jobs/{jobId}/apply	POST	Apply for a job.	Employee	None	{ "message": "Application submitted." }
/employee/jobs/applied	GET	View jobs applied for.	Employee	None	{ "jobs": [{ "id": "...", "title": "...", "status": "applied", "employer": {... } }] }
/employee/jobs/shortlist	POST	Shortlist a job.	Employee	{ "job_id": "..." }	{ "message": "Job shortlisted." }
/employee/j	GET	View	Employee	None	{ "jobs": [

obs/shortlisted		shortlisted jobs.			{... }] }

3.3. Employer Module APIs

These APIs provide employers with the ability to manage their company profile, post jobs, and handle applications [1].

Endpoint	Method	Purpose	Auth Required	Request Payload (JSON)	Response Payload (JSON)
/employer/profile	GET	Retrieve employer profile.	Employer	None	{ "user": {... }, "plan": {... } }
/employer/jobs	POST	Create a new job post.	Employer	{ "title": "...", "description": "...", "salary": "...", "location_id": "...", "category_id": "..." }	{ "job_id": "...", "message": "Job created." }
/employer/jobs/{jobId}	GET	Get details of a single job.	Employer	None	{ "job": {... } }
/employer/jobs/{jobId}	PUT	Update a job post.	Employer	{ "title": "...", "description": "...",... }	{ "message": "Job updated." }
/employer/j	DELETE	Delete a job	Employer	None	{

obs/{jobId}		post.			"message": "Job deleted." }
/employer/jobs/{jobId}/applications	GET	View applications for a job.	Employer	None	{ "applications": [{ "id": "...", "employee": {... }, "applied_at": "..." }] }
/employer/applications/{appId}/status	PUT	Update application status.	Employer	{ "status": "shortlisted" }	{ "message": "Status updated.", "whatsapp_sent": true }

4. UI/UX Functional Specification

The user interface must be intuitive and guided, catering to the distinct needs of each user role. The following sections outline the key user journeys and the functional design of the main pages.

4.1. Employee User Journey

The employee registration process is a crucial multi-step journey designed to mitigate user fatigue and collect comprehensive data incrementally [1].

- **Registration (6-Step Flow):**
  1. **Step 1 (Account Details):** The initial screen will request core credentials like Email, Mobile, Name, Password, and Gender. It must feature clear input validation to ensure data integrity.

2. **Step 2 (Basic Details):** This step collects personal details such as Date of Birth and Address.
3. **Step 3 (Education Details):** A dynamic form that allows the employee to add multiple educational entries with fields for degree, university, and dates.
4. **Step 4 (Experience Details):** A similar dynamic form for professional experience, including company, title, dates, and a description.
5. **Step 5 (Skills Details):** A user-friendly interface for adding and managing skills, potentially using a tag-based input field.
6. **Step 6 (Upload CV):** The final step is a simple file upload interface for the employee's CV [1]. A progress bar or visual indicator should be present across all steps to provide a sense of progression and reduce abandonment.

- **Employee Dashboard:** The employee's central hub will present a clear overview of their activity. It should have dedicated sections or widgets for Shortlisted Jobs, Applied Jobs, and My Plan Details [1]. The dashboard will also feature a download button for a non-professional CV automatically generated from their registration details [1].

4.2. Employer User Journey

The employer interface is streamlined for efficiency, focusing on core tasks of job management and application review [1].

- **Registration:** A single, straightforward registration form is required, capturing the Company Name, Email, Contact, Address, and Industry Type [1]. Upon successful completion, the employer is automatically assigned a default plan [1].
- **Employer Dashboard:** This dashboard serves as the command center for the employer. It must provide clear access to:
  - **Job Management:** A table or list view of all posted jobs with options to Add, Edit, or Delete [1].
  - **Application Viewing:** The ability to view all applications received for each specific job [1].
  - **Plan Details:** A section to display the current plan, its features, and a clear path to upgrade.

4.3. Admin Panel UI

The Admin Panel is a secure, data-driven application with distinct interfaces for each administrative role [1].

- The UI must adhere to the Role-Based Access Control (RBAC) matrix, dynamically displaying only the modules and actions a specific administrator is authorized to use [1]. For example, a Catalog Manager would only see the interface for managing skills and locations, while an Employee Manager would see the tools to manage employee profiles and CV requests [1]. The interface should be built with robust search, filter, and CRUD (Create, Read, Update, Delete) functionality to facilitate efficient data management [1].

5. Business Logic and Workflow Diagrams

This section describes the complex business workflows that govern the platform's core functionalities, translating the high-level requirements into a step-by-step process.

5.1. Job Application and WhatsApp Notification Flow

The job application process is a two-way communication channel facilitated by WhatsApp [1].

1. An **Employee** finds a job and clicks the "Apply" button.
2. The system records the application and sends an API request to the WhatsApp messaging service to notify the **Employer** that a new application has been received.
3. The **Employer** reviews the applications on their dashboard.
4. The **Employer** updates the application status (e.g., to "Shortlisted," "Interview Date," or "Selected").
5. The system detects the status change and triggers a new API request to the WhatsApp messaging service to inform the **Employee** of the update [1].

5.2. Coupon-Based Commission System Workflow

This automated workflow enables sales staff to earn commissions through unique coupon codes [1].

1. A Super Admin or Account-Privileged Admin generates a unique Coupon code and assigns it to a Sales Staff member.
2. The **Sales Staff** shares the code with a user.
3. The **User** applies the coupon at checkout when purchasing a plan [1].

4. The system automatically verifies the coupon, applies the discount to the payment, and calculates a fixed percentage of the transaction as commission [1].
5. This commission amount is then automatically added to the Staff's account balance within the system [1].
6. The Staff member can log in to a dedicated panel to view their commissions, and Super Admins can view all transactions [1].

5.3. Manual Commission Workflow

This workflow addresses commission scenarios where a coupon was not used [1].

1. A **User** completes a plan payment without a coupon code.
2. The **Sales Staff** member who influenced the sale submits the transaction details to the office.
3. A Super Admin or Account-Privileged Admin verifies the transaction details.
4. The admin manually adds the commission amount to the Staff's account balance via the Admin Panel [1].
5. This manual entry is recorded in the system, and the commission becomes visible to the Staff member in their panel [1].

5.4. Role-Based Access Control Matrix

The following matrix specifies the permissions for each administrative role, ensuring security and a clear division of responsibilities [1].

Module	Super Admin	Employee Manager	Employer Manager	Plan Upgrade Manager	Catalog Manager
Employees	CRUD	CRUD	View	View	None
Employers	CRUD	View	CRUD	View	None
Jobs	CRUD	None	CRUD	None	View

Plans	CRUD	None	None	CRUD	None
Catalogs	CRUD	None	None	None	CRUD
Commissions	View All, Manual Add	None	None	View All, Manual Add	None
Content	CRUD	None	None	None	None
Media	CRUD	None	None	None	None

*CRUD stands for Create, Read, Update, and Delete. "View" implies read-only access. "None" signifies no access to the module.

Conclusion

This report provides a comprehensive, expert-level technical blueprint for the development of the job portal platform. By translating the high-level business requirements into detailed architectural diagrams, precise data schemas, and a fully specified API contract, the document eliminates ambiguity and provides developers with a clear roadmap for implementation. The strategic architectural decisions, such as the use of Next.js for SEO and an asynchronous messaging queue for real-time notifications, directly address key business goals and lay the groundwork for a performant and scalable system. The detailed API specifications and schema definitions, which were inferred to fill critical gaps in the source documentation, form the core of this blueprint. Furthermore, the detailed workflow analyses for the plan system, commission models, and notification features provide a robust framework for implementing the platform's core business logic. This documentation is a complete technical guide, ready to be handed to a development team for immediate execution.