COLLEGE CODE: [8223]

COLLEGENAME: [vandayarengineeringcollege]

DEPARTMENT: [Computer Science And Engineering]

STUDENT NM- ID :

[D1B44DEFD2A3219BCA84BFAFCCD771F0]

ROLLNO: [822323104023]

DATE: [26.09.2025]

Completed the project named as

Phase-2

TECHNOLOGY PROJECT NAME: ADMIN DASHBOARD WITH CARDS

SUBMITTED BY,

NAME: [Ramya.M]

MOBILENO: [7094330361]

1. PROBLEM STATEMENT

In the modern software development landscape, organizations and developers face significant challenges in selecting an appropriate technology stack (frontend, backend, database, and deployment platform) for their applications. With the rapid evolution of frameworks, libraries, and cloud services, it becomes difficult to make data-driven decisions that balance performance, scalability, popularity, and ease of use.

Currently, most decisions are made subjectively based on personal preferences or limited experience, which often leads to suboptimal choices, higher costs, and technical debt in the long run. Additionally, administrators and managers lack a centralized dashboard to visualize technology adoption trends, analyze usage statistics, and generate reports that could guide informed decision-making.

2. Tech Stack Selection

A good modern admin dashboard usually uses this stack:

Frontend

- React.js (or Angular / Vue, but React is most popular)
- Tailwind CSS (for styling)
- Charting Library:
    - Recharts (simple & responsive)
    - or Chart.js (for more customization)

Backend

- Node.js (Express.js) or Django / Flask (Python) or Spring Boot (Java) depending on your ecosystem.
- REST API or GraphQL API for data exchange.

Database

- MongoDB (NoSQL, flexible) or
- PostgreSQL / MySQL (Relational, structured data)

Authentication

- JWT (JSON Web Tokens) or OAuth

Deployment

- Docker + Kubernetes (for scaling)
- Cloud: AWS / Azure / GCP
- Simpler: Vercel (Frontend) + Render / Railway (Backend)

Example Features in Dashboard

1. User Management (CRUD)
2. Tech Stack Selection Module – Dropdown or cards for selecting frontend, backend, DB, hosting.
3. Charts for Analysis
    - Pie Chart: % usage of different stacks
    - Line Chart: Performance benchmarks
    - Bar Chart: Popularity comparison

3. UI Structure (Frontend Layout)

1. Sidebar Navigation

- Dashboard (Charts overview)
- Tech Stack Selection
- User Management
- Reports / Logs
- Settings

2. Topbar

- Search bar
- Notifications
- Profile dropdown (Admin info, logout)

3. Main Content (Cards + Charts)

- **Dashboard Page**
    - KPI Cards (e.g., Total Users, Selected Projects, Popular Stack %)
    - Charts: Pie Chart (Frontend popularity), Bar Chart (Backend usage), Line Chart (Trends)
- **Tech Stack Selection Page**
    - Form with dropdowns / radio buttons:
        - Frontend: React, Angular, Vue, Svelte
        - Backend: Node.js, Django, Spring Boot, Flask
        - Database: MongoDB, PostgreSQL, MySQL
        - Deployment: AWS, Azure, GCP, Vercel
    - Submit button → Saves to DB
- **Reports Page**
    - Table with selected stacks, usage stats, timestamps

API Schema Design

Base URL: `/api/v1`

Auth

```
POST /auth/login
POST /auth/register
POST /auth/logout
```

User

```
GET    /users          → List all users
GET    /users/:id      → Get user by ID
POST   /users          → Create new user
PUT    /users/:id      → Update user
DELETE /users/:id      → Delete user
```

Tech Stacks

```
GET    /stacks               → List all available stacks
POST   /stacks/select        → Save user's selected stack
GET    /stacks/selected/:id  → Get selected stack for user/project
```

Data Handling Approach

1. Data Flow Overview

```
UI (React Dashboard)
(REST/GraphQL calls)
Backend API (Node.js/Express or Django/Spring Boot)
(ORM / Query builder)
Database (PostgreSQL / MongoDB)
(Aggregations & Reports)
Analytics Layer (Chart Data)
```

2. Frontend (React/Tailwind + Charts)

- **State Management**:
  - Use **React Query** or **Redux Toolkit** for API data fetching & caching.
  - Form state: Controlled components (React Hook Form for selection page).
- **Data Handling Strategy**:
  - **Charts** get data via `/analytics/*` endpoints.
  - **Forms** send JSON payloads via `/stacks/select`.
  - **Optimistic Updates** (update UI immediately, rollback if API fails).

3. Backend (API Layer)

- **Request Validation**:
  - Middleware (e.g., Joi / Yup / Pydantic) for validating tech stack submissions.
- **Data Transformation**:
  - Normalize request → Map frontend, backend, DB, deployment into DB-friendly schema.
  - Example: Convert `"React"` into `{id: "f1", name: "React"}` internally.
- **Caching**:
  - Use **Redis** for popular analytics queries (e.g., "frontend popularity %").
  - Cache invalidates when new selections are saved.

4. Database Handling

- **Relational (PostgreSQL / MySQL)**
  - Tables: `users`, `projects`, `stacks`, `selections`
  - Foreign keys to keep relationships clear.
- **NoSQL (MongoDB)**
  - Collection: `stackSelections` with embedded documents.

5. Analytics & Charts

- **Aggregation Pipelines** (MongoDB) OR **GROUP BY queries** (SQL):
  - Count how many users picked each stack.
  - Compute popularity % = (count / total) × 100.
- **Trend Analysis**:
  - Time-series queries grouped by `createdAt`.

6. Security & Data Integrity

- **Authentication**: JWT-based → ensures only logged-in users submit/view stacks.
- **Authorization**: Role-based (Admin vs Normal User).
- **Data Validation**: Ensure submitted values match allowed stacks.
- **Audit Logs**: Keep a history of stack changes per user/project.

7. Deployment & Scaling

- Use **message queues (Kafka / RabbitMQ)** if analytics grows large → decouple real-time analytics from user submissions.
- API Gateway with **rate limiting** to protect backend.
- Partition database if millions of submissions come in.

In short:

- **Frontend** → Fetches data via APIs, caches results, sends selections.
- **Backend** → Validates, transforms, saves to DB, caches analytics.
- **Database** → Stores selections, runs aggregations for charts.
- **Analytics Layer** → Serves chart-ready JSON.

4. Module Diagram

Module Breakdown

Frontend (React / Tailwind / Recharts)

- **Dashboard Module** → Displays KPIs + Charts
- **Tech Stack Selection Module** → Form to pick Frontend/Backend/DB/Deployment
- **Reports Module** → Table of projects & selected stacks
- **User Management** → CRUD for admins
- **Auth Module** → Login/Logout/Profile

Backend (Node.js/Express or Django/Spring Boot)

- **Auth Module** → JWT/OAuth handling
- **User Module** → CRUD APIs for users
- **Tech Stack Module** → `/stacks/select`, `/stacks/selected`
- **Analytics Module** → `/analytics/*` (chart data endpoints)
- **Reports Module** → `/reports` with filters
- **Validation & Security** → Middleware for sanitization, RBAC

Database & Analytics

- **Users Table / Collection**
- **Projects Table / Collection**
- **StackSelections Table / Collection** (links user + project + chosen stack)
- **Reports View** (materialized or computed)
- **Analytics Layer** (aggregation queries for charts)

So the **flow** is:

1. User interacts with UI (Tech stack selection / Dashboard).
2. API layer handles logic + validation.

3. DB stores selections & runs analytics queries.
4. Charts module fetches aggregated data.

5. Basic Flow Diagram

Flow Explanation

1. **Admin/User Action** → Logs in, selects tech stack, or views dashboard.
2. **Frontend (UI Layer)** → Sends request (`/stacks/select`, `/analytics/*`) via API.
3. **Backend (API Layer)** → Validates input, stores selections, runs aggregation queries.
4. **Database** → Stores raw selections + computes grouped data for analytics.
5. **Charts Module** → Fetches ready-to-use JSON (e.g., `{name: "React", value: 45}`) and renders Pie/Bar/Line charts.