Sure, here are the updated table configurations:

users

| Column Name | Data Type | Description |
| --- | --- | --- |
| user\_id | integer | Unique identifier for each user |
| email | string | User’s email address |
| password | string | User’s password |
| account\_type | string | User’s account type (job seeker or employer) |

job\_history

| Column Name | Data Type | Description |
| --- | --- | --- |
| job\_history\_id | integer | Unique identifier for each job history entry |
| user\_id | integer | Foreign key referencing the user who owns the job history |
| job\_title | string | Job title |
| company\_name | string | Company name |
| start\_date | date | Start date of the job |
| end\_date | date | End date of the job |

job\_listings

| Column Name | Data Type | Description |
| --- | --- | --- |
| job\_id | integer | Unique identifier for each job listing |
| employer\_id | integer | Foreign key referencing the employer who posted the job |
| job\_title | string | Job title |
| company\_name | string | Company name |
| location | string | Job location |
| job\_description | string | Job description |
| application\_instructions | string | Application instructions |

applications

| Column Name | Data Type | Description |
| --- | --- | --- |
| application\_id | integer | Unique identifier for each job application |
| job\_id | integer | Foreign key referencing the job listing to which the application is submitted |
| user\_id | integer | Foreign key referencing the job seeker who submitted the application |
| resume | string | File path to the job seeker’s resume |
| cover\_letter | string | File path to the job seeker’s cover letter |

Here is an example of an MVP for a job board application that includes the features you mentioned:

## Architecture

The above diagram illustrates the end-to-end flow of data in the job board MVP. The web client communicates with the web server through a RESTful API. The web server interacts with a database to store and retrieve data.

## APIs and Methods

The following API routes will be created for the web client to communicate with the web server:

* /register: Allows users to create accounts as either job seekers or employers.
* /login: Allows users to log in to their accounts.
* /logout: Allows users to log out of their accounts.
* /profile: Allows users to create and edit their profiles, including personal information, job history, skills, and company details.
* /job-listings: Allows employers to post job listings.
* /job-listings/:id: Allows job seekers to view job details for a specific job listing.
* /job-search: Allows job seekers to search for job listings based on location, job type, and industry.
* /apply/:id: Allows job seekers to apply for jobs directly through the platform and upload resumes and cover letters.
* /dashboard: Allows users to manage their profiles, job listings, and job applications.
* /notifications: Allows users to receive email or push notifications about important updates, such as new job listings or application responses.

The following API endpoints or function/methods will be created to allow other clients to use:

* createUser(): Creates a new user account.
* loginUser(): Logs in a user.
* logoutUser(): Logs out a user.
* createProfile(): Creates a new user profile.
* updateProfile(): Updates an existing user profile.
* createJobListing(): Creates a new job listing.
* getJobListing(): Retrieves job details for a specific job listing.
* searchJobListings(): Searches for job listings based on location, job type, and industry.
* applyForJob(): Allows job seekers to apply for jobs directly through the platform and upload resumes and cover letters.
* getDashboard(): Retrieves user dashboard data.
* sendNotification(): Sends email or push notifications to users.

## Data Model

The above diagram illustrates the data model for the job board MVP. The database consists of four tables: users, profiles, job\_listings, and applications. The users table stores user account information, while the profiles table stores user profile information. The job\_listings table stores job listing information, and the applications table stores job application information.

## User Stories

Here are some detailed user stories that the job board MVP will satisfy:

1. As a job seeker, I want to be able to create a profile that includes my personal information, job history, skills, and company details so that employers can find me easily.
2. As an employer, I want to be able to post job listings that include job details such as title, company, location, job description, and application instructions so that job seekers can apply for the job.
3. As a job seeker, I want to be able to search for job listings based on location, job type, and industry so that I can find relevant job openings.
4. As a job seeker, I want to be able to apply for jobs directly through the platform and upload resumes and cover letters so that I can easily apply for jobs.
5. As a user, I want to be able to manage my profiles, job listings, and job applications through a user dashboard so that I can easily keep track of my activity on the platform.
6. As a user, I want to be able to receive email or push notifications about important updates, such as new job listings or application responses so that I can stay up-to-date with the latest information.

I'll provide a textual description of the requested components for your Job Board MVP, including the architecture, APIs and methods, data model, and user stories. Please note that due to the limitations of a text-based response, I cannot provide actual diagrams. You may need to create visual representations of these components using appropriate tools.

Architecture:

The Job Board MVP has a client-server architecture. Here's an overview:

1. Client-Side (Web Client):
   * User Interface: Handles user registration, login, user profiles, job search, application submission, and user dashboard.
   * Interacts with the Web Server through API endpoints.
2. Server-Side (Web Server):
   * Receives requests from the web client.
   * Processes requests and interacts with the database.
   * Sends responses back to the client.
   * Manages user authentication and authorization.
3. Database:
   * Stores user profiles, job listings, job applications, and related data.
   * Supports data retrieval and storage operations for the web server.

APIs and Methods:

For communication between the web client and the web server, you'll need the following API routes:

* User Registration and Profiles:
  + POST /api/users/register: Allows users to create accounts.
  + POST /api/users/login: Handles user login.
  + GET /api/users/:id/profile: Retrieves user profiles.
  + PUT /api/users/:id/profile: Enables users to edit their profiles.
* Job Listings:
  + POST /api/job-listings: Allows employers to post job listings.
  + GET /api/job-listings/:id: Retrieves job listing details.
  + GET /api/job-listings: Provides a list of job listings.
  + PUT /api/job-listings/:id: Allows employers to edit job listings.
  + DELETE /api/job-listings/:id: Allows employers to delete job listings.
* Job Search and Filters:
  + GET /api/job-listings/search: Implements job search functionality.
  + GET /api/job-listings/filters: Provides filter options (e.g., location, job type, industry).
* Application Submission:
  + POST /api/applications: Allows job seekers to submit applications.
  + GET /api/applications/:id: Retrieves application details.
  + GET /api/applications/job/:jobId: Retrieves applications for a specific job listing.
* User Dashboard:
  + GET /api/users/:id/dashboard: Retrieves data for user dashboards (profile, job listings, job applications).
* Notifications:
  + Implement email and push notification systems to notify users about relevant updates (e.g., new job listings, application responses).

Data Model:

The data model includes the following entities:

1. Users: Stores user information (name, email, password), user type (job seeker or employer), and user profiles.
2. Job Listings: Contains details of job listings, including title, company, location, job description, and application instructions.
3. Applications: Records job applications, including the applicant's information, the job applied for, and attached resumes and cover letters.

User Stories:

1. As a job seeker, I want to register an account, create a profile, search for relevant job listings, and submit applications with my resume and cover letter.
2. As an employer, I want to register an account, post job listings, and receive applications from job seekers.
3. As a user (job seeker or employer), I want a user dashboard to manage my profile, job listings, and job applications.
4. As a job seeker, I want to receive notifications about new job listings matching my preferences and updates on my job applications.
5. As an employer, I want to receive notifications about new job applications and updates on my job listings.

These user stories cover the core functionality of your Job Board MVP and ensure that both job seekers and employers can effectively use the platform to find and post jobs.

Web Architecture for Job Board Application:

1. Front-End (Web Client):
   * User Interface (UI): Built using HTML, CSS, and JavaScript.
   * Front-end Framework (e.g., React, Angular, or Vue.js): Provides a responsive and interactive user interface.
   * User Registration and Login Components.
   * User Profile Management Components.
   * Job Listing Pages.
   * Job Search and Filtering Components.
   * Application Submission Forms.
   * User Dashboard Components.
   * Notifications System (may use WebSocket or other technologies for real-time updates).
   * Responsive Design for Mobile Devices.
2. Back-End (Web Server):
   * Web Application Framework (e.g., Express.js, Ruby on Rails, Django, or ASP.NET):
     + Handles HTTP requests and routing.
     + Implements middleware for authentication and validation.
   * User Authentication and Authorization:
     + Manages user accounts and sessions.
     + Utilizes tokens or cookies for user sessions.
   * RESTful API Endpoints:
     + User Management: Registration, Login, Profile Editing.
     + Job Listings: Create, Read, Update, Delete (CRUD) operations.
     + Job Search and Filtering.
     + Application Submission.
     + User Dashboard Data Retrieval.
     + Notifications Handling.
   * Data Validation and Sanitization: Protects against security vulnerabilities.
   * Integration with Third-Party Services: Email notifications, push notifications, payment processing (if applicable).
   * Security Measures: Protection against common web application vulnerabilities.
3. Database:
   * Database System (e.g., PostgreSQL, MySQL, or MongoDB):
     + Stores user profiles, job listings, applications, and other data.
   * Database Schema:
     + Tables for users, job listings, and applications, structured according to your data model.
     + Indexing for efficient data retrieval.
   * Database Caching: Improve performance through caching mechanisms.
4. APIs and Communication:
   * RESTful or GraphQL APIs for communication between the web client and server.
   * Secure Communication: HTTPS to protect data in transit.
5. Scalability and Performance:
   * Horizontal Scalability: Deployment across multiple servers or cloud-based scaling.
   * Optimization: Database query optimization, caching, and use of CDNs for static assets.
6. Security:
   * Security Best Practices: Protection against SQL injection, XSS, CSRF, and other vulnerabilities.
   * Dependency Updates: Regular updates to address security issues.
   * Security Audits and Penetration Testing.
7. Monitoring and Analytics:
   * Monitoring Tools: Track application performance, errors, and health.
   * Analytics: Gather insights into user behavior and application usage patterns.
8. Deployment and Hosting:
   * Hosting Solution: Choose a reliable hosting platform or service (e.g., AWS, Azure, GCP).
   * Deployment Automation: Scripts for automated deployment and updates.
9. Testing and Quality Assurance:
   * Testing: Unit testing, integration testing, and end-to-end testing.
   * Staging Environment: Testing new features and updates before deploying to production.
10. Documentation:
    * Comprehensive documentation for developers and API users.
11. Backup and Disaster Recovery:
    * Regular data backups and a disaster recovery plan to prevent data loss.

This outline serves as a foundation for your Job Board application's architecture. Depending on your technology stack and specific requirements, you will need to make more detailed design and technology choices.

Creating an MVP for a Job Board Application:

Architecture:

Job Board Application MVP Architecture

In the above architecture diagram:

The web client interacts with the web server through RESTful APIs.

Data is stored in a relational database.

Notifications are sent through email or push notification services.

APIs and Methods:

i. API Routes for Web Client:

User Registration and Profiles:

POST /api/users/register - User registration.

POST /api/users/login - User login.

PUT /api/users/:id - Update user profile information.

Job Listings:

POST /api/jobs - Create a new job listing.

GET /api/jobs/:id - Get details of a specific job listing.

GET /api/jobs - Retrieve a list of job listings based on search criteria.

Application Submission:

POST /api/applications - Submit a job application.

GET /api/applications/:id - Get details of a specific application.

User Dashboard:

GET /api/users/:id/dashboard - Retrieve user dashboard information.

Notifications:

Notifications are sent automatically based on various triggers and user preferences.

ii. API Endpoints for Other Clients (for future use):

These endpoints could be used for mobile apps, third-party integrations, etc.

Data Model:

Data Model for Job Board MVP

User:

user\_id (PK)

name

email

password (hashed)

role (job seeker or employer)

UserProfile:

profile\_id (PK)

user\_id (FK)

personal\_info

job\_history

skills

company\_details (if the user is an employer)

JobListing:

job\_id (PK)

employer\_id (FK)

title

company

location

description

application\_instructions

JobApplication:

application\_id (PK)

job\_id (FK)

seeker\_id (FK)

resume

cover\_letter

Notification:

notification\_id (PK)

user\_id (FK)

message

timestamp

status (read/unread)

User Stories:

User Registration:

As a new user, I can register an account by providing my name, email, and password.

As a job seeker, I can provide additional personal information during registration.

Job Posting:

As an employer, I can post a job listing with details like title, company, location, description, and application instructions.

Job Search and Filters:

As a job seeker, I can search for job listings using keywords and apply filters like location, job type, and industry.

Application Submission:

As a job seeker, I can apply for a job by uploading my resume and cover letter through the platform.

User Dashboard:

As a user, I can log in and access my dashboard to manage my profile, view my job listings, and track my job applications.

Notifications:

As a user, I receive notifications about new job listings and application responses through email or push notifications.

This MVP will allow you to create a functional job board application that serves the basic needs of both job seekers and employers while setting a solid foundation for future feature enhancements.