Office of Career Services (OCS), IIT Delhi

Technical team recruitment

Consider SQL tables with the following definition:

```
CREATE TABLE users (
  userid VARCHAR(64) PRIMARY KEY,
  password_hash VARCHAR(64) NOT NULL,
  role VARCHAR(16) NOT NULL -- admin | recruiter | student
);

CREATE TABLE public.profile (
  profile_code   SERIAL PRIMARY KEY,
  recruiter_email VARCHAR(128) NOT NULL,
  company_name    VARCHAR(255) NOT NULL,
  designation     VARCHAR(255),
  FOREIGN KEY (recruiter_email) REFERENCES public.users(userid) ON DELETE CASCADE
);

CREATE TABLE public.application (
  profile_code   INTEGER NOT NULL,
  entry_number   VARCHAR(64) NOT NULL,
  status         VARCHAR(32) NOT NULL DEFAULT 'Applied',  -- Applied | Not Selected | Selected | Accepted
  PRIMARY KEY (profile_code, entry_number),
  FOREIGN KEY (profile_code) REFERENCES public.profile(profile_code) ON DELETE CASCADE,
  FOREIGN KEY (entry_number) REFERENCES public.users(userid) ON DELETE CASCADE
);
```

With the following test data:

userid	password_hash	role	raw_password (note: NOT in DB)
admin	5f4dcc3b5aa765d61d8327deb882cf99	admin	password
student1	aeddf07d1ab10bd6d8dde8b778368511	student	spassword1
student2	269afc90dec9f5bb8da8dfce984a2232	student	spassword2
student3	3a1a4b50a02aa4312cb674d4dae587ce	student	spassword3
recruiter1@techcorp.com	ca290bcb62ce43c5b34c2005ec4a91ed	recruiter	rpassword1
recruiter2@dataworks.com	3e943c50ba499e8b11995c7d76cc7f1f	recruiter	rpassword2

profile_code	recruiter_email	company_name	designation
1001	recruiter1@techcorp.com	TechCorp	Backend Intern
1002	recruiter1@techcorp.com	TechCorp	Frontend Intern
1003	recruiter2@dataworks.com	DataWorks	ML Intern
1004	recruiter2@dataworks.com	DataWorks	Software Engineer

profile_code	entry_number	status
1001	student1	Selected
1001	student2	Applied
1002	student1	Applied
1003	student2	Applied
1004	student3	Accepted

Problem Statement:
Develop a secure REST API and a browser-based client that allows users to authenticate and view/manage data stored in the recruitment database. The system models three roles **admin**, **recruiter**, and **student** and supports profiles (job/internship postings) and student applications

Core constraints:
1) The code for the front end should NOT contain any sensitive information that could be accessed using the Developer console.
2) All data exchange with the SQL table must occur on the server (Through the API endpoint).
3) The password_hash field in the table is the md5 hash of the raw_password.
4) The Browser Client must not transfer raw_password to the server.

Authentication & basic behavior

1) A user logs in by entering `userid` and password in the client. The client computes the MD5 hash of the password and sends `{ userid, password_md5 }` to the server's login endpoint.
2) The server validates the provided hash against `users.password_hash`. On success, the server returns a short-lived session token (JWT or session cookie) encoding the user identity and role. The token is used to authenticate subsequent API requests.

UI & server requirements:

1. Student

- **D**isplay only the student's own user record (no other users).
- **Profiles list:** show a list of all available profiles for the relevant session(s) with essential fields (profile_code,company name, designation). Each profile row should include an **Apply** button for students who are eligible and permitted to apply.
- **Application behavior:** when the student clicks **Apply**, the client calls the server API to create an `application` record with `status = 'Applied'`. The server enforces eligibility (for example, prevents applying if the student already has an `Accepted` offer in

that session).

- **Selected/Accepted flow:** if the student has any application with `status = 'Selected'`, then immediately after login the UI must prominently show the selected profile(s) — **do not** show the profiles list until the selected offer is resolved. The student must get an option to **Accept** or **Reject** the selected offer.
    - If the student **Accepts**, the server sets `status = 'Accepted'` for that application and marks the student/session as accepted (so further applications in that session are blocked). After acceptance the client should show only the student's data and a congratulatory message such as:

      Congratulations! You have accepted an offer from *Company Name (Designation)*.
      and should **not** list any profiles for application.
    - If the student **Rejects**, the server should revert the application status appropriately (for example to `'Not Selected'`) and the client should resume showing the profiles list.

2. Recruiter

- Display the recruiter's own details.
- **Profiles management:** allow the recruiter to create and manage profiles (job/internship postings) that are associated with their `recruiter_email`.
- **Applications view & actions:** list all applications submitted to the recruiter's profiles (joined by `profile_code`). For each application the recruiter may change the application `status` of a student to `'Selected'` (or back to another status). Recruiters can **not** see and thus change applications for profiles they do not own.

3. Admin

- **Full access:** admin users can view all users, all profiles, and all applications. Admins may create profiles for any recruiter and may change application status (including converting a `'Applied'` application to `'Selected'`).
- **Visibility:** admin views may show additional management-only fields; however, sensitive raw data (such as `password_hash`) must be treated carefully.

API expectations (examples)

Provide REST endpoints that support the above flows. Example endpoints (exact names/paths are flexible, document what you implement):

- `POST /api/login` — authenticate with `{ userid, password_md5 }`. Returns token + role.
- `GET /api/users/me` — returns logged-in user record (student/recruiter/admin view as appropriate).
- `GET /api/profiles` — list profiles (include `profile_code`).
- `POST /api/create_profile` — create a profile (recruiter-only or admin).
- `POST /api/apply` — student applies to a profile.
- `POST /api/application/change_status` — recruiter/admin marks application as `Selected`. Or can change any other status
- `POST /api/application/accept` — student accepts a `Selected` application (transitions to `Accepted`).

You are encouraged to open dev tools and monitor any network requests.
Try to host the client and the server code on the cloud. Some recommended services are mentioned at the end.
Required to submit: A public GitHub/Gitlab repository link with proper documentation.
Some recommended services:
1) For hosting the front-end client: vercel
2) For server hosting (API endpoint): vercel
3) For cloud SQL DB service: SupaBase
4) Testing API: postman
5) Generate md5 hashes here
You are free to use any languages/technologies apart from the ones mentioned above.
You are expected not to devote more than 6-8 hours towards this whole process. Even if you cannot complete the assignment, be sure to learn some new things during the same. You may submit the incomplete code as well while explaining the problems faced.
Submission deadline: January 25, 2026 EOD
Submission link: