LEARNLYNK — TECHNICAL TEST

SECTION 1 — Supabase Schema Challenge (45 minutes)

Task:
Create the SQL schema for 3 tables, with constraints + indexes + relationships.

Tables:

1. **leads**
2. **applications**
3. **tasks**

Requirements:

- Every table must include: id, tenant_id, created_at, updated_at
- applications references leads(id)
- tasks references applications(id) (as related_id)
- Add proper FOREIGN KEY constraints
- Add indexing for common queries:
    - fetch leads by owner, stage, created_at
    - fetch applications by lead
    - fetch tasks due today
- Add a constraint:
    - tasks.due_at >= created_at
- Add a check constraint:
    - task.type IN ('call', 'email', 'review')

Deliverable:

A **single .sql file** that we should be able to run in Supabase.

SECTION 2 — RLS & Policies Test (30 minutes)

Task:
Write RLS policies for:

Goal:

A counselor should only be able to read leads that are either:

- assigned to them, OR
- assigned to their team

But **Admins can read all leads**.

Inputs:

- leads.owner_id = counselor user id
- user_teams(user_id, team_id)
- teams(team_id)
- user role in JWT (auth.jwt() -> role: "admin" or "counselor")

Deliverables:

- RLS row policy for **SELECT** on leads
- RLS row policy for **INSERT** on leads

SECTION 3 — Edge Function Task (45 minutes)

Task: Build a Supabase Edge Function that:

1. Accepts POST request:

POST /create-task

{

"application_id": "...",

"task_type": "call",

"due_at": "2025-01-01T12:00:00Z"

}

2. Validates:

- task_type must be one of: call, email, review

- due_at must be in the future

3. Inserts into tasks table (using Supabase client)
4. Emits a Supabase Realtime broadcast event: "task.created"
5. Returns:

```
{
 "success": true,
 "task_id": "..."
}
```

Requirements:

- Write in TypeScript
- Use proper error handling
- Validate input
- Use Supabase client with service role
- Include response codes (400, 200)

SECTION 4 — Mini Frontend Exercise (30 minutes)

Task:
Create a small Next.js page:

/dashboard/today

Requirements:

- Fetch tasks due today from Supabase
- Display in a table
- Include:
    - Task title
    - Related application ID
    - Due date
    - Status
- Add a button "Mark Complete"

- On click → update Supabase using supabase.from("tasks").update()
- Refresh UI using React Query or re-fetch
- Show loading & error states

Why this works:

Weak candidates struggle with:

- State management
- APIs
- Mutations
- React Query
- Async handling

Strong ones breeze through.

SECTION 5 — Integration Task (30 minutes)

This is quick but VERY revealing.

Task:
Explain, in 8–12 lines, how they would implement **Stripe Checkout** for an application fee.

Must mention:

- Creating a Checkout session
- Storing payment_request
- Handling Stripe webhook
- Updating payment status
- Updating application stage or timeline