Instructions

- Create a copy of these assessment questions in Google Docs.
- At the very top of your copied document, add your full name and Discord ID.
- Carefully answer all written questions in the document. Be clear, concise and technical.
- After Answering all questions, set your Google Doc to "Anyone with the link can edit."
- The assessor will review your answers directly in your document and update your status (Qualified/Disqualified).
- If you pass, you will be invited to the next stage of the hiring process.

Note :

- I highly recommend that if you have any doubts about the questions below , then go back and watch the video and read the paradigm documents two or three times before answering these questions.

Important :

- You **must complete all answers** in this document before you can be selected for the hands-on technical test.
- Incomplete or inaccessible docs may disqualify your submission.

Questions

1. Authentication & Group Access

- How would you implement secure email-based and password authentication for the Paradigm App, and how would you manage session or token-based access once a user is verified?.

- Paradigm users are scoped to groups (e.g., Dev/Group-1, Creators/Group-2). How would you persist group membership and validate it securely for every API request across microservices?.

- Paradigm's group permission schema may expand (e.g., roles like 'admin', 'editor', 'viewer'). How would you future-proof your system to support role-based permissions per group without redesigning the entire access model?.

2. Local Development & Testing

- What testing tools or strategies would you use in VS Code to validate the behavior of session tokens, user switching, and API protection across microservices?.

- How would you design a local development environment that supports parallel environments (dev, staging), secret injection, mock n8n testing, and live reload across services?.

3. Connector Integration

- Explain how you would set up a complete build/test pipeline locally using Node.js, Docker, oracle databases with the help of VS Code..

4. Cloud Deployment (Oracle Cloud)

- What steps would you take to deploy a containerized Node.js microservice to Oracle Cloud?.

5. Prompt Logging

- How would you structure a NoSQL database schema to store LLM prompts and responses specific to a group, project, and user?

- How you interpret UI elements from Figma and translate them into backend requirements

6. Developer Collaboration & Future-Proofing

- How would you design your backend to allow the n8n engineer to test flows in staging or local environments using mock data from your services — before going to production?

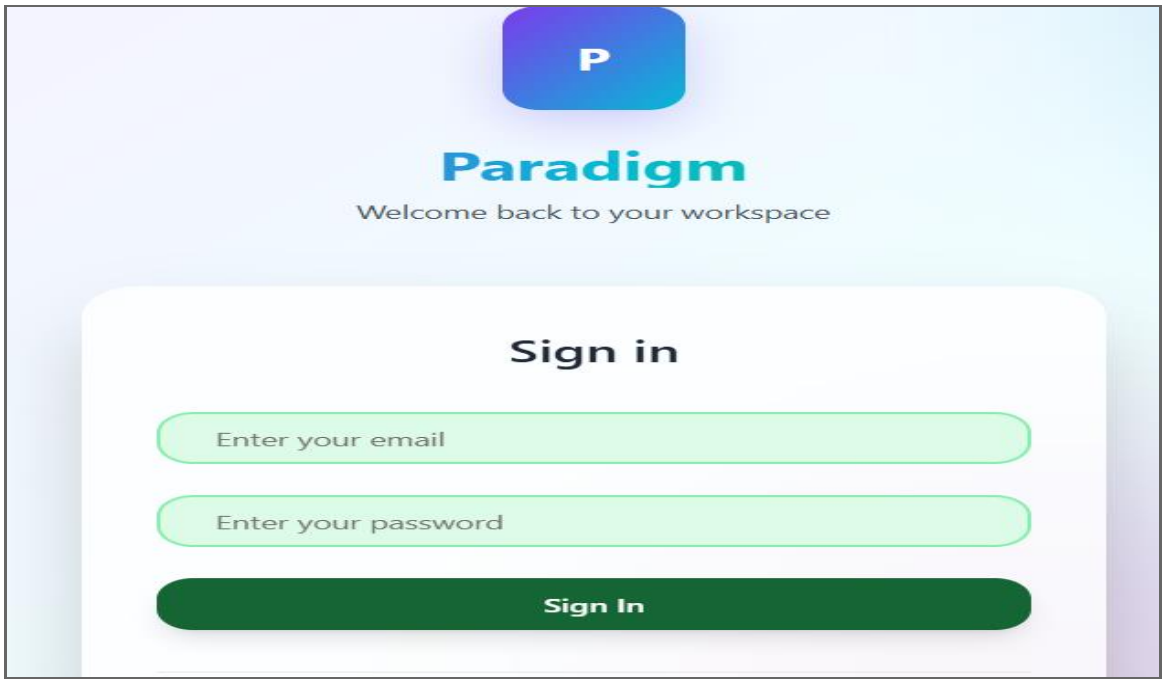
Technical Test – Candidate Instruction

You will undergo a **hands-on backend technical test** based on the Paradigm App. The two most important parts are:

1. **Build a secure user login system** with email, password, and group-based access — tested via n8n calling your VS Code backend.

2. **Expose clean, working API endpoints** (`/auth/validate`, `/groups`, `/n8n/webhook`, `/health`) that return valid data and support group permissions as defined.

Developer Task



1. Paradigm App Login Integration

Your Task: Enable Email + Password Login via API :

In Paradigm, the login process is triggered when a user enters their **email and password** into the frontend form. This data is sent to **n8n**, which then forwards it to your backend for validation.

You must build and expose a secure endpoint in your local backend (VS Code) that supports this authentication check.

Make Your Backend Reachable :

To test the integration:

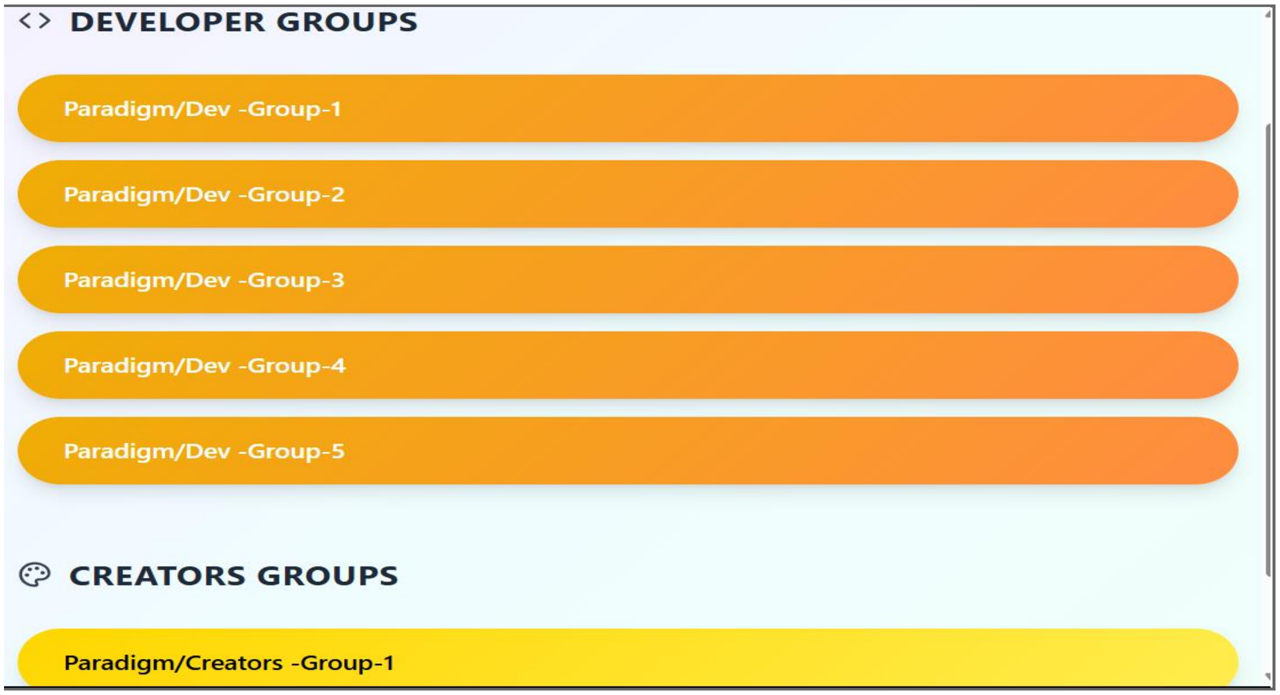
- Run your backend locally using VS Code and Node.js
- Use **ngrok** or any **secure tunnel** to expose your vs code local server
- Share the exposed `/auth/validate` endpoint with us for testing from n8n
- Make sure your backend is reachable by n8n (via ngrok or local tunneling).

Send Your Test Credentials :

Once setup is complete, send:

- Your **test email**
- Your **test password**
- Your **exposed API URL** **(Using services which provide sending email with passwords)**

This will allow us to test your backend login logic using the Paradigm frontend through n8n.



Groups Screen

2. Paradigm App Login Integration

After a user signs in with a valid email and password, your backend must return **only the groups that user is allowed to access**, based on their group membership. These groups will appear as clickable buttons in the UI (like in the image i have provided).

What You Need to Implement:

- Group Membership Schema
- Updated Login Response

Group Scoping Logic

On any future API requests ensure the backend:

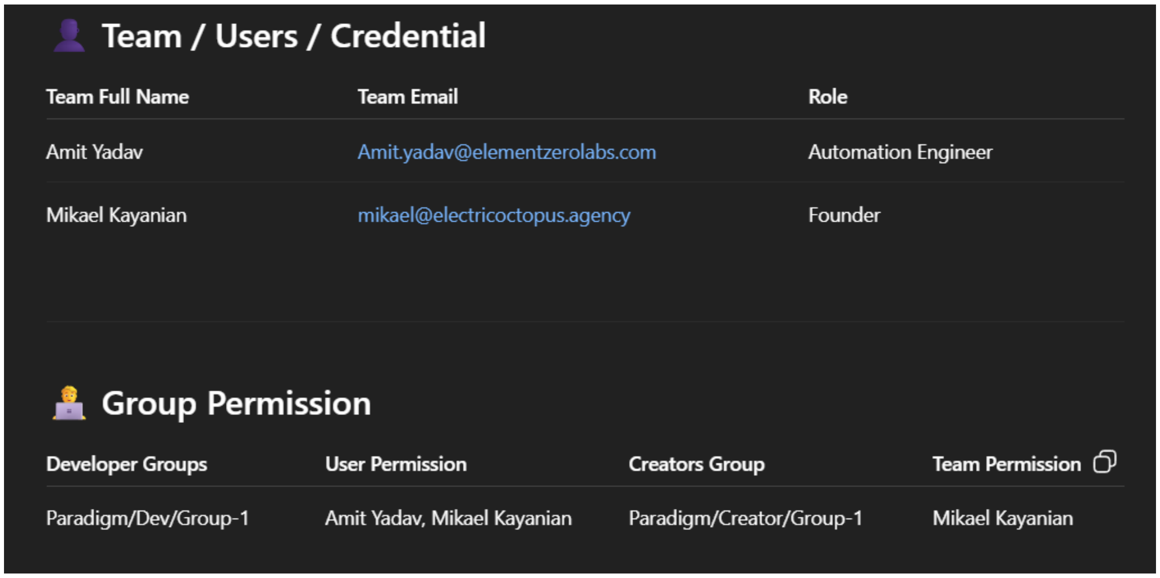
- Checks that the user belongs to the group they are trying to access

- Rejects unauthorized group access attempts

Group Permission Handling :

Each group may eventually have different access levels or features (e.g., only Creator Groups can use design connectors).

- Prepare your schema to support role-based access:



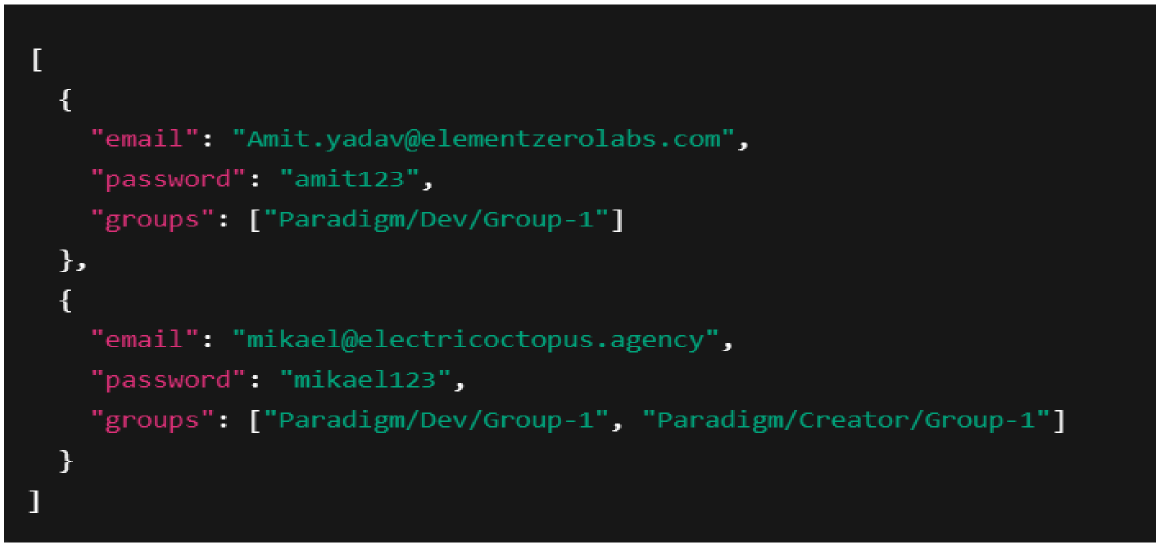
👤 Team / Users / Credential

Team Full Name	Team Email	Role
Amit Yadav	Amit.yadav@elementzerolabs.com	Automation Engineer
Mikael Kayanian	mikael@electricoctopus.agency	Founder

🧑‍💻 Group Permission

Developer Groups	User Permission	Creators Group	Team Permission
Paradigm/Dev/Group-1	Amit Yadav, Mikael Kayanian	Paradigm/Creator/Group-1	Mikael Kayanian

Schema for reference



```
[
  {
    "email": "Amit.yadav@elementzerolabs.com",
    "password": "amit123",
    "groups": ["Paradigm/Dev/Group-1"]
  },
  {
    "email": "mikael@electricoctopus.agency",
    "password": "mikael123",
    "groups": ["Paradigm/Dev/Group-1", "Paradigm/Creator/Group-1"]
  }
]
```

mock user schema for reference only

Test Case Expectations

- You must include **at least 2 users**, each with:

    - Unique email + password
    - Different assigned groups (e.g., one Dev, one Creator)
- Provide this sample data when submitting your project for testing

Note: A user must have access of multiple groups as per requirement , so you have to design your schema accordingly

For testing give mikael permission of both developer and creator group permissions.

e.g. **mock user schema.**