Code Assessment – Email Cadence (Monorepo, TypeScript)

Objective

Build a small application using Next.js + NestJS + Temporal.io (TypeScript SDK) that:
1. Creates an email cadence consisting of steps
2. Enrolls a contact into that cadence
3. Executes the cadence using Temporal.io workflows
4. Allows updating the cadence while the workflow is running, and the workflow continues correctly using the new definition

Technology Requirements

- All code must be TypeScript
- Monorepo structure is required
- Use Temporal.io TypeScript SDK
- No authentication required
- Email sending must be a simple mock that always succeeds
- No test cases required
- UI design is not important

Monorepo Structure (Required)

```
repo/
  apps/
    web/       # Next.js (TypeScript)
    api/       # NestJS (TypeScript)
    worker/    # Temporal.io worker (TypeScript)
  package.json
  tsconfig.base.json
  README.md
```

Cadence Payload Structure (Required Contract)

The entire system must use this JSON format for cadence:

```
{
  "id": "cad_123",
  "name": "Welcome Flow",
  "steps": [
    {
      "id": "1",
      "type": "SEND_EMAIL",
      "subject": "Welcome",
      "body": "Hello there"
    },
    {
      "id": "2",
      "type": "WAIT",
      "seconds": 10
    },
    {
      "id": "3",
      "type": "SEND_EMAIL",
      "subject": "Follow up",
      "body": "Checking in"
    }
  ]
}
```

React (Next.js) Application Requirements

The frontend must only focus on managing this payload and workflow. UI can be extremely basic.

- Create and edit cadence steps using a basic UI that produces the JSON structure above (a JSON textarea editor is acceptable).
- Start workflow by calling POST /enrollments with cadenceId and contactEmail.
- Display workflow state by polling GET /enrollments/:id and showing currentStepIndex, status, and stepsVersion.
- Update a running workflow by calling POST /enrollments/:id/update-cadence with updated steps; the running workflow must adopt the updated steps and continue correctly.

API Requirements (NestJS)

Cadence endpoints:

- POST /cadences – Create cadence
- GET /cadences/:id – Get cadence
- PUT /cadences/:id – Update cadence definition

Enrollment endpoints:

- POST /enrollments – Body: { cadenceId, contactEmail } – Starts Temporal.io workflow
- GET /enrollments/:id – Returns current status

Update-in-flight endpoint:

- POST /enrollments/:id/update-cadence – Body: { steps } – Sends Temporal.io signal to running workflow

Temporal.io Worker Requirements

- Workflow executes steps sequentially.
- WAIT steps use a Temporal.io timer/sleep for the specified seconds.
- SEND_EMAIL steps call an activity that uses a mock function that always returns success.
- Workflow maintains currentStepIndex, stepsVersion, and status.
- Expose a query getState() that returns workflow state.
- Expose a signal updateCadence(steps) that replaces the steps definition at runtime.

Update rules (must implement):

1. Already completed steps remain completed.
2. Keep currentStepIndex.
3. If new steps length <= currentStepIndex, mark workflow COMPLETED.
4. Else continue from currentStepIndex using the new steps.
5. Increment stepsVersion.

Mock Email Requirement

The SEND_EMAIL activity must not call any real email provider. It should log the action and always return success, for example:

```
{
  "success": true,
  "messageId": "string",
  "timestamp": 123456789
}
```

Local Run Requirements

No docker is required for this assessment.

Provide a README that explains how to run:

- apps/api
- apps/worker
- apps/web

Assume Temporal.io is available in the environment. Include placeholders for:

- Temporal.io server address
- Namespace
- Task queue

Monorepo Scripts (Required)

- dev – runs web + api + worker
- dev:web
- dev:api

- dev:worker

Deliverables

- Working TypeScript monorepo
- README with install steps, Temporal.io configuration placeholders, how to start all apps, and example API calls