Scalable Services Assignment

# Group Details

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sl. No. | BITS ID | Name | Contribution (Qualitative) | Contribution % |
| 1 | 2023MT03158 | Bharat Kondam | Individual | 100% |

*Note: Add rows for each member and update contributions as per submission guidelines.*

GIT Repository details:

<https://github.com/bharathkondam/scalableservices/tree/main>

Here’s a cleaner, more human-readable version you can drop into your doc or share with non-tech stakeholders.

# CareConnect Telehealth Platform – Simple Overview

## 1) What is CareConnect?

CareConnect is a digital health app that helps patients find doctors, book online appointments, join video visits, and handle everything after the visit—like prescriptions, bills, and reminders. It needs to handle busy seasons (like flu spikes), respect regional rules, and keep working smoothly even if some parts have issues.

### What we care about most

* **Scalability:** Grow different parts independently (search, booking, reminders).
* **Resilience:** Keep running if something fails (retries, circuit breakers, queues).
* **Extensibility:** Add new specialties, partners, or channels without big rewrites.

## 2) What are the main building blocks?

We split the platform into small services. Each one owns its data and does one job well.

* **Appointment Service:** Search slots, book/reschedule/cancel, enforce availability rules.
* **Notification Service:** Email/SMS/push templates, send messages, track delivery.

## 3) What actions does the system support?

A few examples (C = Command/changes data, Q = Query/reads data):

* **CreateAppointment (C):** Reserve a slot; trigger confirmations.
* **GetAppointment / ListAppointmentsForPatient (Q):** Read appointment details/history.
* **SendNotification / GetNotificationStatus (C/Q):** Send messages and check results.

Typical events we publish so other services can react: **AppointmentConfirmed**, **AppointmentCancelled**, **InvoicePaid**, **ConsultationSummaryReady**.

## 4) How do services talk to each other?

* **Fast reads & critical writes:** Use REST/gRPC for low-latency calls (e.g., check provider availability).
* **Loose coupling with events:** Use a message broker so services can react without direct calls.  
  Example: AppointmentConfirmed → Notification sends a confirmation
* **Long workflows = Saga:** For multi-step processes (book → pay → notify), we orchestrate steps and roll back if something fails.
* **Each service owns its database:** SQL for transactions (e.g., PostgreSQL), document DB for clinical notes (e.g., MongoDB), time-series for metrics (Prometheus).

**Example: Booking Flow (happy path)**

1. App calls **POST /appointments**
2. Gateway authenticates; forwards to **Appointment Service**.
3. Appointment checks patient exists and provider is free.
4. Appointment is saved; publishes **AppointmentConfirmed**.
5. **Notification Service** sends confirmation
6. Metrics, logs, and traces are captured for visibility.

## 5) How do we scale and stay reliable?

* **Scale each service separately:** Containers/Kubernetes with autoscaling (CPU, requests, queue depth).
* **Be failure-tolerant:** Circuit breakers, timeouts, exponential backoff, idempotency keys for commands.
* **Modern infra:** Kubernetes plus a service mesh (Istio/Linkerd) for secure, observable service-to-service traffic.

## 6) Technology choices (reference stack)

* **Services:** Node.js (NestJS/Express
* **Data:** db.js

## 7) What’s in the prototype right now?

We’re building a thin, working slice:

* **Appointment Service:** REST for create/read/cancel; SQLite for demo; emits events.
* **Notification Service:** REST + background worker; consumes appointment events and logs message history.

This slice runs in containers (with sample Kubernetes manifests) and shows how we’ll grow to the full target.

8) Architecture Diagram:

A screenshot of a computer

AI-generated content may be incorrect.

**Implementation Details:**

**Technology Selections**

- **\*\*Language & Framework\*\***: Node.js 20 with Express – fast to bootstrap, broad ecosystem, native async support.

- **\*\*Persistence\*\***: Lightweight JSON file store keeps each service self-contained without native bindings (suited for constrained environments). In production the Appointment

- **\*\*Validation\*\***: Joi schemas enforce payload integrity at the service boundary.

- **\*\*Inter-service communication\*\***:

- REST over HTTP for synchronous interactions. Appointment Service calls the Notification Service when new events occur.

- Asynchronous, event-driven expansion path via a message broker (Kafka/RabbitMQ) described in `docs/architecture.md`. The HTTP call simulates the event emission for the prototype.

**## Repository Layout**

- `appointment-service/` – Self-contained microservice with its own dependency manifest, database, Dockerfile, and deployment descriptors.

- `notification-service/` – Independent microservice with equivalent scaffolding.

- `docs/` – Architecture and implementation notes.

- `k8s/` – Kubernetes manifests defined in Part 3.

Each service can be copy-extracted into its own Git repository; no cross-directory dependencies exist.

**## Running Locally (quickstart)**

1. Start Notification Service: `cd notification-service && npm install && npm start` (defaults to port 3000).

2. Start Appointment Service: `cd appointment-service && npm install && NOTIFICATION\_SERVICE\_URL=http://localhost:3000 npm start`.

3. Create an appointment:

```bash

curl -X POST http://localhost:3100/appointments \

-H "Content-Type: application/json" \

-d '{"patientId":"patient-1","providerId":"provider-42","scheduledFor":"2024-12-01T10:30:00.000Z"}'

```

4. Inspect generated notification:

```bash

curl http://localhost:3000/notifications

The Notification Service logs simulated dispatches to stdout.