Backend + Flutter Hybrid Developer Assessment

What You're Building

A task management application that automatically classifies and organizes tasks based on content analysis.

Example: User creates: *"Schedule urgent meeting with team today about budget allocation"*

System automatically:

- Detects category: **Scheduling** (keywords: meeting, schedule)
- Assigns priority: **High** (keywords: urgent, today)
- Extracts entities: team, budget
- Suggests actions: Block calendar, Send invite, Prepare agenda

---

Deliverables

Item	Required
Backend API (Node.js/FastAPI)	5 core endpoints
Database Schema	PostgreSQL/Supabase with 2 tables
Flutter Mobile App	Single dashboard screen
Live Deployment	Backend on Render.com
Documentation	GitHub README with setup instructions
Testing	Minimum 3 unit tests

Who Should Apply

- Hands-on with **Flutter (Dart)** at production level
- Strong **Node.js** (preferred) or **Python/FastAPI** backend skills
- Experience with **Supabase/PostgreSQL** or similar

- Bonus: Exposure to LangGraph, LangChain, or AI agent frameworks
- Bonus: Familiarity with Render, Cloudflare, or similar deployment platforms

---

Role Expectations

Area	Ownership
Backend (60%)	API design, database architecture, classification workflows, integrations
Flutter (40%)	Production-ready features, performance optimization, clean architecture

What You'll Work On

- Backend Services - API development, data processing, system integrations
- AI Workflows - Document processing, intelligent classification systems
- Core Features - Task management, analytics, workflow automation
- Infrastructure - CI/CD, deployment automation, monitoring

Assessment Task

The Task: Build a "Smart Site Task Manager"

Build a task management system that:

1. Allows creating, viewing, and managing tasks
2. Automatically classifies and prioritizes tasks using intelligent logic
3. Persists data in Supabase/PostgreSQL
4. Provides a polished Flutter mobile experience

Part 1: Backend (Node.js/FastAPI + Supabase)

Database Schema (Supabase)

Design and create these tables:

```
-- Tasks table
tasks (
  id uuid PRIMARY KEY,
  title text NOT NULL,
  description text,
  category text,  -- scheduling, finance, technical, safety, general
  priority text,  -- high, medium, low
  status text,    -- pending, in_progress, completed
  assigned_to text,
  due_date timestamp,
  extracted_entities jsonb,
  suggested_actions jsonb,
  created_at timestamp,
  updated_at timestamp
)

-- Task history/audit log
task_history (
  id uuid PRIMARY KEY,
  task_id uuid REFERENCES tasks(id),
  action text,  -- created, updated, status_changed, completed
  old_value jsonb,
  new_value jsonb,
  changed_by text,
  changed_at timestamp
)
```

API Endpoints

Method	Endpoint	Description
`POST`	`/api/tasks`	Create a new task (with auto-classification)
`GET`	`/api/tasks`	List all tasks (with filters: status, category, priority)
`GET`	`/api/tasks/{id}`	Get task details with history
`PATCH`	`/api/tasks/{id}`	Update task
`DELETE`	`/api/tasks/{id}`	Delete task

Auto-Classification Logic

When a task is created, automatically determine:

Category (based on keywords):

- `scheduling` → meeting, schedule, call, appointment, deadline
- `finance` → payment, invoice, bill, budget, cost, expense
- `technical` → bug, fix, error, install, repair, maintain
- `safety` → safety, hazard, inspection, compliance, PPE
- `general` → default

Priority (based on urgency indicators):

- `high` → urgent, asap, immediately, today, critical, emergency
- `medium` → soon, this week, important
- `low` → default

Entity Extraction (parse from description):

- Dates/times mentioned
- Person names (after "with", "by", "assign to")
- Location references
- Action verbs

Suggested Actions (generate based on category):

```
{
  "scheduling": ["Block calendar", "Send invite", "Prepare agenda", "Set
reminder"],
  "finance": ["Check budget", "Get approval", "Generate invoice", "Update
records"],
  "technical": ["Diagnose issue", "Check resources", "Assign technician",
"Document fix"],
  "safety": ["Conduct inspection", "File report", "Notify supervisor",
"Update checklist"]
}
```

Requirements

- Use appropriate validation (Joi/Zod for Node.js, Pydantic for FastAPI)
- Implement proper error handling with meaningful error messages
- Add pagination for list endpoints (limit, offset)
- Include filtering and sorting options
- Write at least 3 unit tests for the classification logic
- Use environment variables for all secrets

Part 2: Flutter App

Build a mobile app with a single comprehensive screen:

Task Dashboard Screen

Top Section: Summary Cards

- Task counts by status (Pending, In Progress, Completed)
- Quick filters by category and priority
- Floating action button to create new task

Main Section: Task List

- List view of tasks with:
  - Title, category chip (color-coded), priority badge
  - Due date and assigned person
  - Status indicator
- Pull-to-refresh
- Search functionality
- Filter options (by category, priority, status)

Bottom Sheet: Create/Edit Task Form

- Form with:
  - Title (required)
  - Description (multiline, required)

    - Due date picker
    - Assigned to (text field)
- On submit: Show the auto-generated classification before saving
- Allow user to override category/priority if needed

Requirements

- Clean, production-quality UI following Material Design 3
- Proper loading states and skeleton loaders
- Error handling with user-friendly snackbars/dialogs
- Offline indicator (show when no network)
- Use Riverpod or Provider for state management
- Use Dio for API calls with proper interceptors
- Implement form validation

Part 3: Deployment & Documentation

Deploy Backend to Render

- Deploy your service to Render (free tier is fine)
- Provide the live API URL in README

README Must Include

1. Project Overview - What you built and why
2. Tech Stack - All technologies used
3. Setup Instructions - How to run locally (backend + Flutter)
4. API Documentation - All endpoints with request/response examples
5. Database Schema - ER diagram or table descriptions
6. Screenshots - Flutter app screens
7. Architecture Decisions - Why you chose certain approaches
8. What I'd Improve - Given more time, what would you add?

Bonus Features (Optional)

- Dark mode support in Flutter
- Task search with highlighting
- Export tasks to CSV
- Real-time updates using Supabase subscriptions
- Rate limiting on API
- API key authentication
- Swagger/OpenAPI documentation

Evaluation Criteria

Area	Weight	What We're Looking For
Backend Architecture	35%	Clean API design, proper models, error handling, tests
Database Design	15%	Normalized schema, proper indexes, relationships
Flutter UI/UX	25%	Polish, responsiveness, state management, offline handling
Classification Logic	15%	Thoughtful keyword mapping, entity extraction accuracy
Code Quality	10%	Clean structure, naming, comments, git history

---

Submission Requirements

Include

Item	Required
GitHub Repository	Yes - Clean README with setup instructions
Live Backend URL	Yes - Deployed on Render
Screenshots	Yes - Flutter app screens in README

Item	Required
API Documentation	Yes - In README with examples

Submission Checklist

- GitHub repo with comprehensive README
- Backend deployed on Render (live URL)
- Supabase database with schema
- Flutter app with dashboard screen
- API documentation with request/response examples
- At least 3 unit tests for backend
- Screenshots of Flutter app
- Meaningful git commit history (10+ commits)

Navicon Infraprojects