Full-Stack/Backend Take Home Assignment

Tech Stack Overview (Preferred)

- **Backend**: Django + Django Rest Framework (DRF) with PostgreSQL
- **Frontend**: React
- **Database**: PostgreSQL
- **Separation**: Frontend and backend code kept in separate directories or repositories
- **Features**:
    - Landing page with a data table
    - Filtering and bucketing (grouping) of data
    - CRUD APIs for all sections

Approach to be followed -

1. Backend (Django + DRF + PostgreSQL)

- **Purpose**: Handle business logic, serve RESTful APIs, and manage the PostgreSQL database.
- **Setup**:
    - Create a Django project and configure it to use PostgreSQL.
    - Use Django Rest Framework to build CRUD APIs.
    - Define a simple data model (e.g., Item) for the table.
- **APIs**:
    - Provide endpoints for Create, Read, Update, and Delete operations.
    - Enable filtering on fields like category or name.
- **Directory**: backend/

2. Frontend (React)

- **Purpose**: Consume APIs, render the UI, and handle client-side logic like filtering and bucketing.
- **Setup**:
    - Create a React app using create-react-app or vite.
    - Fetch data from Django APIs and display it in a table.
- **Features**:
    - Data table with filtering (e.g., by name) and bucketing (e.g., grouping by category).
    - UI components for CRUD operations (e.g., forms, buttons).
- **Directory**: frontend/

3. Database (PostgreSQL)

- **Purpose**: Store application data persistently.
- **Setup**: Configure PostgreSQL and connect it to Django.

4. Separation & Communication

- Keep frontend and backend in separate folders (e.g., myproject/backend/ and myproject/frontend/).
- Use a proxy in React during development to forward API requests to Django, avoiding CORS issues.

Features Explained

- **Data Table**: Displays items fetched from /api/items/, grouped by category (bucketing).
- **Filtering**: Backend supports filtering via query params (e.g., ?search=term), frontend updates based on user input.
- **Bucketing**: Frontend groups items by category into separate sections or tables.
- **CRUD APIs**:
  - GET /api/items/ (list with filtering)
  - POST /api/items/ (create)
  - GET /api/items/<id>/ (retrieve)
  - PUT /api/items/<id>/ (update)
  - DELETE /api/items/<id>/ (delete)

Project Structure

```
myproject/
  backend/
    myproject/
      settings.py
      urls.py
    myapp/
      models.py
      views.py
      serializers.py
    manage.py
  frontend/
    src/
      App.js
      LandingPage.js
    package.json
```

Notes

- **Development**: Run Django (python manage.py runserver) and React (npm start) simultaneously. The proxy handles API requests.
- **Extensions**: Add pagination, authentication, or styling (e.g., Tailwind CSS or Material-UI) as needed.
- **Assumption**: "Bucketing of section" interpreted as grouping data by a field (e.g., category) on the landing page.
- Use github to maintain code.
- Brownie points If you can deploy it.

Submission:

- Github repository link (make it mobile responsive)
- Live deployed link (test cases bonus pointers)
- Document link (explaining code structure components used & Tech stack library utilised)
- Additional Implementation fetches higher chances of conversion

Review the take home and feel free to ask any questions you may have (soumaya@resollect.com)