

5-Day Backend Demo Task (Django + DRF)

Scope: Backend only (no email, no frontend). All flows are API-based. Invitation/Reset should return **tokens in JSON**.

What we're evaluating

- Django project structure (**multi-app**)
 - DRF API design
 - **Role-based access control (RBAC)**
 - **Multi-tenant scoping**
 - Correct model relations + constraints
 - Basic security hardening (tokens, throttling, no leakage, safe responses)
-

1) Requirements Summary

Roles

- **Admin** (tenant-level)
- **Manager** (tenant-level)
- **Staff** (company-level only)
- **Customer** (tenant-level, read-only access to own products)

Tenant rules

- All users (except superadmin bootstrap) belong to one **Tenant**
- A Tenant has many **Companies**
- **Staff must belong to exactly one Company** and must not access tenant endpoints
- Admin can edit tenant fields **except tenant.name is read-only**
- Manager can view tenant and create/edit companies
- Admin can also create/edit companies

Invitation rules (API only)

- System starts with a **platform superadmin** (Django creates this via `createsuperuser`)

- Superadmin creates an invitation for a new **Tenant Admin**
- Accepting that invitation:
 - Requires `tenant_name`
 - Creates Tenant
 - Creates Admin user under that tenant
- Admin can create invitations for **Managers**
- Accept invitation returns JWT tokens

Staff creation (no invitation)

- Admin or Manager creates Staff directly with password
- Staff logs in using credentials

Products / sharing / customer signup

- Staff can create/update products
 - Product has a **share_token** (unguessable) generated on create
 - Customer uses share_token to sign up and get access to that product
 - Each Product belongs to exactly one Customer (nullable until claimed)
 - Customer can only **view** their own products
 - Admin/manager can view products and customer list/info (read-only)
-

2) Project Structure (Multi-App Required)

Create a Django project with **separate apps** at minimum:

- `accounts` (custom user, auth, RBAC helpers)
 - `tenants` (Tenant, Company)
 - `invitations` (Invitation + accept flow)
 - `products` (Product + public claim)
 - `core` (shared utils: token generation, permissions mixins, throttles)
-

3) Model Schema (Candidate must implement)

accounts app

User (custom user model)

Fields:

- `id` UUID (recommended)
- `email` (unique, used as USERNAME_FIELD)
- `role` (choices: ADMIN, MANAGER, STAFF, CUSTOMER)
- `tenant` FK → `Tenant` (nullable only for platform superadmin)
- `company` FK → `Company` (nullable; **required for STAFF**, null for others)
- `is_active`, `is_staff`, `is_superuser`
- timestamps

Constraints (must enforce):

- If `role == STAFF` → `company IS NOT NULL`
- If `role in {ADMIN, MANAGER, CUSTOMER}` → `company IS NULL`
- If `role != PLATFORM_SUPERADMIN` → `tenant IS NOT NULL`
- Tenant scoping must be applied to all queries in viewsets

tenants app

Tenant

- `id` UUID
- `name` (string)
- timestamps

Company

- `id` UUID
- `tenant` FK → `Tenant`

- `name`
 - `timestamps`
- Constraint:** `company.tenant` must match `user.tenant` for staff assignment

invitations app

Invitation

- `id` UUID
- `email`
- `role` (ADMIN or MANAGER only)
- `tenant` FK nullable (null for bootstrap admin invitation, set for manager invites)
- `token` (random, unique, long)
- `expires_at`
- `used_at` nullable
- `timestamps`

Rules:

- Single-use: cannot accept if `used_at` set
- Must reject if expired

products app

Product

- `id` UUID
- `tenant` FK → Tenant
- `company` FK → Company
- `created_by` FK → User(STAFF)
- `customer` FK → User(CUSTOMER) nullable until claimed
- `name` , `description` (or minimal fields)
- `share_token` (unique random long string)
- `timestamps`

Constraints:

- `created_by.role == STAFF`
- `product.tenant == created_by.tenant`

- `product.company == created_by.company`
 - When customer claims: `customer.tenant` must match `product.tenant`
-

4) Auth & Security Requirements (Must Have)

Authentication

Use **JWT** (SimpleJWT is fine):

- Access + Refresh tokens

Throttling

Enable DRF throttles at least for:

- login
- password reset request

Token handling

- `Invitation.token` must be:
 - random, unguessable (e.g., `secrets.token_urlsafe(32+)`)
 - stored in DB
 - single-use + expiring
- `Product.share_token` also random/unguessable

Safe responses

- Password reset request must not reveal whether an email exists
- All endpoints must enforce tenant scoping (no ID guessing access)

Basic hardening (expected)

- strong password validation (Django validators)
 - permissions checked server-side (no relying on client role)
 - deny-by-default approach in permissions
-

5) Required Endpoints (Backend Only)

Auth (accounts app)

1. `POST /api/auth/login/`

- body: { "email": "", "password": "" }
- response: { "access": "", "refresh": "" }

2. `POST /api/auth/token/refresh/`

- body: { "refresh": "" }
- response: { "access": "" }

3. `POST /api/auth/password-reset/request/`

- body: { "email": "" }
- response: always { "detail": "If the email exists, a reset token has been generated." , "reset_token": "<token-for-demo>" }
 - For demo purposes, you may return reset_token **only if user exists** OR always return a dummy token – but must not leak existence via different messages/status codes.

4. `POST /api/auth/password-reset/confirm/`

- body: { "reset_token": "", "new_password": "" }
- response: { "detail": "Password updated" }

Invitations (invitations app)

5. `POST /api/invitations/bootstrap-admin/` (platform superadmin only)

- body: { "email": "", "expires_in_hours": 48 }
- response: { "invitation_token": "" }

6. `POST /api/invitations/` (Admin only)

- creates Manager invite under admin's tenant
- body: { "email": "", "role": "MANAGER", "expires_in_hours": 48 }
- response: { "invitation_token": "" }

7. `POST /api/invitations/accept/`

- body for bootstrap admin: `{ "token": "", "tenant_name": "", "password": "" }`
- body for manager: `{ "token": "", "password": "" }`
- response: `{ "detail": "Accepted", "access": "", "refresh": "" }`

Tenant / Companies (tenants app)

8. `GET /api/tenant/me/` (Admin/Manager)

- response: tenant info

9. `PATCH /api/tenant/me/` (Admin only)

- must reject updates to `name` (read-only)
- response: updated tenant fields

10. `GET /api/companies/` (Admin/Manager)

11. `POST /api/companies/` (Admin/Manager)

12. `PATCH /api/companies/{id}/` (Admin/Manager)

- all scoped to tenant

Staff Management (accounts or tenants app)

13. `POST /api/staff/` (Admin/Manager)

- body: `{ "email": "", "password": "", "company_id": "" }`
- response: created staff

14. `GET /api/staff/` (Admin/Manager)

Products (products app)

15. `POST /api/products/` (Staff only)

- body: `{ "name": "", "description": "" }`
- response includes: `{ "id": "", "share_token": "" }`

16. `PATCH /api/products/{id}/` (Staff only, only within own company)

17. `GET /api/products/`

- Staff: list products in own company (or those created by them)
- Admin: list all tenant products (read-only)
- Customer: list only their products

18. `GET /api/products/{id}/`

- Admin read-only tenant-scoped
- Staff read-only if in same company
- Customer only if owns it

Customer claim via share token (products app)

19. `POST /api/public/products/claim/`

- body: { "share_token": "", "email": "", "password": "" }
- creates customer user (if not exists) under product tenant
- assigns `product.customer = customer`
- response: { "detail": "Claimed", "access": "", "refresh": "" }

Optional: If customer already exists, just authenticate and claim if allowed.

Admin customer view (accounts/products app)

20. `GET /api/admin/customers/` (Admin only, read-only)

- list customers in tenant + minimal info

21. `GET /api/admin/customers/{id}/products/` (Admin only, read-only)

6) Permission Matrix (Must Match)

- Superadmin: can create bootstrap admin invites only
 - Admin: manage companies, invite managers, create staff, view customers/products, edit tenant except name
 - Manager: manage companies, create staff, view tenant
 - Staff: create/update products, no tenant endpoints, no company management
 - Customer: view own products only
-

7) What to Submit (Candidate Deliverables)

1. **Git repository**

2. **Postman Collection + Environment** (required)

- Include variables: `base_url` , `access` , `refresh` , `bootstrap_invite_token` , etc.

3. **Short video demo (5–10 min)** showing:

- Bootstrapping a tenant admin via invitation token
- Admin inviting manager (token returned), manager accepting
- Admin/manager creating staff, staff login
- Staff creating product → `share_token` generated
- Customer claiming product via `share_token` and seeing product list
- Admin listing customers and products

4. Evidence that:

- Permissions are enforced (show at least 2 forbidden attempts)
- Tenant isolation works (cannot access other tenant resources)

If you want, I can also provide a **Postman flow outline** (folder structure + exact requests ordering + environment vars) so you can hand it to candidates as the expected test script.