A continuación tienes una visión clara (y sin rodeos) de cómo está pensado el flujo en este "mini-Jira/Trello" durante el primer sprint, basándome en los modelos y viewsets que existen en el repositorio:

1. Autenticación
   • Todo parte del usuario autenticado (`settings.AUTH_USER_MODEL`).
   • Cada endpoint relevante (`OrganizationViewSet` , WorkspaceViewSet, etc.) exige `IsAuthenticated` , de modo que siempre sabemos quién es el "actor" de la operación.

2. Organización (Organization) → nivel 1
   • Es la entidad raíz.
   • Campos clave: `id (UUID)` , `name` , `slug` , `owner` .
   • El creador (owner) se guarda y sirve como "super-admin" inicial.
   • Relaciones descendentes:
   - `workspaces` (related_name en Workspace)
   - `memberships` (OrganizationMembership)
   • Métricas útiles ya incluidas: member_count, workspace_count.

   Flujo típico:
   a. Usuario autenticado llama a `POST /organizations/` con `name` , `slug` , etc.
   b. Se crea la organización y se asocia automáticamente al usuario como `owner` .

3. Espacio de trabajo (Workspace) → nivel 2
   • Pertenece a UNA organización (`organization = ForeignKey('organizations.Organization')`).
   • Campos extra: `workspace_type` , `visibility` , `cover_image` , etc.
   • El creador (`created_by`) se añade con perform_create en el WorkspaceViewSet.
   • Restricción de unicidad: `unique_together = ['organization', 'slug']` .
   • Los queries del WorkspaceViewSet.get_queryset() filtran por membresías activas (`members__user=self.request.user`).

   ¿Por qué no "pides" organization_id explícito en el endpoint?
   – Lo envías dentro del JSON del POST (p.ej. `"organization": "<uuid>"`).
   – Si el front ya conoce la organización actual (lo normal en un UI tipo "Jira"), la incluye automáticamente.
   – No se "hereda" implícitamente desde el usuario: se valida con la FK que la organización exista y que el usuario sea miembro.

4. Proyecto (Project) → nivel 3
   • FK directa al Workspace (`workspace = ForeignKey('workspaces.Workspace')`).
   • Restricción: `unique_together = ['workspace', 'key']` (similar a "ABC-123" en Jira).

- Se guarda `created_by` vía el `ProjectViewSet` (mismo patrón).
- Para crear un proyecto, el JSON incluye el `workspace` UUID; de ahí se infiere la organización. No hace falta pasar `organization_id` porque la relación Workspace→Organization ya existe.

5. Membresías & permisos
   - `OrganizationMembership` y `WorkspaceMember` controlan quién ve qué.
   - Los get_queryset() de cada viewset siempre filtran por la tabla de membresías activa, así que un usuario solo ve lo que le toca.
   - No hay un sistema granular de roles todavía (eso seguramente venga en sprints posteriores).

6. Sprint-1 Scope (lo implementado)
   ✓ CRUD completo para Organization, Workspace y Project.
   ✓ Upload de imágenes/archivos (logo de organización, cover de workspace, attachments de proyecto) con helpers a S3.
   ✓ Endpoints de invitaciones y membresías básicas.
   ✘ Aún no hay issues, boards, ni sprints—eso vendrá después.

7. Orden natural en la vida real
   1. Crea tu Organization.
   2. Dentro de ella, crea uno o varios Workspaces (podrías usarlos como "departamentos" o "áreas de negocio").
   3. Dentro de cada Workspace, crea Projects (equivalentes a proyectos Jira).
   4. Luego vendrán Issues, Boards, Sprints, etc.

8. Cosas a mejorar en futuros sprints
   - Validar en los serializers que el usuario realmente pertenezca a la organización cuando pasa el `organization` o `workspace` UUID.
   - Roles y permisos más finos (owner, admin, member).
   - Hooks para que, al crear un Workspace, auto-añada al creador como `WorkspaceMember`.
   - Tests de integración para los `unique_together` y filtros de membresía.

En resumen: la jerarquía es Organization → Workspace → Project. Las claves foráneas están definidas en los modelos, por eso no ves `organization_id` directamente en Project; viene "por transitividad" desde Workspace. Todo el flujo se basa en que el front end pase el UUID correcto del nivel superior inmediato y en que la API valide la pertenencia del usuario.