Django Project Collaboration Portal

Student Name: Nikhil Kumar
Roll No: —240410700033
Year & Section: —2nd Year
Project Title: Project Collaboration Portal (Collab Apps)
Project Type: Application Developer
Stack / Framework: Django, Django ORM, Django Templates, django-guardian (permissions), Tailwind/Bootstrap

---

1. Problem Understanding

1.1 Problem Statement (in your own words)

Teams working on projects require a centralized space to manage tasks, share files, collaborate via comments, and track progress. Communication becomes fragmented when discussions, documents, and tasks are scattered across tools. Small organizations or academic groups often need a lightweight, customizable collaboration tool without the complexity or cost of enterprise portals.

This project solves this by building a **web-based project collaboration portal** enabling teams to work together through tasks, comments, and shared files — all tied to specific projects with role-based permissions.

1.2 Why this problem exists or matters?

For students & teams

- They need one unified workspace for assignments, hackathons, and team projects.

For organizations

- SaaS collaboration tools can be expensive or too heavy for smaller teams.
- A custom solution allows control, extensibility, and privacy.

For developers

- Provides hands-on experience with many-to-many relationships, user permissions, uploads, and real-world collab features.

This system benefits **students, small teams, academic institutions, startups**, and **workgroup managers**.

1.3 Key Inputs and Expected Outputs

Inputs	Process	Expected Outputs
Project creation (name, description, members)	Save project → attach members (many-to-many)	Project page with team list
Task creation (title, due date, assignees)	Validate → save → link to project	Task list under project

Comment (text, mentions, attachments)	Validate → sanitize → store	Threaded or chronological comments
File upload (PDF, images, docs)	Validate → secure store in MEDIA folder	Downloadable project/task files
User permissions	django-guardian → object-level perms	Role-based access to projects, tasks
@Mentions (stretch)	Parse text → notify mentioned users	Alerts/notifications

---

2. Functional Scope

2.1 Core Features (must-haves)

A. Project Management

- Create/edit/delete projects
- Add/remove team members
- Many-to-many relationship: *Project* ↔ *Users*

B. Task Management

- Create/edit/delete tasks
- Link tasks to specific projects
- One-to-many: *Project* → *Tasks*

- Many-to-many: *Task* ↔ *Assignees*

C. Comments & Conversations

- Comment on projects and tasks
- Text-only + timestamps
- Correct task/comment linkage (evaluation hook)

D. File Uploads

- Upload files under projects/tasks (PDF, DOCX, images)
- Display file metadata (owner, uploaded at)
- Restrict max file size (e.g., 5MB)

E. Permissions & Roles

- Use django-guardian for object-level permissions
- Roles:
    - **Owner** (full control)
    - **Editor** (edit tasks/comments/files)
    - **Viewer** (read-only)

2.2 Stretch Goals (if time permits)

@Mentions & Notifications

- Detect "@username" in comments
- Notify users via UI alerts / email
- Mention suggestions via autocomplete

Activity Feed

- "X added a task", "Y uploaded a file"

Real-time updates

- Using Django Channels / WebSockets

2.3 Libraries & Tools

- **Django** (views, forms, models)
- **django-guardian** (permissions)
- **Django Messages** (feedback UI)
- **Bootstrap/Tailwind** (UI)
- **SQLite/PostgreSQL**
- **pytest / Django TestCase**

3. System & Design Thinking

3.1 System Flow / Pipeline

User Login

↓

Dashboard (Projects List)

↓

Project View

- Team Members
- Tasks
- Files
- Comments

↓

Task View

- Assignees
- Status
- Comments

↓

Comment System

- Optional @Mentions

↓

File Upload Engine

↓

Permissions Check (django-guardian)

↓

Response Rendered (HTML Templates)

Upload Pipeline

File Input → File Validation → Save to MEDIA → DB record → Accessible in UI

Permissions Pipeline

Action → Check object permissions → Allow / Deny with message

---

3.2 Data Structures & Algorithms

Models Required

- **Project** (M2M with User)
- **Task** (FK to Project, M2M with User)
- **Comment** (FK to Project or Task, FK to User)
- **File** (FK to Project or Task, FileField)

Relationships

- Project ↔ Users (Many-to-Many)
- Task ↔ Assignees (Many-to-Many)
- Project → Tasks (One-to-Many)
- Tasks/Projects → Comments (One-to-Many)

Algorithms

- @Mention parsing:
    - Regex to detect *@username*
    - Map to user IDs

    - Trigger notifications

- Permission checks: django-guardian per-object lookup
- File validation:
    - Type check
    - Size check
    - Sanitization

3.3 Testing Strategy

Unit Tests (min 6)

- Create project with members
- Task creation & assignment
- Comment linking correctness
- File upload validation
- Permission enforcement
- Edge cases: empty input, oversized file

Integration Tests

- Create→assign→comment→upload pipeline

- Permissions: user with no access attempting edits

Manual UI Testing

- Error messages
- File upload workflows
- Member management flows

---

4. Timeline & Milestones (4 Weeks)

Week	Planned Deliverables	Check
W1	Models (Projects, Tasks, Comments, Files), schema, initial M2M design	☐
W2	Views (CRUD), permissions, dashboards, task/project flows	☐
W3	File uploads, comments, @mentions (optional), UI polish	☐
W4	README, 6+ tests, deployment, demo video	☐

---

5. Risks & Dependencies

5.1 Hardest technical parts

- Many-to-many modeling for teams & assignees
- Correct linking of comments → tasks/projects

- Permissions (django-guardian) integration
- File upload validation

5.2 Dependencies (Mentor Support Needed)

- Review model design to avoid relationship mistakes
- Confirm permission model (roles + actions)
- Decide upload limits and allowed formats
- Help debug deployment if needed

---

6. Evaluation Readiness

6.1 How to prove the project works

- Walkthrough video (2–3 min)
- GitHub repo with:
    - Setup/run instructions
    - Screenshots
    - Tests
    - Architecture notes

- Functional demo:
    - Create project
    - Add members
    - Create tasks
    - Comment
    - Upload files

6.2 Success Metrics

- 100% CRUD for projects & tasks
- Correct task/comment linkage (rubric requirement)
- All tests passing
- File uploads safe + validated
- Clean simple UI
- Permissions enforceable and correct

---

7. Responsibilities

Task	Student	Mentor Notes
Models & schema	Nikhil Kumar	☐

Views & permissions	Nikhil Kumar	☐
File uploads + comments	Nikhil Kumar	☐
@Mentions + notifications (optional)	Nikhil Kumar	☐
Deployment + README + tests	Nikhil Kumar	☐

Signatures (Students):

Mentor Approval:

Date: