AgroInsight

Executive Summary

Project: Project Upload Presets — build backend support for storing per-project upload presets. Objectives: add DB migration (public schema) creating table project_upload_presets with UUID primary key using uuid_generate_v4(), project_id FK, preset JSON, created_at/updated_at timestamps; expose Remix GET (member read) and PUT (admin write) handlers with auth, payload validation, and normalization hooks; provide architecture SVG. Expected outcomes: secure endpoints respecting roles, validated/normalized payload, DB persistence via Prisma, and architecture diagram.

Core Functionalities

- **Project Upload Presets:** Manage per-project file upload presets (allowed types, size limits, storage options) with create/read/update APIs and validation hooks. (Priority: **High**)
- **Authentication & Authorization:** Role-based access control with member read and admin write permissions, integrated with Remix handlers. (Priority: **High**)
- **Schema Migrations & DB Integrity:** Robust Prisma migrations using public schema and uuid_generate_v4() for IDs, ensuring data integrity and constraints. (Priority: **High**)
- **Payload Validation & Normalization:** Request payload validation and normalization hooks to enforce consistent data before DB persistence. (Priority: **Medium**)

- **Architecture Diagrams & Documentation:** Automatically generated SVG architecture diagrams and documentation for onboarding and audits. (Priority: **Low**)

Personas

- **Project Admin** – Manages projects, upload presets, and user access within the platform.

  Goals: Configure upload presets, Manage user permissions

  Pain Points: Complex UI for preset configuration, Lack of clear audit logs

  Key Tasks: Create/modify presets, Assign roles to users

- **Project Member** – Uses the project features and consumes upload presets set by admins.

  Goals: Upload files using presets, Ensure uploads match project requirements

  Pain Points: Confusing preset options, Unexpected validation errors on upload

  Key Tasks: Select preset and upload files, View preset details

- **Developer/Integrator** – Integrates backend endpoints and maintains schemas and migrations.

  Goals: Implement reliable endpoints, Maintain DB migrations

  Pain Points: Ambiguous schema requirements, Inconsistent UUID generation across environments

  Key Tasks: Write migrations, Implement handlers and validation

- **End User/Client** – Consumes uploaded assets or uses front-end features tied to project uploads.

  Goals: Access uploaded assets quickly, Ensure assets meet format requirements

Pain Points: Slow asset availability, Mismatch in expected file formats

Key Tasks: Download/view assets, Report issues to project members

Stakeholders

- **Project Management Team:** Oversees timeline, deliverables, and resource allocation.
- **Development Team:** Frontend, Backend, and DevOps engineers building features and integrations.
- **QA Team:** Tests functionality, performance, and security to ensure quality.
- **Design Team:** UX/UI designers creating interfaces and user flows.
- **Product Owner:** Defines requirements, prioritizes backlog, and aligns stakeholders.
- **Security/Compliance:** Ensures data protection, access controls, and regulatory compliance.
- **End Users:** Members and admins using the system; provide feedback and requirements.

Tech Stack

- **Frontend:** React, Remix
- **Backend:** Node.js, Node.js seed script
- **Frontend/Backend:** TypeScript
- **Database:** Prisma, uuid-ossp, Prisma Migrate
- **Database Management:** PostgreSQL
- **User Authentication:** Clerk

- **Validation:** Zod
- **Containerization:** Docker
- **Automation:** GitHub Actions
- **Storage:** AWS S3
- **Data Storage:** Redis

Project Timeline

Tasks are categorized by complexity to guide time estimations: XS, S, M, L, XL, XXL.

Roles:

- **Frontend Developer** (FE)
- **Backend Developer** (BE)
- **QA Engineer** (QA)
- **DevOps Engineer** (DevOps)

Milestone 1: Setup: repository, environment, DB init (UUID extension), CI; prepare schema.prisma and migration folder

Estimated 226.5 hours

- **Configure Environment:** As a: DevOps engineer, I want to: configure environment variables, local development environment, and resource constraints, So that: developers have a consistent and reproducible setup across projects**(6 hours)** - Environment variables are defined for all services with default values Local dev environment can be started with

a single command Environment config is stored in version control and documented Invalid or missing env vars are flagged during startup

  - INFRA: Add .env.example with all env vars in `apps/api/.env.example` and `apps/web/.env.example` - (M) (1 hours)[FE][BE]
  - INFRA: Create Docker Compose dev command in `docker-compose.yml` to start api and web - (M) (1 hours)[FE][BE]
  - DEV: Implement env validation in `apps/api/main.py` to flag missing/invalid vars - (M) (1 hours)[FE][BE][QA]
  - DEV: Load env defaults in `apps/web/vite.config.ts` and `apps/web/src/config/env.ts - (M) (1 hours)[FE][BE][DevOps]
  - QUALITY: Add startup env check GitHub Action in `.github/workflows/check-env.yml` - (M) (1 hours)[FE][BE]
  - DOCUMENTATION: Add `docs/ENVIRONMENT.md` documenting variables and single-command startup (e.g., `make dev`) - (M) (1 hours)[FE][BE][DevOps]

- **Setup Repository:** As a: DevOps engineer, I want to: initialize and configure the repository with standard branching, hooks, and templates, So that: development workflow is standardized and scalable**(3.5 hours)** - Repository initialization completes with default branch main CI hooks and pre-commit hooks are installed README templates and contribution guidelines are present Access controls and basic permissions are configured for the team

  - Initialize a Git repository with default branch main and create a README.md at repository root to document project purpose and setup. - (XS) (0.5 hours)[FE][BE][DevOps]
  - Create package.json at repository root; define scripts for hooks (lint, test) and CI (build, test) to enable automation. - (S) (0.5 hours)[FE][BE][QA]

    - Infra: Add CI workflow at .github/workflows/ci.yml referencing tests in apps/api, ensuring PRs trigger test execution for API app. - (M) (1 hours)[FE][BE][QA]
    - Install Husky and add pre-commit hook in .husky/pre-commit and configure in package.json to enforce linting on commit. - (S) (0.5 hours)[FE][BE][DevOps]
    - Docs: Add CONTRIBUTING.md and CODEOWNERS at repository root to document contribution guidelines and ownership. - (XS) (0.5 hours)[FE][BE]
    - Ops: Add permissions script scripts/configure_permissions.sh to set GitHub team access and branch protection. - (M) (1 hours)[FE][BE][DevOps]
- **Setup CI Pipeline:** As a: DevOps engineer, I want to: set up continuous integration pipeline with build, test, and artifact publishing, So that: code changes are automatically validated and deployable**(7 hours)** - Pipeline runs on new PRs and pushes Build and test stages complete within defined timeouts Artifacts are published to a registry or artifact store Failure notifications are configured for the team
    - INFRA: Create CI workflow in `.github/workflows/ci.yml` with PR and push triggers and timeouts - (M) (1 hours)[FE][BE]
    - DEV: Add build script `scripts/build.sh` to build project and produce artifact` - (M) (1 hours)[FE][BE]
    - DEV: Add test script `scripts/test.sh` to run unit and integration tests in `apps/api/tests` with timeouts - (M) (1 hours)[FE][BE][QA]
    - INFRA: Create Dockerfile at `Dockerfile` for CI builds - (M) (1 hours)[FE][BE]
    - INFRA: Add publish script `deploy/publish.sh` to upload artifacts to S3/AWS registry - (M) (1 hours)[FE][BE][DevOps]

    - INFRA: Configure failure notifications in `.github/workflows/ci.yml` calling `scripts/notify_failure.sh` - (M) (1 hours)[FE][BE][DevOps]
    - QUALITY: Add README status badge in `README.md` pointing to workflow - (M) (1 hours)[FE][BE]
- **Configure Dev Tooling:** As a: Developer, I want to: configure local tooling and IDE guidelines (linters, formatters, and templates), So that: code quality is maintained and onboarding is efficient**(5.5 hours)** - Linter and formatter configurations are applied IDE templates and starter files are provided Tooling checks run locally and in CI Documentation for local setup is available
    - Configure ESLint: add .eslintrc.cjs in apps/api and apps/web with base rules, extend recommended configurations, parser options for modern JS/TS, and overrides for test dirs. - (M) (1 hours)[FE][BE][DevOps][QA]
    - Add Prettier config in apps/api/prettier.config.cjs and apps/web/prettier.config.cjs aligning with ESLint rules for formatting consistency. - (S) (0.5 hours)[FE][BE][DevOps]
    - Add EditorConfig in apps/api/.editorconfig and apps/web/.editorconfig ensuring consistent indentation and line endings across projects. - (XS) (0.5 hours)[FE][BE][DevOps]
    - Configure Husky pre-commit hooks in apps/api/.husky/pre-commit and update package.json scripts to run lint and format checks before commits. - (M) (1 hours)[FE][BE][DevOps]
    - Create GitHub Actions CI workflow in .github/workflows/ci.yml to run lint & format checks. - (M) (1 hours)[FE][BE]
    - Provide VSCode workspace and settings in .vscode/settings.json and .vscode/extensions.json - (S) (0.5 hours)[FE][BE]
    - Add starter IDE templates and README starter files in apps/web/src/templates/ and apps/api/src/templates/ - (M) (1 hours)[FE][BE]

    - Add dev setup documentation in docs/DEV_SETUP.md describing local checks and IDE setup - (XS) (0.5 hours) [FE][BE][DevOps]

- **Apply Instructions:** As a: migration engineer, I want to: apply the provided migration instructions to the repository, So that: the codebase reflects the latest recommended changes.**(7 hours)** - Instructions are applied without errors to the codebase No conflicting changes are introduced during apply Repository builds successfully after application All relevant files updated matching the README changes Audit log shows applied changes with timestamp

    - INFRA: Create backup script in scripts/backup/create_backup.sh. Implement a shell script to snapshot relevant data/files, with logging, error handling, and idempotent execution. Ensure script is executable and I/O redirection works in CI. - (S) (0.5 hours)[FE][BE]

    - DB: Add migration_README entry in apps/api/db/migrations/README_migration.sql. Create a placeholder entry documenting README migration steps for the API migrations table. - (XS) (0.5 hours)[FE][BE]

    - API: Update migration README model in apps/api/models/migration_readme.py. Extend model to include fields describing README migration steps and status; ensure serialization matches API models. - (S) (0.5 hours)[FE][BE]

    - API: Implement apply_instructions() in apps/api/services/migration/ApplyService.py. Core logic to apply instruction set from README, coordinate with migration locks, and return success/failure with logs. - (M) (1 hours)[FE][BE]

    - INFRA: Update CI pipeline in .github/workflows/ci.yml to run README apply step. Integrate a new job/step to invoke apply_instructions flow within CI. - (M) (1 hours)[FE][BE]

    - SYS: Add migration_lock handling in apps/api/utils/migration_lock.py. Implement acquire/release semantics, timeouts, and deadlock avoidance. - (M) (1 hours)[FE][BE]

    - DOC: Update README docs in docs/migration/README.md and README.md root. Document README migration process and usage. - (S) (0.5 hours)[FE][BE]
    - QA: Add integration test in apps/api/tests/test_apply_instructions.py. Validate end-to-end apply_instructions flow with mocks for dependencies. - (L) (2 hours)[FE][BE][QA]
- **Seed Template:** As a: administrator, I want to: create a seed template that populates initial data for migration verification, So that: we can quickly verify migration outcomes in a sandbox environment.**(5.5 hours)** - Seed template includes representative data set Template can be executed without errors Generated data aligns with schema constraints Documentation includes how-to run the seed and verify results No sensitive data hard-coded
    - DB: Create seed file prisma/seed/seed_template.ts with representative data for table_project_upload_presets and table_field_mappings, ensuring type-safe Prisma client usage and non-production data patterns suitable for MVP seed. - (S) (0.5 hours)[FE][BE]
    - DB: Add seed runner in prisma/seed/index.ts to execute prisma/seed/seed_template.ts deterministically without exposing sensitive data, wired to Prisma CLI seed mechanism and guard rails for environments. - (S) (0.5 hours)[FE][BE][DevOps]
    - API: Expose seed verification endpoint in apps/api/routes/seed.ts to validate seeded data against schema, using existing Prisma Client and input validation, returning deterministic pass/fail results. - (M) (1 hours)[FE][BE][QA]
    - Docs: Add HOWTO in docs/migration/SEED_README.md with run steps and verification queries for table_project_upload_presets and table_migration_readme, including sample queries and expected results against Prisma schema. - (XS) (0.5 hours)[FE][BE]

    - Tests: Create integration test tests/seed/test_seed.ts to run prisma/seed/index.ts and assert schema constraints, including constraints for table_project_upload_presets and table_migration_readme. - (M) (1 hours)[FE][BE][QA]
    - CI: Add GitHub Actions job .github/workflows/seed.yml to run seed and tests in Docker, ensuring deterministic MVP seed environment across CI runs. - (L) (2 hours)[FE][BE][DevOps][QA]
- **Apply Migration:** As a: system administrator, I want to: apply the migrations documented in the README to the target environment, So that: the repository and database are in sync with the latest schema and data conventions**(6 hours)** - Migration script executes without errors on target environment Database schema version updates to latest after migration All sample data remains intact and matches schema changes Migration logs are generated and stored securely System reports migration success to monitoring service
    - DB: Create migration in prisma/migrations/20251030_apply_migration/ preserving Prisma Migrate conventions and ensuring migration contains changes to schema_version table and table_migration_readme references. - (S) (0.5 hours)[FE][BE]
    - Infra: Add backup script scripts/db_backup.sh to back up database before migrations using pg_dump (or equivalent) and store to backups/, with logging. - (S) (0.5 hours)[FE][BE]
    - API: Implement runMigration() in apps/api/services/migration/MigrationService.ts to trigger Prisma migrate in API service layer or orchestrate migration execution path. - (M) (1 hours)[FE][BE]
    - IO: Write migration logs to logs/migrations/migration.log to capture start, progress, and result including timestamps. - (XS) (0.5 hours)[FE][BE]

    - DB: Update schema version in apps/api/models/migration/READMEModel.ts and table table_migration_readme to reflect latest migration version. - (S) (0.5 hours)[FE][BE][QA]
    - API: Implement notifyMonitoring() in apps/api/services/monitoring/MonitoringService.ts to emit migration health/status events to monitoring systems. - (M) (1 hours)[FE][BE]
    - CI: Add workflow .github/workflows/migration.yml to run migration and tests - (M) (1 hours)[FE][BE][QA]
    - Test: Create integration test tests/integration/test_migration.py - (M) (1 hours)[FE][BE][QA]
- **UUID Extension Note:** As a: developer, I want to: extend UUID handling documentation and add notes for UUID generation constraints, So that: future migrations and integrations respect UUID rules.**(4 hours)** - Documentation updated to include UUID extension notes Examples provided for new UUID formats No breaking changes to current UUID usage Docs render without errors in preview
    - Docs: Add UUID extension note to docs/migration/README.md preserving architectural references and providing clear guidance on extended UUID formats. - (S) (0.5 hours)[FE][BE]
    - Docs: Create examples in docs/migration/uuid_extensions.md with new UUID formats demonstrating extended parsing and validation rules. - (S) (0.5 hours)[FE][BE][QA]
    - API: Update parsing in apps/api/services/uuid/UuidService.ts to accept extended UUID formats, ensuring backward compatibility where possible. - (M) (1 hours)[FE][BE]
    - DB: Add comment and note in prisma/migrations/ migration SQL to describe UUID extension for table_project_upload_presets and table_migration_readme. - (XS) (0.5 hours)[FE][BE]

    - CI: Add docs preview job in .github/workflows/docs-preview.yml to render docs before PRs, ensuring extended UUID formats render correctly. - (M) (1 hours)[FE][BE]
    - Docs: Update migration_README row in DB via seed script prisma/seed/migration_readme_seed.ts to reflect UUID extension changes in the database seed. - (S) (0.5 hours) [FE][BE]
- **Migration Notes:** As a: project maintainer, I want to: document migration notes and best practices in README, So that: future migrations are faster and less error-prone**(5 hours)** - Notes include prerequisites and rollback steps Examples of common issues and resolutions Versioned notes aligned with migration scripts Notes are accessible from main README and kept up-to-date Documentation build passes without errors
    - Documentation: Draft Migration Notes in docs/migration/README.md detailing prerequisites, rollback procedures, usage examples, and a version table that references migration scripts in prisma/migrations/ and aligns with project MVP docs workflow. - (S) (0.5 hours)[FE][BE]
    - Database: Create a migration_README record in apps/api/db/migrations/migration_README.sql and update apps/api/db/schema/migration_README to reflect versions in table_migration_readme; ensure cross-reference integrity with prisma/migrations and docs migration notes. - (M) (1 hours)[FE][BE]
    - LOCK: Implement migration_lock handling example and rollback in scripts/migration_lock/README.md and update apps/api/config/migration_lock to reflect table_migration_lock; ensure proper locking semantics and idempotent rollback guidance. - (L) (2 hours)[FE][BE][DevOps]
    - LINK: Add a link to migration notes from the main README and ensure navigation in docs/README.md; adjust internal docs navigation tree to include docs/migration. - (XS) (0.5 hours)[FE][BE]

    - CI: Add docs build check to GitHub Actions in .github/workflows/docs.yml to validate docs/migration/README.md builds without errors - (S) (0.5 hours)[FE][BE]
    - EXAMPLES: Add common issues and resolutions section in docs/migration/FAQ.md with examples referencing prisma/migrations and apps/api/services/auth/AuthService.ts where auth issues appear (cite table_users) - (S) (0.5 hours)[FE][BE]
- **Add migration-lock checksum file:** As a: DevOps engineer, I want to: add a migration-lock checksum file to the repository, So that: I can verify migrations integrity across environments and prevent drift**(5 hours)** - File exists at configured path with correct filename. Checksum file contains a valid hash for the current migration-lock state. System stores the checksum with proper permissions and is readable by the deployment process.
    - DB: Add migration_locks table migration in prisma/migrations/ to track locks with fields (id, migrationHash, lockedBy, expiresAt, createdAt) and ensure Prisma schema migration reflects table in migrations folder - (S) (0.5 hours)[FE][BE]
    - API: Implement computeAndSaveChecksum() in apps/api/services/migration/MigrationService.ts to calculate checksum of current migrations and persist to checksum file path, handling IO errors and returning final checksum - (M) (1 hours)[FE][BE]
    - FS: Create writeChecksumFile() in apps/api/utils/checksum/file.ts to write file at configured path; ensure directory exists, handle atomic write, and set permissions flags - (S) (0.5 hours)[FE][BE][DevOps]
    - Config: Add checksum file path and filename in apps/api/config/paths.ts - (S) (0.5 hours)[FE][BE][DevOps]
    - Perm: Set file permissions after write in apps/api/utils/checksum/file.ts (chmod & ownership) and verify readability by deployment user - (M) (1 hours)[FE][BE][DevOps]

    - CI: Add GitHub Actions step in .github/workflows/ci.yml to validate checksum file exists and hash matches - (S) (0.5 hours)[FE][BE]
    - Test: Add unit test for checksum generation in apps/api/services/migration/__tests__/MigrationService.test.ts - (S) (0.5 hours)[FE][BE][QA]
    - Doc: Document checksum behavior and path in docs/migration-lock.md - (XS) (0.5 hours)[FE][BE]
- **Validate checksum:** As a: DevOps engineer, I want to: validate the migration-lock checksum during deployment, So that: I can detect changes to migrations and prevent unauthorized updates**(5 hours)** - Checksum file is read during deployment. Generated hash matches stored checksum for current migrations. Deployment fails with clear error if checksum mismatch or missing file.
    - Compute a cryptographic hash for all Prisma migrations in prisma/migrations/ by hashing file contents in scripts/computeChecksum.ts; output stored in a canonical checksum file for deployment validation. - (S) (0.5 hours)[FE][BE][DevOps][QA]
    - Migrations: Implement Checksum reader in apps/api/services/migrations/ChecksumService.ts to read and expose the computed checksum during deployment; ensure it tolerates missing file and validates format. - (M) (1 hours)[FE][BE][DevOps]
    - Migrations: Implement MigrationLock comparison in apps/api/services/migrations/MigrationLockService.ts to compare the generated hash against the table_migration_locks entry; return mismatch details to deploy validator. - (M) (1 hours)[FE][BE][DevOps]
    - Deploy: Add validation step in apps/api/deploy/validateMigrationLock.ts to fail deployment if checksum mismatch or checksum file is missing; integrate with existing deployment flow and error handler. - (M) (1 hours)[FE][BE][DevOps][QA]

    - CI: Add GitHub Actions step in .github/workflows/deploy.yml to execute node scripts/computeChecksum.ts and invoke apps/api/deploy/validateMigrationLock.ts as part of the CI deployment pipeline. - (S) (0.5 hours)[FE][BE][DevOps]
    - Test: Create unit tests in apps/api/tests/migrationChecksum.test.ts to cover matching checksum, mismatch, and missing checksum file scenarios. - (S) (0.5 hours)[FE][BE][QA]
    - Error Handling: Add clear error messages in apps/api/middleware/errorHandler.ts for checksum failures to improve debuggability and user feedback. - (XS) (0.5 hours)[FE][BE]
- **Commit migration folder:** As a: developer, I want to: commit the migration folder with the checksum metadata, So that: changes are tracked in version control and reproducible deployments exist**(6 hours)** - Migration folder including checksum file is committed to repository Commit triggers CI to run with checksum verification in pipeline No sensitive data exposed in commit Commit includes a clear message referencing checksum feature Rollback: previous commit containing migrations remains retrievable
    - DB: Create migrations folder and checksum file in `prisma/migrations/20251030_add_migrations/` (include checksum.txt) - (M) (1 hours)[FE][BE]
    - API: Add checksum verification in `apps/api/services/migration/MigrationService.ts` to read `prisma/migrations/…/checksum.txt` - (M) (1 hours)[FE][BE]
    - CI: Update `.github/workflows/ci.yml` to run checksum verification step and fail on mismatch - (M) (1 hours)[FE][BE]
    - Git: Commit migrations with message ‘Add migrations and checksum (migration-lock)’ in repo root via `git commit` to ensure rollback history - (M) (1 hours)[FE][BE]

    - Security: Scan commit to remove secrets, add `.gitignore` rules in `.gitignore` and pre-commit hook in `hooks/pre-commit` - (M) (1 hours)[FE][BE]
    - Docs: Add `docs/migrations.md` describing checksum feature and rollback procedure - (M) (1 hours)[FE][BE]
- **Validate checksum: (0 hours)**
- **Add migration checksum:** As a: database administrator, I want to: Add a checksum to each migration file in the migration folder, So that: I can verify integrity of migrations during deployment and rollback.**(32 hours)** - Checksum is computed for each migration file and stored in a separate manifest file or embedded metadata System validates checksum during migration execution and fails on mismatch Manifest lists all migration files with their checksums and timestamps Checksum recomputed on file modification and triggers alert or rejection if changed Security considerations ensure checksums are stored securely and not exposed publicly
    - Tooling: Add checksum generator script in `apps/api/scripts/migrations/generate_checksums.py` (4 hours)[FE][BE]
    - Manifest: Create `apps/api/prisma/migrations/manifest.json` creation logic in `apps/api/scripts/migrations/generate_checksums.py` (4 hours)[FE][BE]
    - Validation: Integrate checksum validation into migration runner in `apps/api/scripts/migrations/run_migration.py` (4 hours)[FE][BE][QA]
    - Watcher: Add file watcher to recompute checksum in `apps/api/scripts/migrations/watch_migrations.py` (4 hours)[FE][BE]
    - Alerting: Implement alert/reject on checksum mismatch in `apps/api/services/monitoring/AlertService.py` (4 hours)[FE][BE]

    - Security: Store manifest securely in `apps/api/secure/manifest_store.py` with restricted ACLs (4 hours)[FE][BE]
    - Testing: Add unit tests for checksum logic in `apps/api/tests/test_migration_checksums.py` (4 hours)[FE][BE][QA]
    - Docs: Document checksum process in `docs/migrations/checksum.md` and update README` (4 hours)[FE][BE]
- **Create migration folder:** As a: developer, I want to: Create a new migration folder for uploads presets with proper naming, schema and references, So that: Migrations are organized and auditable in the Prisma workflow.**(6 hours)** - New migration folder is created with correct naming convention Folder contains migration.sql and schema.prisma placeholders or templates Hash for folder integrity tracked in an index or manifest Migration script can be committed to repository without errors Tooling validates folder structure on CI to ensure consistency and existence of required files
    - FS: Create migration folder `prisma/migrations/_upload-presets/` with naming convention - (S) (0.5 hours)[FE][BE]
    - DB: Add `migration.sql` template in `prisma/migrations/_upload-presets/migration.sql` - (S) (0.5 hours)[FE][BE]
    - DB: Add `schema.prisma` placeholder in `prisma/migrations/_upload-presets/schema.prisma` - (S) (0.5 hours)[FE][BE]
    - SCRIPT: Implement hash generation and update `prisma/migrations/index.json` in `scripts/generate_migration_index.py` - (M) (1 hours)[FE][BE]
    - SCRIPT: Add validation script `scripts/validate_migration_folder.py` to verify structure and files - (M) (1 hours)[FE][BE][QA]
    - CI: Add GitHub Actions job in `apps/api/.github/workflows/ci.yml` to run `scripts/validate_migration_folder.py` - (L) (2 hours)[FE][BE]

    - DOC: Update `docs/CONTRIBUTING.md` with migration folder naming and commit instructions - (XS) (0.5 hours) [FE][BE]

- **Add README: (20 hours)**
    - Docs: Create `prisma/migrations/README.md` with migration overview and TOC (4 hours)[FE][BE]
    - Docs: Add migration usage examples referencing `prisma/schema.prisma` and `prisma/migrations/*.sql` in `prisma/migrations/README.md` (4 hours)[FE][BE]
    - Docs: Update `apps/api/README.md` with instructions to run migrations in `apps/api/prisma/` and seed scripts `apps/api/prisma/seed.ts` (4 hours)[FE][BE]
    - Docs: Update root `README.md` to reference `prisma/migrations/` and include badges and contribution pointer to `CONTRIBUTING.md` (4 hours)[FE][BE]
    - Docs: Add small CONTRIBUTING.md entry at `CONTRIBUTING.md` with migration workflow and PR checklist (4 hours)[FE][BE]
- **Add migration SQL: (32 hours)** - Story functionality is implemented as specified User interface is responsive and accessible Data is properly validated and stored Error handling provides helpful feedback
    - DB: Create migration SQL file in `prisma/migrations/20251030_add_migration.sql` (4 hours)[FE][BE]
    - DB: Update `prisma/schema.prisma` in `prisma/schema.prisma` to reflect new migration (4 hours)[FE][BE]
    - API: Add migration apply script in `apps/api/scripts/apply_migrations.py` (4 hours)[FE][BE]
    - API: Update models in `apps/api/models/user.py` to match schema changes (4 hours)[FE][BE]
    - API: Add validation and error handling in `apps/api/routes/migrations.py` (4 hours)[FE][BE][QA]

    - Frontend: Update UploadPresets component in `apps/web/components/migration/UploadPresets.tsx` to validate and include migration.sql + hash (4 hours)[FE][BE]
    - Tests: Add integration test in `apps/api/tests/test_migrations.py` to apply migration and verify `table_users` schema (4 hours)[FE][BE][QA]
    - Docs: Update `prisma/README.md` with migration instructions and CI steps in `.github/workflows/migrate.yml` (4 hours)[FE][BE]
- **Update schema.prisma: (8.5 hours)** - Story functionality is implemented as specified User interface is responsive and accessible Data is properly validated and stored Error handling provides helpful feedback
    - DB: Update schema in prisma/schema.prisma to add presets and hash fields - (M) (1 hours)[FE][BE]
    - DB: Create migration files in prisma/migrations/ for new presets table and altered users table - (L) (2 hours)[FE][BE]
    - DB: Run prisma migrate and commit generated client in apps/api/prisma/ - (M) (1 hours)[FE][BE]
    - API: Update Prisma model usages in apps/api/services/presets/PresetsService.ts and apps/api/services/auth/AuthService.ts - (M) (1 hours)[FE][BE]
    - API: Add validation and endpoints in apps/api/routes/presetsRouter.ts - (M) (1 hours)[FE][BE][QA]
    - Frontend: Update types and components in apps/web/src/components/presets/PresetsForm.tsx and apps/web/src/types/preset.ts - (S) (0.5 hours)[FE][BE]
    - Testing: Add integration tests in apps/api/tests/presets.test.ts and apps/web/tests/presets.integration.test.ts - (L) (2 hours)[FE][BE][QA]

- **Add automatic seed: (36 hours)** - Story functionality is implemented as specified User interface is responsive and accessible Data is properly validated and stored Error handling provides helpful feedback
    - DB: Add seed script in `prisma/seed.ts` to populate users table (4 hours)[FE][BE]
    - DB: Add migration in `prisma/migrations/` to run seed after migrate (4 hours)[FE][BE]
    - API: Implement run_seed() in `apps/api/services/seed/SeedService.py` (4 hours)[FE][BE]
    - API: Add router endpoint in `apps/api/routers/seed.py` to trigger seed (4 hours)[FE][BE]
    - Frontend: Build AutoSeedStatus component in `components/seed/AutoSeedStatus.tsx` (4 hours)[FE][BE]
    - Frontend: Create Redux slice in `store/slices/seedSlice.ts` to track seed state (4 hours)[FE][BE]
    - CI: Add `.github/workflows/prisma_seed.yml` to run migrations and seed in CI (4 hours)[FE][BE]
    - Tests: Add integration tests in `tests/seed_test.py` to verify seeding and error handling (4 hours)[FE][BE][QA]
    - Docs: Write seeding guide in `docs/seeding.md` with rollback steps (4 hours)[FE][BE]
- **Add migration checksum: (0 hours)**
- **Add migration SQL: (0 hours)**
- **Update schema.prisma: (0 hours)**
- **Add automatic seed: (0 hours)**
- **Add README: (0 hours)**

- **Add migration README: (20 hours)**
  - DOC: Create `prisma/migrations/README.md` with purpose and conventions (4 hours)[FE][BE]
  - DOC: Add example migration file `prisma/migrations/20251030_add_users_table/migration.sql` with schema snippet for `table_users` (4 hours)[FE][BE]
  - CI: Update `/.github/workflows/ci.yml` to validate migrations in `prisma/migrations/` (4 hours)[FE][BE]
  - DOC: Add usage section to `prisma/migrations/README.md` with prisma migrate commands and rollback steps (4 hours)[FE][BE]
  - TEST: Add integration test `apps/api/tests/migrations_test.py` to verify example migration applies to PostgreSQL (4 hours)[FE][BE][QA]
- **Add migration checksum: (0 hours)**
- **Run migrate dev:** As a: developer, I want to: run prisma migrate dev, So that: I can apply schema changes to the local database and generate migration artifacts for version control**(3 hours)** - Migration command completes successfully without errors Migration generates a new migration folder with up-to-date SQL statements Local database schema is updated to reflect Prisma model changes Migration status is reflected in Prisma status output
  - Prepare Prisma schema changes in prisma/schema.prisma by updating models, relations, and datasource generator according to MVP Postgres settings and existing DB schema. Ensure migrations section is ready for migration generation without applying changes yet. - (S) (0.5 hours)[FE][BE]
  - Create migration by running npx prisma migrate dev in the project root and capture output to scripts/migrate-dev.sh, including captured SQL, migration id, and any warnings/errors. Ensure script is executable and logs are deterministic for MVP scope. - (M) (1 hours)[FE][BE]

    - Ensure local Postgres is reachable and configured in prisma/.env and docker-compose.yml, including host, port, database, user, and password; verify docker-compose up can bring a running Postgres instance accessible to Prisma client. - (S) (0.5 hours)[FE][BE][DevOps]
    - Validate migration folder generated in prisma/migrations/ contains up-to-date SQL that matches the current Prisma schema and intended changes; ensure SQL reflects CREATE/ALTER statements for all modified models and relations. - (S) (0.5 hours)[FE][BE]
    - Run npx prisma migrate status and record status output to docs/migration_status.md, including applied migrations, pending migrations, and any drift warnings; ensure docs entry is formatted and checked in. - (S) (0.5 hours)[FE][BE]
- **Generate Prisma Client:** As a: developer, I want to: generate Prisma Client, So that: I can use the generated client in application code to access the database**(2.5 hours)** - Prisma Client generation completes without errors Generated client files exist under node_modules/@prisma/client Application can import and instantiate Prisma Client in code Prisma Client version matches Prisma schema version
    - Infra: Add Prisma schema at prisma/schema.prisma with users model (references table_users) using Prisma ORM, including datasource to Postgres and a User model with id, email, name, and relation to table_users. Ensure schema.prisma is in apps/api/prisma/schema.prisma and aligns with existing table_users relation. - (S) (0.5 hours)[FE][BE]
    - Infra: Ensure DB env in apps/api/.env and docker-compose.yml for Postgres, wiring DB_HOST, DB_USER, DB_PASSWORD, DB_NAME and ports consistent with Prisma schema and migrations. - (XS) (0.5 hours)[FE][BE]
    - Dev: Run npx prisma generate from apps/api/ and output to node_modules/@prisma/client, ensuring generator config in schema.prisma is used and that generated client is present in the expected path. - (S) (0.5 hours)[FE][BE][DevOps]

    - Dev: Create Prisma client wrapper in apps/api/src/lib/prisma.ts to import and instantiate PrismaClient, exporting a singleton for reuse across services, enabling type-safe DB access. - (S) (0.5 hours)[FE][BE]
    - Quality: Add test apps/api/tests/prismaClient.test.ts to import PrismaClient and assert instantiation and version match - (XS) (0.5 hours)[FE][BE][QA]
    - Infra: Add CI step in .github/workflows/ci.yml to run npx prisma generate in apps/api/ - (S) (0.5 hours)[FE][BE]
- **Commit migration folder:** As a: developer, I want to: commit migration folder, So that: migration artifacts are versioned and shared with the team**(3 hours)** - Migration folder added to git index Commit created with message referencing migration Remote repository contains migration artifacts CI pipeline recognizes new migrations without errors
    - Backend/VCS: Stage the Prisma migrations folder by adding prisma/migrations/ to git index in apps/api/ ensuring folder structure is preserved and empty migrations are avoided; verify git status shows the folder as staged with proper permissions. - (S) (0.5 hours)[FE][BE]
    - VCS/CI: Commit migrations with message ‘chore(prisma): add migration ’ at repo root using git commit -m, ensuring the commit includes prisma/migrations/ and is associated with a explanatory message; verify commit hash and commit message integrity. - (XS) (0.5 hours)[FE][BE]
    - Remote/Network: Push the committed changes to origin main from repo root using git push origin main; ensure correct remote is configured and authentication succeeds; verify remote branch updated with expected commit. - (S) (0.5 hours)[FE][BE][DevOps]
    - CI: Update/validate CI workflow at .github/workflows/ci.yml to run prisma migrate status for apps/api/ and ensure migrations are recognized; include check for exit code 0 and potential failure on missing migrations. - (M) (1 hours)[FE][BE]

    - Infra/Repo settings: Add prisma/migrations/ to repo and remove from .gitignore in root .gitignore if present; ensure subsequent commits reflect the ignore rule change and that migrations are tracked by version control. - (S) (0.5 hours) [FE][BE]

- **Run migrate dev: (0 hours)**
- **Generate Prisma Client: (0 hours)**
- **Commit migration folder: (0 hours)**

Milestone 2: Prisma model & SQL migration: add ProjectUploadPreset model, migration file 20251030_create_project_upload_presets.sql, seed SQL using uuid_generate_v4(), create table and indexes

Estimated 273.5 hours

- **Add migration SQL using uuid_generate_v4() from uuid-ossp:** As a: devops engineer, I want to: add migration SQL that uses uuid_generate_v4() from uuid-ossp, So that: UUIDs are generated at the database level for new records**(4 hours)** - Migration script creates table with UUID default using uuid_generate_v4() uuid-ossp extension is installed or available Migration runs successfully in tests No data loss on apply Rollback path exists
    - DB: Create prisma migration SQL in prisma/migrations/ 20251030_add_uuid_generate_v4/migration.sql to install extension uuid-ossp and create table project_upload_presets with id UUID DEFAULT uuid_generate_v4() - (M) (1 hours)[FE][BE]
    - DB: Add Prisma schema model ProjectUploadPreset in prisma/schema.prisma with @db.Uuid and dbgenerated('uuid_generate_v4()') - (S) (0.5 hours)[FE][BE]
    - Test: Write migration test in tests/migrations/ test_uuid_migration.test.ts to run migration and assert default UUID generation - (M) (1 hours)[FE][BE][QA]

    - Rollback: Add rollback SQL in prisma/migrations/20251030_add_uuid_generate_v4/rollback.sql to drop table and extension safely - (S) (0.5 hours)[FE][BE]
    - CI: Update GitHub Actions workflow /.github/workflows/ci.yml to run migration tests against PostgreSQL with uuid-ossp enabled - (M) (1 hours)[FE][BE][QA]
- **Add Prisma model ProjectUploadPreset (dbgenerated('uuid_generate_v4()') / db.Uuid):** As a: data engineer, I want to: add Prisma model ProjectUploadPreset with a UUID default using uuid_generate_v4(), So that: each project upload preset has a unique identifier in the database**(6.5 hours)** - Model is defined in Prisma schema with id field using UUID type and default uuid_generate_v4() Database migration creates a table for ProjectUploadPreset with primary key on id New table can be migrated without errors and is accessible Default UUID generation is supported by Postgres uuid-ossp extension Existing data unaffected
    - DB: Add ProjectUploadPreset model to prisma/schema.prisma with id String @db.Uuid @default(dbgenerated('uuid_generate_v4()')) - (S) (0.5 hours)[FE][BE]
    - DB: Create migration SQL in prisma/migrations/ to enable extension uuid-ossp and create table ProjectUploadPreset with primary key id - (M) (1 hours)[FE][BE]
    - Dev: Run npx prisma migrate dev and commit migration files in prisma/migrations/ - (S) (0.5 hours)[FE][BE]
    - Dev: Generate Prisma client in apps/api/node_modules/.prisma/client via npx prisma generate - (XS) (0.5 hours)[FE][BE]
    - API: Add service apps/api/services/projectUploadPresetService.ts with basic CRUD using Prisma client - (M) (1 hours)[FE][BE]
    - TEST: Add integration test prisma/tests/projectUploadPreset.test.ts to verify table exists and uuid defaults are generated - (M) (1 hours)[FE][BE][QA]

    - CI: Add migration step to .github/workflows/ci.yml to run npx prisma migrate deploy and ensure uuid-ossp extension present - (L) (2 hours)[FE][BE][DevOps]

- **Add migration SQL using uuid_generate_v4() from uuid-ossp:** As a: devops engineer, I want to: add migration SQL that uses uuid_generate_v4() from uuid-ossp, So that: UUIDs are generated at the database level for new records**(3 hours)** - Migration script creates table with UUID default using uuid_generate_v4() uuid-ossp extension is installed or available Migration runs successfully in tests No data loss on apply Rollback path exists

    - DB: Create migration to enable uuid-ossp extension in prisma/migrations/0001_enable_uuid_ossp/migration.sql with SQL to enable extension in PostgreSQL (CREATE EXTENSION IF NOT EXISTS “uuid-ossp”). Includes minimal verification query. - (S) (0.5 hours)[FE][BE]

    - DB: Add migration SQL to create ProjectUploadPreset table with default uuid_generate_v4() in prisma/migrations/0002_create_project_upload_preset/migration.sql. Creates table with id UUID default uuid_generate_v4(), other fields as needed, and ensure default generated UUID. - (S) (0.5 hours)[FE][BE]

    - DEV: Update Prisma schema in prisma/schema.prisma to use dbgenerated(‘uuid_generate_v4()’) for ProjectUploadPreset.id while preserving existing model fields and relations. - (S) (0.5 hours)[FE][BE]

    - TEST: Add integration test for migrations in tests/migrations/migration.test.ts to run migrations and verify no data loss. - (S) (0.5 hours)[FE][BE][QA]

    - CI: Update .github/workflows/ci.yml to install uuid-ossp or use Postgres image with extension pre-installed. - (S) (0.5 hours)[FE][BE]

    - DOC: Add rollback and notes in prisma/migrations/README.md and README.md describing extension enablement, table creation, rollback steps and considerations. - (XS) (0.5 hours)[FE][BE]

- **Create migration file 20251030_create_project_upload_presets.sql:** As a: DevOps engineer, I want to: Create migration file 20251030_create_project_upload_presets.sql, So that: the project_upload_presets table can be created with proper UUID defaults**(6 hours)** - Migration file name matches convention 20251030_create_project_upload_presets.sql SQL creates project_upload_presets table with id UUID default gen_random_uuid() Migration script is idempotent or safe to apply in test environments Migration applies cleanly in CI Rollback script restores previous schema state

    - DB: Create migration file prisma/migrations/20251030_create_project_upload_presets/20251030_create_project_upload_presets.sql with foundational migration file scaffold. - (XS) (0.5 hours)[FE][BE]

    - DB: Add CREATE TABLE project_upload_presets with id UUID DEFAULT gen_random_uuid() inside prisma/migrations/20251030_create_project_upload_presets/20251030_create_project_upload_presets.sql - (S) (0.5 hours)[FE][BE]

    - DB: Ensure pgcrypto extension exists in prisma/migrations/20251030_create_project_upload_presets/20251030_create_project_upload_presets.sql (CREATE EXTENSION IF NOT EXISTS pgcrypto) - (XS) (0.5 hours)[FE][BE]

    - DB: Make migration idempotent using IF NOT EXISTS guards in prisma/migrations/20251030_create_project_upload_presets/20251030_create_project_upload_presets.sql - (S) (0.5 hours)[FE][BE]

- DB: Add rollback SQL in prisma/migrations/20251030_create_project_upload_presets/down.sql to restore previous schema state - (S) (0.5 hours)[FE][BE]
- Dev: Update prisma/schema.prisma with model ProjectUploadPreset using dbgenerated(‘gen_random_uuid()’) - (M) (1 hours)[FE][BE]
- QA: Add migration test tests/migrations/test_migration_project_upload_presets.py to apply and rollback migration against test DB - (M) (1 hours)[FE][BE][QA]
- Infra: Update .github/workflows/ci.yml to run prisma migrate deploy and migration tests - (L) (2 hours)[FE][BE][DevOps][QA]

• **Add README for migration: (20 hours)**

- Docs: Create README for migrations at `prisma/migrations/README.md` (4 hours)[FE][BE]
- Docs: Document migration creation steps in `prisma/migrations/README.md` with examples of `prisma migrate dev` and `prisma migrate deploy` (4 hours)[FE][BE][DevOps]
- Docs: List affected tables and intents in `prisma/migrations/README.md` referencing `table_project_upload_default_mappings`, `table_project_upload_preset_intervals`, `table_staging_curated_datasets` (4 hours)[FE][BE]
- Docs: Add CI instructions in `prisma/migrations/README.md` describing usage in `.github/workflows/prisma-migrate.yml` (4 hours)[FE][BE]
- Docs: Add troubleshooting and rollback steps in `prisma/migrations/README.md` with commands for `psql` and `prisma migrate resolve` (4 hours)[FE][BE]

- **Add migration-lock checksum: (48 hours)**
  - DB: Add migration-lock checksum column in `prisma/migrations/migration-lock.schema.prisma` (4 hours)[FE][BE]
  - DB: Add migration-lock checksum migration in `prisma/migrations/` (4 hours)[FE][BE]
  - DB: Create migration to populate checksum in `prisma/migrations/20251030_add_migration_lock_checksum/steps.sql` (4 hours)[FE][BE]
  - Script: Create checksum generator `scripts/prisma/generate_migration_lock.ts` to compute checksum for `prisma/migrations/` (4 hours)[FE][BE]
  - API: Implement checksum compute and lock handling in `apps/api/services/migrations/MigrationLockService.ts` (4 hours)[FE][BE]
  - API: Implement MigrationLockService in `apps/api/services/db/MigrationLockService.ts` to read/write migration-lock checksum (4 hours)[FE][BE]
  - API: Add endpoint to verify and refresh migration-lock checksum in `apps/api/routers/migrationsRouter.ts` (4 hours)[FE][BE]
  - API: Update migration runner in `apps/api/services/db/PrismaMigrationRunner.ts` to validate checksum before applying migrations (4 hours)[FE][BE]
  - Testing: Add integration tests for migration-lock checksum in `tests/migrations/migration_lock.test.ts` (4 hours)[FE][BE][QA]
  - Tests: Add unit/integration tests in `apps/api/tests/migration_lock.test.ts` (4 hours)[FE][BE][QA]
  - Docs: Document migration-lock checksum behavior in `docs/prisma/migration-lock.md` (4 hours)[FE][BE]

    - CI: Add checksum verification step in `.github/workflows/ci.yml` and `deploy/migrate.sh` (4 hours)[FE][BE][DevOps]

- **Validate UUID defaults using uuid_generate_v4(): (6 hours)** - Story functionality is implemented as specified User interface is responsive and accessible Data is properly validated and stored Error handling provides helpful feedback

    - DB: Enable UUID generation by installing the uuid-ossp extension and creating a Prisma migration under prisma/migrations/ to run uuid_generate_v4() as the default for new records. - (S) (0.5 hours)[FE][BE]

    - DB: Add Prisma migration to set default uuid_generate_v4() on project_upload_default_mappings.id in prisma/migrations/, ensuring new rows have server-generated IDs. - (S) (0.5 hours)[FE][BE]

    - Prisma: Update prisma/schema.prisma to map table/collection with @@map and set default = uuid_generate_v4() for ProjectUploadDefaultMapping model, aligning ORM with DB default. - (S) (0.5 hours)[FE][BE]

    - Prisma: Update prisma/schema.prisma for ProjectUploadPresetInterval and staging_curated_datasets models to use uuid_generate_v4() as the default for their UUID fields. - (S) (0.5 hours)[FE][BE]

    - API: Implement validation and creation using Prisma client in apps/api/services/db/DBService.ts, enforcing UUID presence server-side and relying on DB/server defaults when omitted. - (M) (1 hours)[FE][BE][QA]

    - API Router: Update apps/api/routers/projectUploadPreset.ts to enforce missing UUIDs handling and surface actionable errors to clients. - (M) (1 hours)[FE][BE]

    - Frontend: Update apps/web/components/ProjectUploadPresetForm.tsx to avoid sending client-generated UUIDs, rely on server-generated IDs, and add accessibility checks. - (S) (0.5 hours)[FE][BE]

    - Tests: Add integration test apps/api/tests/uuid_defaults_test.py to create records and assert UUID defaults and storage in table_project_upload_default_mappings. - (M) (1 hours)[FE][BE][QA]
    - Docs: Add migration and usage note in docs/prisma/uuid_defaults.md describing uuid_generate_v4() requirement. - (XS) (0.5 hours)[FE][BE]
- **Ensure Project relation and cascade delete: (0 hours)**
- **Add StagingCuratedDataset model for ingestion: (0 hours)**
- **Add Prisma schema entry for ProjectUploadPreset: (7 hours)** - Story functionality is implemented as specified User interface is responsive and accessible Data is properly validated and stored Error handling provides helpful feedback
    - DB: Add ProjectUploadPreset model in `prisma/schema.prisma` with id dbgenerated('gen_random_uuid()') and fields - (M) (1 hours)[FE][BE]
    - DB: Create migration in `prisma/migrations/` for ProjectUploadPreset table - (M) (1 hours)[FE][BE]
    - API: Add ORM model usage in `apps/api/services/project/ProjectService.ts` - create, read, validate ProjectUploadPreset - (M) (1 hours)[FE][BE]
    - API: Add router endpoints in `apps/api/routers/project.py` for ProjectUploadPreset CRUD - (M) (1 hours)[FE][BE]
    - Frontend: Build ProjectUploadPresetForm component in `components/project/ProjectUploadPresetForm.tsx` with responsive Chakra UI form and validation - (M) (1 hours)[FE][BE][QA]
    - Tests: Add integration tests in `tests/test_project_upload_preset.py` for validation and storage - (M) (1 hours)[FE][BE][QA]

    - Docs: Add docs in `docs/project_upload_preset.md` and update changelog - (M) (1 hours)[FE][BE]

- **Create migration SQL file: (0 hours)**
- **Create seed script for default bovine presets:** As a: DevOps/DB engineer, I want to: Create seed script for default bovine presets, So that: the database has initial presets for bovine-related workflows**(4 hours)** - Seed script inserts predefined bovine presets with valid UUIDs Seed can be idempotent to avoid duplicates Seeding runs without errors in local and test environments Seeded data accessible via Prisma client and queries reflect correct counts
    - DB: Add ProjectUploadPreset model check in prisma/schema.prisma and create migration in prisma/migrations/ - (S) (0.5 hours)[FE][BE]
    - Script: Create idempotent seed script prisma/seed/seedProjectUploadPresets.ts to insert default bovine presets with UUIDs using Prisma client - (M) (1 hours)[FE][BE]
    - Impl: Implement upsert logic in prisma/seed/seedProjectUploadPresets.ts using UUIDs and pgcrypto gen_random_uuid() fallback - (M) (1 hours)[FE][BE]
    - Test: Add integration tests in tests/seed/seedProjectUploadPresets.test.ts to verify idempotency and correct counts via Prisma client - (S) (0.5 hours)[FE][BE][QA]
    - CI: Add GitHub Actions step in .github/workflows/seed.yml to run prisma migrate deploy and node prisma/seed/seedProjectUploadPresets.ts in test matrix - (S) (0.5 hours)[FE][BE][DevOps][QA]
    - Doc: Document seed usage and rollback in docs/development/seeding.md with commands to run in local/test envs - (XS) (0.5 hours)[FE][BE][QA]
- **Add migration SQL using gen_random_uuid() from pgcrypto: (0 hours)**

- **Add payload validation and normalization hooks (mapping & units):** As a: data validator, I want to: add payload validation and normalization hooks for mapping and units, So that: incoming data conforms to expected formats and internal representations**(7.5 hours)** - Validation layer rejects invalid mapping values Normalization converts units to canonical form Hooks applied to incoming payload before persistence Tests cover valid and invalid payload scenarios Performance impact kept minimal
    - Config: Add unit mapping file apps/api/config/unitMappings.json preserving architecture: API layer and config management. Include mapping keys for unit normalization and a default fallback. Ensure file is loaded by UnitNormalizer and PayloadValidator during MVP flow. - (S) (0.5 hours)[FE][BE][DevOps]
    - Types: Add payload types in apps/api/types/payload.ts ensuring structural compatibility with existing Prisma and validation logic; include interfaces for validation result, unit fields, and optional mapping references. - (S) (0.5 hours)[FE][BE][QA]
    - API: Implement PayloadValidator.validatePayload() in apps/api/services/validation/PayloadValidator.ts to enforce required fields, types, and basic unit validation using unitMappings.json. - (M) (1 hours)[FE][BE][QA]
    - API: Implement UnitNormalizer.normalizeUnits() in apps/api/services/normalization/UnitNormalizer.ts to convert unit strings to canonical forms using unitMappings.json; handles pluralization and case-insensitive matching. - (M) (1 hours)[FE][BE]
    - Prisma: Add middleware hook in apps/api/prisma/middleware.ts to run validation and normalization before create/update, wiring PayloadValidator and UnitNormalizer into Prisma lifecycle. - (L) (2 hours)[FE][BE][QA]
    - DB: Update prisma/schema.prisma for ProjectUploadPreset if needed and run prisma/migrations/ to support new fields introduced by payload hooks. - (M) (1 hours)[FE][BE]

    - Tests: Add unit and integration tests in apps/api/tests/payload_hooks.test.ts covering valid and invalid payloads and perf. - (M) (1 hours)[FE][BE][QA]
    - Docs: Add docs/payload_hooks.md explaining hooks and config paths - (XS) (0.5 hours)[FE][BE][DevOps]
- **Update README to document uuid_generate_v4() requirement and migration:** As a: tech writer, I want to: update README with uuid_generate_v4() requirement and migration instructions, So that: developers understand database UUID setup and migration steps**(7 hours)** - README mentions uuid_generate_v4() requirement Migration steps described clearly Examples of UUID usage No broken links or deprecated instructions Documented rollback and testing guidance
    - Docs: Update README.md with uuid_generate_v4() requirement and explanation in `README.md` - (S) (0.5 hours)[FE][BE]
    - Docs: Add migration steps and CLI commands in `docs/migrations/uuid_setup.md` - (S) (0.5 hours)[FE][BE][DevOps]
    - DB: Add SQL migration enabling uuid-ossp in `prisma/migrations/2025xxxx_add_uuid_ossp/steps.sql - (L) (2 hours)[FE][BE]
    - DB: Create Prisma migration files in `prisma/migrations/2025xxxx_prisma_uuid_generate_v4/` with dbgenerated() annotations in `prisma/schema.prisma` - (L) (2 hours)[FE][BE]
    - Docs: Add UUID usage examples in `README.md` and `docs/examples/uuid_examples.md` - (S) (0.5 hours)[FE][BE]
    - Quality: Add rollback and testing guidance in `docs/migrations/uuid_setup.md` and `tests/db/uuid_migration.test.ts` - (M) (1 hours)[FE][BE][DevOps][QA]

    - CI: Add GitHub Actions step in `.github/workflows/checks.yml` to validate migrations and README links - (S) (0.5 hours)[FE][BE]

- **Note app logic: create default preset on project creation:** As a: database administrator, I want to: create a default preset automatically when a new project is created, So that: every project has a baseline configuration ready for use**(9.5 hours)** - When a new project is created, a default preset record is inserted automatically The default preset aligns with the project schema and preset constraints No duplicate default presets are created for the same project Audit log records the creation event with timestamp and project ID

    - DB: Add migration in prisma/migrations/20251030_create_project_upload_presets.sql to create table project_upload_presets and a unique index, using gen_random_uuid() in public schema. - (L) (2 hours)[FE][BE]

    - DB: Add seed insert for existing projects in prisma/migrations/20251030_create_project_upload_presets.sql to create default presets for each project. - (M) (1 hours)[FE][BE]

    - API: Implement DefaultPresetService.create_default_for_project in apps/api/services/presets/DefaultPresetService.py to create a preset for a given project using the new table and share logic with seeding. - (M) (1 hours)[FE][BE]

    - API: Add createDefaultPreset mutation in apps/api/routers/project_upload_presets.py to expose creation and enforce schema for inputs/outputs. - (S) (0.5 hours)[FE][BE]

    - API: Call create_default_preset on project creation in apps/api/routers/projects.py to ensure a default preset is created when a project is created. - (M) (1 hours)[FE][BE]

    - DB: Add trigger or transaction logic in prisma/migrations/20251030_create_project_upload_presets.sql to prevent duplicate default presets per project. - (L) (2 hours)[FE][BE]
    - API: Record audit log in apps/api/services/audit/AuditService.py when default preset is created (timestamp + project_id). - (S) (0.5 hours)[FE][BE]
    - QA: Add unit tests in tests/test_default_preset.py for DefaultPresetService and migration effects. - (M) (1 hours)[FE][BE][QA]
    - CI: Add migration step to .github/workflows/migrate.yml to apply prisma/migrations/20251030_create_project_upload_presets.sql - (S) (0.5 hours)[FE][BE]
- **Use gen_random_uuid() from pgcrypto: (17 hours)** - Given a database with pgcrypto installed, when generating IDs using gen_random_uuid(), then a new UUID is produced for each new row. The migration script uses gen_random_uuid() in appropriate insert statements and compiles without syntax errors. System validates that UUIDs are UUIDv4 format and unique across inserted rows.
    - DB: Create migration file `apps/api/db/migrations/20251030_use_pgcrypto_gen_random_uuid.sql` to CREATE EXTENSION IF NOT EXISTS pgcrypto; alter inserts to use gen_random_uuid() - (M) (1 hours)[FE][BE]
    - DB: Update seed SQL in `apps/api/db/seeds/20251030_seed_project_upload_presets.sql` to use gen_random_uuid() in INSERT statements (4 hours)[FE][BE]
    - API: Add runMigrationWithSeed handler in `apps/api/routes/project_upload_presets/runMigration.ts` to execute migration and seeds using pgcrypto (4 hours)[FE][BE]
    - Testing: Add UUIDv4 uniqueness and format tests in `apps/api/tests/migrations/gen_random_uuid.test.ts` (4 hours)[FE][BE][QA]

    - Docs: Add docs `docs/migrations/20251030_use_pgcrypto.md` describing pgcrypto dependency and usage of gen_random_uuid() (4 hours)[FE][BE]

- **Create migration with gen_random_uuid():** As a: database administrator, I want to: Create a migration that uses gen_random_uuid() for identifiers, So that: Presets and related records have unique IDs generated at runtime.**(5 hours)** - Migration applies without errors All ID columns are populated with valid UUIDs Backward compatibility with existing schema is maintained Migration idempotency is ensured to avoid duplicates on re-run Logs show UUID generation events during migration
    - DB: Create SQL migration file in `apps/api/prisma/migrations/20251030_create_project_upload_presets.sql` to create table and use gen_random_uuid() for id columns - (M) (1 hours)[FE][BE]
    - DB: Add id population step in `apps/api/prisma/migrations/20251030_create_project_upload_presets.sql` using UPDATE … SET id = gen_random_uuid() WHERE id IS NULL - (M) (1 hours)[FE][BE]
    - Infra: Add migration runner script `apps/api/scripts/run_migrations.sh` with logging of UUID generation events - (M) (1 hours)[FE][BE]
    - CI: Add GitHub Actions step in `.github/workflows/migrate.yml` to run `apps/api/scripts/run_migrations.sh` in Docker - (M) (1 hours)[FE][BE]
    - Testing: Create migration test `tests/migrations/test_migration_gen_uuid.py` to verify ids populated and idempotency - (M) (1 hours)[FE][BE][QA]
- **Insert presets for existing projects:** As a: database administrator, I want to: Insert presets for existing projects during migration, So that: Each project has a default preset in public.project_upload_presets**(6 hours)** - Presets are inserted for all existing projects Each preset references a valid

project_id Insertion script runs without errors No orphan presets exist after migration Presets have correct default values as defined in schema

  - DB: Create migration SQL file `migrations/20251030_create_project_upload_presets.sql` to create presets table and insert presets per existing project using gen_random_uuid() - (M) (1 hours)[FE][BE]
  - DB: Implement insertion script in `migrations/20251030_create_project_upload_presets.sql` ensuring preset.project_id references projects table and default values per schema - (M) (1 hours)[FE][BE]
  - DB: Add rollback (DROP INSERTED presets) in `migrations/20251030_create_project_upload_presets.sql` - (M) (1 hours)[FE][BE]
  - QA: Add SQL unit test in `tests/db/test_presets_insertion.sql` to verify no orphan presets and correct default values - (M) (1 hours)[FE][BE][QA]
  - Infra: Add migration run step to `/.github/workflows/ci.yml` to run the new migration in CI using Docker Postgres - (M) (1 hours)[FE][BE]
  - Doc: Update migration notes in `docs/migrations/20251030_create_project_upload_presets.md` with acceptance criteria and run instructions - (M) (1 hours)[FE][BE]

- **Create migration 20251030_create_project_upload_presets.sql: (2.5 hours)** - Migration file 20251030_create_project_upload_presets.sql is created and present in migration path. Table is created with required columns and constraints. Presets are inserted per existing project as expected by business logic.
  - DB: Create migration file prisma/migrations/20251030_create_project_upload_presets.sql preserving repository structure and naming conventions - (XS) (0.5 hours)[FE][BE]

    - DB: Add CREATE TABLE project_upload_presets in prisma/migrations/20251030_create_project_upload_presets.sql with columns id uuid default uuid_generate_v4(), project_id uuid NOT NULL, preset_name text NOT NULL, settings jsonb NOT NULL, created_at timestamptz DEFAULT now(), constraints: FK -> table_projects, UNIQUE(project_id, preset_name) - (M) (1 hours)[FE][BE]
    - DB: Insert presets per existing project in prisma/migrations/20251030_create_project_upload_presets.sql using INSERT … SELECT project_id, 'default'… FROM table_projects - (S) (0.5 hours)[FE][BE]
    - QA: Add migration verification test in apps/api/tests/migrations.test.ts to check table exists and presets inserted - (S) (0.5 hours)[FE][BE][QA]
- **Add INSERT presets for existing projects:** As a: database administrator, I want to: insert presets for existing projects during migration, So that: projects have default presets immediately after deployment**(5 hours)** - Presets are inserted for all existing projects Each preset associates correctly with a project ID No duplicate presets are created on re-run INSERT statements validate with test data Migration script runs cleanly on test environment
    - DB: Add uuid-ossp extension and create migration in `prisma/migrations/20251030_create_project_upload_presets.sql` - (M) (1 hours)[FE][BE]
    - DB: Implement INSERT presets per existing project with idempotent ON CONFLICT in `prisma/migrations/20251030_create_project_upload_presets.sql` - (M) (1 hours)[FE][BE]
    - Script: Create seed script to verify inserts in `scripts/seed_presets.ts` - (M) (1 hours)[FE][BE]
    - Test: Add migration test `tests/migrations/presets.test.ts` to validate INSERT statements and no duplicates - (M) (1 hours)[FE][BE][QA]

    - CI: Add GitHub Actions workflow `.github/workflows/migration-test.yml` to run migration and tests - (M) (1 hours)[FE][BE][QA]

- **Add migration SQL file (uses uuid_generate_v4()): (2.5 hours)** - Migration SQL file can create the table without errors. The script uses uuid_generate_v4() and inserts a UUID for seed data. Existing projects unaffected; migration runs on clean DB and records are created.

    - DB: Create migration script at prisma/migrations/20251030_create_project_upload_presets.sql to create table project_upload_presets and enable the uuid-ossp extension, laying the foundation for UUID primary keys in the migration workflow. - (M) (1 hours)[FE][BE]

    - DB: Seed the project_upload_presets table using uuid_generate_v4() inside prisma/migrations/20251030_create_project_upload_presets.sql to insert at least one preset entry as part of the initial migration. - (S) (0.5 hours)[FE][BE]

    - Testing: Add a test script scripts/db/test_migration.sh to run the migration on a clean database and verify that records in project_upload_presets exist and have valid UUIDs. - (S) (0.5 hours)[FE][BE][QA]

    - Docs: Document the migration in docs/database/migrations.md and note the usage of uuid_generate_v4() within the migration process. - (XS) (0.5 hours)[FE][BE]

- **Insert presets for existing projects:** As a: database administrator, I want to: insert presets for existing projects, So that: associations between projects and presets exist**(3.5 hours)** - Identify all existing projects Insert a preset record for each project with proper foreign key Verify presets count equals projects count No orphan presets created

    - DB: Create migration SQL file migrations/20251030_create_project_upload_presets.sql to create presets table entries using gen_random_uuid() - (M) (1 hours)[FE][BE]

    - DB: Implement insert logic in migrations/20251030_create_project_upload_presets.sql: INSERT INTO preset_field_mappings (id, project_id, …) SELECT gen_random_uuid(), p.id, … FROM projects p - (S) (0.5 hours)[FE][BE]
    - Script: Add verification script scripts/verify_presets_count.py to check presets count equals projects count and no orphan presets - (S) (0.5 hours)[FE][BE]
    - Test: Add test tests/test_presets_migration.py to assert presets count == projects count and FK integrity to table_preset_field_mappings - (M) (1 hours)[FE][BE][QA]
    - CI: Update /.github/workflows/migrations.yml to run migration migrations/20251030_create_project_upload_presets.sql and run scripts/verify_presets_count.py - (S) (0.5 hours)[FE][BE]
- **Create migration SQL file: (0 hours)**
- **Add migration SQL file (uses uuid_generate_v4()):** As a: database administrator, I want to: add a migration SQL file that utilizes uuid_generate_v4() for primary keys, So that: unique identifiers are generated without exposing sequential IDs**(6 hours)** - Migration file includes a valid SQL that calls uuid_generate_v4() Migration runs without errors against a test database The generated IDs are UUIDs and unique per row Documentation references the use of uuid_generate_v4() and its extension Edge case: ensure no collisions with existing IDs
    - DB: Create migration SQL `prisma/migrations/20251030_create_project_upload_presets.sql` to ENABLE EXTENSION uuid-ossp and CREATE TABLE project_upload_presets using uuid_generate_v4() for id - (M) (1 hours)[FE][BE]

    - DB: Add INSERT statements in `prisma/migrations/20251030_create_project_upload_presets.sql` to insert presets using uuid_generate_v4() and reference existing projects - (M) (1 hours)[FE][BE]
    - Dev: Update seed script `scripts/seed.ts` to handle existing IDs and use SELECT uuid_generate_v4() where needed - (M) (1 hours)[FE][BE]
    - Test: Add migration test `tests/migrations/project_upload_presets.test.ts` to run migration against test DB and verify UUID format and uniqueness - (M) (1 hours)[FE][BE][QA]
    - Docs: Create `docs/migrations/project_upload_presets.md` documenting use of uuid_generate_v4() and enabling uuid-ossp extension - (M) (1 hours)[FE][BE]
    - Infra: Add CI job in `.github/workflows/migrate.yml` to run migration against a test Postgres instance - (M) (1 hours)[FE][BE][QA]
- **Ensure uuid_generate_v4() extension note:** As a: database administrator, I want to: note the uuid_generate_v4() extension usage, So that: future maintainers understand the dependency and configuration**(4 hours)** - Documentation notes the dependency on uuid_generate_v4() Migration file mentions extension status and availability No runtime impact expected from the note Review confirms note is clear and actionable
    - DB: Add note to `prisma/migrations/20251030_create_project_upload_presets/20251030_create_project_upload_presets.sql` mentioning uuid_generate_v4() extension status and availability - (M) (1 hours)[FE][BE]
    - Documentation: Add extension note in `docs/database/extensions.md` explaining dependency on uuid_generate_v4() and no runtime impact - (M) (1 hours)[FE][BE]

    - Docs: Update `docs/PR_CHECKLIST.md` to require migration extension notes for `prisma/migrations/` files - (M) (1 hours)[FE][BE]
    - Testing: Add reviewer checklist entry in `tests/migration/README.md` to confirm migration mentions extension and availability - (M) (1 hours)[FE][BE][QA]
- **Skip existing presets when present:** As a: database administrator, I want to: skip inserting presets that already exist for a project, So that: seeding is idempotent and avoids duplicates**(5 hours)** - Seed is idempotent when presets exist No duplicate presets after repeated seeds Existing presets are not overwritten by new seed runs
    - DB: Create migration for presets table in `prisma/migrations/` - (M) (1 hours)[FE][BE]
    - SQL: Update `prisma/seeders/seed_project_upload_presets.sql` to INSERT … ON CONFLICT DO NOTHING and use uuid_generate_v4() where needed - (M) (1 hours)[FE][BE]
    - Script: Implement idempotent seeder in `scripts/seed/presetsSeed.ts` with existence checks and logging - (M) (1 hours)[FE][BE]
    - Test: Add integration tests in `tests/seed/presetsSeed.test.ts` to run seeder twice and assert no duplicates and no overwrites - (M) (1 hours)[FE][BE][QA]
    - Docs: Document seeding behavior in `docs/seeding.md` and reference `prisma/seeders/seed_project_upload_presets.sql` - (M) (1 hours)[FE][BE]
- **Ensure id uses uuid_generate_v4():** As a: database engineer, I want to: ensure the generated IDs for seed presets use uuid_generate_v4(), So that: IDs are universally unique and consistent with existing project seeding strategy**(3 hours)** - Verify id column default uses uuid_generate_v4() Create

preset without id and confirm generated id is a valid UUID v4 Seed operation results in IDs that follow UUID v4 pattern and are unique across presets

    - Database migration: set default value of the id column to uuid_generate_v4() in prisma/migrations/2025xxxx_set_uuid_default/ to ensure new records receive a UUID by default. - (S) (0.5 hours)[FE][BE]
    - Database seed: update prisma/seed/seed_project_upload_presets.sql to omit id so the database default uuid_generate_v4() is used on insert. - (S) (0.5 hours)[FE][BE]
    - API: Update prisma schema model for Preset to set default id using dbgenerated('uuid_generate_v4()') in apps/api/prisma/schema.prisma. - (S) (0.5 hours)[FE][BE]
    - Test: Add integration test tests/integration/seedPresets.test.ts to insert preset without id, assert UUID v4 pattern, and uniqueness across inserted records. - (M) (1 hours)[FE][BE][QA]
    - CI: Update GitHub Actions workflow .github/workflows/seed.yml to run migrations and execute node tests/integration/seedPresets.test.ts. - (S) (0.5 hours)[FE][BE][QA]

- **Add Seed SQL row creation: (24 hours)**
    - DB: Add seed SQL file `db/seed/seed_project_upload_presets.sql` with INSERT using gen_random_uuid() or provided IDs (4 hours)[FE][BE]
    - INFRA: Create Prisma migration in `prisma/migrations/` to support seed insert if schema changes (4 hours)[FE][BE]
    - API: Implement createSeedSQLRow mutation in `apps/api/routers/project_upload_presets.ts` (4 hours)[FE][BE]
    - API: Add SeedService.createSeedRow in `apps/api/services/seed/SeedService.ts` to assemble row data (4 hours)[FE][BE]

    - QUALITY: Add integration test in `apps/api/tests/seed_sql.test.ts` for createSeedSQLRow (4 hours)[FE][BE][QA]
    - DOCS: Update docs in `docs/api/project_upload_presets.md` describing createSeedSQLRow and seed file usage (4 hours)[FE][BE]
- **Add seed SQL row per project: (0 hours)**
- **Skip existing presets: (0 hours)**
- **Use public schema:** As a: database administrator, I want to: ensure seeding operates against the public schema, So that: it seeds under a predictable, shared database namespace**(3.5 hours)** - Seed targets public schema All inserted rows reside in public schema Migration scripts reference public schema consistently
    - DB: Update seed SQL file prisma/seed/seed_project_upload_presets.sql to insert into public.project_upload_presets using public schema qualifiers, ensuring all table references, column names, and literals use public-qualified identifiers. - (S) (0.5 hours)[FE][BE]
    - DB: Modify migration prisma/migrations/*/seed_project_upload_presets.sql to reference public.schema and use uuid_generate_v4() where needed, ensuring extension availability and consistent UUIDs. - (M) (1 hours)[FE][BE]
    - API: Update Prisma seeding script in prisma/seed/seed.ts to set schema to ‘public’ when connecting and to use public. qualifiers in all model references, ensuring Prisma client targets public schema. - (S) (0.5 hours)[FE][BE]
    - TESTING: Add integration test in tests/db/seed_public_schema.test.ts to verify inserted rows reside in public schema and migrations reference public, including cross-checks of data presence and schema attribution. - (M) (1 hours)[FE][BE][QA]

    - DOCS: Update migration docs in docs/db/migrations.md to state public schema usage and examples of qualified inserts, include rationale and examples for uuid_generate_v4(). - (XS) (0.5 hours)[FE][BE]

- **Create migration file:** As a: database engineer, I want to: create a new migration file, So that: I can introduce the necessary schema changes for project upload presets.**(4 hours)** - Migration file can be created and tracked in version control Migration script adheres to project naming conventions Database can apply the migration without errors Migration includes basic table/column definitions as required by the feature

    - DB: Create migration SQL file `prisma/migrations/20251030_create_project_upload_presets.sql` with uuid_generate_v4() and table definitions (schema: public) - (M) (1 hours)[FE][BE]

    - api_development: Add migration script in `package.json` scripts: ‘prisma:migrate:deploy’ pointing to Prisma CLI - (M) (1 hours)[FE][BE][DevOps]

    - testing: Add test to `tests/db/migrations/apply_migration.test.ts` to run prisma migrate deploy against test DB - (M) (1 hours)[FE][BE][DevOps][QA]

    - documentation: Add migration entry in `docs/migrations/README.md` with naming conventions and file path `prisma/migrations/20251030_create_project_upload_presets.sql` - (M) (1 hours)[FE][BE]

- **Add uuid_generate_v4:** As a: database engineer, I want to: add uuid_generate_v4 support to primary keys, So that: records use UUIDs for unique identifiers.**(3 hours)** -

uuid_generate_v4 extension is installed Primary keys utilize UUIDs Existing records have UUIDs generated for new inserts Migration scripts reference UUID generation consistently

  - DB: Update prisma/schema.prisma to set model IDs to UUIDs and default to uuid_generate_v4() in prisma/schema.prisma - (M) (1 hours)[FE][BE]
  - API: Update seed/inserts in prisma/seed.ts to call uuid_generate_v4() or generate UUIDs for new records - (S) (0.5 hours)[FE][BE]
  - Testing: Add migration test in tests/migrations/migration_uuid.test.ts to verify extension installed and UUID PKs - (S) (0.5 hours)[FE][BE][QA]
  - Infra: Add DB migration step to CI in .github/workflows/ci.yml to run prisma migrate deploy before tests - (M) (1 hours)[FE][BE][DevOps][QA]

- **Add staging flags:** As a: deployment engineer, I want to: add staging flags to the migration flow, So that: we can toggle feature behavior in staging environments without affecting prod.**(3.5 hours)** - Staging flags defined in migration context Flags default to safe values Migration and application read flags correctly from env/config Unit tests validate flag behavior in staging scenarios

  - DB: Add staging flags columns and defaults in apps/api/prisma/migrations/20251030_create_project_upload_presets.sql preserving migration-based change and default values for staging flags in the Prisma migration script. - (XS) (0.5 hours)[FE][BE]
  - DB: Add migration context reader in apps/api/src/migrations/context.ts to expose staging flags to the runtime/type-safe access layer, integrating with existing context mechanism. - (S) (0.5 hours)[FE][BE]
  - API: Implement staging flags loader in apps/api/src/config/stagingFlags.ts (read from env/config) leveraging existing config system to load STAGING_FLAG_* values at startup for runtime usage. - (S) (0.5 hours)[FE][BE][DevOps]

    - API: Update env schema and examples in .env.example to include STAGING_FLAG_* entries ensuring validation and documentation parity across environments. - (XS) (0.5 hours)[FE][BE][DevOps][QA]
    - API: Make runMigrations use flags in apps/api/scripts/runMigrations.ts to pass into apps/api/src/migrations/context.ts - (M) (1 hours)[FE][BE]
    - TEST: Add unit tests for staging flags behavior in apps/api/tests/stagingFlags.test.ts - (S) (0.5 hours)[FE][BE][QA]
    - DOC: Update docs/migrations.md with staging flags description and safe defaults - (XS) (0.5 hours)[FE][BE]
- **Seed presets helper:** As a: data engineer, I want to: seed presets helper data into the database, So that: development and testing environments have preset configurations available. **(5.5 hours)** - Seed helper inserts valid presets Seeds idempotent behavior on reruns Seed data adheres to schema constraints Rollback mechanism in migrations undoes seeded data
    - DB: Add seed presets SQL in apps/api/prisma/seeders/seedPresets.sql (insert with ON CONFLICT for idempotency). Preserve seedPresets structure and idempotent behavior to support repeated runs without duplication. - (M) (1 hours)[FE][BE]
    - API: Implement seed helper in apps/api/services/seed/seedPresetsHelper.ts (use Prisma raw SQL + transactions). Expose helper to orchestrate seed run, support rollback path via transaction. Includes idempotent execution and error handling. - (L) (2 hours)[FE][BE]
    - Migration: Add migration file migrations/20251030_create_project_upload_presets.sql to run seed SQL and register rollback. Ensure timestamped migration and reversibility for seeding changes. - (M) (1 hours)[FE][BE]

    - Testing: Add integration tests in apps/api/tests/seed/seedPresets.test.ts to verify insertion, idempotency, and schema adherence. Tests run against test DB with seeds applied once and repeated runs. - (M) (1 hours)[FE][BE][QA]
    - Docs: Document seed behavior in docs/db/seeds.md and update README.md with rollback instructions. Include sample commands and rollback steps for seed migrations. - (S) (0.5 hours)[FE][BE]
- **Add migration files (public schema, uuid_generate_v4()):** As a: devops engineer, I want to: add migration files in the public schema using uuid_generate_v4(), So that: the database schema can evolve with proper UUID primary keys**(4 hours)** - Migration file creates table with UUID primary key using uuid_generate_v4() Migration applies without errors in a test environment Migration is idempotent and does not recreate existing objects Migration file includes rollback/down script Migration log shows successful application
    - DB: Add rollback/down script in `prisma/migrations/20251030_add_project_upload_presets/down.sql` with DROP TABLE IF EXISTS public.project_upload_presets; - (M) (1 hours)[FE][BE]
    - Dev: Update seed SQL in `apps/api/prisma/seed_project_upload_presets.sql` to use uuid_generate_v4() for inserted rows - (M) (1 hours)[FE][BE]
    - QA: Create test in `apps/api/prisma/tests/apply_migrations.test.ts` to run migrations, assert no errors, verify table exists and uuid default works - (M) (1 hours)[FE][BE][QA]
    - Infra: Add GitHub Actions workflow ` .github/workflows/migration-test.yml` to run `npx prisma migrate deploy` against test DB and log results - (M) (1 hours)[FE][BE][DevOps][QA]

- **Add seed: seed_project_upload_presets.sql (public, uuid_generate_v4()):** As a: data engineer, I want to: add seed script to populate presets in public schema using UUIDs, So that: the database has initial project upload presets**(6 hours)** - Seed runs without errors Presets have valid UUIDs generated by uuid_generate_v4() Seed ensures unique presets are created Seed can be re-run idempotently without duplicates Seed results verifiable via count of rows seeded
    - DB: Create seed file `prisma/seed/seed_project_upload_presets.sql` with uuid_generate_v4() inserts (public) - (M) (1 hours)[FE][BE]
    - DB: Add idempotency checks in `prisma/seed/seed_project_upload_presets.sql` using WHERE NOT EXISTS - (M) (1 hours)[FE][BE]
    - DB: Add SQL to validate row count in `prisma/seed/seed_project_upload_presets.sql` and output count - (M) (1 hours)[FE][BE]
    - Infra: Ensure uuid-ossp extension exists in `prisma/migrations/` migration SQL file `prisma/migrations/enable_uuid_ossp.sql` - (M) (1 hours)[FE][BE]
    - QA: Create Node.js seed runner `scripts/run-seeds.ts` that executes `prisma/seed/seed_project_upload_presets.sql` and reports results - (M) (1 hours)[FE][BE][QA]
    - QUALITY: Add test `tests/seed/seed_project_upload_presets.test.ts` to verify idempotency and UUID validity using query to DB - (M) (1 hours)[FE][BE][QA]
- **Add Remix GET/PUT handlers with auth, validation, normalization hooks:** As a: backend developer, I want to: add Remix GET/PUT handlers with auth, validation, normalization hooks, So that: endpoints are secure and correctly process inputs**(7 hours)** - GET/PUT endpoints respond with correct

status codes Auth middleware validates tokens Validation hooks enforce input shapes Normalization hooks transform inputs consistently Error handling paths covered by tests

    - API: Create route file GET/PUT in `apps/api/routes/resource.ts` - (M) (1 hours)[FE][BE]
    - Auth: Add token validation middleware in `apps/api/middleware/auth.ts` - (M) (1 hours)[FE][BE][QA]
    - Validation: Implement Zod validation hooks in `apps/api/hooks/validation.ts` - (M) (1 hours)[FE][BE][QA]
    - Normalization: Add normalization hooks in `apps/api/hooks/normalization.ts` - (M) (1 hours)[FE][BE]
    - Service: Implement AuthService.validateToken in `apps/api/services/auth/AuthService.ts` - (M) (1 hours)[FE][BE]
    - Utils: Add response helpers in `apps/api/utils/response.ts` - (M) (1 hours)[FE][BE]
    - Tests: Add integration tests in `apps/api/tests/resource.test.ts` covering GET/PUT and error paths - (M) (1 hours)[FE][BE][QA]

- **Add migration name: 20251030_create_project_upload_presets:** As a: devops engineer, I want to: add a descriptive migration name reflecting its purpose, So that: future maintenance can identify the change easily**(7 hours)** - Migration filename follows timestamp convention Migration header includes description Migration can be applied in test env without conflicts Rollback script exists and matches up/down behavior Documentation notes reference migration name
    - DB: Create migration folder and SQL in `apps/api/prisma/migrations/20251030_create_project_upload_presets/migration.sql` with header ‘Create project_upload_presets’ and timestamp filename - (M) (1 hours)[FE][BE]

    - DB: Implement up script in `apps/api/prisma/migrations/20251030_create_project_upload_presets/migration.sql` to create table project_upload_presets using uuid_generate_v4() from public schema - (M) (1 hours)[FE][BE]
    - DB: Implement down script in `apps/api/prisma/migrations/20251030_create_project_upload_presets/migration.sql` to drop table project_upload_presets - (M) (1 hours)[FE][BE]
    - Test: Add migration apply test in `apps/api/tests/migrations/apply_migration.test.ts` to run migration in test DB - (M) (1 hours)[FE][BE][QA]
    - Test: Add rollback test in `apps/api/tests/migrations/rollback_migration.test.ts` to validate down behavior - (M) (1 hours)[FE][BE][QA]
    - Seed: Create seed SQL in `apps/api/prisma/seed/seed_project_upload_presets.sql` using uuid_generate_v4() in public schema - (M) (1 hours)[FE][BE]
    - Docs: Update `docs/migrations.md` to reference migration name 20251030_create_project_upload_presets - (M) (1 hours)[FE][BE]
- **Add seed script: seed_project_upload_presets.sql (populate bovine presets):** As a: data engineer, I want to: populate bovine presets in seed script, So that: bovine presets are available for testing**(6 hours)** - Seed includes bovine presets with UUIDs Validation ensures bovine presets count matches expectation Seed idempotent on re-run No duplicate entries on re-run Data integrity checks for preset fields
    - DB: Create SQL seed file in `apps/api/prisma/migrations/seed_project_upload_presets.sql` with INSERT … WHERE NOT EXISTS and uuid_generate_v4() - (M) (1 hours)[FE][BE]
    - DB: Add extension check in `apps/api/prisma/migrations/seed_project_upload_presets.sql` for uuid-ossp (CREATE EXTENSION IF NOT EXISTS uuid-ossp) and permission note - (M) (1 hours)[FE][BE]

    - Script: Implement Node seed runner in `apps/api/prisma/seed/seed_presets.ts` that runs SQL idempotently and logs results - (M) (1 hours)[FE][BE]
    - Test: Add validation test in `tests/seed_presets.test.ts` to assert bovine presets count and no duplicates by UUID - (M) (1 hours)[FE][BE][QA]
    - Data: Define expected bovine presets list with UUIDs in `apps/api/prisma/seed/expected_bovine_presets.json` for integrity checks - (M) (1 hours)[FE][BE]
    - Docs: Update `migrations/README.md` with seed usage, idempotency guarantee, and rollback notes - (M) (1 hours)[FE][BE]
- **Add Index:** As a: database administrator, I want to: add an index on preset_id and project_id to speed up lookups, So that: read queries for presets are efficient during project operations**(6 hours)** - Index on preset_id created Index on project_id created Query performance improvement observed for common read patterns by at least 2x in test
    - DB: Create Prisma migration adding index on preset_id in `prisma/migrations/` (file: prisma/migrations/20251030_add_index_preset_id/README.md)` - (M) (1 hours)[FE][BE]
    - DB: Create Prisma migration adding index on project_id in `prisma/migrations/20251030_add_index_project_id/README.md)` - (M) (1 hours)[FE][BE]
    - API: Add migration runner update in `apps/api/prisma/index.ts` to apply new migrations on deploy - (M) (1 hours)[FE][BE][DevOps]
    - API: Update query in `apps/api/services/presets/PresetsService.ts` to use indexed fields (add explicit order/filter on preset_id/project_id) - (M) (1 hours)[FE][BE]
    - Testing: Add integration test measuring read performance in `apps/api/tests/integration/presets/index_performance.test.ts` - (M) (1 hours)[FE][BE][QA]

    - Docs: Document index changes in `docs/db/indexes.md` with before/after query plans - (M) (1 hours)[FE][BE]

- **Create Table:** As a: database administrator, I want to: create the table structure for project upload presets, So that: the system can store preset configurations for project uploads and be queryable by the application**(3 hours)** - Table exists in database Columns for preset_id, project_id, preset_data (JSON), created_at, updated_at are defined Basic constraints and data types validated (primary key, not null)

    - DB: Add Presets model to prisma/schema.prisma with preset_id, project_id, preset_data Json, created_at, updated_at preserving architecture references and Prisma usage; ensure fields types and relations compatible with project_upload_presets concept - (S) (0.5 hours)[FE][BE]

    - DB: Create migration in prisma/migrations/ to create table project_upload_presets with columns matching the Presets model and proper foreign key to projects; includes up/down migration scripts - (XS) (0.5 hours)[FE][BE]

    - API: Implement DB access methods in apps/api/services/presets/PresetsService.ts (create/get) using Prisma client preserving service-layer architecture; ensure type-safe DTOs - (S) (0.5 hours)[FE][BE]

    - API: Add router endpoints in apps/api/routers/presets.py to expose presets CRUD using existing controller patterns; ensure authentication middleware and validations - (S) (0.5 hours)[FE][BE][QA]

    - TEST: Add integration tests in apps/api/tests/presets.test.ts validating table columns and constraints including not-null, foreign keys, and JSON field constraints - (M) (1 hours)[FE][BE][QA]

    - DOC: Update docs/database.md with schema and migration details for project_upload_presets table including column definitions, indexes, and migration references - (XS) (0.5 hours)[FE][BE]

- **Seed Defaults:** As a: database administrator, I want to: seed default presets into the table, So that: new projects have sensible defaults without manual setup**(5 hours)** - Default records inserted No duplicates on re-seed Defaults satisfy business rules (e.g., non-null presets)
    - DB: Add presets table migration in prisma/migrations/ to ensure defaults columns non-null - (M) (1 hours)[FE][BE]
    - DB: Create seed script prisma/seed/presetsSeed.ts inserting default presets idempotently - (M) (1 hours)[FE][BE]
    - API: Implement seedPresets() in apps/api/services/presets/PresetsService.ts with upsert logic - (M) (1 hours)[FE][BE]
    - API: Add dev-only route apps/api/routes/dev/presetsSeedRoute.ts to trigger seedPresets() - (S) (0.5 hours)[FE][BE]
    - Testing: Add integration tests in apps/api/tests/presets/seed.test.ts verifying defaults inserted and no duplicates - (M) (1 hours)[FE][BE][QA]
    - Documentation: Update README section in apps/api/README.md describing seed process and file prisma/seed/presetsSeed.ts - (S) (0.5 hours)[FE][BE]
- **Add Index: (0 hours)**
- **Seed Defaults (Per Project): (0 hours)**

Milestone 3: Seeding: Prisma/manual seed scripts and Seed APIs to create bovine default presets per project (uses uuid_generate_v4())

Estimated 791 hours

- **Remove gen_random_uuid() usage:** As a: Seeder maintainer, I want to: Remove gen_random_uuid() usage, So that: UUID generation relies on standard public schema UUIDs for consistency**(5 hours)** - gen_random_uuid() calls are removed from the Prisma seed script All UUIDs are generated

using public.uuid_generate_v4() or provided UUIDs Seed runs without errors and outputs generated UUIDs to log for verification

    - Code: Replace gen_random_uuid() calls in `prisma/seed/seed_project_upload_presets.ts` with public.uuid_generate_v4() or provided UUIDs - (M) (1 hours)[FE][BE]
    - DB: Add uuid-ossp extension and public.uuid_generate_v4() wrapper in `prisma/migrations/` SQL migration file - (M) (1 hours)[FE][BE]
    - Script: Log generated UUIDs in `scripts/seed.ts` to stdout for verification - (M) (1 hours)[FE][BE]
    - Test: Create integration test to run `scripts/seed.ts` and assert no errors and UUID outputs in `tests/seed/seed.test.ts` - (M) (1 hours)[FE][BE][QA]
    - Docs: Update `README.md` seed instructions to mention uuid-ossp and public.uuid_generate_v4() usage - (M) (1 hours)[FE][BE]

- **Manual run per project:** As a: Developer, I want to: Manually seed defaults per project, So that: Each project gets appropriate preset defaults without global apply**(7 hours)** - Manual seed executed for each project individually Defaults align with project type and preset requirements Run completes with log output for traceability

    - Script: Create seed script in `prisma/seed/seed_project_upload_presets.ts` to accept project UUIDs and use gen_random_uuid() when missing - (M) (1 hours)[FE][BE]
    - CLI: Add runner in `scripts/seed/run_project_seed.ts` to invoke `prisma/seed/seed_project_upload_presets.ts` per project ID - (M) (1 hours)[FE][BE]
    - Service: Add DB helper in `apps/api/services/db/PrismaClient.ts` to expose gen_random_uuid() integration and transaction wrapper - (M) (1 hours)[FE][BE]

    - Config: Create defaults mapping in `prisma/seed/project_defaults.ts` for project types and presets - (M) (1 hours)[FE][BE][DevOps]
    - Logging: Implement seed logger in `scripts/seed/logger.ts` to write run output to `logs/seed.log` - (M) (1 hours)[FE][BE]
    - Docs: Add manual run instructions to `docs/seeding.md` with examples and required project IDs - (M) (1 hours)[FE][BE]
    - Test: Create integration test in `tests/seed/test_seed_project.ts` to run seed for a sample project and verify defaults - (M) (1 hours)[FE][BE][QA]
- **Use public schema defaults:** As a: seed maintainer, I want to: apply defaults from the public schema when seeding, So that: default values are consistently applied across projects without custom overrides.**(5.5 hours)** - Defaults from public schema are applied for all relevant fields No errors when applying defaults across multiple projects Default values are consistent with schema definitions Seeding does not alter non-defaulted fields unless necessary
    - DB: Add public schema defaults migration in prisma/migrations/ to set DEFAULTs using gen_random_uuid() where needed - (L) (2 hours)[FE][BE]
    - API: Update seed_project_upload_presets.ts in prisma/seeders/seed_project_upload_presets.ts to read defaults from public schema and use DEFAULTs when inserting - (M) (1 hours)[FE][BE]
    - API: Implement helper applySchemaDefaults() in apps/api/services/seeds/SeedHelpers.ts to merge schema defaults into seed payloads - (M) (1 hours)[FE][BE]
    - Testing: Add integration tests in tests/seed/defaults.test.ts verifying defaults applied and non-default fields unchanged - (S) (0.5 hours)[FE][BE][QA]

    - Documentation: Update docs/seeding.md with default handling and multi-project usage examples - (M) (1 hours) [FE][BE]

- **Manual run with provided UUIDs:** As a: seed operator, I want to: run seed manually with provided UUIDs for project presets, So that: exact IDs are used for existing or specific projects.**(8 hours)** - Seed accepts a provided list of UUIDs and inserts corresponding records No-duplicate IDs are created; existing records with same IDs are preserved or updated per policy All inserts validate schema constraints (foreign keys, not-null) Operation logs clearly indicate which IDs were supplied and their origin
    - Setup: Add CLI script `scripts/seed/seed_project_upload_presets.ts` to accept UUID list and source flag - (M) (1 hours)[FE][BE][DevOps]
    - DB: Add/verify Prisma schema in `prisma/schema.prisma` for projects and ensure `table_users` FK constraints - (M) (1 hours)[FE][BE]
    - API: Implement SeedService.upsertWithUUIDs in `apps/api/services/seed/SeedService.ts` to accept UUIDs and perform upserts - (M) (1 hours)[FE][BE]
    - DB Script: Create seeder `prisma/seeders/seed_project_upload_presets.ts` that uses provided UUIDs or gen_random_uuid() - (M) (1 hours)[FE][BE]
    - Logging: Add operation logging in `apps/api/services/seed/SeedService.ts` to record supplied IDs and origin source - (M) (1 hours)[FE][BE]
    - Validation: Add FK and not-null validation checks in `prisma/seeders/seed_project_upload_presets.ts` and `apps/api/services/seed/SeedService.ts` - (M) (1 hours)[FE][BE][QA]
    - Test: Add integration tests in `tests/seed/seed_project_upload_presets.test.ts` covering duplicates, existing IDs, and constraint violations - (M) (1 hours)[FE][BE][QA]

    - Docs: Create `scripts/seed/README.md` documenting usage and UUID provenance logging - (M) (1 hours)[FE] [BE]

- **Seed uses public uuid_generate_v4():** As a: Database seeder, I want to: Seed with public uuid_generate_v4() functionality, So that: IDs are generated using the public schema for seeding defaults.**(6 hours)** - Public uuid_generate_v4() is used for at least one inserted default ID No collisions occur when seeding across multiple runs seeding script runs without errors in local dev environment

    - DB: Add migration to enable uuid-ossp extension in `prisma/migrations/2025xxxx_enable_uuid_ossp/steps.sql` - (M) (1 hours)[FE][BE]

    - DB: Add unique default ID constraint for uploads in `prisma/schema.prisma` - (M) (1 hours)[FE][BE]

    - Seed: Update `prisma/seed.ts` to use provided UUIDs or call public.uuid_generate_v4() via prisma. $executeRaw`INSERT … VALUES (public.uuid_generate_v4(), …)` - (M) (1 hours)[FE][BE]

    - Infra: Ensure local Docker Postgres enables extension in `docker/docker-compose.yml` and `docker/initdb.d/ enable_uuid.sh` - (M) (1 hours)[FE][BE]

    - Test: Add idempotent seed test in `scripts/seed.test.ts` to run seed twice and check for no collisions using `table_users` - (M) (1 hours)[FE][BE][QA]

    - Docs: Add seeding instructions in `prisma/README.md` showing use of public.uuid_generate_v4() and local setup - (M) (1 hours)[FE][BE][DevOps]

- **Manual run script:** As a: Devops engineer, I want to: Run the Prisma seed script manually, So that: Defaults are created for provided project IDs in a controlled manner**(6 hours)** -

Manual execution of seed script completes without errors Provided project IDs result in created defaults Logs show seeded IDs and timestamp

  - DB: Add uuid-ossp extension migration in `prisma/migrations/` - (M) (1 hours)[FE][BE]
  - DB: Add projects default data migration in `prisma/migrations/` - (M) (1 hours)[FE][BE]
  - Script: Implement seed logic in `prisma/seeders/seed_project_upload_presets.ts` to accept provided project UUIDs or use uuid_generate_v4() - (M) (1 hours)[FE][BE]
  - Script: Create runner `scripts/run-seed.ts` to execute `prisma/seeders/seed_project_upload_presets.ts` manually and log results - (M) (1 hours)[FE][BE]
  - Logging: Implement logger in `prisma/seeders/logging.ts` to output seeded IDs and timestamp - (M) (1 hours)[FE][BE]
  - Test: Add manual run checklist in `docs/manual-seed.md` with example UUIDs and expected outputs - (M) (1 hours)[FE][BE][QA]

- **Insert bovine defaults:** As a: Admin, I want to: Insert bovine defaults into seed data, So that: System has realistic sample data for testing and demos**(5 hours)** - Bovine defaults present for all required fields Seed sequence remains deterministic Data integrity checks pass (no nulls where prohibited)
  - DB migration to add bovines table under prisma/migrations with required columns: name (text), breed (text), age (int), id (uuid primary key). Include migration up/down scripts and ensure default values respect NOT NULL constraints where applicable; integrate with existing Prisma setup and database baseline. - (M) (1 hours)[FE][BE][DevOps]
  - API: Extend Prisma client usage by adding Bovine model to prisma/schema.prisma and generate client; ensure type-safe access for create/read operations in seed path; align with 4_1 table schema. - (S) (0.5 hours)[FE][BE]

    - API: Implement deterministic UUID helper in apps/api/utils/seedUtils.ts using pgcrypto gen_random_uuid() as primary generator with fallback to crypto UUID v5 or random UUID to guarantee deterministic seeds per environment. - (S) (0.5 hours)[FE][BE][DevOps]
    - API: Implement seed file prisma/seed/bovineDefaults.ts to insert bovine default records using explicit non-null fields (id, name, breed, age); no nullable columns; export a function to run seed with Prisma client. - (S) (0.5 hours)[FE][BE]
    - API: Update apps/api/seed/seed_project_upload_presets.ts to import and execute prisma/seed/bovineDefaults.ts during project seed initialisation; ensure order respects dependencies and error handling. - (S) (0.5 hours)[FE][BE]
    - Quality: Add integration test tests/seed/bovineDefaults.test.ts to verify deterministic sequence and no nulls; test environment should mock Prisma client or run against test DB; verify order and data integrity. - (M) (1 hours)[FE][BE][DevOps][QA]
    - Quality: Add data integrity check script scripts/check_bovine_integrity.py to scan DB for NULLs and schema violations in bovines table; run as part of CI or nightly checks. - (S) (0.5 hours)[FE][BE]
    - Documentation: Update docs/seeding.md with bovine defaults seed instructions and deterministic UUID behavior; include usage examples and caveats. - (XS) (0.5 hours)[FE][BE]
- **Insert bovine defaults:** As a: Admin, I want to: Insert bovine defaults into presets, So that: Default configurations include bovine-related presets for cattle projects**(7 hours)** - Bovine

defaults are present in presets after seed No bovine presets duplicate on re-seed Seed reports bovine defaults in summary log

- DB: Add bovine presets to `apps/api/prisma/seed/seed_project_upload_presets.ts` with preset data and UUID handling - (M) (1 hours)[FE][BE]
- DB: Create migration in `apps/api/prisma/migrations/` to ensure presets table schema supports uuid-ossp and unique constraint on (name, projectId) - (M) (1 hours)[FE][BE]
- API: Implement upsert logic in `apps/api/prisma/seed/utils.ts` to avoid duplicate bovine presets - (M) (1 hours)[FE][BE]
- API: Update `apps/api/prisma/seed/seed_project_upload_presets.ts` to call `utils.upsertPreset()` and aggregate results - (M) (1 hours)[FE][BE]
- QA: Add integration test `apps/api/tests/seed/seed_bovine.test.ts` to run seed twice and assert no duplicates and presence - (M) (1 hours)[FE][BE][QA]
- Docs: Add seed summary logging and artifact note in `docs/seed.md` - (M) (1 hours)[FE][BE]
- Infra: Add CI job in `.github/workflows/seed.yml` to run seed and collect summary log artifact - (M) (1 hours)[FE][BE]

- **Add seed for all projects: (24 hours)**
  - DB: Add users migration in `prisma/migrations/` to ensure seed targets (table_users) (4 hours)[FE][BE]
  - DB: Create seed entrypoint `prisma/seed/index.ts` to run project seeds (4 hours)[FE][BE]
  - DB: Implement seed logic in `prisma/seed/seed_project_upload_presets.ts` to seed all projects (4 hours)[FE][BE]

    - Config: Add npm script in `package.json` to run `prisma/seed/index.ts` for all projects (4 hours)[FE][BE][DevOps]
    - Frontend: Add docs for manual runner in `apps/frontend/src/routes/seed/runner/manual/README.md` referencing route_seed_runner_manual (4 hours)[FE][BE]
    - CI: Add GitHub Actions workflow `/.github/workflows/seed.yml` to run seed in CI optional (4 hours)[FE][BE]
- **Seed single project: (28 hours)**
    - DB: Create Prisma seed file `prisma/seeders/seed_project_upload_presets.ts` (4 hours)[FE][BE]
    - Script: Implement CLI runner in `scripts/seed/runSeed.ts` to call seeder for single project (4 hours)[FE][BE]
    - API: Add seed endpoint in `apps/api/src/routes/seed.ts` to trigger single-project seed (4 hours)[FE][BE]
    - Service: Implement seeding logic in `apps/api/src/services/seedService.ts` to invoke `prisma/seeders/seed_project_upload_presets.ts` (4 hours)[FE][BE]
    - Frontend: Build UI at `apps/web/src/routes/seed/SeedRunnerManual.tsx` for selecting project and triggering seed (route_seed_runner_manual) (4 hours)[FE][BE]
    - Testing: Add integration test in `apps/api/tests/seed.test.ts` for single-project seed (4 hours)[FE][BE][QA]
    - Docs: Add run instructions in `docs/seed.md` and update `README.md` (4 hours)[FE][BE]
- **Idempotent insert check: (20 hours)**
    - DB: Add unique constraint and migration for upload_presets in `prisma/migrations/20251030_add_upload_presets_unique/` (4 hours)[FE][BE]
    - API: Implement idempotent upsert utility in `apps/api/utils/idempotent.ts` (4 hours)[FE][BE]

    - Script: Update `prisma/seed/seed_project_upload_presets.ts` to use `apps/api/utils/idempotent.ts` for idempotent inserts (4 hours)[FE][BE]
    - Test: Add integration test in `apps/api/tests/seed.test.ts` asserting idempotent behavior when running seed script twice (4 hours)[FE][BE][QA]
    - Doc: Update `/seed/runner/manual` docs in `docs/seed_runner_manual.md` with idempotency notes and usage (4 hours)[FE][BE]
- **Docs + run instructions: (28 hours)**
    - Docs: Create top-level README with run instructions in `README.md` (4 hours)[FE][BE]
    - Docs: Add seed runner page docs in `apps/web/src/routes/seed/runner/manual/README.md` (4 hours)[FE][BE]
    - Docs: Document Prisma seed script usage in `prisma/seed/seed_project_upload_presets.ts` (4 hours)[FE][BE]
    - Docs: Add API usage notes in `apps/api/README.md` referencing route_seed_runner_manual (4 hours)[FE][BE]
    - Run: Create npm script `run-seed-manual` in `package.json` to run prisma/seed/seed_project_upload_presets.ts` (4 hours)[FE][BE]
    - Run: Add Docker run example in `docker/seed/README.md` showing how to run seed inside `apps/api` container (4 hours)[FE][BE]
    - CI: Add GitHub Actions job in `.github/workflows/seed.yml` to run seed script (for docs preview) (4 hours)[FE][BE]

- **Create seed per project: (40 hours)** - Seed data file is generated for 100% of selected projects Seed data includes required fields and sample records per project System handles project-specific configuration and validation before write Seed generation completes within 2 minutes for up to 50 projects
    - API: Add runSeedsByIds mutation in `apps/api/routers/seed_runner.py` to accept project IDs and options (4 hours)[FE][BE]
    - API: Implement createProjectSeed in `apps/api/routers/seed_runner.py` to validate project config before queuing (4 hours)[FE][BE][DevOps]
    - Backend: Implement seed worker task in `apps/api/tasks/seed_worker.py` to generate seed file per project using Pandas (4 hours)[FE][BE]
    - Backend: Add project-specific validation module in `apps/api/services/seed/validation.py` (4 hours)[FE][BE][QA]
    - Storage: Save generated seed to S3 in `apps/api/services/seed/storage.py` and record metadata (4 hours)[FE][BE]
    - Frontend: Add ProjectInput wiring in `apps/web/src/routes/seed/script/runner/SeedRunnerProjectInput.tsx` to submit selected IDs (4 hours)[FE][BE]
    - Frontend: Show seed generation status in `apps/web/src/routes/seed/script/runner/SeedRunnerMainPanel.tsx` using fetchSeedStatus and subscriptions (4 hours)[FE][BE]
    - DB: Create project_seeds table migration in `prisma/migrations/` to store seed metadata and status (4 hours)[FE][BE]
    - Testing: Add integration tests in `apps/api/tests/test_seed_generation.py` to cover up to 50 projects within 2 minutes (4 hours)[FE][BE][QA]
    - Docs: Update README section in `docs/seed_runner.md` with usage, config, and acceptance criteria mapping (4 hours)[FE][BE][DevOps]

- **Run seed for IDs:** As a: automation engine, I want to: run seed scripts for specified IDs, So that: targeted seeds can be executed in isolation without affecting others**(36 hours)** - IDs provided are validated against project registry Seed script executes successfully for each ID Logs show per-ID execution status and errors are captured No partial seeds are left in inconsistent state after run
    - API: Add runSeedsByIds mutation in `apps/api/routers/seed_runner.py` (4 hours)[FE][BE]
    - API: Implement validateSeedIds in `apps/api/services/seed/SeedValidator.py` (4 hours)[FE][BE]
    - Service: Create SeedExecutor with transactional per-ID run in `apps/api/services/seed/SeedExecutor.py` (4 hours)[FE][BE]
    - Service: Add ExecutionLog model and persistence in `apps/api/services/seed/ExecutionLogModel.py` (4 hours)[FE][BE]
    - Task: Create Celery task wrapper in `apps/api/tasks/seed_tasks.py` to run seeds asynchronously (4 hours)[FE][BE]
    - Frontend: Update SeedRunner ExecutionPanel in `components/seed/SeedRunnerExecutionPanel.tsx` to call runSeedsByIds (4 hours)[FE][BE]
    - Frontend: Display per-ID logs in `components/seed/SeedRunnerResultsLog.tsx` using fetchExecutionLogs query (4 hours)[FE][BE]
    - Testing: Add integration tests in `tests/api/test_seed_runner.py` for validate+execute+logs (4 hours)[FE][BE][QA]
    - Docs: Update `docs/seed_runner.md` with usage and failure modes (4 hours)[FE][BE]

- **Auto-run per project:** As a: product workflow, I want to: auto-run seed per project based on repository triggers, So that: seeds are kept up-to-date on project changes**(40 hours)** - Trigger configured per project or repo Seed run is triggered automatically on relevant events Run completes within defined SLA Audit trail of auto-runs with timestamps and results
    - API: Add setAutoRunForProject endpoint in `apps/api/routers/seed_runner.py` (4 hours)[FE][BE]
    - DB: Create project_auto_runs migration in `prisma/migrations/` to store auto-run config and logs (4 hours)[FE][BE][DevOps]
    - API: Implement webhook handler for repo triggers in `apps/api/routers/seed_runner.py` (4 hours)[FE][BE]
    - Service: Implement auto-run scheduler in `apps/api/services/seed/AutoRunService.py` (4 hours)[FE][BE]
    - Frontend: Add Auto-Run Triggers Panel component in `apps/web/src/components/seed/AutoRunTriggersPanel.tsx` (4 hours)[FE][BE]
    - API: Add fetchProjectTriggers and fetchTriggerLogs queries in `apps/api/routers/seed_runner.py` (4 hours)[FE][BE]
    - Infra: Configure GitHub Actions integration in `.github/workflows/seed-auto-run.yml` to POST to webhook (4 hours)[FE][BE][DevOps]
    - Service: Integrate Celery task to run seeds in `apps/api/services/seed/tasks.py` (4 hours)[FE][BE]
    - API: Implement setAutoRunConfig mutation in `apps/api/routers/seed_runner.py` (4 hours)[FE][BE][DevOps]
    - Testing: Add integration tests for webhook and auto-run in `apps/api/tests/test_auto_run.py` (4 hours)[FE][BE][QA]
- **Ensure idempotent seed:** As a: automation engineer, I want to: ensure seed runs are idempotent, So that: repeated runs do not corrupt data or add duplicate records**(28 hours)** - Seed

operations check for existing data before insert Duplicate records are avoided with idempotency keys Seed state can be rolled back safely if needed Tests cover repeated seed runs with same input

  - DB: Add table_seed_runs migration in `prisma/migrations/` to store idempotency keys and run state (4 hours)[FE][BE]
  - DB: Add unique constraints to `prisma/migrations/` for target tables to prevent duplicates (project_upload_presets_mappings/ preset_unit_normalizations) (4 hours)[FE][BE]
  - API: Implement idempotent seed logic in `apps/api/services/seed/SeedService.py` (check seed runs, use idempotency keys, support rollback) (4 hours)[FE][BE]
  - API: Update router in `apps/api/routers/seed_router.py` to use SeedService idempotent startSeed and resetSeed endpoints (4 hours)[FE][BE]
  - Test: Add integration tests for repeated seed runs in `tests/seed/test_seed_idempotency.py` covering inserts, duplicates, and rollback (4 hours)[FE][BE][QA]
  - Infra: Add CI job `/.github/workflows/seed.yml` to run seed tests and teardown DB (4 hours)[FE][BE][QA]
  - Docs: Document idempotent seed behavior in `docs/seed-runner.md` and update `apps/web/src/components/seed/SeedRunner.tsx` comments (4 hours)[FE][BE]

- **Validate preset creation:** As a: QA engineer, I want to: validate that seed presets are created correctly, So that: presets can be reused for consistent seeds**(36 hours)** - Preset definitions are stored with versioning Validation checks ensure required fields exist in presets Presets can be retrieved and applied to seed run without errors Corrupt presets are rejected with meaningful error messages
  - DB: Add presets versioning columns migration in `prisma/migrations/` (4 hours)[FE][BE]

    - DB: Create preset schemas for intervals/mappings in `prisma/migrations/` referencing project_upload_presets_intervals and project_upload_presets_mappings (4 hours)[FE][BE]
    - API: Implement validatePreset endpoint in `apps/api/routers/seed_runner/router.py` to call validation util (router_route_seed_script_runner) (4 hours)[FE][BE][QA]
    - API: Implement getPreset and listPresets in `apps/api/routers/seed_runner/router.py` (router_route_seed_script_runner) (4 hours)[FE][BE]
    - Service: Add preset validation util in `apps/api/services/presets/validation.py` to check required fields and versioning (4 hours)[FE][BE][QA]
    - Service: Implement applyPresetToRun in `apps/api/services/presets/service.py` to load preset and apply to run (used by runPreset) (router_route_seed_script_runner) (4 hours)[FE][BE]
    - Frontend: Update Presets & Validation Panel component in `apps/web/src/components/presets/PresetsPanel.tsx` to show validation errors (comp_seed_runner_presets_panel) (4 hours)[FE][BE][QA]
    - Tests: Add unit tests for validation util in `apps/api/tests/test_presets_validation.py` (testing) (4 hours)[FE][BE][QA]
    - Tests: Add integration tests for validatePreset and getPreset in `apps/api/tests/test_seed_runner_api.py` (router_route_seed_script_runner) (4 hours)[FE][BE][QA]

- **Run seed for IDs: (0 hours)**
- **Validate preset creation: (0 hours)**
- **Create seed for projects: (0 hours)**
- **Run seed for IDs: (0 hours)**
- **Validate preset creation: (0 hours)**

- **Idempotent seed run: (0 hours)**
- **Auto-run seed: (0 hours)**
- **Validate preset creation: (0 hours)**
- **Ensure idempotent seed: (0 hours)**
- **Log seed results: (0 hours)**
- **Validate Admin Auth:** Validate that the request is from an admin user before proceeding with seed operation to ensure proper authorization.**(29 hours)** - Given a request, system verifies admin credentials and role. If not admin, operation is rejected with proper error code (401/403). On success, request proceeds to seed logic without leaking security details. Audit log entry is created for admin authentication attempt.
    - DB: Create prisma migration for user roles and sessions in prisma/migrations/20251030_add_roles_sessions/ - (M) (1 hours)[FE][BE]
    - API: Implement validateAdminToken() in `apps/api/services/auth/AuthService.ts` (4 hours)[FE][BE]
    - API: Add adminAuth middleware in `apps/api/middleware/adminAuth.ts` to enforce role check and error codes (4 hours)[FE][BE]
    - API Router: Integrate middleware into `apps/api/routes/project/upload-presets/seed.ts` (router_route_project_upload_presets_seed) (4 hours)[FE][BE]
    - API: Implement audit log creation in `apps/api/services/audit/AuditService.ts` for admin auth attempts (4 hours)[FE][BE]
    - Frontend: Update AdminAuthValidator component in `apps/web/components/project/upload/AdminAuthValidator.tsx` (comp_project_upload_presets_auth) (4 hours)[FE][BE]

    - Tests: Add integration tests in `apps/api/tests/seed_auth.test.ts` covering 401/403 and success flow (4 hours)[FE][BE][QA]
    - Docs: Add docs in `docs/seed_auth.md` and checklist in `docs/code_review/seed_auth_checklist.md` (4 hours)[FE][BE]
- **Idempotent Upsert Preset (non-destructive):** Upsert presets for projects in an idempotent manner to avoid duplicate presets and ensure non-destructive updates.**(9 hours)** - Upsert operation creates preset if missing. If preset exists, system updates it without duplicating records. The operation is atomic to prevent partial updates. System logs changes for traceability.
    - DB: Add unique constraint and migration for presets in `prisma/migrations/` - (M) (1 hours)[FE][BE]
    - API: Add POST handler upsert endpoint in `apps/api/routers/seed/seedRouter.py` to call idempotent upsert - (M) (1 hours)[FE][BE]
    - Service: Implement upsertPresetIdempotent(projectId, presetData) in `apps/api/services/presets/PresetService.ts` - (M) (1 hours)[FE][BE]
    - DB: Implement atomic transaction logic in `apps/api/services/presets/PresetService.ts` using Prisma client in `prisma/client` - (M) (1 hours)[FE][BE]
    - Logging: Add change logging to `apps/api/services/presets/PresetService.ts` writing to `apps/api/logs/preset_changes.log` and DB audit table via `table_preset_unit_normalizations` - (M) (1 hours)[FE][BE]
    - API: Wire router_route_project_upload_presets_seed upsertPresetIdempotent operation in `apps/api/routers/project_upload_presets_seed.py` - (M) (1 hours)[FE][BE]

    - Tests: Add unit tests for PresetService.upsertPresetIdempotent in `apps/api/tests/services/presets/test_preset_service.ts` - (M) (1 hours)[FE][BE][QA]
    - Integration: Add integration test for POST /api/projects/upload-presets/seed in `apps/api/tests/integration/test_seed_endpoint.py` - (M) (1 hours)[FE][BE][QA]
    - Frontend: Update PresetUpsertExecutor component in `apps/web/components/project/upload/PresetsUpsertExecutor.tsx` to call POST /api/projects/upload-presets/seed - (M) (1 hours)[FE][BE][DevOps]
- **Seed Presets per Project (auto): (32 hours)** - Given a list of project IDs, system seeds a preset per project and returns a success status for each. Edge case: if a project already has a preset, system returns a no-op for that project but still reports status. System validates input array is non-empty and contains valid IDs. Response includes per-project seed results and a summary. Users can retry failed seeds with idempotent error handling
    - API: Add POST /api/projects/upload-presets/seed handler in `apps/api/routers/seed_api.py` (4 hours)[FE][BE]
    - Service: Implement seed_presets_for_projects(project_ids: List[int (4 hours)[FE][BE]
    - DB: Add idempotent check and preset create logic in `apps/api/repositories/presets_repo.py` using `table_preset_unit_normalizations` (4 hours)[FE][BE]
    - Task: Create Celery task `apps/api/tasks/seed_tasks.py` -> seed_project_preset(project_id) with retry and idempotency (4 hours)[FE][BE]
    - Validation: Add request validator for non-empty array and ID schema in `apps/api/schemas/seed_schemas.py` and wire in `apps/api/routers/seed_api.py` (4 hours)[FE][BE][QA]

    - Response: Build per-project result DTO and summary in `apps/api/schemas/seed_response.py` and return from `apps/api/routers/seed_api.py` (4 hours)[FE][BE]
    - Tests: Add unit + integration tests in `apps/api/tests/test_seed_api.py` covering existing preset no-op, success, validation, retry (4 hours)[FE][BE][QA]
    - Docs: Add API docs and OpenAPI examples in `apps/api/docs/seed_api.md` and update route_project_upload_presets_seed (4 hours)[FE][BE]
- **Return Operation Summary:** Return a concise summary of seed operation results for all projects.**(5.5 hours)** - Response includes total projects processed. Number of successes and failures per project. List of failed projects with error messages. Response timestamp and request metadata for traceability.
    - DB: Create operation_summary migration in prisma/migrations/ to store totals, successes, failures, errors, timestamp, request_meta - (S) (0.5 hours)[FE][BE]
    - API: Implement OperationSummaryService.computeSummary() in apps/api/services/seed/OperationSummaryService.ts to aggregate results and format response - (M) (1 hours)[FE][BE]
    - API: Add getOperationSummary handler in apps/api/routers/project/upload/presets/seed.ts (router_route_project_upload_presets_seed) to return summary DTO - (S) (0.5 hours)[FE][BE]
    - API: Add response DTO types in apps/api/models/operationSummary.ts including total, per-project successes/failures, failed list with errors, timestamp, request_meta - (S) (0.5 hours)[FE][BE]
    - Frontend: Build OperationSummaryResponder component in apps/web/components/project/upload/OperationSummaryResponder.tsx (comp_project_upload_presets_response) to display totals, per-project results, failed list, timestamp and request metadata - (S) (0.5 hours)[FE][BE]

    - API: Add unit tests for OperationSummaryService in apps/api/tests/operation_summary.test.ts covering totals, success/failure counts, and error listings - (M) (1 hours) [FE][BE][QA]
    - Frontend: Add component tests in apps/web/tests/OperationSummaryResponder.test.tsx to verify rendering of totals, failed projects and metadata - (S) (0.5 hours)[FE][BE][QA]
    - Infra: Add logging and request metadata capture in apps/api/middleware/requestLogger.ts and config in apps/api/config/logging.ts - (M) (1 hours)[FE][BE][DevOps]
- **Validate Admin Auth: (0 hours)**
- **Idempotent Upsert Preset: (0 hours)**
- **Return Operation Summary: (0 hours)**
- **Return Operation Summary: (0 hours)**
- **Admin Auth Check: (0 hours)**
- **Return Summary: (26 hours)**
    - DB: Create presets migration in `prisma/migrations/202510_seed_presets.sql` (4 hours)[FE][BE]
    - API: Add request/response schemas in `apps/api/schemas/seed.py` (4 hours)[FE][BE]
    - API: Implement SeedService.create_presets(project_ids) in `apps/api/services/seed/SeedService.py` (4 hours)[FE][BE]
    - API: Implement POST /api/projects/upload-presets/seed handler in `apps/api/routes/project_upload_presets_seed.py` (4 hours)[FE][BE]
    - Infra: Add cache update in `apps/api/services/cache/CacheService.py` (4 hours)[FE][BE]
    - Quality: Add tests for seed flow in `tests/seed_test.py` (4 hours)[FE][BE][QA]

    - Docs: Document API in `docs/api/seed.md` (2 hours)[FE][BE]

- **Generate Files Only:** Only generate seed presets files without applying or returning per-project results.**(28 hours)** - Files generated for each project ID present in input. No persistence or state changes beyond file generation. Return list of generated file paths. Error handling for invalid IDs or generation failures.
    - API: Add route handler in `apps/api/routes/seed.py` for POST /api/projects/upload-presets/seed (4 hours)[FE][BE]
    - API: Implement SeedService.generate_files_for_project(project_id) in `apps/api/services/seed/SeedService.py` (4 hours)[FE][BE]
    - Validation: Implement validate_project_ids(ids) in `apps/api/services/seed/validation.py` with error handling for invalid IDs (4 hours)[FE][BE][QA]
    - IO: Create file writer create_preset_files(project_id, presets, out_dir) in `apps/api/services/seed/file_writer.py` to write to `apps/api/data/presets/{project_id}/` (4 hours)[FE][BE]
    - Orchestration: Add createSeedPresetFiles mutation call in `apps/api/routes/seed.py` to invoke `SeedService.generate_files_for_project` per project (4 hours)[FE][BE]
    - Aggregation: Implement aggregate_generated_paths(results) in `apps/api/services/seed/aggregator.py` to return list of generated file paths (4 hours)[FE][BE]
    - Testing/Docs: Add unit tests in `apps/api/tests/test_seed_service.py` and update docs in `docs/api/seed.md` (4 hours)[FE][BE][QA]

- **Admin Trigger Seed: (48 hours)** - Admin initiates seed for a list of project IDs Seed job is queued and status tracked per project Failure in seed for any project is reported with actionable error
    - API: Add POST /api/project/{id}/upload-presets/seed handler in `apps/api/routers/seed.py` (4 hours)[FE][BE]
    - API: Implement triggerSeed mutation in `apps/api/routers/seed.py` to accept list of project IDs and enqueue Celery tasks (4 hours)[FE][BE]
    - Backend: Create Celery task seed_project(project_id) in `apps/api/tasks/seed_tasks.py` to run seed and update status (4 hours)[FE][BE]
    - DB: Add seed_jobs table migration in `prisma/migrations/` to track job per project (4 hours)[FE][BE]
    - DB: Update Prisma model in `prisma/schema.prisma` to include SeedJob model referencing projects and table_preset_unit_normalizations (4 hours)[FE][BE]
    - API: Implement getSeedStatus endpoint in `apps/api/routers/seed.py` to return per-project job status (4 hours)[FE][BE]
    - Frontend: Build Action Panel component in `apps/web/src/components/seed/ActionPanel.tsx` (route_seed_api -> comp_seed_api_action_panel) to trigger seed for selected projects (4 hours)[FE][BE]
    - Frontend: Build Confirmation & Feedback component in `apps/web/src/components/seed/Confirmation.tsx` (route_seed_api -> comp_seed_api_confirmation) to show per-project status and errors (4 hours)[FE][BE]
    - Frontend: Add Redux slice in `apps/web/src/store/seedSlice.ts` to manage request state and statuses (4 hours)[FE][BE]

    - Frontend: Create hook useSeedStatus in `apps/web/src/hooks/useSeedStatus.ts` to poll getSeedStatus (route_seed_api -> comp_seed_request_panel) (4 hours)[FE][BE]
    - Tests: Add API tests in `apps/api/tests/test_seed.py` covering success, partial failure, and error messages (4 hours)[FE][BE][QA]
    - Tests: Add frontend tests in `apps/web/tests/seed.test.tsx` for ActionPanel and Confirmation components (4 hours)[FE][BE][QA]
- **Seed Project Presets: (32 hours)** - Admin can trigger seed for a project and receives confirmation Seed results are stored in the database and include project presets data System logs seed operation with timestamp and project id
    - DB: Create preset_unit_normalizations migration in `prisma/migrations/` (4 hours)[FE][BE]
    - DB: Create seed_logs table migration in `prisma/migrations/` (4 hours)[FE][BE]
    - API: Implement triggerSeedRequest handler in `apps/api/routers/seed.py` (4 hours)[FE][BE]
    - API: Implement SeedService.seed_project_presets in `apps/api/services/seed/SeedService.py` (4 hours)[FE][BE]
    - API: Add fetchSeedPresets query in `apps/api/routers/seed.py` (4 hours)[FE][BE]
    - Frontend: Add Seed button and confirmation UI in `apps/web/src/components/RequestPayloadPanel/SeedRequestPanel.tsx` (comp_seed_request_panel) (4 hours)[FE][BE]
    - DB: Create preset data seeder script in `apps/api/scripts/seed_presets.py` (4 hours)[FE][BE]
    - API: Log seed operation with timestamp in `apps/api/services/logging/LoggingService.py` (4 hours)[FE][BE]

- **Seed Error Handling:** As a: admin, I want to: handle seed errors gracefully, So that: I can recover quickly and maintain system stability**(28 hours)** - Error codes are returned clearly with actionable messages Retry mechanism exists with backoff policy Failed seeds trigger alerting and escalation to admin
    - DB: Create seed_errors table migration in `prisma/migrations/seed_errors/` (4 hours)[FE][BE]
    - API: Add structured error responses in `apps/api/routers/seed.py` for retrySeedResult and retrySeedProject (4 hours)[FE][BE]
    - Worker: Implement retry with exponential backoff in `apps/api/tasks/seed_worker.py` using Celery/Redis (4 hours)[FE][BE]
    - Alert: Implement AlertService in `apps/api/services/alerts/AlertService.py` to notify admins on failed seeds (4 hours)[FE][BE]
    - Frontend: Update ResultsSummary panel in `apps/web/components/seed/ResultsSummary.tsx` to display actionable error codes and retry UI (4 hours)[FE][BE]
    - Infra: Configure Celery/Redis in `apps/api/config/celery.py` and add GitHub Action `/.github/workflows/seed.yml` (4 hours)[FE][BE][DevOps]
    - Tests: Add unit/integration tests in `apps/api/tests/test_seed_errors.py` and `apps/web/tests/test_results_summary.tsx` (4 hours)[FE][BE][QA]
- **Validate Seed Results: (40 hours)** - Seed results are verifiable against expected presets Discrepancies trigger alert and remediation steps Audit log records seed validation outcomes
    - DB: Add audit_logs table migration in `apps/api/prisma/migrations/create_audit_logs` (4 hours)[FE][BE]

    - DB: Add preset_unit_normalizations seed verification fixtures in `apps/api/prisma/seeders/preset_unit_normalizations_seed.py` (4 hours)[FE][BE]
    - API: Implement validateSeedResult endpoint in `apps/api/routers/seed.py` (4 hours)[FE][BE]
    - API: Create ValidationService.validate_result in `apps/api/services/seed/ValidationService.py` (4 hours)[FE][BE][QA]
    - API: Integrate audit logging in `apps/api/services/seed/ValidationService.py` and `apps/api/models/audit.py` (4 hours)[FE][BE][QA]
    - API: Add alerting hook to Redis/Celery task in `apps/api/tasks/alerts.py` to notify discrepancies (4 hours)[FE][BE]
    - Frontend: Display validation summary in `apps/web/src/components/seed/ResultsSummary.tsx` (4 hours)[FE][BE][QA]
    - Frontend: Add confirm/remediation UI in `apps/web/src/components/seed/ResultsPanel.tsx` and `apps/web/src/components/seed/ConfirmationFeedback.tsx` (4 hours)[FE][BE]
    - Frontend: Add seedSlice actions in `apps/web/src/store/seedSlice.ts` to fetchValidationSummary and acknowledgeSeedResult (4 hours)[FE][BE][QA]
    - Testing: Add integration tests for validateSeedResult in `apps/api/tests/test_validation.py` (4 hours)[FE][BE][QA]
- **Admin Trigger Seed: (0 hours)**
- **Validate Seed Results: (0 hours)**
- **Seed Error Handling: (0 hours)**
- **Seed Error Handling: (0 hours)**
- **Validate UUIDs:** As a: data entry assistant, I want to: Validate the array of UUIDs provided for seed input, so that: Only valid project identifiers proceed to per-project preset creation and

mapping.**(32 hours)** - All input UUIDs are validated to proper UUIDv4 format Invalid UUIDs are rejected with a clear error message and no further processing occurs System records the invalid IDs for auditing and user feedback Validation occurs before any per-project processing and mapping Edge case: Empty input array results in a user-friendly prompt and no changes to state

  - API: Add seed UUIDs validation endpoint in `apps/api/routers/seed_input.py` (4 hours)[FE][BE][QA]
  - Service: Implement validate_uuid_array(input) in `apps/api/services/validation/UuidValidator.py` (4 hours)[FE][BE][QA]
  - DB: Create audit table migration for invalid_ids in `prisma/migrations/` (4 hours)[FE][BE]
  - DB: Implement record_invalid_ids(invalid_ids) in `apps/api/services/audit/AuditService.py` (4 hours)[FE][BE]
  - Frontend: Add input validation UI and empty-array prompt in `apps/web/components/SeedInputForm.tsx` (4 hours)[FE][BE][QA]
  - API: Integrate validation call in `apps/api/routers/seed_input.py` to gate per-project processing (4 hours)[FE][BE][QA]
  - Testing: Add unit tests for UuidValidator in `tests/services/test_uuid_validator.py` (4 hours)[FE][BE][QA]
  - Testing: Add integration test for seed_input endpoint in `tests/routers/test_seed_input.py` (4 hours)[FE][BE][QA]

- **Apply Preset Per Project: (32 hours)** - For each valid UUID, a per-project preset is created Intervals and mappings are applied according to predefined rules No data loss occurs during per-project preset creation Operation completes

successfully for all provided UUIDs or returns per-UUID error where applicable Audit trail records per-project preset creation details

    - DB: Create table_presets migration in `prisma/migrations/` and Prisma model in `prisma/schema.prisma` (4 hours)[FE][BE]
    - API: Implement POST /presets/apply in `apps/api/routers/presets.py` (4 hours)[FE][BE]
    - Service: Implement apply_presets_for_projects(uuid_list) in `apps/api/services/presets/PresetsService.py` (4 hours)[FE][BE]
    - Task: Add Celery task apply_presets_task in `apps/api/tasks/preset_tasks.py` to process UUIDs asynchronously (4 hours)[FE][BE]
    - Frontend: Build ApplyPreset component and action in `components/preset/ApplyPreset.tsx` to call POST /presets/apply (4 hours)[FE][BE]
    - Audit: Record audit entries in `apps/api/services/presets/AuditService.py` and store references in `table_presets` (4 hours)[FE][BE]
    - Tests: Add integration tests in `tests/test_presets.py` covering UUID success and per-UUID error handling (4 hours)[FE][BE][QA]
    - Docs: Add usage and acceptance criteria in `docs/presets.md` (4 hours)[FE][BE]

- **Confirm Seed Results:** As a: product owner, I want to: Confirm seed results reflect all processed projects with their presets, so that: The seed operation is observable and verifiable by stakeholders**(24 hours)** - Seed results summary lists all input UUIDs with their resulting preset identifiers Mismatch between input IDs and produced presets is reported

Timeline of seed operation is auditable with start and end timestamps Exportable seed report generated in JSON/CSV No hidden failures; all per-project outcomes are accounted for

  - API: Add /seed/results GET endpoint in `apps/api/routes/seed_routes.py` to return seed summary (4 hours)[FE][BE]
  - Service: Implement gather_seed_results() in `apps/api/services/seed/SeedService.py` to aggregate per-project outcomes (4 hours)[FE][BE]
  - DB: Create seed_results table migration in `prisma/migrations/` recording input_uuid, preset_id, status, start_ts, end_ts (4 hours)[FE][BE]
  - Export: Add JSON/CSV exporter in `apps/api/services/seed/SeedExporter.py` with endpoints in `apps/api/routes/seed_routes.py` (4 hours)[FE][BE]
  - Frontend: Build SeedResults view in `apps/web/src/components/seed/SeedResults.tsx` to display summary, mismatches, and timeline (4 hours)[FE][BE]
  - Testing: Add integration tests in `apps/api/tests/test_seed_results.py` to verify acceptance criteria (4 hours)[FE][BE][QA]

- **Create Seed Input: (32 hours)**
  - DB: Create seed_inputs migration in `prisma/migrations/20251030_create_seed_inputs` (4 hours)[FE][BE]
  - API: Add POST /seed-inputs route in `apps/api/routers/seed_inputs.py` (4 hours)[FE][BE]
  - API: Implement create_seed_input() in `apps/api/services/seed/SeedService.py` (4 hours)[FE][BE]
  - Worker: Add Celery task process_seed_input in `apps/api/tasks/seed_tasks.py` (4 hours)[FE][BE]
  - Frontend: Build SeedInputForm component in `apps/web/src/components/seed/SeedInputForm.tsx` (4 hours)[FE][BE]

    - Frontend: Add Redux slice seedInputSlice in `apps/web/src/store/slices/seedInputSlice.ts` (4 hours)[FE][BE]
    - Testing: Add integration test for /seed-inputs in `apps/api/tests/test_seed_inputs.py` (4 hours)[FE][BE][QA]
    - Docs: Document Seed Input flow in `docs/seed_input.md` (4 hours)[FE][BE]
- **Validate UUIDs: (0 hours)**
- **Confirm Seed Results: (0 hours)**
- **Add bovine intervals preset per-project: create intervals JSON for Peso_nascimento_kg 1–60; Peso_atual_kg 20–900; Idade_meses 0–360; Rendimento_de_carcaça_pct 40–80 and attach to given project IDs: (32 hours)** - Story functionality is implemented as specified User interface is responsive and accessible Data is properly validated and stored Error handling provides helpful feedback
    - DB: Add Intervals model and migration in `prisma/migrations/2025_seed_intervals/` to store project intervals (link to table_projects) (4 hours)[FE][BE]
    - Data: Create bovine intervals JSON in `data/intervals/bovine_intervals.json` with Peso_nascimento_kg 1-60; Peso_atual_kg 20-900; Idade_meses 0-360; Rendimento_de_carcaça_pct 40-80 (4 hours)[FE][BE]
    - API: Implement create_intervals_for_project() in `apps/api/services/intervals/IntervalsService.py` with validation and persistence to projects intervals (4 hours)[FE][BE][QA]
    - API: Add router endpoint POST /projects/{id}/intervals in `apps/api/routers/intervals.py` to attach intervals to given project IDs (4 hours)[FE][BE]
    - Task: Implement Celery task seed_intervals_task in `apps/api/tasks/seed_tasks.py` to attach intervals from `data/intervals/bovine_intervals.json` to multiple projects (4 hours)[FE][BE]

    - Frontend: Build SeedIntervalsForm component in `src/components/seed/SeedIntervalsForm.tsx` to select projects and trigger POST /projects/{id}/intervals (4 hours)[FE][BE]
    - Tests: Add API tests in `tests/api/test_intervals.py` for validation, success, and error cases (4 hours)[FE][BE][QA]
    - Docs: Add usage docs in `docs/seed_intervals.md` and update PR checklist `docs/PR_TEMPLATE.md` (4 hours)[FE][BE]
- **Add bovine intervals preset per-project: create intervals JSON for Peso_nascimento_kg 1–60; Peso_atual_kg 20–900; Idade_meses 0–360; Rendimento_de_carcaça_pct 40–80 and attach to given project IDs: (0 hours)**
- **Verify created presets: confirm project_id linkage and interval values: (32 hours)** - Story functionality is implemented as specified User interface is responsive and accessible Data is properly validated and stored Error handling provides helpful feedback
    - DB: Add presets.project_id and interval fields migration in `prisma/migrations/` (4 hours)[FE][BE]
    - API: Add GET/POST presets endpoints in `apps/api/routes/presets.py` (4 hours)[FE][BE]
    - API: Implement validation and save logic in `apps/api/services/presets/PresetsService.py` (4 hours)[FE][BE][QA]
    - Frontend: Build PresetsForm component in `components/presets/PresetsForm.tsx` to input project_id and intervals (4 hours)[FE][BE]
    - Frontend: Update PresetsList component in `components/presets/PresetsList.tsx` to display interval values and project linkage (4 hours)[FE][BE]
    - Testing: Add unit tests for PresetsService in `apps/api/services/presets/tests/test_presets_service.py` (4 hours)[FE][BE][QA]

    - Testing: Add frontend integration tests in `components/presets/__tests__/PresetsForm.test.tsx` (4 hours)[FE][BE][QA]
    - Docs: Update README section for presets in `docs/presets.md` (4 hours)[FE][BE]
- **Verify created presets: confirm project_id linkage and interval values: (0 hours)**

Milestone 4: API handlers and normalization: implement Remix GET/PUT handlers, auth (member read, admin write), payload validation, normalization & unit hooks, and audit log

Estimated 1003 hours

- **PUT upload presets (admin): (9 hours)** - Story functionality is implemented as specified User interface is responsive and accessible Data is properly validated and stored Error handling provides helpful feedback
    - DB: Create upload_presets table migration in `prisma/migrations/` and update `prisma/schema.prisma` - (M) (1 hours)[FE][BE]
    - API: Add Zod validation schema in `apps/api/schemas/uploadPresetSchema.ts` - (M) (1 hours)[FE][BE][QA]
    - API: Implement updatePreset() in `apps/api/services/uploadPresets/UploadPresetService.ts` - (M) (1 hours)[FE][BE]
    - API: Implement PUT handler in `apps/api/routes/api/project/$projectId/upload-presets.ts` (auth: admin) - (M) (1 hours)[FE][BE]
    - Frontend: Build UploadPresetsForm component in `apps/web/components/admin/UploadPresetsForm.tsx` - (M) (1 hours)[FE][BE]

- Frontend: Add admin route in `apps/web/routes/admin/project/$projectId/upload-presets.tsx` - (M) (1 hours)[FE][BE]
- Tests: Add API tests in `apps/api/tests/uploadPresets.test.ts` for PUT behavior and error cases - (M) (1 hours)[FE][BE][QA]
- Tests: Add UI tests in `apps/web/tests/UploadPresetsForm.test.tsx` for accessibility and responsiveness - (M) (1 hours)[FE][BE][QA]
- Docs: Document endpoint in `docs/api/upload-presets.md` and update API reference - (M) (1 hours)[FE][BE]

- **Auth: member read, admin write: (7 hours)** - Story functionality is implemented as specified User interface is responsive and accessible Data is properly validated and stored Error handling provides helpful feedback
  - DB: Create upload_presets migration in `prisma/migrations/20251001_create_upload_presets` - (M) (1 hours)[FE][BE]
  - API: Add isAdmin/isMember checks in `apps/api/services/auth/AuthService.ts` - (M) (1 hours)[FE][BE]
  - API: Implement GET handler in `apps/api/routes/api/project/$projectId/upload-presets.tsx` (member read) - (M) (1 hours)[FE][BE]
  - API: Implement PUT handler in `apps/api/routes/api/project/$projectId/upload-presets.tsx` (admin write) with Zod validation in `apps/api/schemas/uploadPresets.ts` and normalization in `apps/api/utils/normalizers/uploadPresets.ts` - (M) (1 hours)[FE][BE][QA]
  - Frontend: Build UploadPresetsForm in `apps/web/components/UploadPresetsForm.tsx` with role-aware UI - (M) (1 hours)[FE][BE]

    - Tests: Add integration tests in `apps/api/tests/uploadPresets.test.ts` covering member GET and admin PUT - (M) (1 hours)[FE][BE][QA]
    - Infra: Update Dockerfile and GitHub Actions workflow in `.github/workflows/ci.yml` to run migration and tests - (M) (1 hours)[FE][BE][QA]
- **Return Presets (200): (28 hours)** - Response contains a non-empty array of presets Each preset has id and name fields HTTP status is 200 with correct headers
    - API: Add route GET /api/project/{projectId}/upload-presets in `apps/api/routes/presets.py` (4 hours)[FE][BE]
    - API: Implement PresetsService.get_presets(project_id) in `apps/api/services/presets/PresetsService.py` (4 hours)[FE][BE]
    - DB: Add query get_presets_for_project in `apps/api/db/presets_queries.py` (4 hours)[FE][BE]
    - Middleware: Add response header enforcement in `apps/api/middleware/headers.py` (4 hours)[FE][BE]
    - Test: Create integration test for presets endpoint in `tests/api/test_presets.py` (4 hours)[FE][BE][QA]
    - Test: Add fixtures `tests/fixtures/presets.json` with non-empty presets array (4 hours)[FE][BE][QA]
    - Doc: Document endpoint in `docs/api/presets.md` with response schema (id,name) and headers (4 hours)[FE][BE]
- **401 Unauthorized: (20 hours)** - Response status is 401 Error code indicates invalid/absent credentials No sensitive data exposed in body
    - API: Add is_authenticated() in `apps/api/services/auth/AuthService.ts` (checks token/session) (4 hours)[FE][BE]
    - API: Create auth middleware in `apps/api/middleware/auth.py` to enforce authentication and return 401 (4 hours)[FE][BE]

    - API: Update GET /api/project/{projectId}/upload-presets handler in `apps/api/routes/projects.py` to use auth middleware and return 401 with non-sensitive body (4 hours)[FE][BE]
    - TESTING: Add unit tests in `tests/api/test_upload_presets.py` for 401 responses and error codes (4 hours)[FE][BE][QA]
    - DOCUMENTATION: Update `docs/api/errors.md` with 401 behavior and error code (4 hours)[FE][BE]
- **404 Not Found: (32 hours)** - Response status is 404 Error message clearly states resource not found No information leakage about internal structure
    - API: Add router for GET /api/project/{projectId}/upload-presets in `apps/api/routers/project.py` (4 hours)[FE][BE]
    - API: Implement not-found check in `apps/api/services/project_service.py` to return None when presets missing (4 hours)[FE][BE]
    - API: Return sanitized 404 response in `apps/api/routers/project.py` with status 404 and message ‘Resource not found’ using `apps/api/schemas/errors.py` (4 hours)[FE][BE]
    - Schema: Create error schema in `apps/api/schemas/errors.py` with message field and no internals (4 hours)[FE][BE]
    - Middleware: Add error handling middleware in `apps/api/middleware/error_handler.py` to prevent info leakage (4 hours)[FE][BE]
    - Main: Register middleware and router in `apps/api/main.py` (4 hours)[FE][BE]
    - Tests: Add integration test for 404 in `tests/api/test_upload_presets_404.py` asserting status 404 and message (4 hours)[FE][BE][QA]

    - CI: Add test step in `/.github/workflows/ci.yml` to run `tests/api/test_upload_presets_404.py` (4 hours)[FE][BE][QA]

- **Seed Awareness: (36 hours)**
    - DB: Add presets table migration in `prisma/migrations/` to store upload presets (4 hours)[FE][BE]
    - API: Implement GET /api/project/{projectId}/upload-presets handler in `apps/api/routers/presets.py` (4 hours)[FE][BE]
    - Service: Create get_upload_presets(project_id) in `apps/api/services/presets/PresetsService.py` (4 hours)[FE][BE]
    - Cache: Add Redis caching logic in `apps/api/services/presets/cache.py` using Redis key `project:{projectId}:presets` (4 hours)[FE][BE]
    - Frontend: Add route /projects/:id/presets and component `apps/web/src/components/presets/PresetsList.tsx` (4 hours)[FE][BE]
    - Frontend: Create Redux slice in `apps/web/src/store/slices/presetsSlice.ts` to fetch GET /api/project/{projectId}/upload-presets (4 hours)[FE][BE]
    - Infra: Update Dockerfile for `apps/api` to include new service files `apps/api/services/presets/` (4 hours)[FE][BE]
    - Quality: Add unit tests for PresetsService in `apps/api/tests/test_presets_service.py` (4 hours)[FE][BE][QA]
    - Quality: Add integration test for GET handler in `apps/api/tests/test_presets_endpoint.py` (4 hours)[FE][BE][QA]
- **Validate Payload: ensure body matches schema; reject 400 on invalid: (24 hours)** - Admin can submit updated presets via PUT request and receive a 200 response System validates input against preset schema and returns detailed validation errors for invalid fields Presets are persisted to the

data store and retrievable in subsequent GET requests Edge case: updating to an existing preset name updates the existing record without creating duplicates

    - Schema: Add Preset schema in `apps/api/schemas/presets.py` (4 hours)[FE][BE]
    - Infra: Create presets table migration in `prisma/migrations/` (4 hours)[FE][BE]
    - API: Implement validation middleware in `apps/api/middleware/validation.py` to use schema and return 400 with details (4 hours)[FE][BE][QA]
    - API: Implement PUT handler in `apps/api/routes/presets.py` for /api/project/{projectId}/upload-presets (4 hours)[FE][BE]
    - Service: Implement upsert_presets in `apps/api/services/presets_service.py` to persist and handle name collisions (4 hours)[FE][BE]
    - Quality: Add tests in `tests/api/test_presets_put.py` for valid, invalid, and duplicate-name scenarios (4 hours)[FE][BE][QA]

- **Authorization Check: require admin session; return 403 on unauthorized: (28 hours)** - Request with non-admin role is rejected with 403 Admin session/context is verified before processing the request Audit log records the authorization outcome for each request System denies PUT when authorization fails and returns appropriate error message

    - API: Add admin-check middleware in `apps/api/middleware/auth.ts` to verify admin session (4 hours)[FE][BE]
    - API: Update PUT handler in `apps/api/routes/projects/put_upload_presets.ts` to call admin middleware and return 403 on unauthorized (4 hours)[FE][BE]
    - Service: Implement isAdmin() in `apps/api/services/auth/AuthService.ts` to validate session role against `table_users` (4 hours)[FE][BE]

    - Service: Create audit log entry in `apps/api/services/audit/AuditService.ts` recording authorization outcome (4 hours)[FE][BE]
    - DB: Add role column and seed admin user in migration `prisma/migrations/2025xxxx_add_user_role/` referencing `table_users` (4 hours)[FE][BE]
    - Testing: Add integration tests in `apps/api/tests/put_upload_presets.test.ts` for 403 on non-admin and success for admin (4 hours)[FE][BE][QA]
    - Documentation: Update API docs in `docs/api/upload-presets.md` to specify 403 behavior (4 hours)[FE][BE]
- **Upsert Preset: upsert presets in DB (prepare migration/seed files only; do not auto-run); idempotent: (24 hours)** - Preset is inserted when new Preset is updated when existing name matches Upsert operation returns the upserted preset with its id No duplicates are created for the same preset name
    - DB: Create Prisma migration for presets table in `apps/api/prisma/migrations/20251001_create_presets_table/` (4 hours)[FE][BE]
    - DB: Add Preset model to `apps/api/prisma/schema.prisma` (model Preset with id, name, data) (4 hours)[FE][BE]
    - DB: Create idempotent seed file `apps/api/prisma/seeders/upsert_presets_seed.ts` to upsert presets (4 hours)[FE][BE]
    - API: Implement seed runner script in `apps/api/scripts/runSeed.ts` (do not auto-run) referencing upsert logic (4 hours)[FE][BE]
    - Dev: Add TypeScript upsert helper in `apps/api/services/db/presetUpsert.ts` to upsert and return preset with id (4 hours)[FE][BE][DevOps]
    - Test: Add tests for seed idempotency in `apps/api/tests/seed_presets.test.ts` (4 hours)[FE][BE][QA]

- **Return Responses: 200 on success with summary, 403/400 as applicable: (28 hours)** - Response payload includes the updated preset with all fields Status code 200 OK returned on success Response includes the preset’s ID and name Response adheres to predefined response schema
    - API: Implement PUT handler in `apps/api/routes/project/uploadPresets.py` to return 200 with updated preset payload and summary (4 hours)[FE][BE]
    - API: Add request validation middleware in `apps/api/middleware/validation.py` for preset schema and return 400 on invalid (4 hours)[FE][BE][QA]
    - API: Add authorization check in `apps/api/middleware/auth.py` to return 403 when user lacks permission (4 hours)[FE][BE]
    - DB: Implement upsert preset logic in `apps/api/services/presets/PresetsService.py` to persist and return full preset record (4 hours)[FE][BE]
    - Schema: Define response schema in `apps/api/schemas/preset_response.py` and ensure includes id and name (4 hours)[FE][BE]
    - Test: Add integration tests in `apps/api/tests/test_upload_presets.py` for 200/400/403 responses and payload compliance (4 hours)[FE][BE][QA]
    - Docs: Update API docs in `apps/api/docs/openapi.yaml` for PUT /api/project/{projectId}/upload-presets responses (4 hours)[FE][BE]
- **Validate Payload: (0 hours)**
- **Authorization Check: (0 hours)**
- **Return Responses: (0 hours)**

- **GET Upload Preset (Auth: owner/editor): (28 hours)** - System authenticates user as owner or editor for the project Returns the upload preset configuration with all required fields Handles non-existent project or missing preset gracefully with appropriate error codes
    - DB: Create upload_presets table migration in `prisma/migrations/2025_create_upload_presets.sql` (4 hours)[FE][BE]
    - API: Add route GET /api/project/{id}/upload-presets in `apps/api/routes/project/upload_presets.py` (4 hours)[FE][BE]
    - API: Implement auth check is_owner_or_editor() in `apps/api/services/auth/AuthService.ts` (4 hours)[FE][BE]
    - API: Implement get_upload_preset(project_id) in `apps/api/services/project/UploadPresetService.py` (4 hours)[FE][BE]
    - API: Return preset response and error handling in `apps/api/routes/project/upload_presets.py` (4 hours)[FE][BE]
    - TEST: Add unit/integration tests in `apps/api/tests/test_upload_presets.py` (4 hours)[FE][BE][QA]
    - DOC: Update API docs for /api/project/{id}/upload-presets in `docs/api.md` (4 hours)[FE][BE]
- **PUT Update Preset (Auth: admins) - validate uuid_generate_v4 usage & payload schema:** As a: admin, I want to: PUT update the upload preset with proper UUID generation and payload validation, So that: presets are updated securely with valid identifiers**(7.5 hours)** - Payload schema validation for required fields uuid_generate_v4 usage

validated or generated Successful update returns 200 with updated preset Invalid UUID or schema results in 400/422 with clear error messages

    - API: Add payload schema in apps/api/schemas/presetSchema.ts (validate required fields) preserving architecture refs and ensuring the preset payload matches the server schema for update operations. - (S) (0.5 hours) [FE][BE]
    - API: Implement PUT handler in apps/api/routes/api/project/[id] to update a preset; route should validate id param, parse body via presetSchema, call presetService, and return appropriate HTTP statuses using existing control flow. - (M) (1 hours)[FE][BE]
    - Service: Add/update uuid handling in apps/api/services/presetService.ts to validate or call uuid_generate_v4 - (M) (1 hours)[FE][BE]
    - DB: Add migration in prisma/migrations/ to ensure pgcrypto/uuid columns for upload_presets table - (L) (2 hours)[FE][BE]
    - Tests: Add unit tests for schema in apps/api/tests/presetSchema.test.ts - (S) (0.5 hours)[FE][BE][QA]
    - Tests: Add integration tests for PUT in apps/api/tests/presetUpdate.test.ts (200/400/422 cases) - (L) (2 hours)[FE][BE][DevOps][QA]
    - Docs: Update API docs in apps/api/docs/presets.md with error formats and UUID guidance - (S) (0.5 hours)[FE][BE]

- **Auth Enforcement - member read, admin write:** As a: admin, I want to: enforce correct authorization for reading and writing upload presets, So that: only permitted roles can modify presets and access controls are respected**(4.5 hours)** -

GET requests allowed for members PUT requests allowed for admins Unauthorized access (401/403) returns proper status Audit log entry on successful write

    - API: Implement GET/PUT routes in apps/api/routes/project/upload/presets.ts with route guards for member GET and admin PUT, wiring to service layer for authorization checks and audit logging hook on write paths. - (M) (1 hours)[FE][BE]
    - Auth: Implement middleware check in apps/api/middleware/auth.ts to allow GET for role ‘member’ and PUT for role ‘admin’, attaching user role to request for downstream checks. - (S) (0.5 hours)[FE][BE]
    - Service: Implement role check helper in apps/api/services/auth/AuthService.ts::authorizeRequest() to validate role against allowedRoles and extract user identity from token/claims. - (M) (1 hours)[FE][BE]
    - API: Return proper 401/403 in apps/api/routes/project/upload/presets.ts on unauthorized access, ensuring consistent error response structure. - (S) (0.5 hours)[FE][BE]
    - Audit: Log successful PUT in apps/api/services/audit/AuditService.ts::logWrite() and call from apps/api/routes/project/upload/presets.ts - (S) (0.5 hours)[FE][BE]
    - Tests: Add integration tests in apps/api/tests/project_upload_presets.test.ts for member GET, admin PUT, and 401/403 cases - (M) (1 hours)[FE][BE][QA]

- **Validate Preset Payload:** As a: system, I want to: Validate preset payload structure and value types before processing, So that: only well-formed data is accepted**(4 hours)** - Payload must include required fields with correct types Optional fields

validated if present Invalid types or missing required fields yield 400 with detailed errors Validation occurs before persistence or further processing

    - API: Add preset validation schema in apps/api/services/validation/presetSchema.ts (define required/optional fields, types) preserving preset domain shape and ensuring compatibility with existing validation layer (jsonSchema/yup). - (S) (0.5 hours)[FE][BE][QA]
    - API: Implement request validation middleware in apps/api/routes/apiUtils/validateRequest.ts (use presetSchema, return 400 with detailed errors) integrating with presetSchema to validate incoming presets before route handling. - (S) (0.5 hours)[FE][BE][QA]
    - API: Integrate validation into route handler in apps/api/routes/project/upload-presets.ts before persistence and further processing - (M) (1 hours)[FE][BE][QA]
    - TEST: Add unit tests for presetSchema in apps/api/tests/validation/presetSchema.test.ts (valid/invalid/optional cases) - (S) (0.5 hours)[FE][BE][QA]
    - TEST: Add integration tests for route in apps/api/tests/integration/uploadPresets.test.ts (assert 400 with error details on bad payload) - (M) (1 hours)[FE][BE][QA]
    - DOC: Update API docs in docs/api/upload-presets.md with payload spec and error format - (XS) (0.5 hours)[FE][BE]

- **List Presets (Auth: members): (32 hours)**
    - DB: Add presets query in `apps/api/db/presets_queries.py` (4 hours)[FE][BE]
    - API: Implement PresetsService.listPresets in `apps/api/services/presets/PresetsService.py` (4 hours)[FE][BE]
    - API: Add GET /api/project/{id}/upload-presets handler in `apps/api/routers/project/upload_presets.py` (4 hours)[FE][BE]

    - API: Add auth check middleware usage in `apps/api/routers/project/upload_presets.py` (4 hours)[FE][BE]
    - Frontend: Build ApiPresetList component in `apps/web/components/api/ApiPresetList.tsx` (4 hours)[FE][BE]
    - Integration: Register route in `apps/api/main.py` and expose to frontend route_api_project_upload_presets (4 hours)[FE][BE]
    - Tests: Create unit tests for PresetsService in `apps/api/tests/test_upload_presets.py` (4 hours)[FE][BE][QA]
    - Docs: Add API docs in `apps/web/docs/upload_presets.md` (4 hours)[FE][BE]
- **GET Upload Preset (Auth: members) - return default created via app if missing: (24 hours)**
    - DB: Add upload_presets table migration in `prisma/migrations/` (4 hours)[FE][BE]
    - API: Add auth middleware check in `apps/api/middleware/auth.ts` (4 hours)[FE][BE]
    - API: Implement createDefaultUploadPresetIfMissing in `apps/api/services/upload/UploadPresetService.ts` (4 hours)[FE][BE]
    - API: Implement GET handler in `apps/api/routes/project/uploadPresets.ts` for /api/project/{id}/upload-presets (4 hours)[FE][BE]
    - Tests: Add unit tests in `apps/api/tests/uploadPreset.test.ts` (4 hours)[FE][BE][QA]
    - Frontend: Ensure API Preset View handles default preset in `src/components/ApiPresetView.tsx` (comp_api_project_upload_presets_view) (4 hours)[FE][BE]
- **Create Default Preset on Project Create (App logic): (0 hours)**

- **GET Upload Preset (Auth: members) - returns default if missing: (0 hours)**
- **Validate Preset Payload - schema & intervals: (0 hours)**
- **GET Upload Preset - returns default if missing (members):** As a: member, I want to: GET the upload preset for a project and receive a default preset when none exists, So that: I can proceed with uploads without setup friction**(6.5 hours)** - Verify response returns 200 OK with a preset payload If no preset exists, response contains default preset values Response includes project_id and user role context Error handling for missing project or unauthorized access (401/403)
    - API: Implement GET handler in apps/api/routes/api/project/$projectId/upload-presets.ts to return preset or default (includes project_id and role) using UploadPresetService.getUploadPreset, with 200 on success, 401/403 on auth failures. - (M) (1 hours)[FE][BE]
    - Service: Add getUploadPreset(projectId: string, userId: string) in apps/api/services/uploadPresets/UploadPresetService.ts to fetch preset via Prisma and apply defaults if null. - (M) (1 hours)[FE][BE]
    - DB: Add Prisma query in apps/api/services/uploadPresets/UploadPresetService.ts to fetch preset from DB using Prisma client prisma (cite table_users) - (S) (0.5 hours)[FE][BE]
    - Auth: Add role/context check in apps/api/middleware/auth.ts and use in route handler to enforce 401/403 - (M) (1 hours)[FE][BE]
    - Defaults: Implement default preset values in apps/api/services/uploadPresets/defaults.ts and use when DB returns null - (XS) (0.5 hours)[FE][BE]
    - Types: Add types for UploadPreset response in apps/api/types/uploadPreset.ts including project_id and role - (XS) (0.5 hours)[FE][BE]

    - Tests: Create integration tests in apps/api/tests/uploadPresets/getUploadPreset.test.ts covering 200, default fallback, 401, 403 - (L) (2 hours)[FE][BE][QA]
    - Docs: Update API docs in docs/api.md for GET /api/project/{id}/upload-presets with examples - (XS) (0.5 hours)[FE][BE]
- **Flag unit issues (unit_flag) when conversion ambiguous or missing:** As a: data analyst, I want to: flag unit issues when a unit conversion is ambiguous or missing, so that: the dataset integrity is preserved and downstream mappings can be corrected.**(9 hours)** - System detects when unit conversion path is ambiguous or missing Flag is set on records with unit_flag and a reason logged Flag propagation respects existing mapping rules and does not override confirmed good records Audit log records the reason for the flag with timestamp System handles bulk records efficiently without significant performance degradation
    - DB: Add unit_flag and reason columns in prisma migrations to update table_units, enabling storage of per-unit ambiguity flags and rationale for flagged records. - (S) (0.5 hours)[FE][BE]
    - API: Implement conversion ambiguity detection in apps/api/services/normalization/UnitNormalization.ts using existing normalization workflow to set unit_flag when multiple valid unit conversions exist or when primary conversion missing. - (M) (1 hours)[FE][BE]
    - API: Set unit_flag and log reason in apps/api/services/audit/AuditService.ts with timestamp whenever a flag is set, ensuring traceability. - (S) (0.5 hours)[FE][BE]
    - API: Ensure flag propagation rules in apps/api/services/mapping/MappingService.ts does not override confirmed records, preserving prior confirmations while new flags may be added. - (M) (1 hours)[FE][BE]

    - Job: Implement bulk normalization job in apps/api/jobs/normalizationJob.ts using Celery/Redis optimizations to process large datasets efficiently. - (XL) (4 hours)[FE][BE]
    - Tests: Add unit and integration tests in tests/unit/unitNormalization.test.ts and tests/integration/normalization.integration.test.ts covering flagging, propagation, and audit logging. - (M) (1 hours)[FE][BE][QA]
    - API: Expose unit_flag via route in apps/api/routes/normalization.tsx, enabling frontend retrieval of flag state. - (S) (0.5 hours)[FE][BE]
    - Frontend: Add UnitFlagIndicator component in components/normalization/UnitFlagIndicator.tsx to visualize unit flags on unit items. - (S) (0.5 hours)[FE][BE]
- **Emit review_required banner when dataset contains flagged records:** As a: data steward, I want to: emit a review_required banner when the dataset contains flagged records, so that: stakeholders are alerted to review issues before processing continues.**(7.5 hours)** - Banner appears when any flagged records exist in the dataset Banner contains count of flagged records and a link to review Banner persists or refreshes as data changes Banner dismisses only after flagged records are resolved or dataset re-scanned System performance impact remains negligible with small flag sets
    - DB: Add flagged_count and flagged_index migration in prisma/migrations/2025xxxx_add_flagged_fields/ preserving references to prisma migrations and dataset schema. - (M) (1 hours)[FE][BE]
    - API: Implement getFlaggedCount(datasetId) in apps/api/services/dataset/DatasetService.ts to return count from DB using prisma client. - (S) (0.5 hours)[FE][BE]
    - API: Add /datasets/:id/flagged-count route in apps/api/routes/datasets.ts to expose getFlaggedCount via REST. - (S) (0.5 hours)[FE][BE]

    - Job: Create scanFlaggedRecords job in apps/api/jobs/scanFlaggedRecords.ts to update flagged_count - (M) (1 hours)[FE][BE]
    - Frontend: Build ReviewBanner component in apps/web/components/dataset/ReviewBanner.tsx - (S) (0.5 hours)[FE][BE]
    - Frontend: Add useFlaggedCount hook in apps/web/hooks/useFlaggedCount.ts to poll/subscribe to /datasets/:id/flagged-count - (S) (0.5 hours)[FE][BE]
    - State: Integrate flagged count into Redux store in apps/web/store/datasetSlice.ts - (M) (1 hours)[FE][BE]
    - UX: Add review link and dismiss behavior in apps/web/components/dataset/ReviewBanner.tsx - (M) (1 hours)[FE][BE]
    - Testing: Add unit tests for DatasetService.getFlaggedCount in apps/api/__tests__/datasetFlagged.test.ts - (S) (0.5 hours)[FE][BE][QA]
    - Testing: Add frontend tests for ReviewBanner in apps/web/__tests__/ReviewBanner.test.tsx - (S) (0.5 hours)[FE][BE][QA]
    - Docs: Document banner behavior in docs/feature_review_banner.md - (XS) (0.5 hours)[FE][BE]
- **Standardize Names: (30 hours)**
    - DB: Create migration to add standardized_name column in `prisma/migrations/` for table_unit_normalizers (4 hours)[FE][BE]
    - DB: Create migration to add normalized_name field to `prisma/migrations/` for table_unit_mapping_suggestions (4 hours)[FE][BE]
    - API: Implement NameNormalizer class in `apps/api/services/normalizer/NameNormalizer.py` to standardize names (4 hours)[FE][BE]

- API: Add /normalizers/standardize route in `apps/api/routes/normalizers.py` using NameNormalizer (4 hours) [FE][BE]
- Frontend: Build NameStandardizer component in `src/components/normalizer/NameStandardizer.tsx` (4 hours) [FE][BE]
- Frontend: Update Redux slice in `src/store/normalizerSlice.ts` to call /normalizers/standardize (4 hours)[FE][BE]
- Tests: Add integration tests in `tests/api/test_name_normalizer.py` for standardization flow (4 hours) [FE][BE][QA]
- Docs: Add normalizer docs in `docs/normalizer.md` (2 hours)[FE][BE]

- **Suggest Mappings: (28 hours)**
  - DB: Create unit_mapping_suggestions migration in `apps/api/migrations/` (4 hours)[FE][BE]
  - DB: Add UnitMappingSuggestion model in `apps/api/models/unit_mapping.py` referencing table_unit_mapping_suggestions and table_unit_normalizers (4 hours)[FE][BE]
  - API: Implement Suggest Mappings logic in `apps/api/services/mapping/MappingService.py` (uses table_unit_normalizers, table_unit_mapping_suggestions) (4 hours)[FE][BE]
  - API: Add router `/mapping/suggest` in `apps/api/routers/mapping.py` calling MappingService.py (4 hours)[FE][BE]
  - Frontend: Build SuggestMappings component in `apps/web/src/components/unit/SuggestMappings.tsx` (references API `/mapping/suggest`) (4 hours)[FE][BE]

- Worker: Add Celery task in `apps/api/workers/mapping_tasks.py` to generate suggestions asynchronously (writes to table_unit_mapping_suggestions) (4 hours)[FE][BE]
- Tests: Add unit/integration tests in `apps/api/tests/test_mapping.py` for MappingService and `/mapping/suggest` router (4 hours)[FE][BE][QA]

- **Normalize Units: (40 hours)**
  - DB: Create unit_normalizers table migration in `prisma/migrations/` (4 hours)[FE][BE]
  - DB: Create unit_mapping_suggestions table migration in `prisma/migrations/` (4 hours)[FE][BE]
  - API: Add /unit-normalizers POST & GET routes in `apps/api/routers/unit_normalizers.py` (4 hours)[FE][BE]
  - Service: Implement normalizeUnits() in `apps/api/services/normalizer/NormalizerService.py` (4 hours)[FE][BE]
  - Service: Implement suggestMappings() in `apps/api/services/normalizer/MappingSuggestionService.py` (4 hours)[FE][BE]
  - Frontend: Build UnitNormalizerPage in `apps/web/src/pages/UnitNormalizerPage.tsx` (4 hours)[FE][BE]
  - Frontend: Create NormalizerForm component in `apps/web/src/components/normalizer/NormalizerForm.tsx` (4 hours)[FE][BE]
  - Frontend: Add Redux slice unitNormalizerSlice in `apps/web/src/store/slices/unitNormalizerSlice.ts` (4 hours)[FE][BE]
  - API: Add Prisma model and repository in `apps/api/db/unit_normalizer_repo.py` (4 hours)[FE][BE]
  - Tests: Add unit tests for NormalizerService in `apps/api/tests/test_normalizer_service.py` (4 hours)[FE][BE][QA]

- **Map Units: (32 hours)**
  - DB: Create migration for unit_normalizers and unit_mapping_suggestions in `prisma/migrations/2025_create_unit_mappings` (4 hours)[FE][BE]
  - DB Model: Add UnitNormalizer and UnitMappingSuggestion models in `apps/api/models/unit_normalizer.py` (4 hours)[FE][BE]
  - API Service: Implement map_units() in `apps/api/services/unit/UnitMappingService.py` (4 hours)[FE][BE]
  - API Router: Add POST /unit-map in `apps/api/routers/unit_mapping.py` (4 hours)[FE][BE]
  - Frontend: Build UnitMapper component in `apps/web/components/unit/UnitMapper.tsx` (4 hours)[FE][BE]
  - Frontend Store: Add unitSlice with mapping actions in `apps/web/store/unitSlice.ts` (4 hours)[FE][BE]
  - Tests: Add integration tests for mapping in `apps/api/tests/test_unit_mapping.py` (4 hours)[FE][BE][QA]
  - Docs: Document Map Units API and UI in `docs/unit_mapping.md` (4 hours)[FE][BE]
- **Validate Units: (28 hours)**
  - DB: Create prisma migration for unit_normalizers and unit_mapping_suggestions in `prisma/migrations/` (4 hours)[FE][BE]
  - DB: Add Prisma models in `prisma/schema.prisma` for unit_normalizers (table_unit_normalizers) and unit_mapping_suggestions (table_unit_mapping_suggestions) (4 hours)[FE][BE]
  - API: Implement validateUnits() in `apps/api/services/unit_normalizer/UnitValidator.ts` (4 hours)[FE][BE]

    - API: Add router and endpoint POST /unit-normalizers/validate in `apps/api/routers/unit_normalizers.py` (4 hours)[FE][BE]
    - Frontend: Build UnitValidatorForm component in `src/components/unit-normalizer/UnitValidatorForm.tsx` (4 hours)[FE][BE]
    - Frontend: Create Redux slice in `src/store/unitNormalizerSlice.ts` to call /unit-normalizers/validate (4 hours)[FE][BE]
    - Testing: Add unit tests for UnitValidator in `tests/unit/test_unit_validator.py` (4 hours)[FE][BE][QA]
- **Preview Normalization: (36 hours)**
    - DB: Create unit_normalizers migration in `prisma/migrations/20251030_create_unit_normalizers.sql` (4 hours)[FE][BE]
    - DB: Add UnitNormalizer Prisma model in `prisma/schema.prisma` and seed in `prisma/seed.ts` (4 hours)[FE][BE]
    - API: Implement PreviewNormalizer service in `apps/api/services/normalizer/PreviewNormalizer.py` (4 hours)[FE][BE]
    - API: Add router endpoint POST /preview-normalize in `apps/api/routers/preview_normalizer.py` (4 hours)[FE][BE]
    - Frontend: Build PreviewNormalizer component in `src/components/preview/PreviewNormalizer.tsx` (4 hours)[FE][BE]
    - DB: Create unit_mapping_suggestions table migration in `prisma/migrations/20251030_create_unit_mapping_suggestions.sql` (4 hours)[FE][BE]
    - API: Integrate normalization with S3 staging in `apps/api/services/normalizer/PreviewNormalizer.py` (4 hours)[FE][BE]

    - Tests: Add unit tests for PreviewNormalizer in `tests/test_preview_normalizer.py` (4 hours)[FE][BE][QA]
    - Frontend: Add integration test for PreviewNormalizer in `tests/frontend/preview_normalizer.test.tsx` (4 hours)[FE][BE][QA]
- **Store Change Diff:** As a: admin, I want to: store the change diff when an upload-preset is updated, So that: we can audit exactly what changed for accountability**(7.5 hours)** - Change diff is captured and stored for every update Diff captures field-level changes with old and new values System handles updates with no diff gracefully (null diff) Diff is persisted in audit log database with timestamp Access to diff is restricted to admins with proper permissions
    - DB: Create migration for project_upload_presets_audit in prisma/migrations/ to add diff JSONB, user_id FK, timestamp (created_at) preserving existing table structure and ensuring index compatibility. - (S) (0.5 hours)[FE][BE]
    - API: Add audit log model in apps/api/models/ProjectUploadPresetsAudit.ts with fields (id, preset_id, user_id, diff JSONB, created_at) using Prisma schema and corresponding TypeScript type definitions. - (S) (0.5 hours)[FE][BE]
    - API: Implement storeDiff() in apps/api/services/audit/AuditService.ts to compute and persist diff, handle null diff - (M) (1 hours)[FE][BE]
    - API: Add middleware checkAdmin in apps/api/middleware/authorization.ts to restrict diff access to admins using table_user_roles and table_users - (M) (1 hours)[FE][BE]
    - API: Hook storeDiff() into update handler in apps/api/routes/uploadPresets.ts to call AuditService.storeDiff(old, new, userId) - (M) (1 hours)[FE][BE]
    - DB: Add index on created_at in prisma/migrations/ for table_project_upload_presets_audit - (S) (0.5 hours)[FE][BE]

    - Frontend: Create AdminAuditView in apps/web/components/admin/AdminAuditView.tsx to display diffs with old/new values and timestamps - (M) (1 hours)[FE][BE]
    - API: Add GET /audit/upload-presets in apps/api/routes/audit.ts restricted by checkAdmin to return paged audit entries - (M) (1 hours)[FE][BE]
    - Testing: Add unit tests for diff computation in apps/api/services/audit/__tests__/AuditService.test.ts covering null diffs - (S) (0.5 hours)[FE][BE][QA]
    - Documentation: Update docs/audit.md describing schema, API endpoints, permissions and file paths - (S) (0.5 hours)[FE][BE]
- **Tie Update to User:** As a: admin, I want to: tie each upload-presets update to the performing user, So that: the audit log shows who performed the change for accountability**(9 hours)** - Update record includes user_id of the actor Audit log entry links to admin user profile System enforces non-empty user_id for updates Query can filter updates by user Error handling for missing user context
    - DB: Create migration adding user_id to project_upload_presets_audit in `prisma/migrations/XXXX_add_user_id_to_project_upload_presets_audit/` - (M) (1 hours)[FE][BE]
    - DB: Update Prisma model in `prisma/schema.prisma`: add user_id relation to project_upload_presets_audit - (M) (1 hours)[FE][BE]
    - API: Add user enforcement middleware in `apps/api/middleware/auth.ts` to require user context - (M) (1 hours)[FE][BE]
    - API: Implement createAuditEntry(update) in `apps/api/services/audit/AuditService.ts` to include user_id and link to users - (M) (1 hours)[FE][BE]

    - API: Update upload-preset update route in `apps/api/routes/uploadPresets.ts` to pass actor user_id to AuditService - (M) (1 hours)[FE][BE]
    - API: Add query endpoint filter by user in `apps/api/routes/auditLogs.ts` with user_id filter param - (M) (1 hours)[FE][BE]
    - Frontend: Add user filter UI in `apps/web/components/audit/AuditFilter.tsx` and connect to `routes/audit` fetch with user_id param - (M) (1 hours)[FE][BE]
    - Testing: Add integration tests in `apps/api/tests/audit.test.ts` for missing user context and filtering by user - (M) (1 hours)[FE][BE][QA]
    - Docs: Update README section in `docs/audit.md` describing user_id requirement and API filter - (M) (1 hours)[FE][BE]
- **Timestamp Entry:** As a: admin, I want to: record a precise timestamp for the update in the audit log, So that: we have exact timing for each admin action**(6.5 hours)** - Timestamp recorded on every update Timezone handling and consistency Timestamp format standardized in logs Audit queries can filter by time range Performance impact negligible
    - DB: Add ‘updated_at_tz’ timestamptz column and index in `prisma/migrations/xxxx_add_updated_at_tz/` preserving migrations path and index creation for updated_at_tz to track UTC with timezone. - (S) (0.5 hours)[FE][BE]
    - DB/Model: Update Prisma model in `prisma/schema.prisma` for model project_upload_presets_audit with updated_at_tz to store UTC timestamp with timezone. - (S) (0.5 hours)[FE][BE]
    - API: Implement timestamp write in `apps/api/services/audit/AuditService.ts` to set UTC with timezone info - (M) (1 hours)[FE][BE]

    - API: Add DB trigger function in `prisma/migrations/xxxx_add_trigger_updated_at/` to enforce timestamp on UPDATE - (M) (1 hours)[FE][BE]
    - API: Expose audit query endpoint in `apps/api/routes/audit.ts` with time range filters - (M) (1 hours)[FE][BE]
    - Frontend: Build TimeFilter component in `apps/web/components/audit/TimeFilter.tsx` to select ranges and timezone - (M) (1 hours)[FE][BE]
    - Config: Create timezone normalization util in `apps/api/config/timezone.ts` - (XS) (0.5 hours)[FE][BE][DevOps]
    - Testing: Add unit/integration tests in `apps/api/tests/audit/timestamp.test.ts` for timestamp correctness and format - (S) (0.5 hours)[FE][BE][QA]
    - Testing: Add perf test in `apps/api/tests/perf/audit_perf.test.ts` to ensure negligible impact - (S) (0.5 hours)[FE][BE][QA]
- **Authorization owner/editor doc:** As a: docs contributor, I want to: ensure authorization for owner/editor on GET schema docs, So that: access control is clear and enforced in documentation**(24 hours)** - Docs reflect correct access permissions for owner and editor roles Examples show restricted data exposure according to role No leakage of sensitive fields in docs Documentation updated for access control rules Review checklist ensures policy consistency
    - Docs: Create GET schema doc skeleton in `apps/api/docs/schemas/get_schema.md` (4 hours)[FE][BE]
    - API: Add owner/editor access comments and helper in `apps/api/services/auth/AuthService.ts` (4 hours)[FE][BE]
    - API: Implement role-based field filtering examples in `apps/api/controllers/schema_controller.py` (4 hours)[FE][BE]
    - Docs: Add examples showing restricted data for owner and editor in `apps/api/docs/schemas/get_schema.md` (4 hours)[FE][BE]

    - Quality: Add tests to verify docs match behavior in `apps/api/tests/test_docs_access.py` (4 hours)[FE][BE][QA]
    - Quality: Create review checklist in `docs/review_checks/access_control.md` (4 hours)[FE][BE]
- **Validate payload examples for GET:** As a: API consumer, I want to: validate payload examples for GET requests, So that: I can ensure correctness of requests and prevent schema violations**(20 hours)** - Payload examples validate against JSON Schema with no violations All required fields present in payload examples Invalid payloads rejected with proper error messages Edge case: maximum field length and special characters handled Tests cover both positive and negative scenarios
    - Add JSON Schema file `apps/api/schemas/intervals/defaultFieldMappings.json` (4 hours)[FE][BE]
    - Implement JsonSchemaValidator.validate_example() in `apps/api/services/validation/JsonSchemaValidator.py` (4 hours)[FE][BE][QA]
    - Integrate validation into GET handler in `apps/api/routers/intervals.py` to validate examples and return errors (4 hours)[FE][BE][QA]
    - Create positive and negative tests in `apps/api/tests/test_intervals_schema.py` covering edge cases (4 hours)[FE][BE][QA]
    - Add schema path to settings in `apps/api/config/settings.py` and CI step in `.github/workflows/ci.yml` (4 hours)[FE][BE][DevOps]
- **Validate payload examples for GET: (0 hours)**
- **Authorization owner/editor doc: (0 hours)**
- **Add JSON examples: (20 hours)**
    - API: Create example JSON file in `apps/api/schemas/intervals_defaultFieldMappings_examples.json` (4 hours)[FE][BE]

- API: Update GET /schema/intervals/defaultFieldMappings in `apps/api/routers/schema_router.py` to include examples from `apps/api/schemas/intervals_defaultFieldMappings_examples.json` (4 hours) [FE][BE]
- API: Add OpenAPI example metadata in `apps/api/main.py` for intervals/defaultFieldMappings schema (4 hours)[FE] [BE]
- TEST: Add unit/integration test in `apps/api/tests/test_schema_examples.py` verifying examples present in GET response (4 hours)[FE][BE][QA]
- DOC: Update docs file `docs/api/schemas.md` with JSON examples and usage (4 hours)[FE][BE]

- **Example edge cases: (32 hours)**
  - API: Add GET /schema endpoint in `apps/api/app/main.py` returning intervals/defaultFieldMappings schema JSON (4 hours)[FE][BE]
  - API: Create schema file `apps/api/app/schemas/intervals_schema.json` with examples and edge cases (4 hours)[FE][BE]
  - API: Implement loader in `apps/api/app/services/schema_loader.py` to read `apps/api/app/schemas/intervals_schema.json` (4 hours)[FE][BE]
  - Frontend: Build fetcher in `apps/web/src/services/schemaService.ts` to GET /schema and validate response (4 hours)[FE][BE]
  - Frontend: Add example viewer component in `apps/web/src/components/SchemaViewer.tsx` to display schema and examples (4 hours)[FE][BE]
  - Testing: Add API unit tests in `apps/api/tests/test_schema_endpoint.py` covering edge cases (4 hours) [FE][BE][QA]

    - Testing: Add frontend tests in `apps/web/src/__tests__/SchemaViewer.test.tsx` for rendering and error states (4 hours)[FE][BE][QA]
    - Docs: Update API docs in `apps/api/docs/schema.md` with examples and edge case explanations (4 hours)[FE][BE]
- **GET schema: intervals/defaultFieldMappings examples: (20 hours)** - Response contains a valid JSON object with defaultFieldMappings structure Examples align with schema definitions for intervals No missing required fields in example payloads Examples include at least one edge case (empty or minimal mappings) Schema-example consistency verified against official JSON Schema
    - API: Add GET /schema/intervals/defaultFieldMappings handler in `apps/api/routes/schema.py` (4 hours)[FE][BE]
    - API: Implement schema generation logic in `apps/api/services/schema/IntervalSchemaService.py` (4 hours)[FE][BE]
    - Data: Add example payloads file `apps/api/services/schema/examples/intervals/defaultFieldMappings_examples.json` (4 hours)[FE][BE]
    - Test: Create unit tests in `apps/api/tests/test_schema_intervals.py` validating schema vs examples (4 hours)[FE][BE][QA]
    - Docs: Update OpenAPI components in `apps/api/docs/openapi_schema.yaml` with examples (4 hours)[FE][BE]
- **Validate intervals JSON:** As a: system, I want to: Validate intervals JSON, So that: Payload intervals adhere to expected schema and constraints**(32 hours)** - Intervals object is present and is an array Each interval has start and end in ISO8601 and start < end Invalid intervals are rejected with clear error

message System accepts valid intervals and stores them in payload structure Edge cases: overlapping intervals are either rejected or flagged

  - API: Add intervals schema definition in `apps/api/schemas/intervals_schema.py` (4 hours)[FE][BE]
  - API: Add request validation middleware in `apps/api/middleware/validation.py` to validate intervals payload (4 hours)[FE][BE][QA]
  - API: Implement validate_intervals() in `apps/api/services/validation/intervals_validator.py` to check ISO8601, start<end, overlap rules (4 hours)[FE][BE][QA]
  - API: Update PUT handler to use validator in `apps/api/routers/schema_router.py` (4 hours)[FE][BE]
  - DB: Add payload storage mapping update in `apps/api/models/payload_model.py` to store validated intervals (4 hours)[FE][BE]
  - TESTING: Add unit tests for intervals validator in `apps/api/tests/test_intervals_validator.py` (4 hours)[FE][BE][QA]
  - TESTING: Add integration tests for PUT /schema in `apps/api/tests/test_put_schema_endpoints.py` covering invalid/edge cases (4 hours)[FE][BE][QA]
  - DOCS: Update API docs/examples in `docs/api/put_schema.md` with intervals examples and error messages (4 hours)[FE][BE]

- **Validate defaultFieldMappings JSON:** As a: system, I want to: Validate defaultFieldMappings JSON, So that: Default mappings conform to expected keys and types**(24 hours)** - Mappings object is present Keys are from allowed set and values match expected types Invalid keys or types yield precise errors Valid mappings are accepted and preserved in payload

  - API Schema: Add default mappings Pydantic model in `apps/api/schemas/default_mappings.py` (4 hours)[FE][BE]

    - Service: Implement validate_default_field_mappings in `apps/api/services/schema/SchemaValidator.py` (4 hours) [FE][BE]
    - API Router: Integrate validation into PUT handler in `apps/api/routers/schemas.py` (4 hours)[FE][BE][QA]
    - Tests: Add unit tests for validator in `tests/validation/test_default_field_mappings.py` (4 hours)[FE][BE][QA]
    - Integration Test: Add PUT endpoint tests in `tests/integration/test_put_schema.py` (4 hours)[FE][BE][QA]
    - Docs: Update API docs in `docs/api/schemas.md` describing defaultFieldMappings schema and errors (4 hours)[FE][BE]
- **Auth owner or editor:** As a: editor or owner, I want to: Validate user has owner or editor roles for the resource, So that: Unauthorized users cannot modify the schema**(28 hours)** - User role checked against resource policy Only users with owner or editor role succeed Unauthorized users get 403 with clear message Audit log entry created for access attempts
    - DB: Add audit_logs table migration in `prisma/migrations/` (4 hours)[FE][BE]
    - API: Implement role check in `apps/api/services/auth/AuthService.ts` to validate owner or editor (4 hours)[FE][BE]
    - API: Add authorization middleware in `apps/api/middleware/auth.py` to call AuthService.check_owner_or_editor() (4 hours)[FE][BE]
    - API: Enforce authorization in `apps/api/routers/schema_router.py` PUT /schema handler and return 403 with clear message (4 hours)[FE][BE]
    - API: Create AuditService in `apps/api/services/logging/AuditService.py` to record access attempts (4 hours)[FE][BE]

    - TEST: Add tests in `tests/api/test_schema_put.py` for owner/editor allowed and others get 403 and audit log created (4 hours)[FE][BE][QA]
    - DOC: Update `docs/api/auth.md` with auth rules and error messages for PUT /schema (4 hours)[FE][BE]
- **Examples request/response:** As a: system, I want to: Provide example request and response payloads, So that: Consumers understand correct usage and formats**(20 hours)** - Examples include full request snippets including headers Response examples reflect successful and error cases Examples align with current schema definitions Examples tested against schema validation
    - API: Add example request/response JSON in `apps/api/openapi/examples/put_schema_examples.json` (4 hours)[FE][BE]
    - API: Reference examples in OpenAPI route `apps/api/routers/schemas.py` by adding examples to PUT operation docstring (4 hours)[FE][BE]
    - API: Implement request validation test in `apps/api/tests/test_put_schema_examples.py` using pydantic schemas (4 hours)[FE][BE][QA]
    - CI: Add GitHub Action step to run schema example tests in `.github/workflows/ci.yml` (4 hours)[FE][BE][QA]
    - Docs: Add examples to docs site in `apps/docs/content/schemas/put_examples.md` including headers and curl snippets (4 hours)[FE][BE]
- **Validate intervals JSON: (0 hours)**
- **Validate defaultFieldMappings JSON: (0 hours)**
- **Auth owner or editor: (0 hours)**
- **Examples request/response: (0 hours)**

- **GET not found error:** As a: user, I want to: retrieve a resource and receive a not found error when the resource does not exist, So that: I can understand proper error handling and respond with appropriate status code**(28 hours)** - System returns 404 Not Found when resource is missing Response body includes error code and message clarifying resource not found No server error occurs; error path is logged for monitoring Edge case: non-existent IDs return 404 within 200ms Validation of error payload structure adheres to API spec
    - API: Add 404Error schema in `apps/api/schemas/errors.py` (references: apps/api) (4 hours)[FE][BE]
    - API: Implement GET handler with 404 in `apps/api/routers/examples.py` (references: apps/api) (4 hours)[FE][BE]
    - API: Add logging for not-found path in `apps/api/utils/logging.py` (references: apps/api) (4 hours)[FE][BE]
    - Testing: Create unit tests for 404 response in `tests/api/test_examples_not_found.py` (references: apps/api) (4 hours)[FE][BE][QA]
    - Testing: Add integration/perf test ensuring 404 returned within 200ms in `tests/perf/test_examples_404_perf.py` (references: apps/api) (4 hours)[FE][BE][QA]
    - Frontend: Handle 404 error display in `apps/web/src/components/examples/ExampleView.tsx` (references: apps/api) (4 hours)[FE][BE]
    - Docs: Update API spec and examples in `docs/api/examples.md` with error payload schema (references: apps/api) (4 hours)[FE][BE]
- **GET validation error:** As a: user, I want to: retrieve a resource with invalid query parameters and receive a validation error, So that: I can adjust requests and receive clear guidance on correct usage**(36 hours)** - System returns 400 Bad Request when query parameters are invalid Response body includes validation error details per parameter Error

response follows API schema and includes helpful pointer/field info Logs capture invalid request for monitoring and rate-limiting triggers if repeated

    - API: Add Pydantic query schema in `apps/api/schemas/examples.py` (4 hours)[FE][BE]
    - API: Add validation middleware in `apps/api/middleware/validation.py` to enforce schema and raise 400 (4 hours)[FE][BE][QA]
    - API: Update router GET /examples in `apps/api/routes/examples.py` to use schema and return error response per API schema (4 hours)[FE][BE]
    - API: Implement error formatter in `apps/api/utils/errors.py` to produce field-level details and pointers (4 hours)[FE][BE]
    - API: Add logging for invalid requests in `apps/api/utils/logging.py` and hook from middleware (4 hours)[FE][BE]
    - API: Implement rate-limit trigger hook in `apps/api/middleware/rate_limit.py` for repeated invalid requests (4 hours)[FE][BE]
    - Frontend: Update example request handler in `apps/web/src/components/examples/ExampleList.tsx` to surface validation errors (4 hours)[FE][BE][QA]
    - Tests: Add unit/integration tests in `tests/api/test_examples_validation.py` covering 400 responses and error body (4 hours)[FE][BE][QA]
    - Docs: Update `docs/api/examples.md` and OpenAPI examples to show validation error schema (4 hours)[FE][BE][QA]

- **GET success response: (20 hours)**
    - API: Implement GET /examples success handler in `apps/api/app/main.py` (4 hours)[FE][BE]
    - API: Add response schema in `apps/api/app/schemas/example.py` (4 hours)[FE][BE]

    - Frontend: Create fetch hook in `apps/web/src/hooks/useFetchExamples.ts` (4 hours)[FE][BE]
    - Frontend: Build ExamplesList component in `apps/web/src/components/ExamplesList.tsx` (4 hours)[FE][BE]
    - Testing: Add integration test in `apps/api/tests/test_get_examples.py` (4 hours)[FE][BE][QA]
- **GET not found error: (0 hours)**
- **GET validation error: (0 hours)**
- **Validation error: invalid intervals:** As a: backend API client, I want to: receive clear validation errors for invalid intervals in PUT payload, So that: clients can correct the request and proceed successfully**(28 hours)** - The API returns 400 with specific error code for invalid intervals Error message clearly describes which intervals are invalid and why Validation runs for each interval in array and reports first 5 issues Response time under 200ms for validation path No side effects or data mutations on invalid payloads
    - API: Add IntervalValidator in `apps/api/services/validation/IntervalValidator.py` to validate interval formats, ranges, and overlaps; report issues with codes and messages (4 hours)[FE][BE][QA]
    - API: Update PUT handler in `apps/api/routes/examples/put_examples.py` to call IntervalValidator, return 400 with specific error code `INVALID_INTERVALS`, include first 5 issues (4 hours)[FE][BE]
    - API: Create error schema in `apps/api/schemas/errors.py` for interval validation errors with code, message, path (4 hours)[FE][BE][QA]
    - Testing: Add unit tests in `tests/api/test_put_examples_validation.py` covering invalid formats, ranges, overlaps, and ensuring no DB mutation (4 hours)[FE][BE][QA]

    - Integration: Add integration tests in `tests/integration/test_put_examples.py` to assert 400 response, payloads, and response time <200ms (4 hours)[FE][BE][QA]
    - Docs: Document error codes and examples in `docs/api/errors.md` and update API spec `apps/api/openapi.yaml` (4 hours)[FE][BE]
    - CI: Add performance validation step in `.github/workflows/ci.yml` to assert validation path timing <200ms (4 hours)[FE][BE][QA]
- **Validation error: invalid defaultFieldMappings:** As a: backend API client, I want to: receive precise validation errors for invalid defaultFieldMappings in PUT payload, So that: clients can fix mappings**(24 hours)** - The API returns 400 with detailed mapping error indicating invalid keys or values Validation checks defaultFieldMappings object against schema Errors include path to invalid mapping property No data is mutated on invalid input Latency of validation path under 250ms
    - API: Add schema for defaultFieldMappings validation in `apps/api/schemas/put_examples_schema.py` (4 hours)[FE][BE][QA]
    - API: Implement validation middleware in `apps/api/middleware/validation.py` to check defaultFieldMappings and return 400 with path info (4 hours)[FE][BE][QA]
    - API: Update PUT handler to use validation in `apps/api/routes/examples.py` and ensure no mutation of input payload (4 hours)[FE][BE][QA]
    - Testing: Add unit tests for schema in `apps/api/tests/test_put_validation.py` covering invalid keys/values and path in errors (4 hours)[FE][BE][QA]
    - Performance: Add benchmark test in `apps/api/tests/test_validation_performance.py` ensuring <250ms (4 hours)[FE][BE][QA]

    - Docs: Update API error docs in `docs/api/errors.md` with example invalid defaultFieldMappings responses (4 hours) [FE][BE]

- **Successful 200 response:** As a: registered client, I want to: receive a successful 200 response after PUT payload validation passes, So that: client knows the update succeeded**(36 hours)** - Server returns 200 OK with confirmation payload Response includes updated resource version No validation errors present Latency of response under 1000ms Audit log entry created for the update

    - DB: Create migration adding ‘version’ to resource and audit table in `prisma/migrations/` (4 hours)[FE][BE]

    - API: Implement PUT handler in `apps/api/routes/put_examples.py` returning 200 and confirmation payload (4 hours)[FE][BE]

    - Service: Add update_resource() in `apps/api/services/resource_service.py` to apply changes and return new version (4 hours)[FE][BE]

    - Validation: Implement validate_put_payload() in `apps/api/services/validation.py` ensuring no validation errors (4 hours)[FE][BE][QA]

    - Audit: Implement create_audit_entry() in `apps/api/services/audit_service.py` and write to audit table (4 hours) [FE][BE]

    - Tests: Add unit tests in `tests/unit/test_put_examples.py` to assert 200, payload and version (4 hours)[FE][BE][QA]

    - Integration: Add integration test in `tests/integration/test_put_flow.py` including latency check <1000ms (4 hours)[FE][BE][QA]

    - Infra: Add logging and env config in `apps/api/logging.py` and `apps/api/.env.example` for audit (4 hours)[FE][BE] [DevOps]

    - Docs: Update `docs/api.md` with PUT /examples response contract and version field (4 hours)[FE][BE]

- **Valid PUT payload: (36 hours)**
    - API: Add PUT /examples endpoint with validation in `apps/api/routers/examples.py` (api_development) (4 hours)[FE][BE][QA]
    - API: Create Pydantic model ExamplePutSchema in `apps/api/schemas/examples.py` with field types and validators (api_development) (4 hours)[FE][BE]
    - API: Implement update_example() in `apps/api/services/example_service.py` to apply validated PUT payload (api_development) (4 hours)[FE][BE]
    - DB: Add migration for examples table in `prisma/migrations/` if needed referencing users table `table_users` (api_development) (4 hours)[FE][BE]
    - Frontend: Build ExamplePutForm component in `apps/web/src/components/examples/ExamplePutForm.tsx` to send PUT payloads (frontend_component) (4 hours)[FE][BE]
    - Frontend: Add route /examples/edit to `apps/web/src/routes/examples.tsx` and integrate ExamplePutForm (frontend_component) (4 hours)[FE][BE]
    - Testing: Add integration test test_put_example_success in `apps/api/tests/test_examples.py` asserting 200 and valid response (testing) (4 hours)[FE][BE][QA]
    - Testing: Add frontend E2E test put_example_validation in `apps/web/tests/e2e/examples.spec.ts` to verify validation errors shown (testing) (4 hours)[FE][BE][QA]
    - Docs: Add PUT example payload docs in `docs/api/examples.md` with sample request/response (documentation) (4 hours)[FE][BE]

- **Validation error: invalid intervals: (0 hours)**
- **Validation error: invalid defaultFieldMappings: (0 hours)**

- **Successful 200 response: (0 hours)**

Milestone 5: UI & integration: mapping UI, dashboard, upload flows, banner and prepopulate presets tying frontend to API

Estimated 1133.5 hours

- **Advanced Overrides:** As a: data analyst, I want to: configure and apply advanced override rules during data mapping, So that: I can ensure higher fidelity mappings when legacy data formats require custom transformations**(52 hours)** - System allows defining custom override rules with at least 3 parameters Override rules can be saved and loaded from project scope Override application is logged with a timestamp and user for audit System prevents conflicting override rules from being saved Override rules apply in mapping preview results
    - DB: Add overrides table migration in `prisma/migrations/` referencing table_project_upload_presets and table_users (4 hours)[FE][BE]
    - API: Create overrides router and CRUD in `apps/api/routers/overrides.py` (4 hours)[FE][BE]
    - API: Implement OverrideService.save_override() in `apps/api/services/overrides/OverrideService.py` with conflict detection (4 hours)[FE][BE]
    - API: Implement apply_override() in `apps/api/services/overrides/OverrideService.py` and preview endpoint in `apps/api/routers/preview.py` (4 hours)[FE][BE]
    - DB: Add audit_log table migration in `prisma/migrations/` linked to table_users for override application logs (4 hours)[FE][BE]
    - API: Add audit logging in `apps/api/services/audit/AuditService.py` for override applications (log timestamp and user) (4 hours)[FE][BE]

    - Frontend: Build OverrideRuleEditor component in `src/components/flow_mapping/OverrideRuleEditor.tsx` to define rules with ≥3 parameters (4 hours)[FE][BE]
    - Frontend: Integrate overrides save/load in `src/store/slices/projectSlice.ts` and API calls in `src/services/api/overrides.ts` (4 hours)[FE][BE]
    - Frontend: Add mapping preview integration in `src/components/flow_mapping/MappingPreview.tsx` to apply overrides via preview endpoint (4 hours)[FE][BE]
    - Frontend: Prevent conflicting rules in UI in `src/components/flow_mapping/OverrideConflictChecker.tsx` and validation in `src/utils/overrides/validate.ts` (4 hours)[FE][BE][QA]
    - Testing: Add backend unit tests for OverrideService in `apps/api/tests/test_overrides.py` (4 hours)[FE][BE][QA]
    - Testing: Add frontend tests for OverrideRuleEditor and conflict checker in `src/__tests__/OverrideRuleEditor.test.tsx` (4 hours)[FE][BE][QA]
    - Docs: Document override rule format and storage in `docs/features/advanced_overrides.md` (4 hours)[FE][BE]
- **Outlier Review Queue:** As a: data steward, I want to: review and triage outlier records in a queue, So that: I can flag, edit, or discard anomalies to improve data quality**(40 hours)** - Outlier queue supports pagination and filtering by sensitivity and source Triage actions (flag, edit, discard) persist to history Preview shows impact of action on downstream models System logs decisions for audit and traceability System handles large volumes with responsive UI
    - DB: Add outlier_flags table migration in `prisma/migrations/2025_add_outlier_flags/` (4 hours)[FE][BE]
    - API: Create outliers router in `apps/api/routers/outliers.py` with pagination & filtering (4 hours)[FE][BE]

    - API: Implement OutlierService.triageAction() in `apps/api/services/outliers/OutlierService.py` to persist flag/edit/discard and record history (4 hours)[FE][BE]
    - Frontend: Build OutlierQueue component in `apps/web/components/outlier/OutlierQueue.tsx` with pagination, filters by sensitivity & source (4 hours)[FE][BE]
    - Frontend: Implement OutlierPreview modal in `apps/web/components/outlier/OutlierPreview.tsx` showing downstream model impact (4 hours)[FE][BE]
    - API: Add audit logging middleware in `apps/api/middleware/audit.py` to log decisions to `table_outlier_flags` history (4 hours)[FE][BE]
    - Infra: Configure Redis & Celery in `docker-compose.yml` and `apps/api/celery_worker.py` for preview async jobs (4 hours)[FE][BE][DevOps]
    - API: Implement preview computation endpoint in `apps/api/routers/outliers.py` that triggers Celery task `apps/api/tasks/compute_preview.py` (4 hours)[FE][BE]
    - Frontend: Connect preview flow to `apps/web/components/outlier/OutlierQueue.tsx` calling `/api/outliers/preview` and showing `OutlierPreview.tsx` (4 hours)[FE][BE]
    - QA: Add integration tests in `tests/api/test_outliers.py` and `tests/frontend/test_outlier_queue.spec.ts` for pagination, filters, triage actions (4 hours)[FE][BE][QA]
- **Make Review Mandatory Toggle:** As a: data steward, I want to: make the review step mandatory via a toggle in the data mapping UI, So that: users must review mappings before completion, improving data integrity**(48 hours)** - Toggle is present in UI and accessible Toggling enforces required review state across sessions Validation prevents proceeding without review when enabled State persists in user profile and session storage UI reflects current mandatory state after refresh
    - Frontend: Add ‘Mandatory Review’ Toggle UI in `components/mapping/ReviewToggle.tsx` (4 hours)[FE][BE]

- Frontend: Create Redux slice for review toggle in `store/slices/reviewToggleSlice.ts` (api_development) (4 hours) [FE][BE]
- Frontend: Persist toggle to sessionStorage in `utils/session/sessionStorage.ts` (frontend_component) (4 hours)[FE][BE]
- API: Add GET/PUT preference endpoints in `apps/api/routes/user_preferences.py` (api_development) (4 hours) [FE][BE]
- DB: Add user_preferences column to users table via migration in `prisma/migrations/` referencing `table_users` (documentation) (4 hours)[FE][BE]
- Backend: Implement preference read/write in `apps/api/services/user/UserPreferencesService.py` -> saves to `table_users` (api_development) (4 hours)[FE][BE]
- Integration: Hook Redux actions to API in `services/api/userPreferencesApi.ts` and call from `store/slices/reviewToggleSlice.ts` (api_development) (4 hours)[FE][BE]
- Frontend: Enforce validation in mapping completion flow in `components/mapping/MappingCompletionButton.tsx` to block when review required (frontend_component) (4 hours)[FE][BE][QA]
- Backend: Validate mapping completion request in `apps/api/routes/mapping.py` using `apps/api/services/user/UserPreferencesService.py` to enforce review requirement (api_development) (4 hours)[FE][BE]
- Testing: Add unit tests for Redux slice and component in `tests/frontend/review_toggle.test.tsx` (testing) (4 hours) [FE][BE][QA]
- Testing: Add backend tests for preferences endpoints in `tests/backend/test_user_preferences.py` (testing) (4 hours)[FE][BE][QA]

    - Docs: Update README and UI docs in `docs/flow_mapping_ui.md` describing toggle behavior (documentation) (4 hours)[FE][BE]

- **Review/Edit Intervals Banner:** As a: data analyst, I want to: review or edit the intervals banner displayed in the data mapping UI, So that: I can quickly adjust time-window intervals to align with data quality checks and validation rules**(36 hours)** - User can open the Review/Edit Intervals Banner from the mapping UI Banner loads with current interval values and is editable Changes to intervals are validated against allowed ranges and saved to the mapping profile System preserves previous interval values for rollback Acceptance: banner reflects updated intervals in the mapping results

    - DB: Add intervals column to `prisma/migrations/2025xxxx_add_mapping_intervals/` for table_project_upload_presets (table_project_upload_presets) (4 hours)[FE][BE]

    - API: Implement GET /mappings/{id}/intervals in `apps/api/routers/mappings.py` to read intervals (table_project_upload_presets) (4 hours)[FE][BE]

    - API: Implement PUT /mappings/{id}/intervals in `apps/api/routers/mappings.py` with validation and save to DB (table_project_upload_presets) (4 hours)[FE][BE][QA]

    - API: Implement rollback storage for intervals in `apps/api/services/mappings.py` to record previous values (table_project_upload_presets) (4 hours)[FE][BE]

    - Frontend: Build IntervalsBanner component in `src/components/mapping/IntervalsBanner.tsx` to open from mapping UI (frontend_route_mapping) (4 hours)[FE][BE]

    - Frontend: Add mapping redux slice and async thunks in `src/store/mappingSlice.ts` to fetch/save intervals (frontend_route_mapping) (4 hours)[FE][BE]

    - Integration: Wire component to API in `src/services/api/mappings.ts` to call GET/PUT endpoints (apps/api/routers/mappings.py) (4 hours)[FE][BE]
    - Testing: Add backend tests in `tests/api/test_mappings.py` for GET/PUT and rollback (table_project_upload_presets) (4 hours)[FE][BE][QA]
    - Testing: Add frontend tests in `src/__tests__/IntervalsBanner.test.tsx` to validate UI open/edit/validation/save (src/components/mapping/IntervalsBanner.tsx) (4 hours)[FE][BE][QA]
- **Unit Ambiguity Flagging:** As a: QA engineer, I want to: flag units with ambiguity in unit mappings, So that: data units lacking clear mapping can be reviewed and resolved**(32 hours)** - Ambiguity indicators appear next to affected units Flagging can be triggered manually or via heuristics Flagged units can be filtered or grouped Resolved flags clear the unit status Audit trail records flag changes and user actions
    - DB: Create outlier_flags migration in `prisma/migrations/20251001_add_outlier_flags/` (4 hours)[FE][BE]
    - API: Add flag endpoints in `apps/api/routers/flags.py` to create/list/update flags (4 hours)[FE][BE]
    - Service: Implement heuristic flagging in `apps/api/services/flags/FlagService.py` (4 hours)[FE][BE]
    - Frontend: Add ambiguity indicator component in `apps/web/src/components/mapping/AmbiguityBadge.tsx` (4 hours)[FE][BE]
    - Frontend: Add flagging UI and filters in `apps/web/src/routes/Mapping/UnitFlagsPanel.tsx` (4 hours)[FE][BE]
    - API: Add audit trail logging in `apps/api/services/audit/AuditService.py` and router `apps/api/routers/audit.py` (4 hours)[FE][BE]

    - Testing: Add unit & integration tests in `apps/api/tests/test_flags.py` and `apps/web/src/__tests__/UnitFlagsPanel.test.tsx` (4 hours)[FE][BE][QA]
    - Docs: Document flagging feature in `docs/flow_mapping_ui/unit_ambiguity_flagging.md` (4 hours)[FE][BE]
- **Outlier Threshold Override:** As a: data scientist, I want to: override outlier thresholds within the mapping UI, So that: I can adjust sensitivity for data quality checks during validation**(44 hours)** - Override control available in mapping UI Overridden thresholds apply to current validation pass Overrides are user-scoped and saved to the profile System validates new thresholds against sane bounds Audit log records threshold changes
    - DB: Add user_outlier_thresholds column in `prisma/migrations/` to update table_users profile (4 hours)[FE][BE]
    - API: Create outlier override endpoint POST /outlier/override in `apps/api/routers/outlier.ts` (4 hours)[FE][BE]
    - API: Implement OutlierService.saveOverride(userId, thresholds) in `apps/api/services/outlier/OutlierService.ts` (4 hours)[FE][BE]
    - API: Add threshold validation logic in `apps/api/services/outlier/OutlierService.ts` to enforce sane bounds (4 hours)[FE][BE][QA]
    - API: Write audit entry on override in `apps/api/services/audit/AuditService.ts` (4 hours)[FE][BE]
    - Frontend: Build ThresholdOverride component in `apps/web/src/components/mapping/ThresholdOverride.tsx` (4 hours)[FE][BE]
    - Frontend: Add Redux slice thresholdSlice in `apps/web/src/store/slices/thresholdSlice.ts` (4 hours)[FE][BE]
    - Frontend: Integrate override control into Mapping UI route in `apps/web/src/routes/MappingRoute.tsx` (4 hours)[FE][BE]

- Integration: Apply overridden thresholds to current validation pass in `apps/api/services/validation/ValidationService.ts` (4 hours)[FE][BE][QA]
- Tests: Add backend tests for override and bounds in `apps/api/tests/test_outlier_override.py` (4 hours)[FE][BE][QA]
- Tests: Add frontend tests for ThresholdOverride in `apps/web/tests/ThresholdOverride.test.tsx` (4 hours)[FE][BE][QA]

- **Column Mapping: (40 hours)**
  - DB: Create project_upload_presets migration in `prisma/migrations/` to add mapping_columns (4 hours)[FE][BE]
  - API: Implement ColumnMapping schema and endpoint in `apps/api/services/mapping/MappingService.ts` (4 hours)[FE][BE]
  - API: Add router for /mappings in `apps/api/routers/mapping_router.py` to call MappingService (4 hours)[FE][BE]
  - Frontend: Build ColumnMapper component in `src/components/mapping/ColumnMapper.tsx` (4 hours)[FE][BE]
  - Frontend: Add route /mapping in `src/routes/MappingRoute.tsx` and connect to ColumnMapper (4 hours)[FE][BE]
  - Frontend: Create Redux slice for mapping state in `src/store/slices/mappingSlice.ts` (4 hours)[FE][BE]
  - API: Implement mapping persistence in `apps/api/services/mapping/MappingService.ts` to store mapping into table_project_upload_presets (4 hours)[FE][BE]
  - Frontend: Implement CSV column preview parser in `src/utils/csvParser.ts` used by ColumnMapper (4 hours)[FE][BE]

    - Quality: Add unit tests for ColumnMapper in `src/__tests__/ColumnMapper.test.tsx` (4 hours)[FE][BE][QA]
    - Quality: Add integration test for mapping API in `apps/api/tests/test_mapping.py` (4 hours)[FE][BE][QA]
- **Unit Normalization: (32 hours)**
    - Frontend: Create UnitNormalizationForm component in `components/mapping/UnitNormalizationForm.tsx` (4 hours)[FE][BE]
    - Frontend: Add unit normalization Redux slice in `store/slices/unitNormalizationSlice.ts` (api_development) (4 hours)[FE][BE]
    - Frontend: Implement normalization utilities in `utils/normalization/unitNormalization.ts` (frontend_component) (4 hours)[FE][BE]
    - API: Add /normalize endpoint handler in `apps/api/routers/normalize.py` (api_development) (4 hours)[FE][BE]
    - API: Implement Normalization service in `apps/api/services/normalization/NormalizationService.py` (api_development) (4 hours)[FE][BE]
    - DB: Add outlier_flags update migration in `prisma/migrations/` referencing table_outlier_flags (documentation) (4 hours)[FE][BE]
    - Quality: Create unit tests for normalization utils in `tests/unit/test_unit_normalization.py` (testing) (4 hours)[FE][BE][QA]
    - Quality: Add integration tests for /normalize in `tests/integration/test_normalize_api.py` (testing) (4 hours)[FE][BE][QA]
- **Auto-Format Detection: (32 hours)**
    - Frontend: Build FormatDetector component in `src/components/mapping/FormatDetector.tsx` (4 hours)[FE][BE]

- API: Add /infer-format endpoint in `apps/api/routers/format_infer.py` (4 hours)[FE][BE]
- API Service: Implement infer_format() in `apps/api/services/format/FormatService.py` (4 hours)[FE][BE]
- Backend: Add format inference utilities in `apps/api/utils/format_detection.py` (4 hours)[FE][BE]
- DB: Add project_upload_presets migration in `prisma/migrations/` to store detected_format in table_project_upload_presets (4 hours)[FE][BE]
- Frontend: Integrate FormatDetector with mapping upload flow in `src/pages/mapping/UploadMappingPage.tsx` (4 hours)[FE][BE]
- Testing: Add unit tests for format detection in `tests/api/test_format_infer.py` and `src/__tests__/FormatDetector.test.tsx` (4 hours)[FE][BE][QA]
- Doc: Update mapping README in `docs/mapping/FORMAT_DETECTION.md` (4 hours)[FE][BE]

• **Outlier Flagging: (48 hours)**

- DB: Create outlier_flags table migration in `prisma/migrations/` (4 hours)[FE][BE]
- API: Add Prisma model for OutlierFlag in `prisma/schema.prisma` (4 hours)[FE][BE]
- API: Implement OutlierFlagService in `apps/api/services/outlier_flags/OutlierFlagService.py` (4 hours)[FE][BE]
- API: Create router endpoints in `apps/api/routers/outlier_flags.py` (4 hours)[FE][BE]
- Frontend: Build OutlierFlagPanel component in `apps/web/components/outlier/OutlierFlagPanel.tsx` (4 hours)[FE][BE]
- Frontend: Add page MappingOutliers in `apps/web/pages/MappingOutliers.tsx` and route (4 hours)[FE][BE]

    - Frontend: Implement API client in `apps/web/services/api/outlierFlags.ts` (4 hours)[FE][BE]
    - Frontend: Create Redux slice in `apps/web/store/outlierFlagsSlice.ts` (4 hours)[FE][BE]
    - Tasks: Add Celery task for asynchronous flagging in `apps/api/tasks/outlier_tasks.py` (4 hours)[FE][BE]
    - QA: Add API tests in `apps/api/tests/test_outlier_flags.py` (4 hours)[FE][BE][QA]
    - QA: Add frontend tests in `apps/web/tests/OutlierFlagPanel.test.tsx` (4 hours)[FE][BE][QA]
    - Docs: Create docs/outlier_flagging.md describing feature and API (4 hours)[FE][BE]
- **Missing Field Suggestions: (36 hours)**
    - DB: Create migration for outlier_flags in `prisma/migrations/2025xxxx_create_outlier_flags.sql` (4 hours)[FE][BE]
    - DB: Create migration for project_upload_presets in `prisma/migrations/2025xxxx_create_project_upload_presets.sql` (4 hours)[FE][BE]
    - API: Implement SuggestionService in `apps/api/services/suggestions/SuggestionService.ts` (4 hours)[FE][BE]
    - API: Add suggestions router in `apps/api/routers/suggestions.py` (4 hours)[FE][BE]
    - Frontend: Build SuggestionBox component in `src/components/mapping/SuggestionBox.tsx` (4 hours)[FE][BE]
    - Frontend: Integrate SuggestionBox into MappingEditor in `src/pages/flow_mapping_ui/MappingEditor.tsx` (4 hours)[FE][BE]
    - Testing: Add API tests in `tests/api/test_suggestions.py` (4 hours)[FE][BE][QA]

    - Testing: Add frontend tests in `tests/frontend/test_suggestion_ui.tsx` (4 hours)[FE][BE][QA]
    - Docs: Add docs in `docs/flow_mapping_ui/missing_field_suggestions.md` (4 hours)[FE][BE]
- **Project Default Rules: (40 hours)**
    - DB: Add project_default_rules table migration in `prisma/migrations/20251030_add_project_default_rules/` (4 hours)[FE][BE]
    - DB: Create Prisma model ProjectDefaultRule in `prisma/schema.prisma` (4 hours)[FE][BE]
    - API: Implement createDefaultRule() in `apps/api/services/projectRules/ProjectRulesService.ts` (4 hours)[FE][BE]
    - API: Add routes for project default rules in `apps/api/routers/project_rules.py` (4 hours)[FE][BE]
    - Frontend: Build DefaultRulesList component in `apps/web/src/components/project/DefaultRulesList.tsx` (4 hours)[FE][BE]
    - Frontend: Build DefaultRuleForm in `apps/web/src/components/project/DefaultRuleForm.tsx` (4 hours)[FE][BE]
    - Frontend: Add Redux slice projectDefaultRulesSlice in `apps/web/src/store/slices/projectDefaultRulesSlice.ts` (4 hours)[FE][BE]
    - Frontend: Add route /projects/:id/default-rules in `apps/web/src/routes/projects.tsx` (4 hours)[FE][BE]
    - Tests: Add API tests for project rules in `apps/api/tests/test_project_rules.py` (4 hours)[FE][BE][QA]
    - Tests: Add frontend component tests in `apps/web/src/__tests__/DefaultRules.test.tsx` (4 hours)[FE][BE][QA]
- **Advanced Overrides: (0 hours)**
- **Outlier Review Queue: (0 hours)**

- **Review/Edit Intervals: (0 hours)**
- **Make Review Mandatory: (0 hours)**
- **Unit Ambiguity Flagging: (0 hours)**
- **Outlier Threshold Override: (0 hours)**
- **Apply bovine presets:** As a: dataset editor, I want to: apply bovine presets, So that: I can quickly preconfigure datasets with domain-specific settings**(8 hours)** - User can apply a preset to the dataset with a single action Preset changes are reflected in dataset preview in real-time System validates that the preset is compatible with the current dataset type and flags incompatibilities
    - API: Add applyPresetToDataset mutation in apps/api/routers/mapping/presets.py with input validation, role-based access, and tracing integration. - (M) (1 hours)[FE][BE][QA]
    - API: Implement seedBovinePresets mutation in apps/api/routers/mapping/presets.py to seed presets into DB with validation and idempotency safeguards. - (S) (0.5 hours)[FE][BE][QA]
    - Service: Implement apply_preset() in apps/api/services/presets/PresetsService.py with compatibility checks - (M) (1 hours)[FE][BE]
    - DB: Add presets table migration in apps/api/migrations/versions/ and model in apps/api/models/presets.py - (S) (0.5 hours)[FE][BE]
    - Frontend: Build PresetsPicker component in apps/web/src/components/dataset/PresetsPicker.tsx (single-action apply) - (M) (1 hours)[FE][BE]
    - Frontend: Create PresetCompatibilityAlert in apps/web/src/components/dataset/PresetCompatibilityAlert.tsx - (S) (0.5 hours)[FE][BE]

    - Frontend: Add presetsSlice in apps/web/src/store/slices/presetsSlice.ts and actions to call applyPresetToDataset API - (S) (0.5 hours)[FE][BE]
    - Realtime: Wire onDatasetPreviewUpdate subscription in apps/web/src/hooks/useDatasetPreview.ts to update preview - (M) (1 hours)[FE][BE]
    - Integration: Connect PresetsPicker to presetsSlice in apps/web/src/pages/DatasetPreviewPage.tsx - (S) (0.5 hours)[FE][BE]
    - Tests: Add backend tests in apps/api/tests/test_presets.py for apply and compatibility - (S) (0.5 hours)[FE][BE][QA]
    - Tests: Add frontend tests in apps/web/src/__tests__/PresetsPicker.test.tsx for UI apply and alert - (S) (0.5 hours)[FE][BE][QA]
    - Docs: Document feature in docs/features/dataset-presets.md and API in docs/api/presets.md - (XS) (0.5 hours)[FE][BE]
- **Allow quick edit intervals:** As a: user, I want to: allow quick edit intervals, So that: I can adjust dataset parameters rapidly during preview**(8.5 hours)** - Users can adjust a set of common interval parameters Changes reflect in real-time in the preview panel Validation ensures intervals are within allowed ranges and not conflicting with existing settings
    - Frontend: Add IntervalControls component in `src/components/dataset/IntervalControls.tsx` preserving architecture references to Dataset UI and interval editing features. - (S) (0.5 hours)[FE][BE]
    - Frontend: Update DatasetPreviewPanel to consume intervals in `src/components/dataset/DatasetPreviewPanel.tsx` preserving DatasetPreviewPanel integration. - (M) (1 hours)[FE][BE]
    - API: Add updateInterval mutation handler in `apps/api/routers/mapping_ui/intervals.py` preserving API routing structure. - (M) (1 hours)[FE][BE]

    - API: Add fetchIntervals query in `apps/api/routers/mapping_ui/intervals.py` preserving API routing structure. - (M) (1 hours)[FE][BE]
    - Frontend/API: Implement realtime subscription onIntervalChanged in `src/services/api/intervals.ts` and `apps/api/routers/mapping_ui/intervals.py` preserving real-time data flow. - (L) (2 hours)[FE][BE]
    - Validation: Implement interval validation utils in `src/utils/intervalValidation.ts` preserving utility module for validators. - (S) (0.5 hours)[FE][BE][QA]
    - Integration: Wire IntervalControls to Redux store in `src/store/slices/intervalsSlice.ts` and `src/components/dataset/IntervalControls.tsx` preserving Redux integration. - (M) (1 hours)[FE][BE]
    - Testing: Add unit tests for validation in `tests/utils/test_intervalValidation.test.ts` preserving test suite location. - (XS) (0.5 hours)[FE][BE][QA]
    - Testing: Add integration tests for preview updates in `tests/integration/test_dataset_preview_intervals.test.ts` preserving test suite location. - (M) (1 hours)[FE][BE][QA]
- **Enforce modal for critical uploads:** As a: admin, I want to: enforce modal for critical uploads, So that: I ensure user focus and confirm critical actions**(6 hours)** - Modal appears for critical upload actions Users must confirm before proceeding State persists across navigation and is testable via automated tests
    - Frontend: Implement CriticalUploadModal component at src/components/uploads/CriticalUploadModal.tsx. This component renders a modal enforcing user confirmation before critical upload actions, wiring to redux state for visibility and to API layer for subsequent upload flow. Includes accessibility, keyboard handling, and responsive behavior. - (M) (1 hours)[FE][BE]

    - Frontend: Integrate modal trigger and route wiring in src/routes/DatasetPreviewRoute.tsx to conditionally open CriticalUploadModal based on dataset state and route transitions. Ensure that normal previews are unaffected and that critical uploads flow through the modal when required. - (S) (0.5 hours)[FE][BE]
    - State: Create redux slice in src/store/slices/criticalUploadSlice.ts to persist modal state across navigation, including visibility, datasetId, and flag for required confirmation. Provide actions to show/hide and to set dataset context. - (M) (1 hours)[FE][BE]
    - API: Add forceUpload mutation handler in apps/api/routers/upload_router.py to check confirmation flag before proceeding with upload. Enforce that without confirmation flag, mutation is rejected with clear error and status code 400/403 as appropriate. - (L) (2 hours)[FE][BE]
    - DB: Add review_required column migration in apps/api/migrations/versions/ to support critical upload flag on table_users or uploads table. The migration should be idempotent and reversible, with down migration. - (S) (0.5 hours)[FE][BE]
    - Tests: Add integration tests in tests/integration/test_critical_upload.py to verify modal enforcement and persistence, including modal visibility across navigation, dataset context preservation, and forced upload path when confirmation provided. - (M) (1 hours)[FE][BE][QA]
- **Preview upload mappings:** As a: data analyst, I want to: preview mappings of uploaded dataset fields to target schema, So that: I can verify that columns align before import**(7 hours)** - System loads uploaded dataset and displays column headers UI shows mapping suggestions and allows manual override Validation ensures all required target fields are mapped Error

handling for missing column mappings with clear message
Performance: mapping view loads within 2 seconds for datasets up to 1000 rows

- Frontend: Implement CSV/Excel parser and upload component in src/components/dataset/Upload.tsx, wired to API upload endpoint; supports parsing headers and sample rows, drag-and-drop, progress, and error handling. - (S) (0.5 hours)[FE][BE]
- API: Create upload endpoint in apps/api/routers/datasets.py to accept files and return headers, validating content type and size; on success, return extracted headers and sample rows for mapping preview. - (M) (1 hours)[FE][BE]
- API: Implement mapping suggestion service in apps/api/services/mapping.py, consuming headers and sample data to propose default target-field mappings; include configurable rules and fallback suggestions. - (M) (1 hours)[FE][BE][DevOps]
- Frontend: Build MappingView UI in src/components/dataset/MappingView.tsx with manual override controls, showing suggested mappings, allowing user adjustments, and exposing current mapping state to API validators. - (M) (1 hours)[FE][BE]
- API: Add mapping validation in apps/api/validators/mapping_validator.py to ensure required target fields are mapped, returning detailed error messages for missing fields and mismatches. - (M) (1 hours)[FE][BE][QA]
- Frontend: Add ErrorBanner component in src/components/common/ErrorBanner.tsx for missing mappings/errors, reusable across MappingView and Upload components. - (S) (0.5 hours)[FE][BE]
- Quality: Add tests for mapping flow in tests/test_mapping.py and performance test for 1000 rows, including unit and integration tests across Upload, MappingView, and validator. - (L) (2 hours)[FE][BE][QA]

- **Show non-blocking banner:** As a: user, I want to: show non-blocking banner, So that: I can see important notices without interrupting workflow**(9.5 hours)** - Banner appears without blocking main UI Banner can be dismissed or auto-hides after a duration Banner content is fetched asynchronously and does not delay initial render
    - Frontend: Build NonBlockingBanner component in src/components/banner/NonBlockingBanner.tsx with React + TypeScript, export default, self-contained styles import, prop types aligned with docs - (S) (0.5 hours)[FE][BE]
    - Frontend: Add banner styles in src/components/banner/NonBlockingBanner.module.css using CSS modules, responsive positioning and animation hooks referenced by component - (XS) (0.5 hours)[FE][BE]
    - State: Create banner slice in src/store/bannerSlice.ts (api_development) to manage banner state in Redux store with actions: show/hide/setContent and selector helpers - (M) (1 hours)[FE][BE]
    - API Client: Implement fetchBannerContent in src/api/bannerClient.ts (api_development) to retrieve banner data from backend API - (M) (1 hours)[FE][BE]
    - Integration: Connect NonBlockingBanner to Redux in src/components/banner/NonBlockingBanner.tsx (frontend_component) to read content from store and dispatch show/hide as needed - (M) (1 hours)[FE][BE]
    - UX: Add dismiss and auto-hide logic in src/components/banner/NonBlockingBanner.tsx (frontend_component) to auto-hide after timeout and provide dismiss button - (M) (1 hours)[FE][BE]
    - Feature Flag: Add feature toggle in src/config/featureFlags.ts and gate banner rendering (frontend_component) - (S) (0.5 hours)[FE][BE][DevOps]
    - Testing: Add unit tests for banner component in src/components/banner/__tests__/NonBlockingBanner.test.tsx (testing) - (M) (1 hours)[FE][BE][QA]

    - E2E: Add integration test in cypress/integration/banner_spec.ts (testing) - (L) (2 hours)[FE][BE][QA]
    - Docs: Document banner props and behavior in docs/features/non-blocking-banner.md (documentation) - (M) (1 hours)[FE][BE]
- **Quick edit intervals:** As a: data analyst, I want to: quickly edit data intervals (e.g., time groupings) in the preview, So that: I can adjust sampling or windowing before export**(5 hours)** - User can select an interval type (daily, hourly) and apply to preview Preview reflects updated intervals in real-time Edge case: invalid interval configurations show helpful error Saved interval settings persist for current session Performance: interval computation completes within 1 second for small datasets
    - Frontend: Implement IntervalSelector component in src/components/dataset/IntervalSelector.tsx using React and TypeScript. The component should manage interval selection state, expose onChange, support presets, keyboard navigation, and accessibility roles. Integrates with existing dataset page layout and consumes interval configuration from store/hooks. - (M) (1 hours)[FE][BE][DevOps]
    - Frontend: Add PreviewIntervalRenderer in src/components/dataset/PreviewIntervalRenderer.tsx to render a compact, read-only preview of selected intervals. It should consume interval data from IntervalSelector or session and present in a responsive layout. - (S) (0.5 hours)[FE][BE]
    - API: Create interval compute endpoint in apps/api/routers/intervals.py that computes derived interval data on the backend. Implement routing, input validation, and a compute function that accepts base intervals and returns computed metrics. - (M) (1 hours)[FE][BE][QA]

    - State: Implement session persistence for intervals in src/hooks/useSessionIntervals.ts to preserve user intervals across page reloads. Utilize localStorage or sessionStorage and provide API to read/write intervals, with fallback strategies. - (S) (0.5 hours)[FE][BE]
    - Validation: Add interval validation util in src/utils/intervals/validateInterval.ts that validates interval definitions (start <= end, non-overlapping, within allowed range). - (S) (0.5 hours)[FE][BE][QA]
    - Performance: Add memoized interval computation in src/utils/intervals/computeIntervals.ts to cache results for identical inputs and avoid recomputation. - (S) (0.5 hours)[FE][BE]
    - Testing: Add unit tests for computeIntervals in tests/utils/computeIntervals.test.ts - (XS) (0.5 hours)[FE][BE][QA]
    - QA: Add e2e test for UI preview in tests/e2e/interval_preview.spec.ts - (S) (0.5 hours)[FE][BE][QA]
- **Manage Upload Presets:** As a: project admin, I want to: manage upload presets, So that: I can control what file types and size limits are allowed for project uploads**(6.5 hours)** - User can create a new upload preset with defined name, allowed extensions, and size limit System validates allowed extensions and numeric size limit Preset appears in the preset list with correct metadata and can be edited or deleted Uploading a file uses the selected preset to enforce constraints Edge case: attempting to save a preset with duplicate name is rejected
    - DB: Create presets table migration in prisma migrations folder with proper schema and indices; ensure migrations are idempotent and seedable for presets usage - (XS) (0.5 hours)[FE][BE]
    - DB: Add preset_unit_normalizations seed in prisma seeds to populate normalization rules used by presets - (XS) (0.5 hours)[FE][BE]

    - API: Implement PresetsService.createPreset() in apps/api/services/presets/PresetsService.ts to handle validation, persistence, and uniqueness constraints - (M) (1 hours)[FE][BE][QA]
    - API: Add presets router in apps/api/routes/presets.py with create/read/update/delete endpoints - (S) (0.5 hours)[FE][BE]
    - Frontend: Build PresetsPanel component in components/project/settings/PresetsPanel.tsx (lists, edit/delete actions) - (M) (1 hours)[FE][BE]
    - Frontend: Build PresetForm component in components/project/settings/PresetForm.tsx (create/edit with validation) - (M) (1 hours)[FE][BE][QA]
    - API: Enforce preset constraints in upload endpoint apps/api/routes/uploads.py using apps/api/services/presets/PresetsService.ts - (M) (1 hours)[FE][BE]
    - Quality: Add backend tests in apps/api/tests/test_presets.py covering validation and duplicate name rejection - (S) (0.5 hours)[FE][BE][QA]
    - Quality: Add frontend tests in components/project/settings/__tests__/PresetsPanel.test.tsx covering UI create/edit/delete and display - (S) (0.5 hours)[FE][BE][QA]
    - Docs: Add presets docs in docs/project-settings/presets.md - (XS) (0.5 hours)[FE][BE]
- **Seed Preset for Projects:** As a: admin, I want to: seed a preset for projects, So that: default settings are available for new projects**(5.5 hours)** - Default preset exists for new projects New project creation automatically applies seed preset when no custom preset is chosen Seed preset can be

overridden per project Audit log records seed preset application Edge case: seed preset not applied when project inherits settings from template

    - DB: Add seed preset and default flag in prisma/migrations/2025XXXX_add_seed_preset/ and update prisma/schema.prisma - (M) (1 hours)[FE][BE]
    - API: Implement applySeedPresetDuringProjectCreation in apps/api/services/projects/ProjectService.ts - (M) (1 hours)[FE][BE]
    - Frontend: Integrate Preset Seeding Runner in apps/web/src/routes/project/settings/ProjectSettings.tsx (comp_project_settings_seed) to auto-apply UI - (S) (0.5 hours)[FE][BE]
    - API: Add endpoint toggle to override seed preset in apps/api/routes/projects/routers.ts and handler in apps/api/services/projects/ProjectService.ts - (M) (1 hours)[FE][BE]
    - API: Record audit log for seed application in apps/api/services/audit/AuditService.ts and migration in prisma/migrations/ - (S) (0.5 hours)[FE][BE]
    - Tests: Add integration tests for project creation seed behavior in apps/api/tests/integration/test_project_seed.py - (S) (0.5 hours)[FE][BE][QA]
    - Edge Case: Skip seed when inheriting template in apps/api/services/projects/ProjectService.ts and apps/web/src/routes/project/settings/ProjectSettings.tsx - (M) (1 hours)[FE][BE]

- **Configure Intervals Validation:** As a: admin, I want to: configure intervals validation, So that: interval data input is validated against business rules**(9 hours)** - Validation rules can be added/edited/removed Existing intervals validate against updated rules on save Invalid intervals are rejected

with user-friendly messages Changes persist and affect subsequent project operations Audit trail records validation rule changes

  - DB: Create validation_rules table migration in `prisma/migrations/` - (M) (1 hours)[FE][BE][QA]
  - API: Add ValidationRules router in `apps/api/routers/validation_rules.py` - (M) (1 hours)[FE][BE][QA]
  - API: Implement apply_validation_rules() in `apps/api/services/validation/ValidationService.py` - (M) (1 hours)[FE][BE][QA]
  - Frontend: Build ValidationRulesEditor component in `apps/web/src/components/project/settings/ValidationRulesEditor.tsx` - (M) (1 hours)[FE][BE][QA]
  - Frontend: Integrate ValidationRulesEditor into /project/settings in `apps/web/src/routes/project/settings/ProjectSettings.tsx` - (M) (1 hours)[FE][BE][QA]
  - API: Add audit trail logging in `apps/api/services/audit/AuditService.py` - (M) (1 hours)[FE][BE]
  - Backend: Add unit tests for validation logic in `apps/api/tests/test_validation_rules.py` - (M) (1 hours)[FE][BE][QA]
  - Frontend: Add UI tests for ValidationRulesEditor in `apps/web/tests/validation_rules_editor.test.tsx` - (M) (1 hours)[FE][BE][QA]
  - DB: Add sample interval data migration update in `prisma/migrations/` to trigger revalidation - (M) (1 hours)[FE][BE][QA]

- **Review Intervals Banner:** As a: project member, I want to: review the intervals banner, So that: I understand project cadence and upcoming milestones**(10 hours)** - Banner renders with current interval data Banner updates when interval settings change Accessed from project settings page and

persists across reloads No visual regression on banner across devices Edge case: no intervals configured shows a helpful placeholder

    - API: Create intervals route in `apps/api/src/routes/intervals.ts` - (M) (1 hours)[FE][BE]
    - DB: Add preset_unit_normalizations seed/migration in `prisma/migrations/` - (M) (1 hours)[FE][BE]
    - API: Implement IntervalsService in `apps/api/src/services/intervals/IntervalsService.ts` - (M) (1 hours)[FE][BE]
    - Frontend: Build IntervalsReviewBanner component in `apps/web/src/components/project-settings/IntervalsReviewBanner.tsx` - (M) (1 hours)[FE][BE]
    - Frontend: Add route integration in `apps/web/src/routes/project/SettingsRoute.tsx` to include comp_project_settings_intervals - (M) (1 hours)[FE][BE]
    - Persistence: Implement client cache & localStorage sync in `apps/web/src/hooks/useIntervals.ts` - (M) (1 hours)[FE][BE]
    - Testing: Add unit tests in `apps/web/src/components/project-settings/__tests__/IntervalsReviewBanner.test.tsx` - (M) (1 hours)[FE][BE][QA]
    - E2E: Add Cypress test in `apps/web/cypress/e2e/project_settings.cy.ts` for banner rendering and persistence - (M) (1 hours)[FE][BE][QA]
    - Accessibility: Verify responsive styles in `apps/web/src/components/project-settings/IntervalsReviewBanner.tsx` and update `apps/web/src/styles/theme.ts` if needed - (M) (1 hours)[FE][BE]
    - Docs: Document banner behavior in `docs/project-settings/intervals.md` - (M) (1 hours)[FE][BE]

- **Manage Upload Presets: (0 hours)**
- **Review Intervals Banner: (0 hours)**

- **Show Review CTA: (32 hours)**
  - Frontend: Build ReviewCTABanner component in `src/components/banner/ReviewCTABanner.tsx` (4 hours)[FE][BE]
  - Frontend: Add route/link to Review screen in `src/routes/AppRoutes.tsx` (4 hours)[FE][BE]
  - State: Create reviewBanner slice in `src/store/slices/reviewBannerSlice.ts` (api_development) (4 hours)[FE][BE]
  - Integration: Trigger banner from `src/components/banner/BannerContainer.tsx` and map to ReviewCTABanner (4 hours)[FE][BE]
  - API: Add telemetry endpoint in `apps/api/app/routers/telemetry.py` to record CTA impressions/clicks (4 hours)[FE][BE]
  - DB: Add telemetry_events table migration in `prisma/migrations/` referencing table_project_upload_preset_intervals (4 hours)[FE][BE]
  - Testing: Add unit tests for `src/components/banner/ReviewCTABanner.test.tsx` and `src/store/slices/reviewBannerSlice.test.ts` (4 hours)[FE][BE][QA]
  - Docs: Update README in `docs/feature/review_cta.md` documenting behavior and file paths (4 hours)[FE][BE]
- **Edit Intervals: (32 hours)**
  - DB: Add columns for intervals in `prisma/migrations/` migration file for project_upload_preset_intervals table (4 hours)[FE][BE]
  - API: Add PATCH /projects/{id}/intervals handler in `apps/api/routers/projects/intervals.py` (4 hours)[FE][BE]
  - API: Implement update_intervals() in `apps/api/services/projects/ProjectService.py` (4 hours)[FE][BE]

    - Frontend: Build EditIntervalsModal component in `src/components/flow/banner/EditIntervalsModal.tsx` (4 hours)[FE][BE]
    - Frontend: Add intervals slice and actions in `src/store/slices/intervalsSlice.ts` (4 hours)[FE][BE]
    - Frontend: Wire modal trigger in `src/components/flow/banner/BannerActions.tsx` (4 hours)[FE][BE]
    - Integration: Add e2e test for edit intervals in `tests/e2e/flow_edit_intervals.test.ts` (4 hours)[FE][BE][QA]
    - Docs: Update README and add API spec in `docs/api/projects_intervals.md` (4 hours)[FE][BE]
- **Auto-apply presets on upload:** As a: data analyst or system user, I want to: auto-apply preset configurations to bovine interval data upon upload, So that: uploaded data is immediately usable with standard presets without manual intervention**(36 hours)** - Uploaded file triggers preset application workflow automatically Presets are correctly applied to 100% of supported data types If presets fail, system logs error and falls back to default state User receives notification that presets were applied successfully
    - DB: Create migrations for presets tables in `prisma/migrations/20251030_add_presets_tables` (4 hours)[FE][BE]
    - API: Add upload endpoint and trigger in `apps/api/routers/upload.py` (4 hours)[FE][BE]
    - Backend: Implement ApplyPresetService.apply_presets() in `apps/api/services/presets/ApplyPresetService.py` (4 hours)[FE][BE]
    - Tasks: Create Celery task enqueue_preset_application in `apps/api/tasks/preset_tasks.py` (4 hours)[FE][BE]
    - Frontend: Update UploadForm handleUpload in `apps/web/components/upload/UploadForm.tsx` to call API and show progress (4 hours)[FE][BE]

    - Frontend: Implement NotificationService.notifyPresetApplied in `apps/web/services/notifications/NotificationService.ts` (4 hours)[FE][BE]
    - Infra: Add Celery config in `apps/api/celery.py` and Dockerfile updates in `Dockerfile` (4 hours)[FE][BE][DevOps]
    - Logging: Add preset application logging in `apps/api/utils/logging.py` and fallback handling in `apps/api/services/presets/ApplyPresetService.py` (4 hours)[FE][BE]
    - Quality: Add tests in `tests/api/test_presets.py` and `tests/frontend/test_upload.tsx` (4 hours)[FE][BE][QA]
- **Project override presets:** As a: project manager, I want to: override global presets at project level, So that: project-specific rules take precedence over defaults for interval presets**(40 hours)** - Project-level presets can override defaults Override is reflected in subsequent uploads and previews Audit log records preset overrides with timestamps and user Validation prevents invalid preset configurations
    - DB: Add project_upload_presets row & project_upload_preset_intervals migration in `prisma/migrations/` (4 hours)[FE][BE]
    - API: Create endpoints to GET/PUT project presets in `apps/api/routers/presets.py` (4 hours)[FE][BE]
    - Service: Implement override logic in `apps/api/services/presets/PresetsService.py` (4 hours)[FE][BE]
    - Frontend: Build ProjectPresetsForm component in `apps/web/src/components/presets/ProjectPresetsForm.tsx` (4 hours)[FE][BE]
    - State: Add Redux slice in `apps/web/src/store/slices/projectPresetsSlice.ts` (4 hours)[FE][BE]
    - Upload: Apply project presets during upload in `apps/api/services/upload/UploadService.py` (4 hours)[FE][BE]

    - Preview: Update preview generation in `apps/web/src/utils/preview/generatePreview.ts` to respect project presets (4 hours)[FE][BE]
    - Audit: Add audit log records for overrides in `apps/api/services/audit/AuditService.py` and `table_project_upload_presets` entries (4 hours)[FE][BE]
    - Validation: Implement preset validation in `apps/api/validators/presets_validator.py` and tests in `apps/api/tests/test_presets_validation.py` (4 hours)[FE][BE][QA]
    - Tests: Integration tests for upload/preview with overrides in `apps/api/tests/test_project_presets_integration.py` (4 hours)[FE][BE][QA]
- **Show non-blocking review banner:** As a: user, I want to: see a non-blocking banner suggesting review of presets after upload, So that: I can verify presets without interrupting workflow**(40 hours)** - Banner appears after upload with clear CTA to review presets Banner is non-blocking and does not block further actions Banner respects user dismissal preferences and does not reappear unnecessarily Presets preview is accessible from banner for quick validation
    - Frontend: Add ReviewBanner component in `src/components/ui/ReviewBanner.tsx` (4 hours)[FE][BE]
    - Frontend: Add Redux slice reviewBannerSlice in `src/store/slices/reviewBannerSlice.ts` (4 hours)[FE][BE]
    - Frontend: Trigger banner on upload success in `src/components/upload/UploadForm.tsx` (4 hours)[FE][BE]
    - Frontend: Create PresetsPreview modal in `src/components/upload/PresetsPreview.tsx` (4 hours)[FE][BE]
    - API: Add GET presets endpoint in `apps/api/routers/presets.py` (4 hours)[FE][BE]
    - API: Add user prefs endpoint to persist dismissal in `apps/api/routers/user_prefs.py` (4 hours)[FE][BE]

    - DB: Add optional migration for user_banner_prefs in `prisma/migrations/` (4 hours)[FE][BE]
    - Devops: Add telemetry event in `src/lib/analytics.ts` (4 hours)[FE][BE]
    - Testing: Add unit tests for banner in `src/__tests__/ReviewBanner.test.tsx` (4 hours)[FE][BE][QA]
    - Testing: Add API tests for prefs in `apps/api/tests/test_user_prefs.py` (4 hours)[FE][BE][QA]
- **Auto-apply presets: (0 hours)**
- **Show review banner: (0 hours)**
- **Project presets override: (0 hours)**
- **Upload Data:** As a: data analyst, I want to: upload data files into the dashboard from my local machine or cloud storage, So that: I can begin data processing and visualization workflows within the dashboard**(10 hours)** - User can successfully upload a data file (CSV/JSON) via the dashboard System validates file format and size limits (e.g., CSV/JSON, max 100MB) Uploaded data appears in the in-app data catalog with a preview Error handling for invalid formats or corrupted files is user-friendly and informative
    - DB: Create uploads table migration in `apps/api/migrations/create_uploads_table.py` - (M) (1 hours)[FE][BE]
    - API: Implement upload endpoint in `apps/api/routers/uploads.py` - (M) (1 hours)[FE][BE]
    - API: Build UploadService.save_file() in `apps/api/services/uploads/UploadService.py` with S3 integration` - (M) (1 hours)[FE][BE]
    - Frontend: Build UploadForm component in `src/components/uploads/UploadForm.tsx` and mount in `src/pages/dashboard/index.tsx` (route_dashboard_index) - (M) (1 hours)[FE][BE]

    - Frontend: Implement UploadPreview in `src/components/uploads/UploadPreview.tsx` showing CSV/JSON preview (route_dashboard_layout) - (M) (1 hours)[FE][BE]
    - Frontend: Add uploads redux slice in `src/store/uploadsSlice.ts` for state and async thunks - (M) (1 hours)[FE][BE]
    - Validation: Implement file validation (type/size) in `apps/api/services/uploads/UploadService.py` and client-side in `src/components/uploads/UploadForm.tsx` - (M) (1 hours)[FE][BE][QA]
    - DB/API: Insert upload record into `apps/api/models/uploads.py` and `table_uploads` in `apps/api/routers/uploads.py` upon successful save - (M) (1 hours)[FE][BE]
    - Testing: Add unit/integration tests in `tests/test_uploads.py` covering valid/invalid files and previews - (M) (1 hours)[FE][BE][QA]
    - Docs: Write feature docs in `docs/features/upload_data.md` and error message guidelines in `docs/errors/upload_errors.md` - (M) (1 hours)[FE][BE]
- **Data Curation:** As a: data steward, I want to: apply basic data curation (cleanup, deduplication, normalization) on uploaded data, So that: data quality improves for downstream analysis**(13 hours)** - User can run cleaning operations (trim spaces, normalize case) on selected fields Deduplicate records based on chosen key(s) Data quality report summarizes improvements and remaining issues Curation changes are reversible to a previous version
    - DB: Add curations table migration in apps/api/migrations/versions/ preserving existing schema conventions and ensuring backward compatibility with curation_version table. - (S) (0.5 hours)[FE][BE]
    - API: Create curation models in apps/api/models/curation.py reflecting domain object with relations to curation_version and audit fields. - (M) (1 hours)[FE][BE]

    - API: Implement curation service methods in apps/api/services/curation/CurationService.py providing create, run, dedupe, and report workflows. - (L) (2 hours)[FE][BE]
    - API: Add endpoints in apps/api/routers/curations.py for run_clean, dedupe, report, revert - (M) (1 hours)[FE][BE]
    - Worker: Create Celery task in apps/api/tasks/curation_tasks.py to run curation jobs using Pandas - (M) (1 hours)[FE][BE]
    - Frontend: Build CurationPanel component in apps/web/components/curation/CurationPanel.tsx - (M) (1 hours)[FE][BE]
    - Frontend: Add Redux slice in apps/web/store/slices/curationSlice.ts for job state and versions - (M) (1 hours)[FE][BE]
    - Frontend: Add route integration in apps/web/routes/dashboard/FlowDashboard.tsx linking to route_dashboard_index - (XS) (0.5 hours)[FE][BE]
    - DB: Store curation versions in apps/api/models/curation_version.py and table_curations - (M) (1 hours)[FE][BE]
    - API: Implement deduplication logic in apps/api/services/curation/dedupe.py using Pandas - (XL) (4 hours)[FE][BE]
- **Visualize Results:** As a: data analyst, I want to: visualize analysis results with charts and dashboards, So that: insights are communicated effectively**(8 hours)** - User can create and configure at least 3 chart types (bar, line, scatter) Charts render correctly from chosen dataset Interactive filters apply to visualizations Export visualizations as image/PDF is available
    - DB: Create visualizations table migration in apps/api/migrations/ to support storing visualization metadata, user associations, and chart preferences for VisualizeResults feature. - (S) (0.5 hours)[FE][BE]

    - API: Implement Visualizations router in apps/api/routes/visualizations.py to expose REST endpoints for listing, creating, retrieving, and deleting visualizations, wiring to VisualizationService and route-level validation. - (M) (1 hours)[FE][BE][QA]
    - API: Add VisualizationService in apps/api/services/visualization/VisualizationService.py implementing business logic for create, read, update, delete, and export preparation hooks, using repository layer and input validation. - (M) (1 hours)[FE][BE][QA]
    - Frontend: Build VisualizeResults page in src/pages/dashboard/VisualizeResults.tsx to render dashboards, fetch visualizations, and provide UI hooks for filtering and exporting visuals, integrating with ChartContainer and API routes. - (M) (1 hours)[FE][BE]
    - Frontend: Create ChartContainer and charts in src/components/dashboard/ChartContainer.tsx and src/components/dashboard/charts/ to render Line, Bar, and Scatter charts using a React chart library, driven by Visualization data. - (M) (1 hours)[FE][BE]
    - Frontend: Implement ChartControls.tsx (filters) to drive filtering like date range, chart type, and quick-filters wired to VisualizeResults API calls. - (S) (0.5 hours)[FE][BE]
    - Frontend/API: Connect VisualizeResults to API (fetch) in src/pages/dashboard/VisualizeResults.tsx -> uses apps/api/routes/visualizations.py. - (S) (0.5 hours)[FE][BE]
    - Feature: Implement export to image/PDF util in src/utils/export/chartExport.ts and backend export endpoint in apps/api/routes/visualizations.py - (L) (2 hours)[FE][BE]
    - Testing: Add API tests in tests/api/test_visualizations.py and frontend tests in src/__tests__/VisualizeResults.test.tsx - (S) (0.5 hours)[FE][BE][QA]
- **Seed Bovine Defaults:** As a: system administrator, I want to: seed default bovine configurations and parameters into the Dashboard seed data set, So that: the platform has baseline

data for analytics, testing, and onboarding.**(7.5 hours)** - System seeds default bovine configuration data on initial setup Seed data includes at least 3 default bovine records with sane defaults Seed operation idempotent and can run multiple times without duplication Data seeded is persisted to the dashboard database and available for display on charts Error handling reports seed status in logs and UI

- DB: Create bovines table migration in apps/api/migrations/ (add table_bovines -> cite table_samples as related) - (S) (0.5 hours)[FE][BE]
- DB Model: Create Bovine model in apps/api/models/ bovine.py (maps to table_bovines) – cite table_samples - (S) (0.5 hours)[FE][BE]
- API: Implement seeder in apps/api/seeders/ bovine_seeder.py (seed 3 default bovines, idempotent) – cite table_samples - (M) (1 hours)[FE][BE]
- API: Add seed router in apps/api/routers/seed.py to trigger seeding and report status – cite route_dashboard_index - (M) (1 hours)[FE][BE]
- Service: Add bovine service in apps/api/services/ bovine_service.py (upsert logic) – cite table_samples - (M) (1 hours)[FE][BE]
- Frontend: Add SeedStatus component in src/components/ seed/SeedStatus.tsx to show seed status/errors – cite route_dashboard_index - (S) (0.5 hours)[FE][BE]
- Frontend: Update dashboard index in src/routes/dashboard/ index.tsx to call apps/api/routers/seed.py and display SeedStatus – cite route_dashboard_index - (S) (0.5 hours) [FE][BE]
- Infra/Logging: Add logging utility in apps/api/utils/logger.py and log seeding outcomes – cite table_samples - (S) (0.5 hours)[FE][BE]

    - Testing: Add integration test in apps/api/tests/test_bovine_seeder.py to assert idempotency and persistence – cite table_samples - (M) (1 hours)[FE][BE][QA]
    - Docs: Document seeding operation in docs/seed_bovine.md including how to run apps/api/routers/seed.py - (M) (1 hours)[FE][BE]
- **Apply Preset on Upload:** As a: user, I want to: apply a predefined preset during file upload, So that: uploads are automatically configured according to my saved preferences. **(5.5 hours)** - User can select a preset before or during upload Preset is applied to upload automatically without errors System validates preset compatibility with file type and size Uploaded file metadata includes preset reference for traceability
    - DB migration: Add preset_id and preset_meta to apps/api/prisma/migrations/ migration for table_uploads preserving existing schema and ensuring backward compatibility with current Upload model - (S) (0.5 hours)[FE][BE]
    - API: Add applyPresetOnUpload endpoint in apps/api/routers/uploads.py to trigger preset application during upload flow, returning updated upload resource and status hooks - (S) (0.5 hours)[FE][BE]
    - API: Implement preset compatibility validation in apps/api/services/uploads/UploadService.py to ensure chosen preset is compatible with file type, size, and user permissions - (S) (0.5 hours)[FE][BE][QA]
    - Frontend: Add PresetSelector component in apps/web/src/components/uploads/PresetSelector.tsx for route_dashboard_index enabling preset selection UI - (S) (0.5 hours)[FE][BE]
    - Frontend: Integrate preset selection into UploadForm.tsx within route_dashboard_index to reflect and persist user preset choice - (S) (0.5 hours)[FE][BE]

    - API: Persist preset reference in upload record in apps/api/services/uploads/UploadService.py so that uploads carry preset metadata - (S) (0.5 hours)[FE][BE]
    - Frontend: Apply preset automatically on file drop in apps/web/src/components/uploads/UploadForm.tsx and update metadata - (S) (0.5 hours)[FE][BE]
    - Testing: Add integration tests for upload with preset in apps/api/tests/test_uploads.py and apps/web/src/tests/UploadForm.test.tsx - (M) (1 hours)[FE][BE][QA]
    - Docs: Document preset-on-upload behavior in docs/features/preset_on_upload.md - (M) (1 hours)[FE][BE]
- **Manage Presets UI:** As a: user, I want to: manage presets via a dedicated UI, So that: I can create, update, or delete presets to tailor my workflows.**(9 hours)** - UI allows create/read/update/delete operations for presets Changes persist in backend storage UI shows validation messages for invalid presets Presets list supports search and filter by name and type Access controls ensure only authorized users can modify presets
    - DB: Create presets table migration in prisma/migrations/. Create a new Prisma migration to add the presets table with fields (id, name, description, createdAt, updatedAt) and ensure it aligns with existing Prisma schema and migrate the database in a safe, idempotent way. - (XS) (0.5 hours)[FE][BE]
    - API: Add presets router and CRUD in apps/api/routers/presets.py. Implement endpoints for create, read, update, delete, and list, wiring to Prisma Client and Preset model. - (S) (0.5 hours)[FE][BE]
    - API: Implement Preset model and Prisma client usage in apps/api/models/preset.py. Define Prisma-backed model mapping and helper for CRUD operations. - (S) (0.5 hours)[FE][BE]

    - API: Add authorization middleware checks in apps/api/middleware/auth.py for presets endpoints. Enforce role-based access (e.g., admin) for create/update/delete, read maybe open. - (M) (1 hours)[FE][BE]
    - Frontend: Build PresetsList component in components/presets/PresetsList.tsx. Render list of presets from API and support basic UI state. - (S) (0.5 hours)[FE][BE]
    - Frontend: Build PresetForm component with validation in components/presets/PresetForm.tsx. Support create/update with form validation - (S) (0.5 hours)[FE][BE][QA]
    - Frontend: Integrate presets routes into /dashboard in routes/route_dashboard_index.tsx referencing route_dashboard_index - (S) (0.5 hours)[FE][BE]
    - Frontend: Implement search and filter logic in components/presets/PresetsList.tsx - (M) (1 hours)[FE][BE]
    - API: Add search/filter query params handling in apps/api/routers/presets.py - (M) (1 hours)[FE][BE]
    - Frontend: Add ACL UI controls using Clerk in components/presets/PresetsList.tsx and components/presets/PresetForm.tsx - (M) (1 hours)[FE][BE]
    - Tests: Add API tests for presets CRUD in apps/api/tests/test_presets.py - (M) (1 hours)[FE][BE][QA]
    - Tests: Add frontend component tests for PresetsList and PresetForm in tests/components/presets.test.tsx - (M) (1 hours)[FE][BE][QA]
- **Generate Sample Data:** As a: data analyst, I want to: generate sample dataset within the dashboard, So that: I can demo features without external data and test pipelines**(9 hours)** - User can generate a predefined sample dataset with selectable size and schema Sample data is created in a

temporary workspace and visible in data catalog Option to download sample data as CSV is available System ensures sample data does not impact real datasets

- DB: Create samples table migration in apps/api/migrations/ and reference table_samples. - (XS) (0.5 hours)[FE][BE]
- API: Implement POST /api/samples/generate in apps/api/routers/samples.py to trigger sample creation. - (S) (0.5 hours)[FE][BE]
- Service: Add generate_sample() in apps/api/services/samples/SampleService.py using Pandas/NumPy. - (M) (1 hours)[FE][BE]
- Worker: Create Celery task create_sample_workspace in apps/api/tasks/sample_tasks.py to build temp workspace and save CSV to S3. - (M) (1 hours)[FE][BE]
- Frontend: Build SampleGenerator component in apps/web/src/components/dashboard/SampleGenerator.tsx on route route_dashboard_index - (S) (0.5 hours)[FE][BE]
- Frontend: Add Redux slice samplesSlice in apps/web/src/store/slices/samplesSlice.ts to manage sample generation state - (S) (0.5 hours)[FE][BE]
- API: Implement GET /api/samples/:id in apps/api/routers/samples.py to fetch sample metadata for data catalog - (S) (0.5 hours)[FE][BE]
- Frontend: Add SampleCatalog item in apps/web/src/components/dashboard/SampleCatalogItem.tsx to display sample in data catalog (route_dashboard_index) - (S) (0.5 hours)[FE][BE]
- Frontend: Implement CSV download handler in apps/web/src/components/dashboard/SampleCatalogItem.tsx calling /api/samples/:id/download - (S) (0.5 hours)[FE][BE]
- API: Implement GET /api/samples/:id/download in apps/api/routers/samples.py streaming CSV from S3 or temp workspace - (L) (2 hours)[FE][BE]

- Testing: Add unit tests for apps/api/services/samples/SampleService.py in tests/api/test_samples.py - (S) (0.5 hours)[FE][BE][QA]
- Testing: Add integration test for sample generation end-to-end in tests/integration/test_sample_generation.py - (M) (1 hours)[FE][BE][QA]

- **Search References:** As a: researcher, I want to: search references within the dashboard, So that: I can quickly locate prior studies and related datasets**(9.5 hours)** - Search supports keywords and filters (author, year) Results display relevant references with metadata Clicking a reference opens detail view System handles empty results gracefully
  - DB: Add full-text index on references.title and references.authors in apps/api/migrations/ preserving migration context and enabling Elasticsearch-like search capabilities via PostgreSQL tsquery/GIN index for fast text search. - (S) (0.5 hours)[FE][BE]
  - API: Implement /api/references/search endpoint in apps/api/routers/references.py to accept query params, authenticate, validate inputs, and return structured reference results matching search criteria. - (M) (1 hours)[FE][BE]
  - API: Create search logic in apps/api/services/reference_search.py using Elasticsearch and SQL fallback to provide robust search results for references with ranking and snippet highlighting. - (L) (2 hours)[FE][BE]
  - API: Add saved references lookup in apps/api/routers/references.py to join table_saved_references ensuring user saved flags are reflected in results. - (M) (1 hours)[FE][BE]
  - Frontend: Build ReferenceSearchBar component in src/components/references/ReferenceSearchBar.tsx to capture query, filters, and trigger API search with debounced requests. - (S) (0.5 hours)[FE][BE]

    - Frontend: Build ReferenceResultsList component in src/components/references/ReferenceResultsList.tsx to render list of references with title, authors, and snippet plus loading/error states. - (S) (0.5 hours)[FE][BE]
    - Frontend: Build ReferenceDetailModal in src/components/references/ReferenceDetailModal.tsx to show full reference details when selected. - (XS) (0.5 hours)[FE][BE]
    - Frontend: Add references Redux slice in src/store/referencesSlice.ts with actions for search, setFilters, selectReference to manage search state and results globally. - (M) (1 hours)[FE][BE]
    - Integration: Wire search into DashboardPage at src/pages/dashboard/index.tsx route_dashboard_index to display search UI and render results within dashboard layout. - (M) (1 hours)[FE][BE]
    - Testing: Add unit tests for search API in apps/api/tests/test_references_search.py to validate endpoint, service, and edge cases. - (M) (1 hours)[FE][BE][QA]
    - Testing: Add frontend tests for ReferenceResultsList in src/components/references/__tests__/ReferenceResultsList.test.tsx to cover rendering and states. - (S) (0.5 hours)[FE][BE][QA]
    - Docs: Document API usage and frontend props in docs/features/search_references.md to provide developer guidance and usage examples. - (XS) (0.5 hours)[FE][BE]
- **Saved References:** As a: researcher, I want to: save references for later use, So that: I can build a personal reference library within the dashboard**(9 hours)** - Users can save references from search results Saved references persist across sessions Users can organize references into collections System shows a count of saved references in the header
    - DB: Create saved_references migration in apps/api/alembic/versions/ - (S) (0.5 hours)[FE][BE]

- DB: Add SavedReference model in apps/api/models/saved_reference.py - (S) (0.5 hours)[FE][BE]
- API: Implement service methods in apps/api/services/saved_references.py - (M) (1 hours)[FE][BE]
- API: Add router endpoints in apps/api/routers/saved_references.py - (M) (1 hours)[FE][BE]
- Frontend: Create Redux slice in src/store/savedReferencesSlice.ts - (M) (1 hours)[FE][BE]
- Frontend: Build SavedReferencesPanel in src/features/savedReferences/SavedReferencesPanel.tsx - (M) (1 hours)[FE][BE]
- Frontend: Build CollectionsView in src/features/savedReferences/CollectionsView.tsx - (M) (1 hours)[FE][BE]
- Frontend: Add save button integration in src/features/search/SearchResultItem.tsx - (S) (0.5 hours)[FE][BE]
- Frontend: Display saved count in header src/components/header/Header.tsx - (S) (0.5 hours)[FE][BE]
- Tests: Add API tests in tests/test_saved_references_api.py - (M) (1 hours)[FE][BE][QA]
- Tests: Add frontend tests in tests/test_saved_references_ui.tsx - (M) (1 hours)[FE][BE][QA]

- **Apply Upload Preset: (28 hours)**
  - Frontend: Add ‘Apply Preset’ button in `src/components/dashboard/UploadRow.tsx` (4 hours)[FE][BE]
  - Frontend: Implement applyPreset action in `src/store/uploads/uploadsSlice.ts` (api_development) (4 hours)[FE][BE]
  - API: Create POST /uploads/{id}/apply-preset handler in `apps/api/routers/uploads.py` (4 hours)[FE][BE]

- Backend: Implement apply_preset() in `apps/api/services/uploads/UploadService.py` (4 hours)[FE][BE]
- DB: Add project_upload_default_mappings update in `prisma/migrations/` or `apps/api/db/migrations/` (4 hours)[FE][BE]
- Frontend: Update DashboardPage to call applyPreset in `src/pages/dashboard/DashboardPage.tsx` (frontend_component) (4 hours)[FE][BE]
- Testing: Add integration test for apply preset in `apps/api/tests/test_uploads.py` (testing) (4 hours)[FE][BE][QA]

- **Preview Curated Rows: (24 hours)**
  - API: Add get_curated_rows handler in `apps/api/routers/curations.py` (4 hours)[FE][BE]
  - API: Implement query in `apps/api/services/curation/CurationService.ts` to fetch curated rows from `table_curations` and `table_uploads` (4 hours)[FE][BE]
  - Frontend: Build PreviewCuratedRows component in `src/components/dashboard/PreviewCuratedRows.tsx` (4 hours)[FE][BE]
  - Frontend: Integrate PreviewCuratedRows into dashboard page `src/pages/dashboard/index.tsx` (route_dashboard_index) (4 hours)[FE][BE]
  - Testing: Add API unit tests in `apps/api/tests/test_curations.py` (4 hours)[FE][BE][QA]
  - Testing: Add component tests in `src/components/dashboard/__tests__/PreviewCuratedRows.test.tsx` (4 hours)[FE][BE][QA]

- **Toggle Review Required: (36 hours)**
  - DB: Add ‘review_required’ boolean column migration in `prisma/migrations/` for table `table_curations` (4 hours)[FE][BE]

    - DB: Update Prisma model in `apps/api/prisma/schema.prisma` to include review_required on model Curations (table_curations) (4 hours)[FE][BE]
    - API: Implement toggle endpoint PATCH `/curations/{id}/review_required` in `apps/api/routers/curations.py` (4 hours)[FE][BE]
    - API: Add service method toggleReviewRequired(id, value) in `apps/api/services/curations/CurationService.py` (4 hours)[FE][BE]
    - Frontend: Build ReviewToggle component in `apps/web/src/components/dashboard/ReviewToggle.tsx` referencing route_dashboard_index (4 hours)[FE][BE]
    - Frontend: Update DashboardPage in `apps/web/src/pages/dashboard/DashboardPage.tsx` to include ReviewToggle and map to curations data (route_dashboard_index) (4 hours)[FE][BE]
    - State: Add curations slice and async thunk toggleReviewRequired in `apps/web/src/store/curationsSlice.ts` (4 hours)[FE][BE]
    - Testing: Add backend tests in `apps/api/tests/test_curations.py` for PATCH /curations/{id}/review_required (table_curations) (4 hours)[FE][BE][QA]
    - Testing: Add frontend unit test in `apps/web/src/__tests__/ReviewToggle.test.tsx` for ReviewToggle component (route_dashboard_index) (4 hours)[FE][BE][QA]
- **Upload CSV/Excel:** As a: user, I want to: upload CSV/Excel files, So that: I can bring data into the system for processing**(32 hours)** - User can select CSV or Excel file and initiate upload System validates file format and size Uploaded

file is stored in a temporary staging area with a reference ID
Unsupported formats are rejected with a helpful error message

- Frontend: Build UploadControls component in `components/data-upload/UploadControls.tsx` to allow CSV/Excel selection (accept attribute) and start upload (4 hours)[FE][BE]
- Frontend: Add Upload & Sample Data Panel in `components/data-upload/UploadPanel.tsx` to show upload progress, sample link, and error messages (uses comp_upload_panel) (4 hours)[FE][BE]
- API: Implement getPresignedUploadUrls in `apps/api/routers/data_upload.py` to return presigned S3 URLs (router_route_data_upload) (4 hours)[FE][BE]
- API: Implement uploadFile mutation in `apps/api/routers/data_upload.py` to create staging record in `prisma/migrations/` and `table_uploads` with reference ID and metadata (router_route_data_upload, table_uploads) (4 hours)[FE][BE]
- Backend: Add server-side validation service in `apps/api/services/validation/UploadValidator.py` to validate format and size, reject unsupported formats (router_route_data_upload) (4 hours)[FE][BE][QA]
- Infra: Configure S3 upload staging bucket and presigned policy in `infra/s3/config.py` and GitHub Actions deploy secrets (4 hours)[FE][BE][DevOps]
- Testing: Add unit tests for UploadValidator in `apps/api/services/validation/tests/test_upload_validator.py` and integration tests for `apps/api/routers/data_upload.py` endpoints (4 hours)[FE][BE][QA]
- Docs: Add usage and error messages to `docs/features/data_upload.md` and update route `route_data_upload` docs (4 hours)[FE][BE]

- **Validate & Curate Upload: auto-normalize formats & units; flag outliers (signal only, exclude from downstream use): (32 hours)** - System validates required columns present Data quality checks (nulls, duplicates) performed Invalid rows are reported with line numbers and reasons Curated dataset stored as cleaned version in staging area
    - API: Add validateUpload endpoint in `apps/api/routers/data_upload.py` to validate required columns and trigger processing (4 hours)[FE][BE]
    - Backend: Implement validation worker in `apps/api/services/validation/validate_worker.py` to run Pandas checks (nulls, duplicates) and line-numbered reports (4 hours)[FE][BE][QA]
    - Backend: Implement normalization rules processor in `apps/api/services/normalization/normalizer.py` to auto-normalize formats & units using getUnitConversionMap API (4 hours)[FE][BE]
    - Backend: Implement outlier detection module in `apps/api/services/validation/outliers.py` to flag outliers (signal only) and mark for exclusion (4 hours)[FE][BE][QA]
    - API: Add curateUpload mutation in `apps/api/routers/data_upload.py` to store curated dataset in staging and create validation report record (4 hours)[FE][BE][QA]
    - DB: Create validation_reports entry in `prisma/migrations/` and update `apps/api/models/validation_report.py` to store line numbers and reasons (4 hours)[FE][BE][QA]
    - Frontend: Add validation summary UI in `apps/web/src/components/data_curation/ValidationSummary.tsx` to display invalid rows with line numbers and reasons (uses fetchValidationResults) (4 hours)[FE][BE][QA]
    - Frontend: Add preview curated dataset button in `apps/web/src/components/data_curation/DatasetPreview.tsx` to call fetchDatasetPreview and show staged cleaned data (4 hours)[FE][BE]

- **Preview Dataset:** As a: user, I want to: preview dataset, So that: I can verify data before final submission**(28 hours)** - Preview shows first N rows and summary stats Column headers display correctly Sorting/filtering in preview works Changes in mapping reflect in preview
    - API: Add fetchDatasetPreview endpoint handler in `apps/api/routers/data_upload/preview.py` to return first N rows and summary stats (4 hours)[FE][BE]
    - API: Add fetchPreviewRows query in `apps/api/routers/data_upload/preview.py` to support sorting/filtering params (4 hours)[FE][BE]
    - Frontend: Build DatasetPreview component in `apps/web/src/components/dataPreview/DatasetPreview.tsx` to render table and stats (4 hours)[FE][BE]
    - Frontend: Update DataUploadPage route component in `apps/web/src/pages/DataUploadPage.tsx` to call fetchDatasetPreview via Redux thunk in `apps/web/src/store/dataUpload/thunks.ts` (4 hours)[FE][BE]
    - State: Create Redux slice and thunks in `apps/web/src/store/dataUpload/previewSlice.ts` to store preview rows, headers, sort/filter state (4 hours)[FE][BE]
    - Integration: Wire mapping changes to preview in `apps/web/src/components/mapping/MappingEditor.tsx` to dispatch preview refresh action in `apps/web/src/store/dataUpload/previewSlice.ts` (4 hours)[FE][BE]
    - Testing: Add frontend tests in `apps/web/src/components/dataPreview/DatasetPreview.test.tsx` and API tests in `apps/api/tests/test_preview.py` to verify rows, headers, sorting/filtering, mapping updates (4 hours)[FE][BE][QA]
- **Suggest Missing Fields: (40 hours)**
    - DB: Add suggested_fields column to `apps/api/prisma/migrations/2025_add_suggested_fields` for table `table_uploads` (4 hours)[FE][BE]

- API: Implement POST /uploads/:id/suggest in `apps/api/routers/uploads.py` to generate suggestions (4 hours)[FE][BE]
- Service: Add suggestMissingFields() in `apps/api/services/validation/SuggestService.py` using Pandas to infer columns (4 hours)[FE][BE][QA]
- Worker: Add Celery task `apps/api/tasks/suggest_fields.py` to run suggestMissingFields asynchronously and write to `table_validation_reports` (4 hours)[FE][BE][QA]
- Frontend: Build SuggestFieldsPanel component in `apps/web/src/components/data/SuggestFieldsPanel.tsx` wired to route `route_data_upload` (4 hours)[FE][BE]
- Frontend: Add Redux slice in `apps/web/src/store/slices/suggestSlice.ts` and actions to call `POST /uploads/:id/suggest` (4 hours)[FE][BE]
- Frontend: Integrate SuggestFieldsPanel into `/data/upload` page file `apps/web/src/pages/DataUploadPage.tsx` (4 hours)[FE][BE]
- API: Create DB record in `table_validation_reports` from suggestion results in `apps/api/services/validation/SuggestService.py` (4 hours)[FE][BE][QA]
- Testing: Add unit tests for SuggestService in `apps/api/tests/test_suggest_service.py` (4 hours)[FE][BE][QA]
- Docs: Document API and frontend usage in `docs/features/suggest_missing_fields.md` (4 hours)[FE][BE]

- **Upload CSV/Excel: (0 hours)**
- **Auto Unit Detection: (36 hours)**
  - API: Add endpoint POST /uploads/{id}/detect-units in `apps/api/routers/uploads.py` (4 hours)[FE][BE]
  - Backend: Implement unit detection job in `apps/api/services/processing/unit_detection.py` (4 hours)[FE][BE]

    - DB: Add detection_result columns to `prisma/migrations/` for `table_uploads` (4 hours)[FE][BE]
    - Frontend: Add Detect Units button to `apps/web/src/pages/DataUploadPage.tsx` (route_data_upload) (4 hours)[FE][BE]
    - Frontend: Show detection results component in `apps/web/src/components/UnitDetection/Results.tsx` (route_data_upload) (4 hours)[FE][BE]
    - Worker: Create Celery task detect_units in `apps/api/tasks/detect_units.py` (4 hours)[FE][BE]
    - API: Add service method start_detection in `apps/api/services/uploads/UploadService.ts` (4 hours)[FE][BE]
    - Testing: Add unit tests for detection logic in `apps/api/tests/test_unit_detection.py` (4 hours)[FE][BE][QA]
    - Docs: Document API and frontend in `docs/features/auto_unit_detection.md` (4 hours)[FE][BE]
- **Format Auto-correction: (0 hours)**
- **Suggest Missing Fields: (32 hours)**
    - Frontend: Add ‘Suggest Missing Fields’ button in `apps/web/src/components/data-curation/FieldMapping.tsx` (4 hours)[FE][BE]
    - API: Implement POST /suggest-missing-fields in `apps/api/routers/data_upload.py` (4 hours)[FE][BE]
    - API: Create service logic suggest_missing_fields() in `apps/api/services/data_suggestions.py` (4 hours)[FE][BE]
    - DB: Add migration for project_upload_default_mappings in `prisma/migrations/` to store suggested mappings (4 hours)[FE][BE]

- Frontend: Call /suggest-missing-fields from `apps/web/src/components/data-curation/FieldMapping.tsx` and display suggestions in `apps/web/src/components/data-curation/SuggestionsList.tsx` (4 hours)[FE][BE]
- API: Add fetchExistingMappings query handler in `apps/api/routers/data_upload.py` to include suggested mappings (4 hours)[FE][BE]
- Integration: Persist accepted suggestions via createOrUpdateMapping mutation in `apps/api/routers/data_upload.py` (4 hours)[FE][BE]
- Testing: Add unit tests for suggest_missing_fields in `apps/api/tests/test_data_suggestions.py` (4 hours)[FE][BE][QA]

• **Outlier Flagging (signal only): (0 hours)**

• **Map Fields: (32 hours)**

- API: Add endpoint POST /uploads/{id}/map-fields in `apps/api/routers/uploads.py` to save field mappings (4 hours)[FE][BE]
- DB: Add migration for project_upload_default_mappings in `prisma/migrations/` to store default mappings in table_project_upload_default_mappings (4 hours)[FE][BE]
- Frontend: Build MapFieldsModal component in `apps/web/src/components/data/MapFieldsModal.tsx` and connect to route_data_upload (4 hours)[FE][BE]
- Frontend: Add mapping reducer in `apps/web/src/store/slices/mappingSlice.ts` for saving temporary mappings (4 hours)[FE][BE]
- API: Implement service to apply mappings in `apps/api/services/uploads/MappingService.py` and create validation_reports entries in table_validation_reports (4 hours)[FE][BE][QA]

    - Frontend: Update DataUploadPage at `apps/web/src/pages/DataUploadPage.tsx` to open MapFieldsModal and POST mappings to route in `apps/api/routers/uploads.py` (4 hours)[FE][BE]
    - Testing: Add frontend integration test in `apps/web/tests/MapFieldsModal.test.tsx` to verify mapping flow (4 hours)[FE][BE][QA]
    - Documentation: Document mapping API and frontend usage in `docs/data_upload/map_fields.md` (4 hours)[FE][BE]
- **Preview Dataset: (0 hours)**

Milestone 6: Architecture SVG and SOW docs: generate architecture SVG, migration README, curl examples and docs for migration and handlers

Estimated 499 hours

- **Add Migration:** As a: database administrator, I want to: add a new DB migration script for architecture SVG related endpoints and ingestion, So that: the database schema aligns with the backend diagram and supports new endpoints.**(3.5 hours)** - Migration script file exists and is named clearly Migration can be executed against a local DB without errors Migration includes reversible steps or rollback plan Migration updates schema version in metadata Migration test runs verify schema changes do not affect existing data
    - DB: Create migration file prisma/migrations/20251030_add_columns_schema.sql preserving existing Prisma migration conventions and ensuring idempotent SQL for adding new columns and constraints in the target database. - (XS) (0.5 hours)[FE][BE]
    - Service: Add migration runner in apps/api/scripts/run_migration.py to execute Prisma migrations against the local DB and report status, using existing project tooling and logging conventions. - (S) (0.5 hours)[FE][BE]

    - Service: Implement update schema version in apps/api/services/migrations/MigrationService.py to track applied schema version in a migrations table and expose update functionality. - (M) (1 hours)[FE][BE]
    - DB: Add rollback script prisma/migrations/20251030_add_columns_rollback.sql to revert the changes introduced by 20251030_add_columns_schema.sql, preserving transactional safety. - (XS) (0.5 hours)[FE][BE]
    - Test: Add migration tests in apps/api/tests/test_migrations.py to validate applying migrations, rollback, and schema version updates. - (M) (1 hours)[FE][BE][QA]
    - Infra: Add GitHub Actions job .github/workflows/migrations.yml to run migrations against local DB in CI, including setup of DB, running migrations, and reporting results. - (S) (0.5 hours)[FE][BE][DevOps]
- **Add Seed Script:** As a: database administrator, I want to: add a seed script to populate initial data for the architecture SVG related tables, So that: development and tests have predictable baseline data.**(7 hours)** - Seed script exists and can be executed to populate data Seeding covers essential tables impacted by migration Seed results are idempotent and verifiable Seed data matches expected counts and constraints Seed execution does not violate existing data integrity
    - DB: Create idempotent seed script for users and related tables in `apps/api/prisma/seed.ts` - (M) (1 hours)[FE][BE]
    - DB: Add Prisma migration and seed config in `apps/api/prisma/schema.prisma` and `package.json` scripts - (M) (1 hours)[FE][BE][DevOps]
    - API: Add endpoint fetchSeedScripts in `apps/api/routers/architecture_svg_router.py` to retrieve seed status (router_route_architecture_svg) - (M) (1 hours)[FE][BE]
    - API: Add mutation createSeedScript in `apps/api/routers/architecture_svg_router.py` to trigger seed run (router_route_architecture_svg) - (M) (1 hours)[FE][BE]

    - Testing: Add integration tests for seed idempotency and counts in `apps/api/tests/test_seed.py` - (M) (1 hours)[FE][BE][QA]
    - Docs: Document seed run and verification steps in `docs/seed.md` and update README` - (M) (1 hours)[FE][BE]
    - Infra: Add GitHub Actions workflow `./.github/workflows/seed.yml` to run seeds in CI against test DB - (M) (1 hours)[FE][BE][QA]

- **Add Migration 20251030_create_project_upload_presets:** As a: database administrator, I want to: Add a migration named 20251030_create_project_upload_presets to create presets table for project uploads, So that: Projects can define upload presets for file handling in the backend.**(3.5 hours)** - Migration file exists with correct name 20251030_create_project_upload_presets SQL creates presets table with required columns (id, name, maxSize, allowedTypes) Migration runs without errors in test environment Existing migrations remain reversible and consistent Presets table is accessible via migration history and can be queried
    - DB: Create migration dir `prisma/migrations/20251030_create_project_upload_presets/` and add `migration.sql` creating presets(id UUID DEFAULT uuid_generate_v4(), name text NOT NULL, maxSize int NOT NULL, allowedTypes text[- (S) (0.5 hours)[FE][BE]
    - DB: Update `prisma/schema.prisma` with model Preset and run `prisma migrate dev –name 20251030_create_project_upload_presets` in `apps/api/` - (M) (1 hours)[FE][BE]
    - TEST: Add migration test in `apps/api/tests/migrations/presets_migration.test.ts` to apply migration, query presets table, and rollback - (M) (1 hours)[FE][BE][QA]
    - DOC: Add rollback and migration notes in `prisma/migrations/README.md` and reference migration `20251030_create_project_upload_presets` - (S) (0.5 hours)[FE][BE]

    - CI: Add `/.github/workflows/migration-test.yml` step to run migration tests in CI for `apps/api/` - (S) (0.5 hours)[FE][BE][QA]

- **Add SVG with uuid_generate_v4():** As a: backend engineer, I want to: Add an SVG rendering endpoint that uses uuid_generate_v4() for unique IDs, So that: Each SVG element and diagram node can be uniquely identified in the architecture SVG backend.**(6 hours)** - SVG endpoint returns valid SVG payload with proper UUIDs for nodes uuid_generate_v4() is invoked for each new diagram element creation No collision or duplication of UUIDs on repeated requests SVG persists in an in-memory or persistent store with UUIDs preserved System handles invalid payloads gracefully without breaking existing diagrams

    - DB: Apply migration at prisma/migrations/2025_add_svg_uuid/migration.sql to create Svg table with UUIDs generated via uuid_generate_v4() for new rows. - (XS) (0.5 hours)[FE][BE]

    - API: Implement REST endpoint POST /architecture/svg in apps/api/routes/architecture/svg.ts to accept SVG payload and persist via SvgRepo with generated UUIDs per element. - (M) (1 hours)[FE][BE]

    - Service: Add generateSvgWithUUIDs() in apps/api/services/svg/SvgService.ts implementing per-element uuid_generate_v4() invocation for each element in the SVG payload. - (M) (1 hours)[FE][BE]

    - DB: Create Svg model and repository in apps/api/models/Svg.ts and apps/api/repos/SvgRepo.ts to persist SVG with UUIDs. - (M) (1 hours)[FE][BE]

    - API: Add validation middleware for SVG payload in apps/api/middleware/validateSvg.ts to enforce required fields and UUID presence. - (S) (0.5 hours)[FE][BE][QA]

    - Frontend: Add /architecture/svg route component in apps/web/routes/architecture/svg.tsx to call endpoint and render SVG. - (S) (0.5 hours)[FE][BE]

    - Testing: Write integration tests for SVG endpoint in apps/api/tests/architecture_svg.test.ts to assert UUID uniqueness and persistence. - (M) (1 hours)[FE][BE][QA]
    - Monitoring: Add logging of UUID generation in apps/api/services/svg/SvgService.ts and metrics in apps/api/monitoring/metrics.ts. - (S) (0.5 hours)[FE][BE]
- **PUT curl example: update upload preset: (28 hours)**
    - API: Implement updateUploadPreset mutation in `apps/api/services/upload_presets/UploadPresetsService.py` (4 hours)[FE][BE]
    - API Router: Add PUT handler in `apps/api/routers/project/upload_presets.py` to call UploadPresetsService.update_upload_preset (4 hours)[FE][BE]
    - Frontend: Add PUT curl example section component in `apps/web/src/routes/api/project/upload-presets/PutExample.tsx` (4 hours)[FE][BE]
    - Frontend: Update API Preset Edit Component in `apps/web/src/components/api/UploadPresetEdit.tsx` to include example usage and call PUT endpoint (4 hours)[FE][BE]
    - Tests: Add backend unit test in `apps/api/tests/test_upload_presets.py` for updateUploadPreset (4 hours)[FE][BE][QA]
    - Tests: Add frontend test in `apps/web/src/__tests__/PutExample.test.tsx` to verify example renders and invokes PUT (4 hours)[FE][BE][QA]
    - Docs: Document PUT curl example in `docs/api/upload_presets.md` with sample curl command (4 hours)[FE][BE]
- **PUT curl example - retrieve and update upload preset: (0 hours)**

- **PUT curl example - validation error handling: (28 hours)**
  - API: Add validation and error responses in `apps/api/routers/project/upload_presets.py` (4 hours)[FE][BE][QA]
  - API: Implement updateUploadPreset() in `apps/api/routers/project/upload_presets.py` with detailed error codes (4 hours)[FE][BE]
  - Frontend: Build PUT Curl Example UI in `apps/web/src/components/PutCurlExample/PutCurlExample.tsx` showing validation errors (4 hours)[FE][BE][QA]
  - Frontend: Hook PUT example to route component `route_api_project_upload_presets` in `apps/web/src/routes/ProjectUploadPresets.tsx` (4 hours)[FE][BE]
  - Testing: Add API integration tests for validation errors in `apps/api/tests/test_upload_presets.py` (4 hours)[FE][BE][QA]
  - Testing: Add frontend component tests for error rendering in `apps/web/src/components/PutCurlExample/PutCurlExample.test.tsx` (4 hours)[FE][BE][QA]
  - Docs: Update API docs example for PUT curl in `docs/api/project_upload_presets.md` with validation error samples (4 hours)[FE][BE][QA]
- **GET upload preset curl: (28 hours)**
  - API: Implement getUploadPresetCurl handler in `apps/api/routers/router_route_api_project_upload_presets.py` (4 hours)[FE][BE]
  - API: Add fetchUploadPresetExample service in `apps/api/services/uploadPresetsService.py` (4 hours)[FE][BE]
  - Frontend: Build GET Curl Example Section component in `apps/web/src/components/api/GetCurlExample.tsx` (4 hours)[FE][BE]

    - Frontend: Create API client method getUploadPresetsCurl in `apps/web/src/services/api/uploadPresets.ts` (4 hours) [FE][BE]
    - Tests: Add API unit test in `apps/api/tests/test_upload_presets.py` (4 hours)[FE][BE][QA]
    - Tests: Add frontend test in `apps/web/tests/test_GetCurlExample.test.tsx` (4 hours)[FE][BE][QA]
    - Docs: Add curl example docs in `docs/api/upload_presets.md` (4 hours)[FE][BE]
- **GET upload-preset curl example: (0 hours)**
- **Generate migration files: (24 hours)**
    - DB: Update Prisma schema in `prisma/schema.prisma` to add users table fields (4 hours)[FE][BE]
    - DB: Create migration using `npx prisma migrate dev –name add_users_table` and save under `prisma/migrations/` (4 hours)[FE][BE]
    - DB: Add seed script `prisma/seed.ts` and seed users in `prisma/seed.ts` (4 hours)[FE][BE]
    - API: Add migration runner in `apps/api/services/migrations/MigrationService.py` to invoke prisma migrate and seed (4 hours)[FE][BE]
    - Infra: Add npm script in `package.json` and CI step in `.github/workflows/ci.yml` to run migrations and seed (4 hours)[FE][BE]
    - Quality: Add migration tests in `tests/migrations/test_migrations.py` (4 hours)[FE][BE][QA]
- **Add seed per project: (28 hours)**
    - DB: Create per-project seed folder `apps/api/prisma/seed/{project}` and seed template in `apps/api/prisma/seed/_template.py` (4 hours)[FE][BE]

    - DB: Add seed runner script in `apps/api/prisma/seed/run_seeds.py` to load per-project seeds (4 hours)[FE][BE]
    - API: Implement SeedService.trigger_project_seed(project_id) in `apps/api/services/seed/SeedService.py` (4 hours)[FE][BE]
    - API: Add /seed/:project_id endpoint in `apps/api/routers/seed_router.py` to call SeedService (4 hours)[FE][BE]
    - CLI: Add management CLI command in `apps/api/manage_seed.py` to run `apps/api/prisma/seed/run_seeds.py` (4 hours)[FE][BE]
    - Testing: Create integration test `apps/api/tests/test_seed.py` to verify per-project seed creates data in `table_users` (4 hours)[FE][BE][QA]
    - Docs: Document seed usage in `docs/seed.md` with examples and safety notes` (4 hours)[FE][BE]
- **Include review notice: (28 hours)**
    - Frontend: Add ReviewNotice component in `src/components/flow/ReviewNotice.tsx` (4 hours)[FE][BE]
    - API: Add endpoint to fetch review flags in `apps/api/routers/flow/review.py` (4 hours)[FE][BE]
    - DB: Create migration note file in `prisma/migrations/` documenting review_notice column on `table_users` (4 hours)[FE][BE]
    - State: Add reviewNotice slice in `src/store/slices/reviewNoticeSlice.ts` (4 hours)[FE][BE]
    - Integration: Wire ReviewNotice into `src/pages/FlowPage.tsx` and `src/routes.tsx` (4 hours)[FE][BE]
    - Tests: Add unit tests for ReviewNotice in `tests/components/ReviewNotice.test.tsx` (4 hours)[FE][BE][QA]
    - Docs: Update README and release notes in `docs/release_notes.md` about review notice (4 hours)[FE][BE]

- **Validate intervals & mappings: (32 hours)**
  - API: Add PUT /example endpoint in `apps/api/routes/example.py` (4 hours)[FE][BE]
  - API: Define request schema in `apps/api/schemas/example.py` with intervals and defaultFieldMappings validation (4 hours)[FE][BE][QA]
  - Service: Implement validate_intervals_and_mappings() in `apps/api/services/example/ExampleService.py` (4 hours)[FE][BE]
  - DB Check: Add query to verify related users in `apps/api/services/example/ExampleService.py` using table_users (4 hours)[FE][BE]
  - Frontend: Build ExampleForm component in `apps/frontend/components/example/ExampleForm.tsx` to send PUT payload (4 hours)[FE][BE]
  - Frontend: Add Redux action in `apps/frontend/store/exampleSlice.ts` for PUT example request (4 hours)[FE][BE]
  - Tests: Add backend unit tests in `apps/api/tests/test_example.py` for schema and service (4 hours)[FE][BE][QA]
  - Tests: Add frontend tests in `apps/frontend/tests/ExampleForm.test.tsx` for mapping and validation (4 hours)[FE][BE][QA]
- **Confirm 200 response: (24 hours)**
  - API: Add PUT /examples endpoint in `apps/api/routers/examples.py` to return 200 on valid payload (4 hours)[FE][BE]
  - API: Create Pydantic schema in `apps/api/schemas/example.py` for intervals & defaultFieldMappings (4 hours)[FE][BE]

    - Service: Implement validate_and_process(payload) in `apps/api/services/example_service.py` (4 hours)[FE][BE]
    - Tests: Add integration test asserting 200 in `tests/api/test_examples.py` using `tests/fixtures/example_payload.json` (4 hours)[FE][BE][QA]
    - Frontend: Update client PUT call in `apps/web/src/api/examples.ts` to handle 200 response (4 hours)[FE][BE]
    - Docs: Document endpoint and 200 contract in `docs/api.md` (4 hours)[FE][BE]
- **PUT example response: 200 with preset: (36 hours)** - Response status code is 200 for valid input Response payload matches the predefined preset structure and fields No error messages in valid flow; error path not triggered Performance: response returned within 200ms under load test Security: response does not leak sensitive information in body
    - DB: Create preset field migration in `prisma/migrations/20251030_add_preset_to_examples.sql` (4 hours)[FE][BE]
    - API: Add response schema in `apps/api/schemas/example.py` defining preset structure (4 hours)[FE][BE]
    - API: Implement PUT /examples handler in `apps/api/routes/examples.py` to return 200 with preset (4 hours)[FE][BE]
    - Service: Create mapper populate_preset in `apps/api/services/example_service.py` (4 hours)[FE][BE]
    - Middleware: Add validation for intervals & defaultFieldMappings in `apps/api/middleware/validation.py` (4 hours)[FE][BE][QA]
    - Testing: Add unit tests in `tests/test_examples_put.py` for 200 response and payload match (4 hours)[FE][BE][QA]
    - Testing: Add load test in `tests/load/test_examples_put_load.py` to verify <200ms under load (4 hours)[FE][BE][QA]

    - Frontend: Update mock request in `apps/web/src/components/ExampleForm.tsx` to use PUT and preset payload (4 hours)[FE][BE]
    - Docs: Update API docs in `docs/api.md` for PUT /examples response 200 with preset (4 hours)[FE][BE]
- **PUT example response: Upsert succeeds:** As a: system, I want to: ensure the upsert operation succeeds and returns a confirmation payload, So that: the caller can rely on a successful operation and appropriate response data**(24 hours)** - Operation completes without error on upsert Response includes success flag and upserted IDs No data loss during upsert Error handling path returns meaningful messages
    - DB: Create upsert migration for presets in `prisma/migrations/` to ensure idempotent inserts (4 hours)[FE][BE]
    - API: Implement PUT /examples upsert handler in `apps/api/routers/examples.py` to call service and return payload (4 hours)[FE][BE]
    - Service: Implement upsert_example() in `apps/api/services/example_service.py` with transaction and return upserted IDs (4 hours)[FE][BE]
    - Response: Add response schema in `apps/api/schemas/example_schemas.py` including success flag and ids (4 hours)[FE][BE]
    - Testing: Add unit and integration tests in `tests/test_examples_upsert.py` covering success, partial failure, and data integrity (4 hours)[FE][BE][QA]
    - Docs: Update API docs in `docs/api/examples.md` with example PUT request and success response (4 hours)[FE][BE]
- **Include intervals & mappings:** As a: system, I want to: include intervals and mappings in the upsert payload, So that: the caller has full context for scheduling and data relationships**(36 hours)** - Intervals data present for relevant

presets Mappings data correctly reference related entities Payload validation ensures intervals/mappings align with schema No missing mappings or orphan references

    - DB: Create presets and mappings migration in `prisma/migrations/` - add tables table_presets, table_intervals, table_mappings (4 hours)[FE][BE]
    - DB: Add Prisma models in `prisma/schema.prisma` for Preset, Interval, Mapping referencing table_presets and table_users (4 hours)[FE][BE]
    - API: Implement upsert_presets endpoint in `apps/api/routes/presets.py` to accept intervals and mappings payload (4 hours)[FE][BE]
    - API: Implement validation in `apps/api/services/presetService.py` - validate intervals/mappings schema and reference integrity (4 hours)[FE][BE][QA]
    - API: Add DB persistence logic in `apps/api/services/presetService.py` - upsert intervals into `table_intervals` and mappings into `table_mappings` (4 hours)[FE][BE]
    - Frontend: Update PresetEditor in `components/presets/PresetEditor.tsx` to include intervals and mappings fields and payload builder (4 hours)[FE][BE]
    - Frontend: Add payload validation in `components/presets/validation.ts` to mirror backend schema (4 hours)[FE][BE][QA]
    - Tests: Add unit tests in `apps/api/tests/test_presets.py` for validation and upsert behavior (4 hours)[FE][BE][QA]
    - Docs: Update API docs in `apps/api/docs/presets.md` with intervals and mappings examples (4 hours)[FE][BE]

- **Return 200 payload:** As a: system, I want to: return a 200 OK payload with the upserted preset data, So that: the caller receives the necessary data and status in a standard HTTP response**(24 hours)** - HTTP status 200 is returned for

successful upsert Payload contains upserted preset data Content-Type is application/json No sensitive data exposed in payload

    - DB: Create presets table migration in `prisma/migrations/` or `alembic/versions/` (4 hours)[FE][BE]
    - API: Implement upsert_preset() in `apps/api/services/presets/PresetService.py` (4 hours)[FE][BE]
    - API: Add PUT /presets/{id} handler in `apps/api/routers/presets.py` to call PresetService.upsert_preset and return 200 JSON (4 hours)[FE][BE]
    - Schema: Create/Update `apps/api/schemas/preset.py` to serialize preset and exclude sensitive fields (4 hours)[FE][BE]
    - Testing: Add integration test in `tests/test_presets.py` to assert 200 status, JSON Content-Type, and payload contains upserted preset data (4 hours)[FE][BE][QA]
    - Docs: Update OpenAPI in `apps/api/openapi.yaml` and README to document 200 payload for PUT /presets/{id} (4 hours)[FE][BE]

- **PUT example response: Upsert succeeds: (0 hours)**
- **Return 200 payload: (0 hours)**
- **Include intervals & mappings: (0 hours)**
- **Preset include intervals:** As a: data consumer, I want to: retrieve presets with interval configuration included, So that: I can understand timing windows for each preset and schedule actions accordingly.**(3.5 hours)** - Each preset in the response includes intervals field Intervals are in valid format and within allowed ranges System returns 0-30 minute intervals or equivalent per preset No duplicates in intervals per preset

    - DB: Add presets.intervals column migration in `prisma/migrations/20251001_add_presets_intervals/` - (XS) (0.5 hours)[FE][BE]

    - API: Add intervals field to example schema in `apps/api/schemas/example_schema.py` - (XS) (0.5 hours)[FE][BE]
    - API: Implement generate_intervals() in `apps/api/services/preset/PresetService.py` to produce 0-30min non-duplicate intervals - (M) (1 hours)[FE][BE]
    - API: Update GET /example handler in `apps/api/routes/example.py` to include presets.intervals using `apps/api/services/preset/PresetService.py` - (S) (0.5 hours)[FE][BE]
    - TEST: Add unit tests for intervals format and uniqueness in `apps/api/tests/test_presets.py` - (S) (0.5 hours)[FE][BE][QA]
    - Frontend: Render intervals in `apps/web/components/presets/PresetList.tsx` and fetch via `apps/web/services/apiClient.ts` - (S) (0.5 hours)[FE][BE]
    - DOC: Update API docs in `docs/api/examples.md` to state presets.intervals contract - (XS) (0.5 hours)[FE][BE]
- **List presets retrieval:** As a: data consumer, I want to: list available presets for a project, So that: I can display and select presets for the project workflow.**(4.5 hours)** - User can fetch a list of presets for a given projectId Response includes at least one preset and its basic fields (id, name) System handles invalid projectId with appropriate error Response time under 1s for small datasets
    - DB: Add presets table migration in prisma/migrations/ to support presets linked to users via relation; Prisma ORM integration in MVP stack. - (XS) (0.5 hours)[FE][BE]
    - API: Implement GET /projects/{projectId}/presets in apps/api/routers/presets.py using FastAPI; fetch via PresetsService and return 200 with array; handle 404 for missing project. - (M) (1 hours)[FE][BE]
    - API: Add Preset model and query in apps/api/models/preset.py with relation to table_users; reflect ORM relation and ensure proper joins. - (M) (1 hours)[FE][BE]

    - API: Add service method fetchPresets(projectId) in apps/api/services/presets/PresetsService.py to retrieve presets for a project, returning list of PresetDTOs. - (S) (0.5 hours) [FE][BE]
    - Frontend: Implement fetchPresets in src/services/presets/presetsApi.ts and component use in src/pages/ProjectPresets.tsx to display presets after API call. - (S) (0.5 hours)[FE][BE]
    - Testing: Add integration test for GET /projects/{projectId}/presets in tests/integration/test_presets.py covering success and error paths. - (S) (0.5 hours)[FE][BE][QA]
    - Quality: Add API docs and error handling in apps/api/routers/presets.py and apps/api/exceptions.py to document endpoint and standardized errors. - (S) (0.5 hours)[FE][BE]
- **Preset include default mappings:** As a: data consumer, I want to: receive presets with default mappings included, So that: I can apply default field mappings without extra requests**(4 hours)** - Each preset includes a defaultMappings object Mappings cover at least 2 essential fields Defaults are applied when mappings are missing Validation ensures mappings reference existing fields
    - DB: Add presets table migration with defaultMappings column in prisma/migrations/ - (S) (0.5 hours)[FE][BE]
    - API: Implement applyDefaultsAndValidate in apps/api/services/presets/PresetService.ts to ensure defaultMappings and validation of referenced fields - (M) (1 hours)[FE][BE][QA]
    - API: Update GET /examples handler in apps/api/routers/presets.py to include defaultMappings for each preset - (S) (0.5 hours)[FE][BE]
    - Frontend: Apply defaultMappings in apps/web/components/presets/PresetCard.tsx when rendering presets - (S) (0.5 hours)[FE][BE]

    - Tests: Add unit tests in apps/api/tests/test_presets.py covering defaults applied and validation - (M) (1 hours)[FE][BE][QA]
    - Docs: Add docs for presets defaultMappings in docs/presets.md - (XS) (0.5 hours)[FE][BE]
- **Return project presets: (30 hours)**
    - DB: Create presets table migration in `prisma/migrations/20251030_add_presets_table.sql` (4 hours)[FE][BE]
    - API: Implement get_example() route in `apps/api/routes/example.py` (4 hours)[FE][BE]
    - API: Implement PresetsService.get_presets in `apps/api/services/presets/PresetsService.py` (4 hours)[FE][BE]
    - API: Wire Prisma client in `apps/api/db/prisma_client.py` (4 hours)[FE][BE]
    - Frontend: Add example API client in `src/services/api/example.ts` to call GET /example (4 hours)[FE][BE]
    - Frontend: Build PresetsView component in `src/components/presets/PresetsView.tsx` (4 hours)[FE][BE]
    - Tests: Add API tests in `tests/api/test_example.py` for GET /example (4 hours)[FE][BE][QA]
    - Docs: Document endpoint in `docs/api/example.md` (2 hours)[FE][BE]
- **Include intervals: (36 hours)**
    - DB: Create presets table migration in `prisma/migrations/20251030_create_presets_table/` (4 hours)[FE][BE]
    - API: Implement ExampleService.get_example() in `apps/api/services/examples/ExampleService.py` (4 hours)[FE][BE]
    - API: Add GET /examples/{project_id} router in `apps/api/routers/examples.py` (4 hours)[FE][BE]

    - Frontend: Add apiClient.fetchExample(projectId) in `src/services/apiClient.ts` (4 hours)[FE][BE]
    - Frontend: Create examples Redux slice in `src/store/examplesSlice.ts` (4 hours)[FE][BE]
    - Frontend: Build ExampleViewer component in `src/components/ExampleViewer.tsx` (4 hours)[FE][BE]
    - Frontend: Add ExamplePage route in `src/pages/ExamplePage.tsx` and update `src/App.tsx` routes (4 hours)[FE][BE]
    - Tests: Add API integration test in `tests/test_examples_api.py` (4 hours)[FE][BE][QA]
    - Tests: Add frontend component test in `src/__tests__/ExampleViewer.test.tsx` (4 hours)[FE][BE][QA]
- **Show default flags: (28 hours)**
    - API: Add default flags field to GET example response in `apps/api/routers/examples.py` (4 hours)[FE][BE]
    - API: Update Example model to include default_flags in `apps/api/models/example.py` (4 hours)[FE][BE]
    - Frontend: Add DefaultFlags component in `apps/web/src/components/examples/DefaultFlags.tsx` (4 hours)[FE][BE]
    - Frontend: Update example view to render default flags in `apps/web/src/pages/ExamplePage.tsx` (4 hours)[FE][BE]
    - State: Add defaultFlags to Redux slice in `apps/web/src/store/slices/exampleSlice.ts` (4 hours)[FE][BE]
    - API Test: Add integration test for default_flags in `apps/api/tests/test_examples_api.py` (4 hours)[FE][BE][QA]
    - Frontend Test: Add component test for DefaultFlags in `apps/web/src/components/examples/__tests__/DefaultFlags.test.tsx` (4 hours)[FE][BE][QA]

- **GET 403 example: (34 hours)**
  - API: Add errors router in `apps/api/routers/errors.py` (4 hours)[FE][BE]
  - API: Implement GET /example/403 in `apps/api/routers/errors.py` returning 403 payload (4 hours)[FE][BE]
  - Service: Add errors service in `apps/api/services/errors_service.py` to format 403 response and log to `apps/api/services/errors_service.py` (4 hours)[FE][BE]
  - API: Register errors router in `apps/api/main.py` (2 hours)[FE][BE]
  - Frontend: Create ForbiddenExample page in `src/pages/ErrorExamples/ForbiddenExample.tsx` that calls GET /example/403 (4 hours)[FE][BE]
  - Frontend: Add API call in `src/services/errors.ts` for GET /example/403 (4 hours)[FE][BE]
  - Tests: Add backend test in `tests/test_errors.py` asserting 403 response for /example/403 (4 hours)[FE][BE][QA]
  - Tests: Add frontend test in `src/__tests__/ForbiddenExample.test.tsx` for UI handling of 403 (4 hours)[FE][BE][QA]
  - Docs: Update `docs/errors.md` with GET 403 example usage and payload (4 hours)[FE][BE]

Milestone 7: Documentation & Deployment: deployment instructions, monitoring, release notes, run migration/seed instructions

Estimated 239 hours

- **Setup Monitoring:** Set up and configure monitoring for deployed services and CI/CD pipelines, including alerts and dashboards to ensure system health and rapid incident response.**(6 hours)** - Monitoring agent is installed on all relevant services and containers Alerts trigger on defined

thresholds (CPU, memory, error rate) and are sent to on-call channel Dashboards display key metrics (uptime, latency, error rate) with 95th percentile response times Logs are centralized and searchable for assigned services Monitoring configuration is version-controlled and auditable

    - Infra: Add monitoring agent install to `apps/api/Dockerfile` for containers with emphasis on enabling agent across CI builds and multi-arch images - (S) (0.5 hours)[FE][BE]
    - Infra: Create agent config `infra/monitoring/agent-config.yaml` with CPU/memory/error thresholds - (S) (0.5 hours)[FE][BE][DevOps]
    - API: Integrate metrics exporter in `apps/api/services/metrics/metrics.py` to expose 95th percentile response times - (M) (1 hours)[FE][BE]
    - Infra: Add alert rules in `infra/monitoring/alerts.yaml` and webhook to on-call channel - (S) (0.5 hours)[FE][BE]
    - Infra: Create dashboards in `infra/monitoring/dashboards/uptime_latency_errors.json` - (S) (0.5 hours)[FE][BE]
    - Logging: Configure centralized logging in `apps/api/config/logging.py` to forward to `infra/logging/elasticsearch.yml` - (M) (1 hours)[FE][BE][DevOps]
    - CI: Add monitoring config pipeline in `.github/workflows/monitoring-ci.yml` to validate and version-control `infra/monitoring/` - (M) (1 hours)[FE][BE][DevOps]
    - Testing: Add integration tests in `tests/monitoring/test_alerts.py` to simulate CPU/memory/error thresholds - (S) (0.5 hours)[FE][BE][QA]
    - Docs: Add monitoring runbook in `docs/monitoring/README.md` with playbooks and audit steps - (S) (0.5 hours)[FE][BE]

- **Verify Migration Applied:** As a: DevOps Engineer, I want to: verify that the database migration has been applied successfully in the target environment, So that: the application

schema matches the latest migrations and runtime behavior is correct**(24 hours)** - Migration runs and completes without errors Database schema version matches latest migration hash Migration logs contain success markers and no fatal errors Attempting to rollback fails or is unnecessary because migration is stable System confirms that application can connect and use the migrated schema without errors

  - Infra: Add GitHub Actions workflow `.github/workflows/verify-migration.yml` to run verification (4 hours)[FE][BE]
  - Script: Implement `apps/api/scripts/verify_migration.py` to run `prisma migrate status` and validate hash (4 hours)[FE][BE]
  - API: Add DB helper `apps/api/services/db/migrations.py` with get_current_migration_hash()` (4 hours)[FE][BE]
  - Logs: Implement parser `apps/api/utils/logs.py` to search migration success markers in logs (4 hours)[FE][BE]
  - Test: Create integration test `tests/integration/test_migration_verification.py` to assert schema and connectivity (4 hours)[FE][BE][QA]
  - Docs: Add runbook `docs/runbooks/verify_migration.md` documenting verification steps and rollback guidance (4 hours)[FE][BE]

- **Create Preset per Project: (24 hours)**
  - DB: Add Preset model and create migration in `prisma/migrations/` (create presets table referencing table_users) (4 hours)[FE][BE]
  - API: Implement createPreset(projectId, data) in `apps/api/services/preset/PresetService.ts` (4 hours)[FE][BE]
  - API: Add POST /projects/:id/presets route in `apps/api/routes/preset.py` using PresetService (4 hours)[FE][BE]
  - Frontend: Build PresetForm component in `apps/web/components/preset/PresetForm.tsx` and connect to Redux in `apps/web/store/presetSlice.ts` (4 hours)[FE][BE]

    - Testing: Add integration test in `apps/api/tests/test_preset.py` to verify preset creation and DB relation to table_users (4 hours)[FE][BE][QA]
    - Docs: Add usage docs in `docs/presets.md` and update README in `apps/api/README.md` (4 hours)[FE][BE]
- **Verify Migration Applied: (0 hours)**
- **Add upload presets migration:** As a: database administrator, I want to: apply a new migration that adds upload presets table and related constraints, So that: the system can store and manage user-uploaded presets.**(24 hours)** - Migration runs without errors on local, staging, and production schemas New table and constraints exist in database schema after migration Migration does not overwrite or duplicate existing data Migration rollback is possible and clean Seed data, if any, remains idempotent after migration
    - DB: Update `prisma/schema.prisma` to add UploadPreset model with relations and constraints (4 hours)[FE][BE]
    - DB: Create migration files in `prisma/migrations/` to add upload_presets table and FK constraints (4 hours)[FE][BE]
    - DB: Implement seed logic in `prisma/seed.ts` for upload presets (idempotent) (4 hours)[FE][BE]
    - Test: Add migration tests in `tests/migrations/upload_presets_migration.test.ts` verifying creation, constraints, and rollback (4 hours)[FE][BE][QA]
    - CI: Update `.github/workflows/prisma-migrate.yml` to run migrate –preview-feature and rollback checks in CI (4 hours)[FE][BE]
    - Docs: Add deployment runbook `docs/db/migrations/upload_presets.md` with staging/production steps and verification queries (4 hours)[FE][BE][DevOps]
- **Verify seed idempotency:** As a: developer, I want to: verify that seed scripts are idempotent when applying migrations, So that: repeated seed executions do not create duplicate data or

errors.**(24 hours)** - Seed runs without creating duplicates on first run and on re-run No errors occur when seeding repeatedly Database state is consistent after multiple seed executions Seed idempotency validated across environments (dev/stage) Audit logs show idempotent behavior for seed operations

    - DB: Create idempotent seed script in `prisma/seed.ts` - use upsert for users (4 hours)[FE][BE]
    - API: Add seed wrapper in `apps/api/services/db/seed.ts` to run `prisma/seed.ts` idempotently and return audit info (4 hours)[FE][BE]
    - DB: Add migration files in `prisma/migrations/` ensuring schema for `table_users` matches seed expectations (4 hours)[FE][BE]
    - API: Implement audit logging in `apps/api/services/audit/AuditService.ts` for seed operations (4 hours)[FE][BE]
    - Testing: Add idempotency tests in `tests/seed_idempotency.test.ts` to run seed twice against test DB (4 hours)[FE][BE][QA]
    - Infra: Add CI workflow `.github/workflows/seed-idempotency.yml` to run migrations and seed in dev/stage matrix (4 hours)[FE][BE]

- **Add upload presets migration: (0 hours)**
- **Verify seed idempotency: (0 hours)**
- **Create migration model:** As a: database administrator, I want to: create a migration model, So that: the system can track and apply database schema changes reliably**(3.5 hours)** - Migration model is defined with fields id, version, appliedAt

and status Migration records can be created and persisted to the migrations table Applied migrations can be queried and filtered by status and version

    - DB: Add Migration model to prisma/schema.prisma with fields id, version, appliedAt, status preserving existing Prisma schema conventions and relations; ensure model is exported in Prisma client generation. - (XS) (0.5 hours)[FE][BE]
    - DB: Create migration in prisma/migrations/ and run prisma migrate to create migrations table, ensuring Prisma Migrate tracks applied migrations and schema history. - (S) (0.5 hours)[FE][BE]
    - API: Implement MigrationRepository.ts under apps/api/services/migrations to provide create() and queryByStatusAndVersion() leveraging Prisma client for DB access and returning appropriate domain models. - (M) (1 hours)[FE][BE]
    - Tests: Add integration tests in apps/api/__tests__/migration.test.ts for create and query operations - (M) (1 hours)[FE][BE][QA]
    - Docs: Add docs/migrations.md describing model fields and usage - (S) (0.5 hours)[FE][BE]

- **Add migration.sql file: (20 hours)**
    - DB: Create migration.sql in `prisma/migrations/migration.sql` (4 hours)[FE][BE]
    - DB: Reference new migration in `apps/api/db/init.sql` (4 hours)[FE][BE]
    - infra: Update CI in `.github/workflows/ci.yml` to run migrations (4 hours)[FE][BE]
    - testing: Add smoke test in `tests/migrations/test_migration.py` (4 hours)[FE][BE][QA]

    - documentation: Add entry in `docs/migrations.md` describing `prisma/migrations/migration.sql` (4 hours)[FE][BE]

- **Add README with notes: (20 hours)**
    - DOC: Create migration README skeleton in `apps/api/migrations/README.md` (4 hours)[FE][BE]
    - DOC: Add migration notes in `docs/migration_notes.md` referencing tables and steps (4 hours)[FE][BE]
    - DOC: Update `apps/api/README.md` with link to migrations and summary (4 hours)[FE][BE]
    - INFRA: Update CI link-check step in `.github/workflows/ci.yml` to validate README links (4 hours)[FE][BE]
    - QUALITY: Add link checker script `scripts/link_checker.sh` and include in CI (4 hours)[FE][BE]
- **Add prisma schema models: (20 hours)**
    - DB: Add User and Session models in `prisma/schema.prisma` with relations and fields (4 hours)[FE][BE]
    - Infra: Create migration folder in `prisma/migrations/` and SQL migration files for users and sessions tables (4 hours)[FE][BE]
    - API: Update `apps/api/services/auth/AuthService.ts` to use Prisma Client types and new models (4 hours)[FE][BE]
    - Dev: Add Prisma client import usages in `apps/api/routes/session.ts` and type-safe queries for sessions (4 hours)[FE][BE]
    - Quality: Add integration test in `apps/api/__tests__/prisma_models.test.ts` asserting User and Session create/read (4 hours)[FE][BE][QA]
- **Add seed script stub:** As a: developer, I want to: add a seed script stub, So that: there is a placeholder to seed initial data once migrations are in place**(1.5 hours)** - Seed script file

exists and is executable Seed script contains a placeholder log or comment indicating intended data to seed Script can be invoked without errors (dry-run)

    - Create seed script file apps/api/scripts/seed.sh with placeholder log using Bash. Script should emit a basic message indicating seed placeholder and exit code 0. Location aligns with apps/api/scripts path under the API project. - (XS) (0.5 hours)[FE][BE]
    - Make apps/api/scripts/seed.sh executable (chmod +x) as part of CI config or local script to ensure script can run in CI and locally. Ensure CI environment respects executable flag in repository checkout. - (S) (0.5 hours)[FE][BE][DevOps]
    - Add dry-run npm script seed:dry in apps/api/package.json that invokes bash apps/api/scripts/seed.sh –dry-run. Ensure script passes –dry-run flag to seed script and that script handles it (even if placeholder). Update package.json scripts field accordingly. - (S) (0.5 hours)[FE][BE]
    - Add README entry in apps/api/README.md describing seed script and dry-run usage. Include example commands and expected output placeholder. Ensure docs stay in sync with the implemented features. - (XS) (0.5 hours)[FE][BE]

- **Add API handlers: (0 hours)**
- **Create Migration:** As a: admin, I want to: create a Prisma migration for the new upload presets schema, So that: I can apply structural changes to the database for project presets**(4.5 hours)** - Migration script runs without errors on local and CI environments Migration generates correct SQL for create statements matching Prisma schema Migration is versioned and recorded in migration history table with timestamp

    - Infra: Add Prisma schema and create migration in prisma/schema.prisma and prisma/migrations/ - (M) (1 hours)[FE][BE]

    - DB: Create users table migration SQL in prisma/migrations/ with CREATE TABLE matching table_users - (S) (0.5 hours) [FE][BE]
    - API: Add migration run script in apps/api/scripts/ run_migrations.sh and apps/api/pyproject.toml - (M) (1 hours)[FE][BE]
    - CI: Add GitHub Actions workflow /.github/workflows/ migrate.yml to run apps/api/scripts/run_migrations.sh against test DB - (M) (1 hours)[FE][BE][QA]
    - QA: Add migration test in apps/api/tests/test_migrations.py to assert SQL and migration history entry - (S) (0.5 hours) [FE][BE][QA]
    - Docs: Document migration steps in docs/migrations.md and update changelog CHANGELOG.md - (XS) (0.5 hours)[FE] [BE]
- **Add GET/PUT Handlers:** As a: developer, I want to: add GET and PUT API handlers for project upload presets, So that: clients can retrieve and update presets programmatically**(10 hours)** - GET endpoint returns correct preset data for a given project PUT endpoint updates preset data and persists changes Error handling for non-existent projects and invalid input Security: validate inputs and enforce authorization
    - DB: Create presets migration in `prisma/migrations/` to add presets table and relations - (M) (1 hours)[FE][BE]
    - Prisma: Update schema in `prisma/schema.prisma` and run `prisma generate` - (M) (1 hours)[FE][BE]
    - API: Add GET /projects/:id/presets handler in `apps/api/ routes/projects.ts` - (M) (1 hours)[FE][BE]
    - API: Add PUT /projects/:id/presets handler in `apps/api/ routes/projects.ts` - (M) (1 hours)[FE][BE]
    - Service: Implement getPresets(projectId) in `apps/api/ services/presets/PresetsService.ts` - (M) (1 hours)[FE][BE]

    - Service: Implement updatePresets(projectId, data) in `apps/api/services/presets/PresetsService.ts` - (M) (1 hours)[FE][BE]
    - Auth: Add authorization middleware in `apps/api/middleware/auth.ts` to enforce project access using Clerk - (M) (1 hours)[FE][BE]
    - Validation: Add input validation schemas in `apps/api/validation/presets.ts` and apply in `apps/api/routes/projects.ts` - (M) (1 hours)[FE][BE][QA]
    - Tests: Add integration tests for GET/PUT in `apps/api/tests/presets.test.ts` - (M) (1 hours)[FE][BE][QA]
    - Docs: Update API docs in `docs/api/presets.md` with endpoints, errors, and auth - (M) (1 hours)[FE][BE]
- **Add migration folder: (20 hours)**
    - FS: Create migration folder `prisma/migrations/` in repo root (4 hours)[FE][BE]
    - DB: Add migration SQL file `prisma/migrations/20251030_create_project_upload_presets.sql` to create tables for project_upload_default_mappings and project_upload_preset_intervals (4 hours)[FE][BE]
    - API: Update schema in `apps/api/prisma/schema.prisma` to include models for project_upload_default_mappings and project_upload_preset_intervals (4 hours)[FE][BE]
    - CI: Add migration step in `/.github/workflows/ci.yml` to run `npm run migrate:deploy` for `apps/api` before tests (4 hours)[FE][BE][DevOps][QA]
    - QA: Add migration verification test in `apps/api/tests/migrations.test.ts` to assert tables (table_project_upload_default_mappings, table_project_upload_preset_intervals) exist (4 hours)[FE][BE][QA]

- **Add schema.prisma update: (32 hours)**
    - DB: Update `schema.prisma` to add ProjectUploadPreset and related models in `prisma/schema.prisma` (4 hours) [FE][BE]
    - DB: Create migration in `prisma/migrations/` for project_upload_presets and default_mappings (4 hours)[FE] [BE]
    - DB: Add project_upload_preset_intervals model in `prisma/schema.prisma` with relations to ProjectUploadPreset (4 hours)[FE][BE]
    - API: Update Prisma client usage in `apps/api/src/prismaClient.ts` to include new models (4 hours)[FE][BE]
    - API: Create migration runner script in `apps/api/scripts/runMigrations.ts` to apply `prisma/migrations/` (4 hours) [FE][BE]
    - API: Add repository methods in `apps/api/services/projectUpload/ProjectUploadService.ts` to manage presets (4 hours)[FE][BE]
    - Testing: Add integration tests in `apps/api/tests/projectUpload/presets.test.ts` for CRUD on ProjectUploadPreset (4 hours)[FE][BE][QA]
    - Docs: Update migration docs in `docs/database/migrations.md` describing schema changes and rollback (4 hours)[FE][BE]
- **Add Seed Per Project:** As a: admin, I want to: seed initial preset data per project during migration, So that: new projects have default presets available**(6 hours)** - Seed files run without errors in migration Seed data matches Prisma model fields and constraints Idempotent seeds (no duplicates on rerun)
    - DB: Create project-specific seed files in `prisma/seed/projectA/seed.ts` and `prisma/seed/projectB/seed.ts` - (M) (1 hours)[FE][BE]

    - DB: Add idempotent upsert logic in `prisma/seed/projectA/seed.ts` and `prisma/seed/projectB/seed.ts` (use unique keys) - (M) (1 hours)[FE][BE]
    - Config: Update `prisma/seed/index.ts` to load seed per project based on env VAR and run during migration - (M) (1 hours)[FE][BE][DevOps]
    - CI: Update `.github/workflows/prisma.yml` to run `node prisma/seed/index.ts` after `prisma migrate deploy` - (M) (1 hours)[FE][BE][DevOps]
    - Test: Add integration test that runs migrations and seeds using `tests/integration/seed.test.ts` - (M) (1 hours)[FE][BE][QA]
    - Docs: Document seed process and env vars in `docs/prisma/seeding.md` - (M) (1 hours)[FE][BE]

Milestone 8: Maintenance & Support: ongoing monitoring, support, and maintenance tasks

Estimated 36 hours

- **Monitor System Health:** As a: system administrator, I want to: monitor system health metrics and alert on anomalies, So that: I can ensure high availability and rapid incident response**(11.5 hours)** - System collects CPU, memory, disk, and network metrics every minute Alerts trigger when any metric crosses predefined threshold Dashboards display health status with up-to-date values Incident response protocol is triggered automatically for critical states
    - API: Create metrics table migration in `apps/api/migrations/20251001_create_metrics_table.sql` - (M) (1 hours)[FE][BE]
    - Backend: Implement metrics collector Celery task in `apps/api/tasks/metrics/collect_metrics.py` - (L) (2 hours)[FE][BE]
    - Backend: Add metrics ingestion endpoint in `apps/api/routers/metrics.py` - (M) (1 hours)[FE][BE]

    - Backend: Implement storage logic in `apps/api/services/metrics/StorageService.py` - (M) (1 hours)[FE][BE]
    - Backend: Implement alert evaluator in `apps/api/services/alerts/AlertService.py` - (M) (1 hours)[FE][BE]
    - Infra: Add Celery beat schedule in `apps/api/config/celery_app.py` to run `collect_metrics.py` every minute - (S) (0.5 hours)[FE][BE][DevOps]
    - Frontend: Build HealthDashboard component in `apps/web/src/components/health/HealthDashboard.tsx` using Plotly - (M) (1 hours)[FE][BE]
    - Frontend: Create Redux slice for metrics in `apps/web/src/store/slices/metricsSlice.ts` - (S) (0.5 hours)[FE][BE]
    - Alerting: Create alert rules config in `ops/alerts/thresholds.yaml` and load in `apps/api/config/alerts.py` - (M) (1 hours)[FE][BE][DevOps]
    - Incident: Implement incident webhook handler in `apps/api/routers/incidents.py` to trigger protocol - (S) (0.5 hours)[FE][BE]
    - Monitoring: Add Prometheus exporter in `apps/api/metrics/prometheus_exporter.py` - (S) (0.5 hours)[FE][BE]
    - Testing: Add integration tests for metrics collection in `tests/integration/test_metrics_collection.py` - (M) (1 hours)[FE][BE][QA]
    - Docs: Document runbook and alert response in `docs/runbooks/incident_response.md` - (S) (0.5 hours)[FE][BE]
- **Performance Optimization:** As a: system administrator, I want to: optimize performance of critical services, So that: I can reduce latency and improve user experience**(5 hours)** - Baseline performance metrics captured Implement and

validate at least one optimization (caching, indexing, or configuration tuning) Performance improvement measurable under load test No regression in functionality after changes

  - Measure: Capture baseline metrics using apps/api/monitoring/collect_metrics.py (collect CPU, memory, DB queries) - (XS) (0.5 hours)[FE][BE]
  - API: Add query timing middleware in apps/api/middleware/timing.py to log DB latency - (S) (0.5 hours)[FE][BE]
  - DB: Add index on users.email via migration apps/api/migrations/202510_create_users_email_index.sql - (S) (0.5 hours)[FE][BE]
  - Cache: Implement Redis caching for user profile in apps/api/services/user/UserService.py (use Redis client in apps/api/lib/redis_client.py) - (M) (1 hours)[FE][BE]
  - Load Test: Create load test script tests/load/test_user_profile_load.py using locust/requests to measure improvements - (M) (1 hours)[FE][BE][QA]
  - Validation: Add performance regression tests in tests/perf/test_regression.py and CI workflow .github/workflows/perf.yml - (M) (1 hours)[FE][BE][QA]
  - Docs: Document metrics and tuning steps in docs/performance/README.md - (S) (0.5 hours)[FE][BE]

- **Security Updates:** As a: security engineer, I want to: apply security updates and verify patch integrity, So that: I can reduce vulnerability exposure**(7 hours)** - All critical and high-severity patches applied Patch verification checksums match Regression tests pass after patching Audit log shows patch application details and user No downtime beyond maintenance window
  - Infra: Update dependency versions in apps/api/requirements.txt (apply critical/high patches) with attention to transitive dependencies and compatibility across Python environment and container image build. - (M) (1 hours)[FE][BE][DevOps]

    - API: Implement patch application in apps/api/services/patch/PatchService.py using internal patching logic to apply diff patches to runtime codebase, with validation hooks. - (M) (1 hours)[FE][BE][QA]
    - API: Add maintenance route in apps/api/routes/maintenance.py to trigger patches via PatchService with proper auth and validation. - (S) (0.5 hours)[FE][BE][QA]
    - Security: Add checksum verification in apps/api/services/patch/Checksum.py to validate patch integrity. - (S) (0.5 hours)[FE][BE]
    - DB: Create AuditLog model in apps/api/models/AuditLog.py and migration in apps/api/migrations/ to record patch activities. - (S) (0.5 hours)[FE][BE]
    - CI: Add patch pipeline job in infra/ci/patch_job.yml (GitHub Actions) to run patches and checksum checks - (M) (1 hours)[FE][BE]
    - Testing: Implement regression tests in apps/api/tests/test_regression.py and add to CI - (S) (0.5 hours)[FE][BE][QA]
    - Infra: Update Dockerfile in apps/api/Dockerfile and deployment scripts in infra/deploy/ to ensure zero-downtime deployment - (L) (2 hours)[FE][BE][DevOps]
- **Backup & Recovery:** As a: IT admin, I want to: ensure backups are completed and recoverable, So that: I can restore services quickly after failure**(5.5 hours)** - Backups completed and stored in secure location Restore procedure tested quarterly RPO/RTO meet defined targets Backup integrity verification passes Alerts if backup job fails
    - Infra: Configure S3 bucket and IAM in infra/aws/s3_backup.tf using AWS S3 bucket policies, IAM roles/users, and encryption at rest. Ensure least privilege and versioning; wire with backup service ACLs and cross-account access if needed. - (S) (0.5 hours)[FE][BE][DevOps]

    - API: Implement backup job in apps/api/services/backup/BackupService.py using Python, exposing a REST endpoint to trigger a backup, coordinating with DB dump and S3 upload, with idempotent handling and retry policy. - (M) (1 hours)[FE][BE]
    - DB: Add backup SQL dump task in apps/api/utils/backup_db.py using pg_dump (or equivalent) with schema/table filtering, compression, and integrity hash generation for verify step. - (M) (1 hours)[FE][BE]
    - Worker: Schedule Celery task in apps/api/tasks/backup_tasks.py to orchestrate backup pipeline initiation and status polling, ensuring idempotency and safe retries; integrate with broker/backend. - (S) (0.5 hours)[FE][BE]
    - Verification: Implement integrity check in apps/api/utils/verify_backup.py to compare dump hash with stored value and verify S3 upload integrity, including runbook for mismatch scenarios. - (S) (0.5 hours)[FE][BE]
    - Restore: Create restore script in apps/api/utils/restore_db.py and runbook docs/runbooks/restore.md to fetch latest backup, restore to target DB, and validate data integrity post-restore. - (M) (1 hours)[FE][BE][QA]
    - Monitoring: Add GitHub Actions job in .github/workflows/backup_test.yml to run quarterly restore tests - (S) (0.5 hours)[FE][BE][QA]
    - Alerts: Add failure alerts integration in apps/api/notifications/alerting.py to notify on backup/restore failures via Slack or PagerDuty - (S) (0.5 hours)[FE][BE]
- **Documentation Maintenance:** As a: technical writer, I want to: update and maintain system documentation, So that: I can ensure accurate knowledge base for support and onboarding**(7 hours)** - Documentation reflects latest system architecture

Changelog updated with every deployment Users can locate relevant docs within 2 clicks Documentation reviews occur monthly No outdated or conflicting documentation in repo

- Doc: Create `docs/ARCHITECTURE.md` reflecting system architecture and components file paths (cite: table_users) - (S) (0.5 hours)[FE][BE]
- Doc: Update `CHANGELOG.md` and add release script in `scripts/update_changelog.py` - (S) (0.5 hours)[FE][BE]
- API: Add/refresh doc comments in `apps/api/services/auth/AuthService.ts` and update `apps/api/README.md` - (M) (1 hours)[FE][BE]
- DB: Create `docs/database.md` with `table_users` schema and migrations references in `migrations/` - (M) (1 hours)[FE][BE]
- Infra: Add GitHub Action `/.github/workflows/docs.yml` to build docs and run `scripts/check-doc-links.py` - (L) (2 hours)[FE][BE]
- UX: Add docs index and 2-click nav in `docs/index.md` and `apps/web/src/components/docs/Nav.tsx` - (S) (0.5 hours)[FE][BE]
- Quality: Add monthly review issue template `.github/ISSUE_TEMPLATE/doc-review.md` and schedule workflow `/.github/workflows/doc-review.yml` - (S) (0.5 hours)[FE][BE]
- Tooling: Implement `scripts/check-doc-links.py` to ensure no broken links - (M) (1 hours)[FE][BE]

Total Hours: 4200.5