RERC Review Portal — Project Context Pack

Generated: February 8, 2026
Repository: `rerc-review-portal`
Version: 0.1.0

---

0. Executive Summary

System Purpose

The **RERC Review Portal** is a web-based Research Ethics Review Committee (RERC) management system built for De La Salle University Manila. It streamlines the workflow for managing research ethics protocol submissions—from initial receipt through classification, review, and final decision—while providing mail-merge letter generation and SLA tracking capabilities.

Tech Stack

Layer	Technology
Backend	Node.js 18+, Express 5.x, TypeScript 5.x
Frontend	React 18, Vite 5.x, TypeScript, React Router 6
Database	PostgreSQL (via Prisma ORM 7.x)
Libraries	Axios, docx (Word generation), json2csv, csv-parse
Infra	Local development (no CI/CD or containerization yet)

Highest-Risk Areas / Bottlenecks

- **Authentication is stub-only**: Header-based auth (`X-User-*` headers) is for development only—no production-ready JWT/session implementation exists.
- **Hardcoded user IDs**: Several endpoints use `createdById: 1` or `changedById: 1` instead of actual authenticated user.
- **No rate limiting**: API has no protection against abuse or DoS.
- **CSV/file import security**: Seed script parses external CSV without comprehensive sanitization.
- **No automated tests running in CI**: Tests exist but no CI/CD pipeline.
- **Large monolithic route files**: Some route files exceed 800 lines.

Where to Start Reading the Code (Top 5 Files)

1. `backend/src/server.ts` — Express app entry point, middleware setup, route mounting
2. `backend/prisma/schema.prisma` — Complete data model with all enums and relationships
3. `backend/src/routes/dashboardRoutes.ts` — Dashboard queues and activity endpoints
4. `frontend/src/pages/DashboardPageNew.tsx` — Main dashboard UI (2300+ lines)
5. `docs/SECURITY.md` — RBAC design and audit logging approach

---

1. Repository Map

```
rerc-review-portal/
├── README.md                           # Project overview & setup
├── FILE_EXPLANATIONS.txt              # Detailed file documentation
├── package.json                       # Root workspace scripts
│
├── ai/                                # AI assistant context (mostly empty)
│   ├── context.md
│   ├── decisions.md
│   ├── plan.md
│   └── tasks.md
│
├── backend/                           # Express.js API Server
│   ├── package.json                   # Backend dependencies
│   ├── tsconfig.json                  # TypeScript config
│   ├── prisma.config.ts               # Prisma configuration
│   ├── prisma/
│   │   ├── schema.prisma              # Database schema (687 lines)
│   │   └── migrations/                # 12 migration folders
│   └── src/
│       ├── server.ts                  # Express entry point
│       ├── config/
│       │   ├── prismaClient.ts        # Prisma client singleton
│       │   ├── seed.ts                # Database seeder (1361 lines)
│       │   └── importCSV.ts           # CSV import utilities
│       ├── routes/
│       │   ├── index.ts               # Route exports
│       │   ├── healthRoutes.ts        # Health check endpoints
│       │   ├── committeeRoutes.ts     # Committee/panel endpoints
│       │   ├── dashboardRoutes.ts     # Dashboard queues (543 lines)
│       │   ├── projectRoutes.ts       # Project CRUD (337 lines)
│       │   ├── submissionRoutes.ts    # Submissions/reviews (893 lines)
│       │   └── mailMergeRoutes.ts     # Letter generation (603 lines)
│       ├── services/
│       │   └── letterGenerator.ts     # DOCX letter builder
│       ├── utils/
│       │   ├── index.ts
│       │   ├── csvUtils.ts            # CSV escape utilities
│       │   └── slaUtils.ts            # Working days calculator
│       └── generated/prisma/          # Generated Prisma client
│
├── frontend/                          # React SPA
│   ├── package.json                   # Frontend dependencies
│   ├── index.html                     # Vite entry HTML
│   ├── vite.config.ts                 # Vite bundler config
│   ├── tsconfig.json                  # TypeScript config
│   └── src/
│       ├── App.tsx                    # React app root & routing
│       ├── main.tsx                   # React entry point
│       ├── constants.ts               # App-wide constants
│       ├── components/                # Reusable UI components
│       │   ├── AttentionStrip.tsx
│       │   ├── CommandBar.tsx
│       │   ├── EmptyState.tsx
│       │   ├── LetterReadinessPanel.tsx
│       │   ├── QueueTable.tsx
│       │   ├── SLAStatusChip.tsx
│       │   ├── SummaryCards.tsx
│       │   └── Timeline.tsx
│       ├── hooks/                    # Custom React hooks
│       │   ├── useDashboardQueues.ts
│       │   ├── useDashboardActivity.ts
│       │   ├── useDashboardOverdue.ts
│       │   ├── useProjectDetail.ts
│       │   └── useSubmissionDetail.ts
│       ├── pages/
│       │   ├── DashboardPageNew.tsx   # Main dashboard (2335 lines)
│       │   ├── DashboardPage.tsx      # Legacy dashboard
│       │   ├── ProjectDetailPage.tsx
│       │   ├── SubmissionDetailPage.tsx
│       │   ├── LoginPage.tsx
│       │   └── ForgotPasswordPage.tsx
│       ├── services/
│       │   └── api.ts                 # API client (201 lines)
│       ├── styles/
│       │   ├── globals.css
│       │   └── dashboard.css
│       ├── types/
│       │   └── index.ts               # TypeScript interfaces
│       └── utils/
│
├── packages/shared/                   # Shared code (minimal usage)
│   ├── package.json
│   └── src/
│
├── samples/                           # Sample CSV exports
│   ├── initial_ack.csv
│   └── initial_approval.csv
│
└── docs/
    ├── SECURITY.md                    # Security & RBAC documentation
    ├── fixes.md                       # Integration notes
    └── context-pack/                  # This document
```

2. How to Run (Local Development)

Prerequisites

Requirement	Version	How to Check
Node.js	18+ (LTS)	`node -v`
npm	9+	`npm -v`
PostgreSQL	14+	`psql --version`

Environment Variables

Backend (`backend/.env`):

Variable	Required	Description
`DATABASE_URL`	Yes	PostgreSQL connection string
`PORT`	No	Server port (default: 3000)
`CORS_ORIGINS`	No	Allowed origins (default: localhost:5173)
`FRONTEND_URL`	No	Frontend URL for CORS

Frontend (`.env` or Vite build):

Variable	Required	Description
`VITE_API_URL`	No	API base URL (default: http://localhost:3000)

Installation & Setup

```
# 1. Clone and enter repo
cd rerc-review-portal

# 2. Install all dependencies (root, backend, frontend)
npm install
npm run install:all

# 3. Create backend/.env with DATABASE_URL
echo 'DATABASE_URL=postgresql://USER@localhost:5432/rerc' > backend/.env

# 4. Ensure PostgreSQL is running and database exists
createdb rerc  # or use pgAdmin/psql

# 5. Generate Prisma client and run migrations
npm run db:generate
npm run db:migrate

# 6. (Optional) Seed with sample data
npm run db:seed

# 7. Start development servers
npm run dev
```

Access Points

- **Backend API:** http://localhost:3000
- **Frontend UI:** http://localhost:5173 (default route redirects to `/login`)
- **Health check:** http://localhost:3000/health

Common Failures & Fixes

Issue	Solution
`P1001: Can't reach database`	Check PostgreSQL is running, `DATABASE_URL` is valid
`Cannot find module prisma`	Run `npm run db:generate`
CORS errors in browser	Ensure `CORS_ORIGINS` includes frontend URL
Port 3000/5173 already in use	Kill existing process or change port in config
Missing CSV file during seed	Ensure `[Intern Copy] RERC Protocol Database 2024 - 2024 Submission.csv` exists at repo root

---

3. Architecture Overview

Component Diagram

```
┌─────────────────────────────────────────────────────────────────────┐
│                        CLIENT BROWSER                               │
│  ┌────────────────────────────────────────────────────────────────┐ │
│  │    React SPA (Vite @ port 5173)                                │ │
│  │  ┌─────────────┐  ┌─────────────┐  ┌────────────────────────┐  │ │
│  │  │ LoginPage   │  │ Dashboard   │  │ Project/SubmissionPage │  │ │
│  │  └─────────────┘  └─────────────┘  └────────────────────────┘  │ │
│  │         │               │                     │                │ │
│  │         └───────────────┼─────────────────────┘                │ │
│  │                         │ Axios API Client                     │ │
│  └─────────────────────────┼──────────────────────────────────────┘ │
└────────────────────────────┼────────────────────────────────────────┘
                             │ HTTP/JSON
                             ▼
┌─────────────────────────────────────────────────────────────────────┐
│                   EXPRESS.JS SERVER (port 3000)                     │
│  ┌────────────────────────────────────────────────────────────────┐ │
│  │                        Middleware                              │ │
│  │  CORS → JSON Parser → (Future: Auth Middleware)                │ │
│  └────────────────────────────────────────────────────────────────┘ │
│                              │                                      │
│  ┌───────────────────────────┴──────────────────────────────────┐  │
│  │                      Route Modules                             │   │
│  │  ┌────────┐ ┌───────────┐ ┌──────────┐ ┌─────────────────┐  │   │
│  │  │ Health │ │ Committee │ │Dashboard │ │     Project     │  │   │
│  │  └────────┘ └───────────┘ └──────────┘ └─────────────────┘  │   │
│  │                                                              │   │
│  │  ┌─────────────┐ ┌───────────────────────────────────────┐  │   │
│  │  │ Submission  │ │ MailMerge (CSV + DOCX generation)     │  │   │
│  │  └─────────────┘ └───────────────────────────────────────┘  │   │
│  └──────────────────────────────────────────────────────────────┘   │
│                              │                                      │
│  ┌──────────────────────────┼───────────────────────────────┐   │
│  │                    Prisma ORM Client                      │   │
│  └──────────────────────────┼───────────────────────────────┘   │
└──────────────────────────────┼───────────────────────────────────┘
                               │ PostgreSQL Wire Protocol
                               ▼
┌──────────────────────────────────────────────────────────────────┐
│                      POSTGRESQL DATABASE                         │
│   Tables: User, Committee, Panel, Project, Submission, Review, etc. │
└──────────────────────────────────────────────────────────────────┘
```

Request Lifecycle (Client → API → DB)

```
1. User clicks "View Submission #5" in Dashboard
   │
2. React Router renders SubmissionDetailPage
   │
3. useSubmissionDetail hook fires
   │
4. api.ts → GET /submissions/5
   │
5. Express router matches submissionRoutes.ts
   │
6. Handler calls prisma.submission.findUnique({ where: { id: 5 }, include: {...} })
   │
7. Prisma generates SQL, executes against PostgreSQL
   │
8. JSON response travels back through layers
   │
9. React state updates, component re-renders
```

Auth Lifecycle (Development Mode)

Current Implementation (Stub):

```
┌─────────────┐     ┌────────────────────────────────────────────┐
│   Client    │────▶│  Include headers in each request:          │
│             │     │  X-User-ID, X-User-Email, X-User-Roles     │
└─────────────┘     └────────────────────────────────────────────┘
       │
       ▼
┌──────────────────────────────────────────────────────────────────┐
│  authenticateUser middleware (planned but not fully wired)       │
│  - Extracts user info from headers                               │
│  - Attaches req.user object                                      │
│  - Role-based authorization on specific endpoints                │
└──────────────────────────────────────────────────────────────────┘
```

Production Recommendation: Replace with JWT tokens or session cookies with proper login flow.

---

4. Data Model (Database)

ERD Overview (Key Entities)

```
┌──────────────┐         ┌──────────────┐         ┌──────────────┐
│     User     │◄───────►│  Committee   │◄───────►│    Panel     │
│──────────────│ N:M via │──────────────│  1:N    │──────────────│
│ id           │Committee│ id           │         │ id           │
│ email        │Member   │ code         │         │ name         │
│ fullName     │         │ name         │         │ committeeId  │
│ roles[]      │         │ isActive     │         └──────────────┘
└──────────────┘         └──────────────┘                │
       │                        │                        │
       │                        │ 1:N                    │
       │                        ▼                        │
       │                 ┌──────────────┐                │
       │                 │   Project    │                │
       │                 │──────────────│                │
       │                 │ id           │                │
       │                 │ projectCode  │                │
       │                 │ title        │                │
       │                 │ piName       │                │
       │                 │ fundingType  │                │
       │                 │ overallStatus│                │
       │                 │ committeeId  │                │
       │                 └──────────────┘                │
       │                        │                        │
       │                        │ 1:N                    │
       │                        ▼                        │
       │                 ┌──────────────┐         ┌──────────────┐
       │                 │  Submission  │ ───────►│Classification│
       │                 │──────────────│  1:1    │──────────────│
       │                 │ id           │         │ reviewType   │
       │                 │ projectId    │         │ panelId      │
       │                 │ submissionType│        │ classifiedById│
       │                 │ receivedDate │         └──────────────┘
       │                 │ status       │                │
       └────────────────►│ finalDecision│                │
          assignedTo     │ createdById  │                │
          reviews        └──────────────┘                │
                                │                        │
                         ┌──────┴──────┐                 │
                         │             │                 │
                         ▼             ▼                 │
                  ┌──────────┐  ┌─────────────────┐      │
                  │  Review  │  │StatusHistory    │      │
                  │──────────│  │─────────────────│      │
                  │ reviewerId│ │ oldStatus       │      │
                  │ decision │  │ newStatus       │      │
                  │ remarks  │  │ effectiveDate   │      │
```

```
                    └──────────┘  │ changedById      │       │
                                  └──────────────────┘       │
```

Key Enumerations

RoleType (User permissions):

- `CHAIR` — Committee chair, full committee access
- `MEMBER` — Committee/panel member, view-only for assigned
- `RESEARCH_ASSOCIATE` — RA/Secretariat, manages protocols
- `RESEARCH_ASSISTANT` — RA support, limited data entry
- `REVIEWER` — External reviewer, assigned submissions only
- `ADMIN` — System administrator, full access

SubmissionStatus (Workflow states):

1. `RECEIVED` — Initial receipt
2. `UNDER_COMPLETENESS_CHECK` — Documents being verified
3. `AWAITING_CLASSIFICATION` — Ready for review type decision
4. `UNDER_CLASSIFICATION` — Classification in progress
5. `CLASSIFIED` — Review type assigned
6. `UNDER_REVIEW` — Active reviewer assessment
7. `AWAITING_REVISIONS` — Revisions requested from PI
8. `REVISION_SUBMITTED` — PI resubmitted revisions
9. `CLOSED` — Final decision issued
10. `WITHDRAWN` — Applicant withdrew

ReviewType (Classification outcomes):

- `EXEMPT` — No further review needed
- `EXPEDITED` — Minimal-risk, expedited review
- `FULL_BOARD` — Full committee review required

ReviewDecision (Reviewer/final outcomes):

- `APPROVED`
- `MINOR_REVISIONS`
- `MAJOR_REVISIONS`
- `DISAPPROVED`
- `INFO_ONLY`

Critical Invariants (Code-Implied)

1. **Unique project codes**: `projectCode` has unique constraint, enforced by Prisma
2. **One classification per submission**: `submissionId` is unique on `Classification`
3. **Sequential submissions per project**: `@@unique([projectId, sequenceNumber])`
4. **Reviewer assignment uniqueness**: `@@unique([submissionId, reviewerId])` on Review

Migration History

Migration	Purpose
`20251211100203_init_user`	User model initialization
`20251211103050_rerc_core`	Core RERC models (Committee, Project)

`20251211130711_add_classification`	Classification model
`20251211132529_add_reviews`	Review and reviewer models
`20251211134028_add_submission_status`	Status history tracking
`20251211135517_add_config_sla`	SLA configuration tables
`20251211142937_extend_sla_stages`	Extended SLA stage enum
`20251211150930_add_panel_member_and_deadlines`	Panel membership model
`20251211155533_add_submission_final_decision`	Final decision fields
`20251215021511_init`	Re-initialization
`20251215025323_init`	Additional init
`20260114130242_add_csv_fields`	Fields for CSV import compatibility

Seed Script Notes

The seed script (`backend/src/config/seed.ts`) reads CSV data from `[Intern Copy] RERC Protocol Database 2024 - 2024 Submission.csv` at the repo root and creates:

- Default committee (RERC-HUMAN)
- Users (RA, Chair, reviewers)
- Projects with submissions from CSV

---

5. API Surface

Route Inventory by Module

Health & Status

Method	Endpoint	Behavior	Auth
GET	`/`	Server status JSON	None
GET	`/health`	Database connection check	None

Committee Management

Method	Endpoint	Behavior	Auth
GET	`/committees`	List all committees with panels	None
GET	`/committees/:code/panels`	Get panels for a committee	None
GET	`/panels/:id/members`	Get panel members	None

Dashboard

Method	Endpoint	Behavior	Auth
GET	`/dashboard/queues`	Submission queues by status	None

GET	`/dashboard/activity`	Recent status change activity	None
GET	`/dashboard/overdue`	Overdue reviews and endorsements	None
GET	`/dashboard/upcoming-due`	Submissions with upcoming deadlines	None
GET	`/ra/dashboard`	RA-specific dashboard data	None
GET	`/ra/submissions/:id`	RA submission detail view	None

Project Management

Method	Endpoint	Behavior	Auth Required
POST	`/projects`	Create new project	CHAIR, RA, ADMIN
GET	`/projects`	List all projects	None
GET	`/projects/search`	Search projects	None
GET	`/projects/:id`	Get project by ID	None
GET	`/projects/:id/full`	Get project with all relations	None
POST	`/projects/:projectId/submissions`	Create submission for project	CHAIR, RA, RA_ASST

Submission & Review

Method	Endpoint	Behavior	Auth Required
GET	`/submissions/:id`	Get submission with relations	None
PATCH	`/submissions/:id/status`	Update submission status	CHAIR, RA, ADMIN
PATCH	`/submissions/:id/overview`	Update submission overview	CHAIR, RA, ADMIN
POST	`/submissions/:submissionId/classifications`	Add/update classification	CHAIR, ADMIN
POST	`/submissions/:submissionId/reviews`	Add reviewer assignment	CHAIR, RA, ADMIN
GET	`/submissions/:id/sla-summary`	Get SLA deadline summary	None
POST	`/reviews/:reviewId/decision`	Submit review decision	REVIEWER, CHAIR
POST	`/submissions/:id/final-decision`	Record final committee decision	CHAIR, RA, ADMIN

Mail Merge & Letter Generation

Method	Endpoint	Behavior	Auth
GET	`/mail-merge/initial-ack.csv`	Bulk acknowledgment CSV	None
GET	`/mail-merge/initial-approval.csv`	Bulk approval CSV	None
GET	`/mail-merge/initial-ack/:submissionId`	Single ack letter data	None
GET	`/mail-merge/initial-ack/:submissionId/csv`	Single ack CSV	None
GET	`/mail-merge/initial-approval/:submissionId`	Single approval letter data	None
GET	`/mail-merge/initial-approval/:submissionId/csv`	Single approval CSV	None
GET	`/letters/initial-ack/:submissionId.docx`	Generate DOCX ack letter	None
GET	`/letters/initial-approval/:submissionId.docx`	Generate DOCX approval letter	None

Background Jobs / Cron / Queues

None implemented. All operations are synchronous request-response. Potential future needs:

- SLA violation notification jobs
- Automated reminder emails
- Audit log archival

---

6. Frontend

Routing Structure

```
// frontend/src/App.tsx
<Routes>
  <Route path="/" element={<Navigate to="/login" />} />
  <Route path="/login" element={<LoginPage />} />
  <Route path="/forgot-password" element={<ForgotPasswordPage />} />
  <Route path="/dashboard" element={<DashboardPage />} />
  <Route path="/projects/:projectId" element={<ProjectDetailPage />} />
  <Route path="/submissions/:submissionId" element={<SubmissionDetailPage />} />
</Routes>
```

Key Pages & Their API Calls

Page	Purpose	API Endpoints Used
`LoginPage`	User authentication (stub)	None (TODO: add auth endpoint)
`ForgotPasswordPage`	Password reset (stub)	None (TODO: add reset endpoint)
`DashboardPageNew`	Main RA dashboard	`/dashboard/queues`, `/dashboard/activity`, `/dashboard/overdue`, `/projects/search`

`ProjectDetailPage`	Project details & letter export	`/projects/:id/full`, `/mail-merge/*`, `/letters/*`
`SubmissionDetailPage`	Submission details, editing, timeline	`/submissions/:id`, `/submissions/:id/sla-summary`, `/committees`, `PATCH /submissions/:id/overview`

State Management Approach

- **No global state library** (no Redux, Zustand, etc.)
- **React hooks pattern**: Custom hooks (`useDashboardQueues` , `useSubmissionDetail` , etc.) encapsulate data fetching with `useState` / `useEffect`
- **URL-based state**: Filter selections stored in URL query params
- **Local storage**: Dashboard preferences (collapsed panels, density)

Forms & Validation Strategy

- **Controlled inputs**: Form state managed via `useState`
- **Client-side validation**: Basic required field checks before submission
- **Server-side validation**: Express handlers validate enum values, required fields
- **No form library**: No Formik, React Hook Form, or Zod integration

---

7. Security & Privacy Checklist

Authentication Method

Aspect	Current State	Risk Level
Auth mechanism	**Header-based stub** (`X-User-*` headers)	HIGH
Session management	Not implemented	HIGH
Password storage	`passwordHash` field exists, unused	MEDIUM
Token refresh	Not implemented	HIGH

RBAC/Authorization Approach

- **Middleware exists** (`authenticateUser`) but not consistently applied
- **Role definitions**: 6 roles defined in enum (CHAIR, MEMBER, RA, RA_ASST, REVIEWER, ADMIN)
- **Endpoint protection**: Designed but partially implemented (see SECURITY.md)
- **Field-level access**: Utility functions exist but not fully wired

Input Validation & Sanitization

Location	Validation Type
Route handlers	Basic required field checks, enum validation
CSV import (seed.ts)	`safeTrim`, date parsing, enum mapping
CSV export (mailMergeRoutes)	`csvEscape` for special characters
Frontend forms	Controlled inputs, basic validation

Gap: No comprehensive XSS/injection sanitization library (e.g., DOMPurify, validator.js)

File Upload / CSV Import Threat Notes

- **CSV seed parsing**: Reads external file directly, minimal sanitization
- **Document links**: Stored as URLs (e.g., Google Drive), not file uploads
- **DOCX generation**: Built server-side with `docx` library, controlled content
- **No file upload endpoint exists** (future risk if added)

Logging/Auditing Approach

- **Audit log model designed** in SECURITY.md but **not fully implemented**
- **Console logging**: `console.error` on all catch blocks
- **No centralized logging** (no Winston, Pino, or log aggregation)
- **No request logging middleware**

Obvious Secret Exposure Risks

- No hardcoded secrets found in source code
- `DATABASE_URL` must be in `.env` (not checked into repo)
- No `.env.example` file visible in workspace (mentioned in README but not found)

---

8. Testing & Quality

Existing Tests

Path	Type	Description
`backend/tests/unit/workingDays.test.ts`	Unit	Working days calculation tests
`backend/tests/api/security.test.ts`	API	Security/RBAC tests
`backend/tests/integration/schemaIntegrity.test.ts`	Integration	Database schema tests
`backend/tests/helpers/prismaCleanup.ts`	Helper	Test data cleanup utility

How to Run Tests

```
cd backend
npm test                 # Run all tests
npm run test:unit        # Unit tests only
npm run test:integration  # Integration tests
npm run test:api         # API tests
```

Lint/Format Hooks

- **ESLint**: Configured for frontend (`frontend/package.json` has `lint` script)
- **Prettier**: Not found in config
- **Husky/lint-staged**: Not configured (no pre-commit hooks)

Coverage Gaps (Inferred)

1. **No frontend tests**: No test files in `frontend/src`
2. **No E2E tests**: No Playwright, Cypress, or similar
3. **Limited API test coverage**: Only security tests found

4. **No CI pipeline**: Tests not automated in GitHub Actions/similar

---

9. Current Known Issues / TODO Extraction

TODO/FIXME Comments

File	Line	Comment
`projectRoutes.ts`	59	`TODO: replace with real logged-in user later`
`submissionRoutes.ts`	201	`TODO: replace with authenticated user later`
`submissionRoutes.ts`	511	`TODO: replace with authenticated user later`
`LoginPage.tsx`	121	`TODO: Replace with actual API call once backend auth is implemented`
`ForgotPasswordPage.tsx`	20	`TODO: Replace with actual API call`

Likely Fragile Modules

Module	Reason	File Reference
`DashboardPageNew.tsx`	2335 lines, complex state, should be split	`frontend/src/pages/DashboardPageNew.tsx`
`submissionRoutes.ts`	893 lines, multiple responsibilities	`backend/src/routes/submissionRoutes.ts`
`seed.ts`	1361 lines, complex CSV parsing	`backend/src/config/seed.ts`
Auth middleware	Exists but not consistently wired	`docs/SECURITY.md` describes design
SLA calculations	Simple working-days only, no holiday handling	`backend/src/utils/slaUtils.ts`

Technical Debt Observations

1. **Large files**: Multiple files exceed 500 lines, violating single-responsibility
2. **Auth incomplete**: Header-based auth is insecure, needs production implementation
3. **Hardcoded IDs**: `createdById: 1` used throughout
4. **No API versioning**: Routes at root, no `/api/v1` prefix
5. **Mixed concerns**: Route files contain business logic, should extract to services
6. **No error handling middleware**: Individual try/catch in each handler

---

10. Consultant-Ready "Next Questions"

High Priority (Security/Architecture)

1. **What authentication provider will production use?** (LDAP, OAuth, custom JWT?)
2. **What is the user provisioning process?** (Manual creation, SSO sync?)
3. **Are there compliance requirements?** (Data retention, audit trail mandates?)
4. **What is the expected concurrent user load?** (Sizing for database/server)
5. **Is there an existing deployment infrastructure?** (On-prem, cloud, containerized?)

Medium Priority (Functionality)

6. **How should SLA deadlines handle holidays?** (The `Holiday` table exists but isn't used)
7. **What email service will send notifications?** (SMTP, SendGrid, etc.)
8. **Are there other committees beyond RERC-HUMAN?** (Multi-committee support is modeled)
9. **What reports do chairs/admins need beyond the dashboard?**
10. **How should the letter templates be managed?** (Hardcoded vs. admin-configurable)

Data & Integration

11. **Is there existing data to migrate beyond the sample CSV?**
12. **Does this system need to integrate with other university systems?** (HRMS, LMS, etc.)
13. **What is the document storage strategy?** (Currently just URLs, need upload?)
14. **How should continuing reviews be scheduled and reminded?**
15. **What happens when a project approval period expires?**

Process Clarification

16. **What is the exact workflow for full-board reviews?** (Meeting scheduling, voting)
17. **How are reviewer honoraria processed?** (Model exists, process unclear)
18. **What defines "completeness" for a submission?** (Checklist, documents?)
19. **Who can reassign submissions to different staff?**
20. **What audit records must be retained and for how long?**

Appendix A: Quick Reference Commands

```
# Start development (both servers)
npm run dev

# Backend only
npm run dev:backend

# Frontend only
npm run dev:frontend

# Database commands
npm run db:generate    # Generate Prisma client
npm run db:migrate     # Run migrations
npm run db:seed        # Seed database

# Open Prisma Studio (DB GUI)
cd backend && npx prisma studio

# Run tests
cd backend && npm test
```

Appendix B: Security Checklist Confirmation

- ✓ **No real secrets/tokens in this document**
- ✓ **DATABASE_URL shown as placeholder only**

- ✓ **No real user emails or passwords exposed**
- ✓ **Noted security gaps are design concerns, not exploits**

Document generated by analyzing repository structure, source files, and documentation. Some details are inferred where explicit documentation was incomplete.