
Technical Project Report

A comprehensive technical analysis of the CollabSpace full-stack team collaboration workspace — architecture, implementation, security posture, and improvement roadmap.

AUTHOR

ELMAOUI YAHYA

LICENSE

MIT

FRONTEND

React 19 + Vite 8

BACKEND

NestJS 11 + TypeScript

DATABASE

MySQL via TypeORM

00 — INDEX

TABLE OF CONTENTS

01	Project Overview	3
02	System Architecture	5
	Monorepo layout & communication pattern	5
03	Frontend Application	8
	Technology stack, routing, component architecture, state management	8
04	Backend Application	12
	NestJS modules, services, database configuration	12
05	Database Schema	16
	All 7 tables with columns, types, and constraints	16
06	API Reference	19
	Auth, Teams, Projects, Tasks endpoints	19
07	User Workflow & Roles	22
	Role matrix, onboarding flow, project delivery flow	22
08	Security Analysis	24
09	Testing Coverage	25
10	Installation & Setup	26
11	Development Scripts	28
12	Conclusion & Roadmap	30

Project Overview

CollabSpace is a full-stack monorepo for team collaboration, enabling organisations to manage projects, track tasks, coordinate teams, and monitor progress through a unified platform.

The application is structured as a monorepo containing two independently runnable applications: a React single-page application serving the user interface, and a NestJS REST API responsible for all business logic and data persistence. Communication between them is handled over HTTP with JWT-based authentication.

Key Metrics

2 APPLICATIONS	5 API MODULES	7 DB TABLES	3 USER ROLES
11 TEST SPECS	25+ ENDPOINTS		

Core Features

Project Management Create, assign, and track projects within teams. Monitor status through a full lifecycle from pending to completion with deadline tracking.	Task Tracking Granular task management with a three-stage lifecycle: pending → ongoing → completed. Assignee-based delegation per project member.
Team Management Managers create and govern teams, assign members, and oversee project delivery. Role-based access enforces clear responsibility boundaries.	Analytics Dashboard Interactive charts built with Chart.js display project and task insights, completion rates, and team productivity metrics in real time.
Authentication System JWT-based sign-in combined with an admin-gated guest registration flow. New users submit a request that an administrator must approve before access is granted.	Calendar & Deadlines Full and mini calendar components alongside deadline proximity queries keep teams aligned with project milestones and individual task due dates.

System Architecture

The project follows a clear client-server separation within a monorepo, with the React frontend consuming the NestJS REST API over HTTP.

```
[ Browser (Client) ]
    |
(HTTP + Bearer JWT)
    ▼
[ React 19 SPA (Port 5173) ]
    |
  (REST API)
    ▼
[ NestJS 11 API (Port 3000) ]
    |
  (TypeORM)
    ▼
[ MySQL 8 DB (Port 3306) ]
```

Monorepo Directory Layout

```
CollabSpace/
├─ frontend/           React 19 SPA – served on port 5173
│  └─ src/
│     ├─ main.jsx      Entry point (AuthProvider + BrowserRouter)
│     ├─ App.jsx       Root component with route rendering
│     ├─ api/client.js  HTTP client with Bearer token injection
│     ├─ context/      AuthContext – JWT state management
│     ├─ routes/       Public and protected route definitions
│     ├─ pages/        Page-level components
│     ├─ components/   Reusable UI components
│     ├─ layouts/      Sidebar + main content wrapper
│     └─ Style/        CSS Module files
├─ backend/           NestJS 11 REST API – served on port 3000
│  └─ src/
│     ├─ main.ts        Bootstrap, CORS configuration
│     ├─ app.module.ts  Root module, TypeORM configuration
│     ├─ auth/          Authentication module
│     ├─ users/         User profile module
│     ├─ teams/         Teams module
│     ├─ projects/      Projects module
│     └─ tasks/         Tasks module
├─ docs/              Project documentation
└─ package.json       Monorepo root scripts
```

Communication Pattern

The React frontend communicates with the NestJS backend exclusively via HTTP REST calls. CORS is restricted to `http://localhost:5173` with credentials enabled, configured explicitly in `main.ts`. All authenticated calls attach a `Authorization: Bearer <token>` header, injected by the `api/client.js` HTTP client module.

Layer	Technology	Port	Role
Frontend	React 19 + Vite 8	5173	User interface, client-side routing
Backend	NestJS 11 + Express	3000	REST API, business logic
Database	MySQL 8	3306	Persistent data storage

Frontend Application

A React 19 Single Page Application built with Vite 8, featuring client-side routing, context-based authentication, animated UI components, and Chart.js data visualisation.

Technology Stack

Library	Version	Purpose
React	19	UI framework
Vite	8	Bundler and development server with HMR
react-router-dom	7	Client-side routing
motion (Framer Motion)	12	Declarative animations
lottie-react / dotlottie-react	latest	Lottie JSON animations
typewriter-effect	2	Typewriter animation for landing
Chart.js / react-chartjs-2	4 / 5	Interactive charts (bar, doughnut)
date-fns	4	Date formatting and arithmetic
FontAwesome	7	Icon library
react-select	5	Enhanced dropdown select components
CSS Modules	—	Scoped component-level styling

Page Routes

Path	Component	Description	Access
/	LandingPage	Public marketing page with animations	Public
/SignIn	SignIn	JWT credential login form	Public
/register	Register	Guest registration — creates a pending request	Public
/RequestStatus	RequestStatus	Check status of a submitted registration request	Public
/collabSpace	MainPage	Full authenticated workspace (Dashboard, Projects, Tasks, Teams, Social, Requests)	Authenticated

Component Groups

Group	Components
Pages / Main	Dashboard, MainPage, Projects, Tasks, Teams, Social, Requests, LandingPage
Pages / Auth	SignIn, Register, RequestStatus
Forms	AddProject, AddTask, AddTeamMate, CreateTeam, RegisterForm, SignInForm, StatusForm, AddProjectMember
Charts	BarChart, DoughnutChart, ChartCard
Calendar	MainCalendar, MiniSidebarCalendar
Landing	CardComponent, LandingFeatures, LandingFooter, LandingHeader, LandingHome
StructuralUI	Header, Insights, Members, DataBox, ConfirmationBox, UserBox, RequestList
StructuralUI / feeds	Feeds, ProjectCard, TaskCard, MemberCard, Deadlines, CompletedProject
StructuralUI / projects	ProjectCard, ProjectDash, ProjectList
StructuralUI / teams	TeamCard, TeamDash, TeamList
StructuralUI / social	UserCard, UserDash, UserList
StructuralUI / requests	AdminRequest, MembershipRequestCard, TeamRequestCard

State Management

Mechanism	Scope	What It Stores
AuthContext (React Context)	Global	Current user object (id, role, token), login and logout actions
localStorage	Browser persistent	Serialised user JSON — survives page refreshes
Component state (useState)	Local	Page-level data fetched fresh from the API on each mount

Authentication Flow

The frontend authentication is built around the `AuthContext` provider, which wraps the entire application at the root level in `main.jsx`. Upon a successful sign-in, the user object returned by the API is stored both in React state and serialised to `localStorage` to persist across sessions. All subsequent API calls inject the token via the centralised `api/client.js` module. Logout clears both sources.

Backend Application

A NestJS 11 REST API following the standard Controller → Service → Repository pattern, connected to MySQL via TypeORM 0.3 with CORS restricted to the frontend origin.

Technology Stack

Library	Version	Purpose
NestJS	11	Node.js framework (Express adapter)
TypeScript	5.9	Statically typed language
TypeORM	0.3	ORM for MySQL with QueryBuilder support
mysql2	3	MySQL database driver
class-validator	0.15	Decorator-based DTO input validation
class-transformer	0.5	Request and response object transformation
@nestjs-modules/mailer + nodemailer	2 / 8	Email sending capability
Jest + supertest	30 / 7	Unit and end-to-end testing

Module File Convention

Every feature module follows the standard NestJS structure, ensuring a predictable and consistent codebase:

```

module/
├─ *.module.ts           Module definition – imports, providers,
                        controllers, exports
├─ *.controller.ts       HTTP layer – route handlers, parameter parsing
├─ *.service.ts          Business logic – TypeORM queries, error handling
├─ dto/                  Data Transfer Objects with class-validator
decorators
├─ entities/             TypeORM entity class definitions
├─ *.controller.spec.ts  Controller unit tests
└─ *.service.spec.ts     Service unit tests

```

Auth Module

Manages the full user lifecycle: sign-in via plaintext credential comparison, guest registration creating a `GUEST_REQUEST` record, admin-driven acceptance or rejection of requests, role promotion and demotion, and cascaded user deletion that removes both the `USER` and linked `USERDATA` records.

Users Module

Provides CRUD operations on the `USERDATA` entity, which holds personal profile information (name, CIN, email, phone, birth date, profile picture URL). This entity is linked 1-to-1 with the auth `USER` entity via a shared primary key.

Teams Module

Handles team creation — automatically promoting the creator to the `manager` role — bulk member assignment, individual member removal, team-level project and task insights, a member productivity leaderboard (ranked by completed task count), and team deletion with cascaded member cleanup.

Projects Module

Manages project creation under a team (automatically adding the creator as a member), status lifecycle transitions (`pending → ongoing → completed`), member assignment with availability querying, deadline proximity reports, and both project-level and system-wide admin statistics.

Tasks Module

Covers task creation with assignee linking within a project context, status updates validated against an allowed enum (`pending` | `ongoing` | `completed`), deadline proximity queries for a given user, and user-level aggregated task statistics (total, completed, ongoing, pending counts).

Database Configuration

```
TypeOrmModule.forRoot({
  type      : "mysql",
  host      : "localhost",
  port      : 3306,
  username  : "penguin",
  password  : "penguin123",
  database  : "collabSpaceDB",
  entities  : [User, UserData, guestRequest, Project, projectMember, Task,
Team],
  synchronize : false,    // schema managed externally via ERD file
})
```

Schema Management

synchronize:
false

means
TypeORM
will not
auto-
create or
alter
tables.
The ERD
file
located
at

backend/src/database/collabSpacedb.erd

must be
used to
generate
the
initial
schema
before
the first
run.

Database Schema

Seven relational tables in MySQL (`collabSpaceDB`), using *ENUM* columns for role and status fields and foreign-key constraints across all entity relationships.

USER

Column	Type	Constraint	Notes
id	int	PK, AUTO_INCREMENT	Primary identifier
identifier	varchar	UNIQUE NOT NULL	Login username
password	varchar	NOT NULL	Stored as plaintext — see security section
role	ENUM	admin manager member	Controls access level throughout the app
teamId	int	FK → TEAM.teamID, nullable	Null when user is not assigned to a team
requestId	int	FK → GUEST_REQUEST.id	Links back to the original onboarding request

USERDATA

Column	Type	Notes
id	int (FK)	References USER.id — 1-to-1 relation (shared PK)
firstName / lastName	varchar	Display name, concatenated in queries
cin	varchar	National identity card number
gender	varchar	—
email	varchar	Contact email address
phoneNumber	varchar	—
birthDate	date	Age computed dynamically using TIMESTAMPDIFF(YEAR, birthDate, CURDATE())
picture	varchar	Profile picture URL

GUEST_REQUEST

Column	Type	Notes
id	int	PK, AUTO_INCREMENT
firstName, lastName, cin, gender, email, phoneNumber, birthDate	various	Applicant's personal information
identifier / password	varchar	Desired login credentials submitted by the applicant
status	ENUM	pending accepted rejected — ordered in UI queries
createdAt / updatedAt	timestamp	Auto-managed by TypeORM

TEAM

Column	Type	Notes
teamID	int	PK, AUTO_INCREMENT
Name	varchar	Team display name
memberCount	int	Denormalised count — set to 1 on creation
ManagerID	int	FK → USER.id — the team's designated manager

PROJECTS

Column	Type	Notes
id	int	PK, AUTO_INCREMENT
projectName	varchar	—
description	text	—
status	ENUM	pending ongoing completed
deadLine	date	Project deadline for deadline proximity queries
createdAt / completionDate	timestamp	Completion date used to compute time-since-completion
CreatorID	int	FK → USER.id — user who created the project
teamID	int	FK → TEAM.teamID — owning team

PROJECT_MEMBERS (junction)

Column	Type	Notes
userID	int	FK → USER.id — composite PK
projectID	int	FK → PROJECTS.id — composite PK

Many-to-many junction table between users and projects. Uses `INSERT OR IGNORE` for idempotent bulk assignment.

TASK

Column	Type	Notes
TaskID	int	PK, AUTO_INCREMENT
taskName	varchar	—
description	text	—
status	ENUM	pending ongoing completed — validated in service layer
deadLine	date	Task deadline for upcoming-deadlines queries
createdAt	timestamp	—
assigneeID	int	FK → USER.id — responsible team member
projectID	int	FK → PROJECTS.id — parent project

API Reference

All endpoints are served from `http://localhost:3000`. Authenticated calls must include the header `Authorization: Bearer <token>`.

Authentication

Method	Endpoint	Description
POST	/auth/signin	Validate credentials, return user object (password excluded)
POST	/auth/register	Create guest request with status pending
POST	/auth/existingRequest	Look up a request by CIN, email, and phone number
GET	/auth/requests	List all guest requests, ordered pending → accepted → rejected
GET	/auth/userData	All users with profiles (admin excluded)
GET	/auth/noTeamMembers	Members not yet assigned to any team
GET	/auth/userTeam/:id	Return the team ID for a given user
GET	/auth/taskInfo/:id	Task list for a specific user
GET	/auth/userProjects/:teamId/:userId	Projects belonging to a user within a team
PATCH	/auth/acceptedRequest/:id	Accept request — creates USER and USERDATA records
PATCH	/auth/rejectedRequest/:id	Reject a pending request
PATCH	/auth/updateStatus/:id	Update a user's role
PATCH	/auth/demoteToMember/:id	Demote manager to member and unassign from team leadership
DELETE	/auth/deleteUser/:id	Remove user and their associated profile data

Teams

Method	Endpoint	Description
POST	/teams/createTeam	Create team and promote creator to manager role
POST	/teams/assignMultipleToTeam/:teamId	Bulk-assign an array of users to a team
GET	/teams/teamMembers/:id	List team members with full names
GET	/teams/teamsList	All teams with associated manager information
GET	/teams/projectsInsights/:id	Project statistics for a team (total, completed, ongoing, pending)
GET	/teams/teamMembersTasks/:id	Completed task count per member (productivity leaderboard)
GET	/teams/teamProjects	Completed project count per team (top 10)
DELETE	/teams/deleteTeam/:id	Delete team and unassign all members

Projects

Method	Endpoint	Description
POST	/projects/addProject/:teamId	Create project under a team; auto-adds creator as member
POST	/projects/assignMultipleToProject	Bulk-add members to a project (idempotent)
GET	/projects/projectMembers/:projectId	List project members with full names
GET	/projects/availableMembers/:projectId/:teamId	Team members not yet assigned to this project
GET	/projects/tasksInsights/:id	Task statistics for a project
GET	/projects/projectDeadline/:teamId	Deadline overview with remaining days per project
GET	/projects/adminInsights	System-wide employee and project statistics
PATCH	/projects/updateStatus/:id	Update project status
DELETE	/projects/deleteProject/:id	Remove a project

Tasks

Method	Endpoint	Description
POST	/tasks/createtask/:projectId	Create a task within a project with a specified assignee
GET	/tasks/tasksInsights/:id	Aggregated task statistics for a user
GET	/tasks/taskDeadline/:id	Upcoming task deadlines for a user (incomplete tasks only)
PATCH	/tasks/updateTaskState/:taskId/:newStatus	Update task status (validated: pending ongoing completed)
DELETE	/tasks/deleteTask/:taskId	Delete a task

User Workflow & Roles

CollabSpace enforces a three-tier role system — Admin, Manager, Member — each with clearly delineated responsibilities and access boundaries.

Role Capabilities

Capability	Admin	Manager	Member
View dashboard	System-wide	Team-level	Personal
Manage guest requests	Yes	No	No
Create / delete teams	Yes	Yes (own)	No
Assign users to teams	Yes	Yes	No
Create / delete projects	Yes	Yes	No
Assign project members	Yes	Yes	No
Create / delete tasks	Yes	Yes	Yes
Update task status	Yes	Yes	Yes
Update project status	Yes	Yes	No
Manage user roles	Yes	No	No

Onboarding Flow

A new user submits a registration form at `/register`, which creates a `GUEST_REQUEST` record with status `pending`. The admin reviews all pending requests via the Requests page and either accepts or rejects each. Upon acceptance, the system automatically creates a `USER` record and a linked `USERDATA` profile using the information from the original request. The user can then sign in at `/SignIn`.

Project Delivery Flow

A manager creates a team, becoming its designated manager with automatic role promotion. Members are assigned to the team from the pool of users without a current team assignment. The manager then creates one or more projects under the

team, selecting project members from the team roster. Team members create tasks within those projects, assigning them to specific individuals. Task status is updated progressively from `pending` to `ongoing` and finally `completed`. The dashboard charts reflect this progress in real time via the insights endpoints.

Security Analysis

An objective assessment of the current security posture, identifying vulnerabilities with severity ratings and concrete remediation recommendations.

Severity	Issue	Current State	Recommendation
Critical	Plaintext password storage	Passwords stored and compared as raw strings in both USER and GUEST_REQUEST tables	Replace with bcrypt or Argon2 hashing in the auth service before any deployment
Critical	No JWT guards on API routes	The API validates credentials on sign-in but does not protect subsequent endpoints	Implement <code>@UseGuards(AuthGuard('jwt'))</code> using Passport.js or a custom guard
High	Hardcoded database credentials	MySQL username and password are hardcoded in <code>app.module.ts</code>	Migrate to environment variables using <code>@nestjs/config</code> and a <code>.env</code> file
High	No global ValidationPipe	DTO validation decorators exist but no global pipe is registered in <code>main.ts</code>	Add <code>app.useGlobalPipes(new ValidationPipe({ whitelist: true })))</code>
Medium	No rate limiting on authentication	Sign-in and registration endpoints are unprotected against brute-force attacks	Add <code>@nestjs/throttler</code> with appropriate limits on auth routes
Medium	No HTTPS enforcement	The application communicates over plain HTTP	Configure TLS termination via a reverse proxy (Nginx) for any production deployment
Low	CORS configuration	Correctly restricted to <code>http://localhost:5173</code> with credentials	Update the allowed origin list when deploying to production

Production Readiness The two critical vulnerabilities — plaintext passwords and unguarded API routes — must be resolved before this application is deployed to any environment accessible outside localhost.

Testing Coverage

The backend uses Jest 30 with supertest. The frontend currently has no test framework configured.

Backend Specification Files

File	Type	Current State
test/app.e2e-spec.ts	End-to-end	Stub — contains only "should be defined"
src/app.controller.spec.ts	Unit	Stub
src/auth/auth.controller.spec.ts	Unit	Stub
src/auth/auth.service.spec.ts	Unit	Stub
src/users/users.controller.spec.ts	Unit	Stub
src/users/users.service.spec.ts	Unit	Stub
src/projects/projects.controller.spec.ts	Unit	Stub
src/projects/projects.service.spec.ts	Unit	Stub
src/tasks/tasks.controller.spec.ts	Unit	Stub
src/tasks/tasks.service.spec.ts	Unit	Stub
src/teams/teams.controller.spec.ts	Unit	Stub

Action Required All 11 specification files are currently stubs. A meaningful test suite should include service tests with mocked TypeORM repositories, controller tests with mocked services, and E2E tests covering the full HTTP request lifecycle for each module.

Frontend Testing Gap No testing framework is configured in the frontend application. It is recommended to add Vitest and React Testing Library as development dependencies to cover component behaviour and user interaction flows.

Installation & Setup

CollabSpace requires Node.js $\geq 18.0.0$ and a MySQL 8 instance running on `localhost:3306` with a pre-created database and user.

Prerequisites

Requirement	Minimum Version	Notes
Node.js	18.0.0	LTS release recommended
npm	9.0.0	Bundled with Node.js
MySQL	8.x	Running on localhost, port 3306
Database	—	Named <code>collabSpaceDB</code> — must be created manually
MySQL user	—	Username <code>penguin</code> , password <code>penguin123</code> , full access on collabSpaceDB

Installation Steps

```
# 1. Clone the repository
git clone <repository-url>
cd CollabSpace

# 2. Install all dependencies (frontend and backend)
npm run install-all

# 3. Prepare the MySQL database
mysql -u root -p -e "CREATE DATABASE collabSpaceDB;"
mysql -u root -p -e "CREATE USER 'penguin'@'localhost' IDENTIFIED BY
'penguin123';"
mysql -u root -p -e "GRANT ALL PRIVILEGES ON collabSpaceDB.* TO
'penguin'@'localhost';"
# Import the schema using: backend/src/database/collabSpacedb.erd

# 4. Start the backend (Terminal 1)
cd backend
npm run start:dev
# Server listening on http://localhost:3000

# 5. Start the frontend (Terminal 2)
cd frontend
npm run dev
# Application available at http://localhost:5173
```

Development Scripts

Monorepo Root

Script	Description
<code>npm run install-all</code>	Install dependencies for both frontend and backend in a single command
<code>npm run clean</code>	Remove all <code>node_modules</code> directories across the monorepo
<code>npm run lint-all</code>	Run ESLint on both the frontend and backend projects

Frontend (cd frontend/)

Script	Description
<code>npm run dev</code>	Start the Vite development server with Hot Module Replacement on port 5173
<code>npm run build</code>	Compile and bundle for production into the <code>dist/</code> directory
<code>npm run preview</code>	Serve the production build locally for verification
<code>npm run lint</code>	Run ESLint on the frontend source

Backend (cd backend/)

Script	Description
<code>npm run start:dev</code>	Start in watch mode — TypeScript is compiled on save and the server auto-reloads
<code>npm run start:prod</code>	Run the compiled production build from dist/
<code>npm run build</code>	Compile TypeScript source to JavaScript in dist/
<code>npm run test</code>	Run all Jest unit tests
<code>npm run test:cov</code>	Run tests with a coverage report
<code>npm run test:e2e</code>	Run end-to-end tests using the jest-e2e configuration
<code>npm run lint</code>	Run ESLint with automatic fix on the backend source
<code>npm run format</code>	Apply Prettier formatting to all source and test files

Conclusion & Roadmap

CollabSpace demonstrates a well-structured full-stack implementation with a solid architectural foundation. The following assessment identifies both strengths and the steps required to reach production readiness.

Strengths

Area	Assessment
Architecture	Standard NestJS layered pattern is clean, predictable, and straightforward to extend with new modules
Frontend	React 19 with a rich library set (Framer Motion, Lottie, Chart.js) delivers a polished user experience
Access Control	Three well-defined roles provide clear responsibility boundaries across all application features
Data Queries	TypeORM QueryBuilder is used appropriately for complex joins and aggregated analytics, avoiding N+1 patterns
Monorepo Structure	Clean separation between frontend and backend enables independent development and deployment

Prioritised Improvement Roadmap

Priority	Item	Action Required
Critical	Password hashing	Implement bcrypt or Argon2 in the auth service for all password operations
Critical	JWT route protection	Add <code>@UseGuards</code> to all protected endpoints using Passport.js or a custom guard
High	Environment configuration	Replace all hardcoded secrets with environment variables via <code>@nestjs/config</code>
High	Test coverage	Replace all stub spec files with meaningful unit tests and at least one E2E test suite
Medium	Frontend testing	Integrate Vitest and React Testing Library for component-level test coverage
Medium	Rate limiting	Add <code>@nestjs/throttler</code> to protect authentication and registration endpoints
Low	Containerisation	Write a Docker Compose file to package the backend and MySQL for reproducible environments
Low	CI/CD pipeline	Set up GitHub Actions workflows for automated lint, test, and build checks on every push

Once the critical and high-priority items above are addressed, CollabSpace will be fully suitable for a production deployment and represent a robust, maintainable platform for team collaboration at scale.