Product Requirements Document

Project: DevHub – Developer Task & Project Management App (Training Build)

Objective

Build a small but realistic **Ruby on Rails application** that simulates a production-grade environment.
The purpose is twofold:

1. Teach engineers to apply the **Rails architecture and idioms** through hands-on coding.
2. Deliver a working internal tool for managing projects, tasks, and activity history — structured similarly to the **customer's modular Rails ecosystem**.

After Week 2, the app should be deployable, tested, and demonstrate all core competencies required for contributing to the customer's codebase.

High-Level Description

DevHub lets users:

- Create and manage *Projects* and *Tasks*
- Assign tasks to *Users* (polymorphic association)
- Track updates asynchronously via background jobs
- Expose all data through a **GraphQL API** (for the React client)
- Organize the codebase into **modular Rails Engines** (Core + Admin)
- Authenticate users with **Authlogic**
- Process background jobs with **Sidekiq/ActiveJob**
- Be easily extendable (e.g., reporting, notifications)

Architecture Overview

```
app/
 ├── models/
 │    ├── project.rb
 │    ├── task.rb
 │    ├── user.rb
 │    └── activity.rb
 ├── services/
 │    └── task_status_updater.rb
 ├── graphql/
 │    ├── types/
 │    ├── queries/
 │    └── mutations/
 ├── engines/
 │    ├── admin/   (dashboard, reports)
 │    └── core/    (main business logic)
 ├── jobs/
 │    └── activity_logger_job.rb
```

Frontend:
A small **React + Redux + Apollo** app consumes the GraphQL API for task dashboards.

Functional Requirements

#	Feature	Description	Acceptance Criteria
1	**User Authentication (Authlogic)**	Users can sign up, log in/out, and manage sessions.	Authlogic configured; users stored in DB; sessions persist.
2	**Projects CRUD**	Create/edit/delete projects.	Rails views and GraphQL mutations work; validations enforced.

3	**Tasks CRUD**	Each project has many tasks; tasks have a polymorphic assignee.	Tasks linked to Projects and Users; CRUD works via API/UI.
4	**ActiveRecord Scopes**	Implement reusable query scopes.	.completed, .recent, .assigned_to (user) return expected results.
5	**Service Objects**	Move business logic out of models/controllers.	At least one service (TaskStatusUpdater) handles updates.
6	**Rails Engines**	Split code into Core and Admin engines.	Admin engine mounted at /admin; namespaces and routes isolated.
7	**GraphQL API**	Implement queries & mutations for Projects/Tasks.	GraphiQL explorer returns expected data; mutations persist.
8	**Background Jobs (Sidekiq)**	Log activity asynchronously.	ActivityLoggerJob runs in Sidekiq when a task changes status.
9	**Testing**	Add RSpec and Minitest coverage for core components.	All tests passing in Jenkins pipeline.
10	**React Client**	Front-end consumes GraphQL API with Redux + Apollo.	UI lists projects/tasks and reflects live data changes.

Data Model (simplified)

User

- id, name, email, password_digest

Project

- id, name, description

Task

- id, title, description, status, project_id, assignee_type, assignee_id

Activity

- id, record_type, record_id, action, created_at

Tech Stack

- **Backend:** Ruby 3.x, Rails 7.x
- **DB:** PostgreSQL
- **Auth:** Authlogic
- **Jobs:** Sidekiq + ActiveJob
- **API:** graphql-ruby
- **Frontend:** React 18 + Redux Toolkit + Apollo Client
- **Tests:** RSpec + Minitest
- **Infra:** Render / Heroku, Bitbucket, Jenkins

Development Plan

Week	Focus	Key Deliverables
Week 1	Rails Core, CRUD, Scopes, Engines, Service Objects	Running Rails app with Projects / Tasks / Users, one Engine mounted, service object demo
Week 2	Authlogic, GraphQL, Sidekiq, React Integration, Testing	Auth, background jobs, GraphQL API, connected React client, passing tests

Deliverables

- Fully functional Rails 7 application with modular structure.
- GraphQL endpoint and example queries.
- React client consuming live data.
- Test suite and Jenkins pipeline.
- Documentation: architecture diagram, setup README, schema ERD.

Success Metrics

- App deploys and runs locally for all participants.
- Background job executes correctly.
- GraphQL queries and mutations verified via GraphiQL and React UI.
- All tests pass in CI.
- Mentor approval for code style, architecture, and idiomatic Ruby usage.