SOFTWARE ARCHITECTURE
N02 • GROUP 2

Phenikaa University
School of Computing

Kyte

Kanban Board Application

Modern Software Architecture & Engineering • End-to-End Type Safety • Real-Time Collaboration

Trần Thành Long
Full-Stack Implementation

Nguyễn Xuân Mạnh
Architecture & Documentation

Feb 7, 2026
CSE703110-1-2-25(N02)

Mô tả dự án

Project Description

What is Kyte?

A **real-time collaborative Kanban system** designed for modern software teams. Built with a focus on performance, type safety, and seamless multi-user collaboration.

Core Goal

Solving the **latency and data integrity issues** inherent in multi-user collaboration scenarios. Ensuring smooth drag-and-drop experiences with sub-100ms interaction times.

Key Advantage

Using a **"Type-Safe" bridge** between client and server through Eden Treaty, eliminating an entire class of production errors and API contract mismatches.

System Highlights

- **<100ms Latency** — Optimistic UI
- **Real-Time Sync** — WebSocket Events
- **RBAC Security** — Workspace Isolation
- **Modular Design** — Clean Boundaries

Phân tích yêu cầu

Requirements Analysis

Quality Attributes (NFR)

Performance

Interaction latency under 100ms for drag-and-drop using Optimistic UI

<100ms

Security

RBAC enforced at service layer with workspace membership validation

100%

Maintainability

Strict separation of concerns with controller/service/repository pattern

3-Tier

Additional Quality Attributes

Scalability
Stateless API nodes with DB session storage

Data Integrity
ACID compliance for all mutations

ASR: Architecturally Significant Requirements

ASR-1 Business Logic Independence

Keep business logic independent from UI to allow future clients

→ Enforces layered design with clean module boundaries

ASR-2 Server-Side Access Validation

All write operations must validate board and workspace access on server

→ Requires auth plugin and service-level permission checks

ASR-3 Concurrent Update Handling

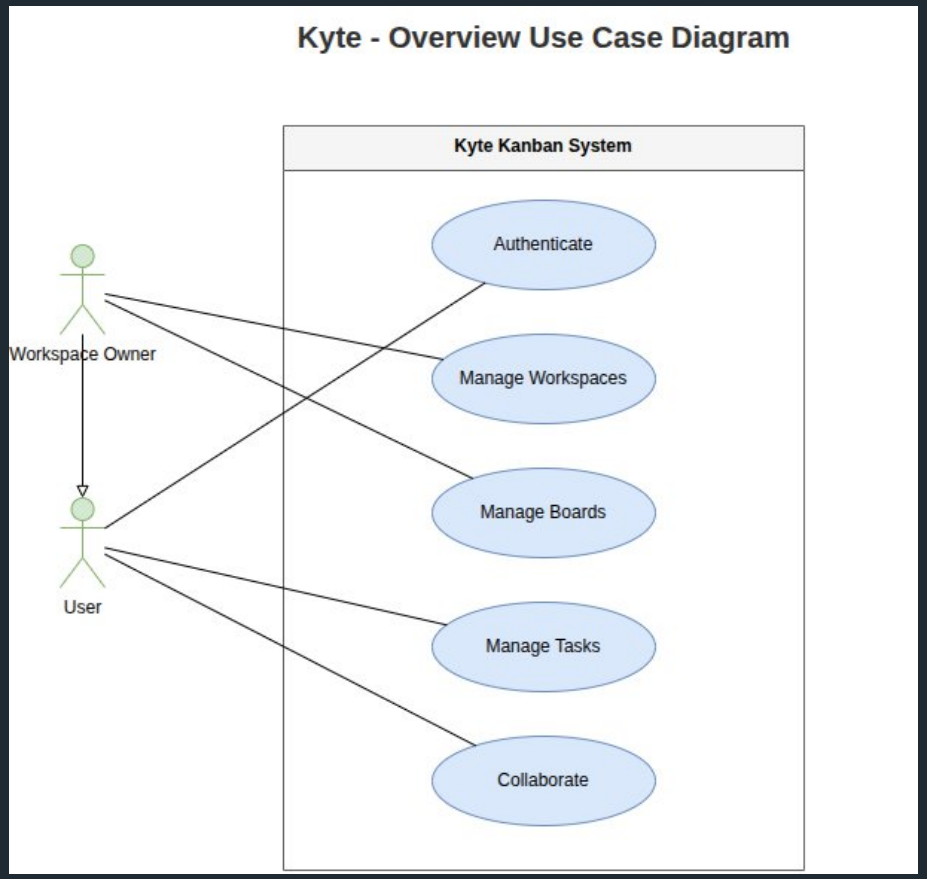
Task ordering must remain correct under concurrent updates

→ Uses fractional indexing and optimistic locking version fields

Sơ đồ chức năng

Function Use Case

Kyte - Overview Use Case Diagram



Actors

Workspace Owner / Admin

Management role with full control over workspace lifecycle, member invitations, and board governance

Board Member

Operational role focused on task management, collaboration, and day-to-day board activities

Key Actions

01 **Create & Manage Workspace**
Multi-tenant container initialization

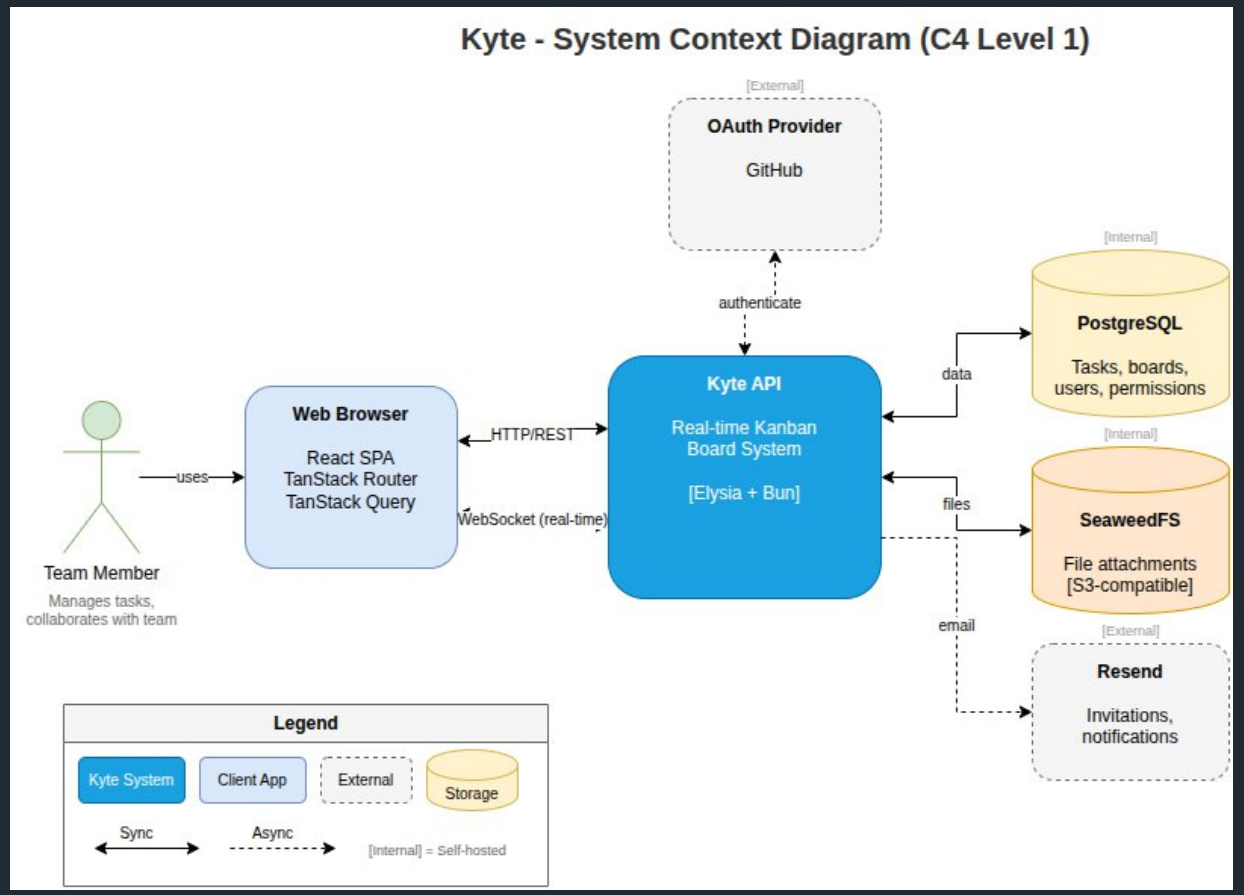
02 **Manage Tasks**
CRUD operations with drag-and-drop

03 **Real-time Collaboration**
WebSocket-based live synchronization

C4 Model • Level 1

System Context

Kyte - System Context Diagram



System Boundary

The Kyte System sits at the center, mediating interactions between users and external services. It defines the relationship between the core Kanban application and all external actors.

Core System

Real-time Kanban Board System built with Elysia + Bun, handling business logic, real-time sync, and authentication

External Integration

GitHub OAuth
Identity provider for user authentication

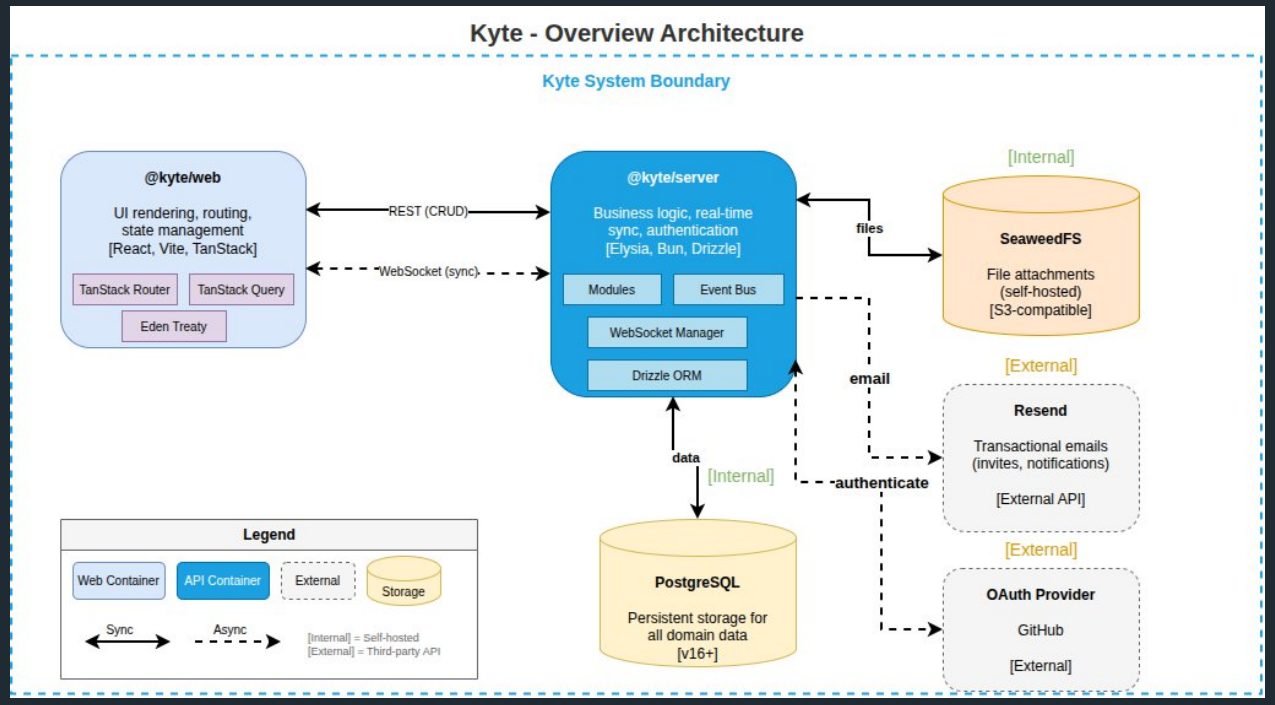
Resend
Transactional emails for invitations & notifications

SeaweedFS
S3-compatible file storage for attachments

C4 Model • Level 2

High-Level Architecture

Kyte - Overview Architecture



Architecture Pattern

Client-Server Architecture

Clear separation between UI and business logic to ensure scalability and security. Overview Container view showing all major system components.

Core Technologies

React SPA

Single Page Application with TanStack Router & Query

Bun / Elysia API

High-performance runtime with type-safe web framework

Communication

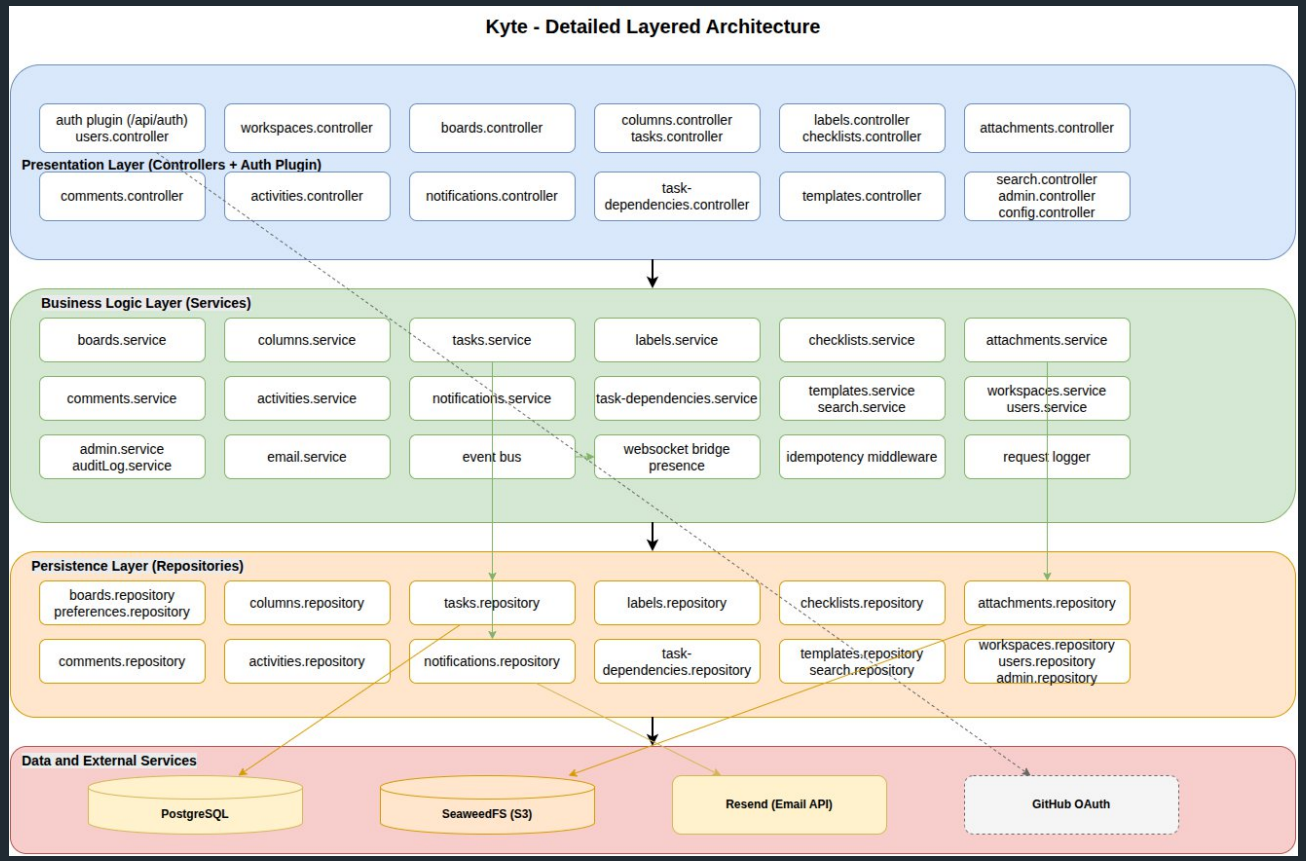
HTTPS (REST)
CRUD Operations

WebSockets
Live Sync

Kiến trúc chi tiết

Detailed Layered Architecture

Kyte - Detailed Layered Architecture



3-Tier Structure

L1 Presentation Layer

Elysia Controllers + Auth Plugin handling HTTP/WS requests

L2 Service Layer

Core domain logic with Event Bus for internal coordination

L3 Persistence Layer

Drizzle ORM with PostgreSQL for type-safe data access

Real-Time Layer

Event Bus → **WebSocket Bridge** → **Broadcast**

Event-driven architecture ensuring real-time synchronization across all connected clients

Cấu trúc Mô-đun

Pattern: Modular Monolith

Why Modular Monolith?

The best of both worlds: **Simplicity of a monolith** with the **isolation of microservices**. Single deployment with shared types, while maintaining strict module boundaries.

Module Structure

```
packages/server/src/modules/
├── task/
├── board/
├── workspace/
├── auth/
└── ...
```

Each module contains: controller, service, repository, and model files with explicit boundaries

Module Isolation

Task Module

Handles task lifecycle, drag-and-drop ordering, attachments, and comments. Strictly isolated from board logic.

Board Module

Manages board structure, columns, and preferences. Communicates with Task module only via Event Bus.

Workspace Module

Multi-tenancy, member management, and invitations. Enforces workspace-level access control.

Future-Proof Architecture

Can be refactored or split into microservices with minimal friction

Tech Selection

Technology Stack

Backend

Bun

High-performance JavaScript runtime with built-in test runner and bundler

ElysiaJS

Type-safe web framework with Eden Treaty for end-to-end type inference

Drizzle ORM

TypeScript-first ORM with SQL-like syntax and perfect type inference

Database

PostgreSQL 18

ACID-compliant relational database with Full-Text Search and JSONB support

Better Auth

Comprehensive authentication framework with OAuth and session management

SeaweedFS

S3-compatible distributed file system for task attachments

Frontend

React 19

Modern concurrency features with useOptimistic for drag-and-drop

TanStack Router

File-based routing with 100% type safety for URL parameters

TanStack Query

Server state management with caching and synchronization

Rationale: End-to-End Type Safety

All technology choices were made to support **End-to-End Type Safety**. Eden Treaty exports server type definitions to the frontend, eliminating manual API documentation and preventing an entire class of production errors. This type-safe bridge ensures that changes in the API are immediately reflected in the client, maintaining perfect synchronization between client and server.

100%
Type Coverage

Zero
API Mismatches

Kiểm thử

Testing & Verification Matrix

Testing Approach

"Real-World Integration Testing"

Unlike traditional architectures that rely on mocking, Kyte uses a **live PostgreSQL 18 instance** for all integration tests to verify complex SQL constraints, transaction atomicity, and database triggers.

Test Results

92	32	15
Integration Tests	Unit Tests	System Tests

139

Total Tests Verified • 100% Pass Rate

Verification Matrix: ASRs to Test Results

Architectural Goal	Test Suite	Evidence	Status
Data Integrity	concurrency.test.ts	100% rejection of stale version updates	PASS
Fault Tolerance	idempotency.test.ts	Zero duplicate entries on 5x retried POST	PASS
Security	admin-guard.test.ts	Unauthorized access blocked with 403 Forbidden	PASS
Scalability	position.test.ts	Lexicographic order maintained for 10,000+ items	PASS
Side-Effects	notifications.test.ts	Event → Subscriber → DB Notification chain verified	PASS
Collaboration	presence.test.ts	Presence state correctly shared among board members	PASS

Conclusion

Individual Contributions & Reflection

Trần Thành Long

Full-Stack Developer

- End-to-end full-stack implementation (Server & Web)
- Database schema design and optimization
- Real-time infrastructure with WebSocket bridge
- API logic and business rule implementation

Nguyễn Xuân Mạnh

Architectural Designer & QA

- System architecture design with C4 Model
- Detailed technical diagrams and documentation
- Quality assurance and verification strategy
- Comprehensive test suite development (139 tests)

Key Reflection

"**Architecture is not just about drawing**, it's about building a solid foundation for code."

Performance
Event-driven updates reduce perceived latency

Modularity
Strong boundaries enable extension without regressions

Indexing
Fractional indexing solves drag-and-drop ordering

Software Architecture Course
N02 • Group 2 • Feb 7, 2026

Thank You
Questions & Discussion