Mattermost Project Structure

The secure collaboration platform for technical teams. A high-performance Go server combined with a rich React/Redux frontend.

Updated 2025-12-30

#mattermost #go #react #redux #messaging #enterprise #monorepo

PNG PDF Copy Prompt

Project Directory

```
mattermost/
  server/ Go Backend Serv…
    channels/ Core messaging …
      app/ Business logic …
      api4/ REST API v4
      store/ Database interf…
      model/ Data structures
    cmd/ Entry points
      mattermost/ Server binary
      mmctl/ CLI tool
    enterprise/ Commercial feat…
    go.mod Go dependencies
  webapp/ React Frontend
    channels/ Main web applic…
      src/
    platform/ Shared componen…
      client/ API Client
      components/ UI Library
    package.json JS dependencies
  e2e-tests/ Cypress & Playw…
  Makefile Build automation
  docker-compose.yml Dev environment
```

Repository Info

Repository - mattermost/mattermost

Stars - 30k+

License - AGPL-3.0 / Commercial

Last Analyzed - December 2025

Tech Stack

Backend - Go 1.22+

Frontend - React + Redux

Database - PostgreSQL / MySQL

Build Tool - Make / Webpack

Realtime - WebSocket

Architecture Notes

Mattermost uses a layered architecture in Go. The `api4` layer handles HTTP requests, the `app` layer contains business logic, and the `store` layer handles database access. This separation makes the codebase highly testable and maintainable. The frontend is a React SPA that heavily relies on Redux for state management, which is crucial for a real-time messaging app.

Key Directories

server/channels/app/ - The 'Service Layer'. This is where the core business rules live (e.g., 'PostMessage', 'CreateChannel'). It is independent of the HTTP API.

server/channels/store/ - The 'Repository Layer'. Abstracts the database interactions, allowing Mattermost to support both PostgreSQL and MySQL.

webapp/channels/src/ - The main React application. It consumes the API and manages the WebSocket connection for real-time updates.

Why This Structure?

Mattermost is the gold standard for Go web applications. Its structure is clean, idiomatic, and scales well. It also demonstrates how to manage an 'Open Core' business model where Enterprise features sit alongside the open-source code.