Remote Work Tracker: Project Structure Overview

Version: 1.0

Date: September 3, 2025

1. Introduction

This document provides a comprehensive breakdown of the directory and file structure for the **Remote Work Tracker** project. The project is organized into a monorepo containing three distinct but interconnected components: a client-side daemon, a backend server, and a frontend dashboard. This structure is designed to promote a clean separation of concerns, streamline development, and simplify deployment.

2. Top-Level Directory Structure

The project root contains the three core components and global configuration files.

3. Part 1: client-daemon/

This directory contains the lightweight, cross-platform Python application that runs on an employee's computer to monitor and report activity.

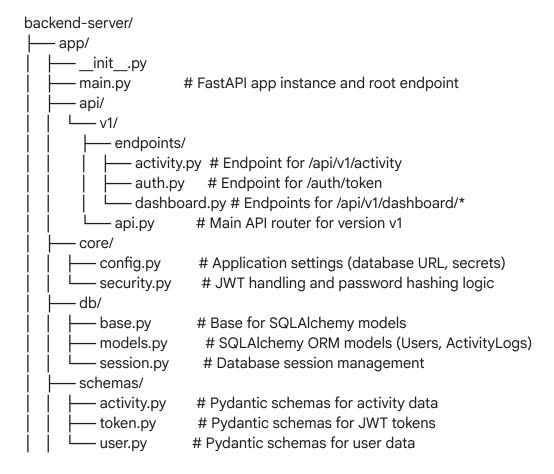
```
client-daemon/
 - src/
   – main .py
                      # Main entry point to start the daemon
                     # Handles loading settings from config.ini
  — config.py
  — tracker.py
                     # Core logic for tracking activity and idle time
   - api client.py
                      # Logic for sending data to the backend
config.ini.template
                       # An example configuration file for users
requirements.txt
                       # Python dependencies (psutil, pynput, requests, etc.)
                    # Python-specific gitignore (e.g., pycache , venv)
- .gitignore
```

File Descriptions:

- src/_main__.py: The primary executable script. It initializes the configuration, starts the activity listeners, and schedules the data transmission tasks.
- src/config.py: Contains functions to locate, read, and validate the config.ini file.
- src/tracker.py: Houses the ActivityTracker class. This file contains the core logic for monitoring the active window, detecting idle states using keyboard/mouse listeners, and buffering the log data.
- src/api_client.py: Responsible for constructing the JSON payload and securely sending the buffered data to the backend API via HTTPS POST requests. It also handles connection errors and retries.
- config.ini.template: A template file that users will copy and rename to config.ini to configure their employee id and the backend url.
- requirements.txt: Lists all Python libraries required for the daemon to run.
- .gitignore: Excludes Python virtual environments, cache files, and build artifacts from version control.

4. Part 2: backend-server/

This directory contains the FastAPI application that serves as the central hub for data ingestion, processing, storage, and API exposure.



└── services/		
categorization.py # Logic for rule-based and AI activity classification		
— Dockerfile	# Instructions to build the backend Docker image	
— docker-compose.	yml # Orchestrates the backend and database containers	
requirements.txt	# Python dependencies for the server	
└─ .gitignore	# Python/FastAPI specific gitignore	

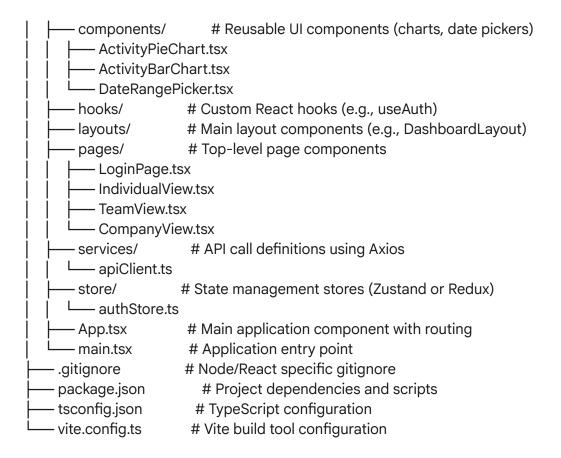
Directory & File Descriptions:

- app/main.py: The entry point for the FastAPI application. It creates the main FastAPI instance and includes the main API router.
- app/api/v1/api.py: Combines all the individual endpoint routers (activity, auth, dashboard) into a single API router for the /api/v1 path.
- app/api/v1/endpoints/: This package holds the API logic for each resource.
 - o activity.py: Handles the data ingestion from the client daemons.
 - o auth.py: Manages user authentication and JWT issuance.
 - dashboard.py: Contains the role-based logic to serve data to the frontend dashboard.
- app/core/: Contains core application logic and configuration.
 - o config.py: Loads environment variables and application settings.
 - security.py: Implements password hashing, JWT creation, and decoding.
- app/db/: Manages database interactions.
 - o models.py: Defines the Users and ActivityLogs tables using the SQLAlchemy ORM.
 - o session.py: Provides logic to create and manage database sessions.
- app/schemas/: Contains Pydantic models for data validation and serialization (request/response shapes).
- app/services/: Holds business logic that is decoupled from the API endpoints.
 - categorization.py: Implements the rule-based and AI-powered engine to classify user activities.
- Dockerfile: Defines the steps to create a container image for the backend application.
- docker-compose.yml: A configuration file for Docker Compose to easily run the backend server and its PostgreSQL database together in development.
- requirements.txt: Lists all Python libraries required for the server.

5. Part 3: frontend-dashboard/

This directory contains the React Single Page Application (SPA) that provides the user interface for data visualization and analysis.

frontend-dashboard/	
— public/	
index.html	# Main HTML file



Directory & File Descriptions:

- public/: Contains static assets that are publicly accessible.
- src/components/: Holds small, reusable React components that are used across multiple pages, such as charts and form elements.
- src/hooks/: Contains custom React hooks for shared logic, like managing authentication state (useAuth).
- src/layouts/: Defines the main structural components of the app, such as DashboardLayout which would include the sidebar and header.
- src/pages/: Contains the top-level components for each view or page in the application, which are mapped to specific routes. These components are responsible for fetching data and composing layouts and smaller components.
- src/services/apiClient.ts: Configures the Axios instance, including setting the base URL and interceptors to automatically attach the JWT to outbound requests.
- src/store/authStore.ts: Defines the Zustand (or Redux) store for managing global application state, such as the user's authentication token and profile information.
- src/App.tsx: The root component of the React application. It sets up the client-side routing.
- src/main.tsx: The entry point for the React application, where the App component is rendered into the DOM.

- package.json: Defines project metadata, dependencies, and scripts (dev, build, preview).
- tsconfig.json: The configuration file for the TypeScript compiler.
- vite.config.ts: The configuration file for the Vite build tool.