Pradeep Kalla

Smart Task Planner – Project Overview and Technical Documentation

Github - Smart_Task_Planner

Page 1: Project Overview & Core Mission

---

1. Introduction

The **Smart Task Planner – AI-Based Productivity Analyzer** is a cross-platform task management application inspired by **Time Doctor 2**. Its core mission is to go beyond simple task reminders by providing a comprehensive productivity ecosystem. It not only manages tasks but also analyzes user behavior and task execution to deliver intelligent insights.

2. Project Objectives

The application aims to achieve the following:

- **Behavioral Analysis:** Monitor and log tool/application usage, as well as idle versus active time.
- **Time Alignment:** Analyze the actual time spent on a task against the planned duration.
- **Geolocation Tracking:** Use location-based data to verify and complete physical tasks.
- **Intelligent Reporting:** Provide detailed reports with productivity scores and focus insights.
-

3. Technical Stack (Frontend & Services)

The current frontend is built with modern web technologies and integrated services:

Category	Technology / Service	Purpose
Frontend	**React.js**	The core JavaScript library for building the user interface.
Styling	**CSS Modules**	Scoped styling to prevent CSS conflicts.
Routing	**React Router DOM**	Handles navigation and single-page application routing.
Charting	**Recharts**	Used to create complex data visualizations (charts, graphs) for reports.
Mapping	**React Leaflet**	Integration of **Leaflet** maps for the Physical Task Tracker feature.

Category	Technology / Service	Purpose
Backend/Cloud	**Firebase**	Provides authentication (Email/Password) and cloud services (Storage, potentially Firestore).

Page 2: Key Modules and Features

---

4. Key Modules

The application is conceptually divided into several modules, with the frontend simulating the planned full desktop/mobile application features.

4.1. Technical Task Analyzer (Web-based Simulation)

This module simulates the core functionality of a desktop application for technical work:

- **Application & Window Tracking:** Simulates logging of active applications and window titles to determine **focused vs. distracted time**.
- **Idle Time Detection:** Tracks periods of user inactivity.
- **Productivity Scoring:** Calculates a real-time productivity score based on the simulated usage data.

4.2. Physical Task Tracker (Web-based Simulation)

This simulates the features of a mobile companion app for tasks requiring physical presence:

- **Geolocation Integration:** Allows users to use their current device location to automatically fill in the location for a physical task.
- **Location-based Completion:** Functionally demonstrates how a task could be automatically or manually marked as complete upon arrival at a designated location.

4.3. Cloud-Based Backend (Firebase)

- **Data Synchronization:** Used to sync user data, task lists, and productivity logs across devices.
- **Authentication:** Manages user sign-up and login securely.

Page 3: Data and Dependencies: package.json

---

5. Project Dependencies (package.json)

The package.json file is the manifest for the project, detailing its name, version, scripts, and all necessary external packages (dependencies).

5.1. Core Dependencies

Dependency	Version (Example)	Brief Explanation
react / react-dom	^19.1.0	The primary libraries for building and rendering the UI.
react-scripts	5.0.1	Scripts for development (start), building (build), and testing (test) the React application.
firebase	^12.0.0	The official SDK for integrating Firebase services (Auth, Storage, etc.).
react-router-dom	^7.7.0	Enables declarative routing for the application.

5.2. Specialized Feature Dependencies

Dependency	Version (Example)	Brief Explanation
recharts	^3.1.0	A composable charting library used for displaying productivity reports and graphs (e.g., bar charts, line charts).
leaflet / react-leaflet	^1.9.4 / ^5.0.0	The core mapping library and its React wrapper, essential for the **Physical Task Tracker's** geolocation features.

5.3. Testing Dependencies

The project includes standard React testing libraries, indicating a commitment to quality and reliable code:

- @testing-library/react
- @testing-library/jest-dom
- @testing-library/user-event

The package-lock.json file supplements this by precisely defining the full dependency tree, ensuring that all contributors use the *exact same* versions of packages for reliable builds.

Page 4: Firebase Configuration and Deployment

---

6. Firebase Integration Files

The project relies heavily on **Firebase** for its backend infrastructure. Two key configuration files manage this setup.

6.1. .firebaserc

This file is a simple, yet critical, configuration file used by the Firebase CLI (Command Line Interface).

- **Purpose:** It links the local project to the correct Firebase project on the cloud.
- **Content:**

JSON

```
{
  "projects": {
    "default": "smart-task-manager-app"
  }
}
```

- **Significance:** The "smart-task-manager-app" ID is the **default project alias** used for all CLI commands (like deployment), ensuring all actions are directed to the correct cloud environment.

6.2. firebase.json

This file declares the hosting and storage configurations for the Firebase project.

- **Purpose:** To define how the application is hosted and how cloud resources are managed.
- **Content:**

JSON

```
{
  "storage": {
    "rules": "storage.rules"
  }
}
```

- **Significance:** It explicitly states that the **Firebase Storage** (used for storing files like user profile pictures or task-related media) will use the security rules defined in the file named **storage.rules**.

Page 5: Security and Version Control

---

7. Firebase Storage Security Rules (storage.rules)

Security rules are the backbone of data protection in Firebase, controlling who can read and write data.

- **Purpose:** To define access control for the files stored in Firebase Storage.
- **Current State:**
- service firebase.storage {

- match /b/{bucket}/o {
- match /{allPaths=**} {
- allow read, write: if false;
- }
- }
- }

- **Security Implication:** The current rule is a **highly restrictive default**, setting **allow read, write: if false;**. This means **no one** (including authenticated users) can read or write any files to storage.
- **Next Steps:** This rule must be updated to allow authenticated users to upload and retrieve their own files, typically by checking for the user's authentication token: allow read, write: if request.auth != null; (or more complex, user-specific checks).

8. Version Control Management (.gitignore)

This file is essential for proper version control using Git.

- **Purpose:** It instructs Git to **ignore** specific files and directories, preventing them from being accidentally committed to the repository.
- **Key Ignored Items:**
    - **/node_modules:** The massive directory containing all project dependencies.
    - **/build:** The folder containing the optimized, final production code.
    - **Environment Files (.env.local, etc.):** Files containing sensitive data like API keys and Firebase configuration details.
- **Significance:** Ignoring these files ensures that the repository remains clean, focused on source code, and does not expose private credentials.

Page 6: Reporting and User Interface

---

9. Data Visualization and Reporting

The application features a dedicated **Reports Module** to analyze and visualize user productivity data using the **Recharts** library. Key visual reports include:

- **Productivity Distribution (Pie Chart):** Shows a breakdown of time spent across different categories (e.g., Productive, Neutral, Distracting).
- **Time Spent Per Task (Bar Chart):** Compares the actual time spent on tasks.
- **Focus Score Throughout the Day (Line Chart):** Tracks the user's focus level over time, an output of the Technical Task Analyzer.

10. User Interface (UI) Screenshots

The project includes several UI mockups/screenshots that illustrate the key views of the application:

- **Dashboard:** The main landing page, likely featuring summarized metrics and quick access to features.
- **Profile Page:** Allows users to view and manage their personal and application settings.
- **Task Page (Task Manager):** The core interface for viewing, managing, and interacting with tasks.
- **New Task Forms (Technical/Physical):** Specialized forms for creating the two distinct types of tasks, highlighting the required input fields like location for physical tasks.
- **Settings Page:** For configuring application behavior and preferences.

The uploaded image file (Screenshot 2025-07-31 143611.png) is an example of the visual components or a specific page within this user interface.

THANK YOU