Software Requirements Specification Ver1.0

Developer: Zhenqi Zhang

Table of Contents

[Introduction 4](#_Toc208560965)

[Purpose 4](#_Toc208560966)

[Scope 4](#_Toc208560967)

[Audience 4](#_Toc208560968)

[Functional Requirements 5](#_Toc208560969)

[Task Management 5](#_Toc208560970)

[Calendar View 5](#_Toc208560971)

[Time Tracking 5](#_Toc208560972)

[Notification 5](#_Toc208560973)

[Dashboard & Analytics 5](#_Toc208560974)

[AI-Powered Features (Optional) 6](#_Toc208560975)

[Non-Functional Requirements 7](#_Toc208560976)

[Usability 7](#_Toc208560977)

[Performance 7](#_Toc208560978)

[Reliability 7](#_Toc208560979)

[Scalability 7](#_Toc208560980)

[Security 7](#_Toc208560981)

[System Architecture 8](#_Toc208560982)

[Frontend (React) 8](#_Toc208560983)

[Backend (Express) 8](#_Toc208560984)

[Database (PostgreSQL) 8](#_Toc208560985)

[Electron 9](#_Toc208560986)

[DevOps & CI/CD (Docker) 9](#_Toc208560987)

[Hosting & Development (To be conducted) 9](#_Toc208560988)

[System Architecture View 10](#_Toc208560989)

[Layer 1: Desktop Application 10](#_Toc208560990)

[Layer 2: API Server 10](#_Toc208560991)

[Layer 3: Database 10](#_Toc208560992)

[UI/UX Design (Figma) 11](#_Toc208560993)

[Development Tools & Stack Summary 12](#_Toc208560994)

[Project Milestones 13](#_Toc208560995)

[Planning and Research 13](#_Toc208560996)

[Design and Prototyping 13](#_Toc208560997)

[Development 13](#_Toc208560998)

[Testing 13](#_Toc208560999)

[Launch 13](#_Toc208561000)

[Final Documentation 13](#_Toc208561001)

# Introduction

## Purpose

This document outlines the requirements for developing a productivity software application focused on task management, time tracking, and data analysis. The software will allow users to organize their to-do lists on daily, weekly, monthly, and yearly levels, with a calendar view, time tracking features, and intelligent suggestions based on user behaviour. The purpose of this software is to provide users with a good planning platform, allowing them to record the plans they think of at any time.

## Scope

This software aims to:

Provide a calendar-based task management system

Allowing drag-and-drop scheduling

Track time for tasks

Offer dashboard with analytics

Send helpful notification

Offer AI-based suggestions for planning.

## Audience

This document is intended for: Developers, Designers, Project Managers and Future Contributors

# Functional Requirements

## Task Management

* Create tasks with titles, descriptions, tags and priorities
* Set task as daily, weekly, monthly or yearly
* Assign due dates and time blocks
* Support recurring tasks
* Allow categorization by type (Work, Personal)

## Calendar View

* View tasks in day/week/month/year formats
* Navigate to any date via calendar
* Drag and drop tasks to schedule them
* Adjust task duration by dragging
* Color-coded task types

## Time Tracking

* Start/Stop timers for tasks
* View historical records of time spent
* Assign time logs to tasks
* Generate time reports per day/week/month

## Notification

* Remind users of upcoming tasks
* Notify users after long continuous work sessions
* Set custom reminders

## Dashboard & Analytics

* Display task completion rate
* Visualize productivity by type/category
* View time spent per task or category
* Display trends over time (Line charts, pie charts)

## AI-Powered Features (Optional)

* Daily summaries generated by AI
* Suggest daily plans based on past habits
* Smart reminders based on typical schedule

# Non-Functional Requirements

## Usability

* Clean and intuitive UI/UX
* Responsive design for desktop and tablet
* Can easily add their plans to the application

## Performance

* Fast rendering calendar and task list
* Time tracking accurate to the second
* Provide support with planning, through AI

## Reliability

* Save data locally and/or in the cloud
* Auto-save task data

## Scalability

* Able to support large number of tasks and users

## Security

* User authentication
* Data encryption for personal tasks and logs.

# System Architecture

## Frontend (React)

React provides a component-based architecture that's perfect for building interactive user interfaces for habit tracking applications. Its virtual DOM ensures smooth performance when displaying lists of habits, charts, and real-time updates. React's extensive ecosystem offers pre-built components for data visualization, date pickers, and form handling, which are essential for tracking applications. The declarative nature of React makes it easier to manage complex state changes as users add, edit, and complete habits. When combined with Electron, React allows developers to leverage web development skills while creating a native desktop experience. The component reusability also means that when mobile versions are eventually built, significant portions of UI logic can be shared with React Native.

## Backend (Express)

Express.js offers a lightweight, flexible framework for building REST APIs that can handle all habit tracking business logic. Its middleware architecture allows easy addition of authentication, logging, and data validation layers as applications grow. Express integrates seamlessly with PostgreSQL through ORMs like Sequelize or Prisma, making database operations straightforward and secure. The framework's simplicity means faster development cycles and easier debugging, which is crucial for desktop-first applications that need to be reliable and responsive. Express also provides excellent support for real-time features through WebSocket integration, enabling live updates and notifications. Its stateless nature makes it easy to scale horizontally when eventually moving to cloud deployment for mobile access.

## Database (PostgreSQL)

PostgreSQL is an enterprise-grade relational database that provides ACID compliance, ensuring habit tracking data remains consistent and reliable. Its advanced data types, including JSON support, allow storage of flexible habit configurations and user preferences without sacrificing query performance. PostgreSQL's excellent indexing capabilities ensure fast searches through large amounts of historical tracking data, which becomes important as users accumulate months or years of habit records. The database's robust backup and recovery features protect against data loss, which is critical for personal productivity applications. PostgreSQL also offers powerful analytical functions that can generate insights and statistics about user habits over time. Its mature ecosystem and extensive documentation make it a safe choice for long-term development, and its ability to run both locally and in the cloud provides flexibility for deployment strategies.

## Electron

Electron is a cross-platform desktop application framework that enables building native desktop apps using web technologies like HTML, CSS, and JavaScript. For EverydayTracking, Electron acts as the bridge between the React frontend and the operating system, providing access to native features like system notifications, file system access, and system tray integration. Its main process handles application lifecycle events while renderer processes display the React UI, with preload scripts ensuring secure communication between components. This architecture is particularly beneficial for habit tracking applications because it enables native desktop features like startup on boot, persistent habit reminders, and offline functionality. Electron also provides automatic updates and can package the entire application into a single executable, eliminating the need for users to install separate dependencies like Node.js or PostgreSQL.

## DevOps & CI/CD (Docker)

Docker containerization simplifies the development and deployment process by ensuring applications run consistently across different environments. For desktop applications, Docker allows packaging of PostgreSQL and Express servers into containers that can be easily distributed with Electron apps or run separately for development. This approach eliminates "works on my machine" problems and makes it easier for team members to set up development environments. Docker Compose can orchestrate multi-container setups, automatically starting databases and API servers when users launch applications. The containerized approach also prepares projects for future cloud deployments, as the same Docker images can be deployed to cloud platforms with minimal configuration changes. Additionally, Docker enables implementation of automated testing pipelines that run in isolated environments, ensuring code quality and reliability.

## Hosting & Development (To be conducted)

The hosting strategy for desktop applications involves both local and cloud considerations. Initially, applications will run entirely on users' machines with local PostgreSQL instances and API servers, requiring no external hosting for basic functionality. However, planning for cloud hosting enables features like data synchronization across devices, collaborative habit tracking, and remote backups. Cloud platforms like AWS, Google Cloud, or Azure can host Express APIs and PostgreSQL databases, allowing users to access their data from multiple devices. This hybrid approach gives users the reliability of local data storage while providing the convenience of cloud synchronization. Development environments should mirror this architecture, using Docker to simulate both local and cloud configurations. Consider implementing environment-specific configurations that allow seamless switching between local-only and cloud-connected modes, giving users choice in how they want to manage their data

# System Architecture View

## Layer 1: Desktop Application

* React Frontend (Electron Renderer Process)
  + Habit Tracking User Interface
* Electron Main Process
  + Native OS Integration

## Layer 2: API Server

* Express API Server (localhost:3001)
  + Business Logic & API Layer
  + Handles HTTP/API calls from desktop app

## Layer 3: Database

* PostgreSQL Database (localhost:5432)
  + Data Persistence
  + Receives database queries from API server

# UI/UX Design (Figma)

# Development Tools & Stack Summary

# Project Milestones

## Planning and Research

* Define the project requirements and create a detailed project plan.

## Design and Prototyping

* Develop the wireframes and prototypes for the UI.
* Start Implementation on Figma and gather feedback on the UI Design.

## Development

* Implement the core features of the software.
* Conduct regular testing and debugging to ensure functionality.

## Testing

* Allow selected group of users to try out the application.
* Collect feedback and make necessary improvements.

## Launch

* Launch the software to the public on specified platform.

## Final Documentation

* Finalise the Documentation.
* Implement ReadMe File.