Software Requirements Specification

for

Pomi

Version 0.1

Prepared by Pawel Grimm

26 August 2020

Table of Contents

Table of Contents ii

Revision History ii

1. Introduction 1

1.1 Purpose 1

1.2 Document Conventions 1

1.3 Intended Audience and Reading Suggestions 1

1.4 Project Scope 1

1.5 References 1

2. Overall Description 2

2.1 Product Perspective 2

2.2 Product Features 2

2.3 User Classes and Characteristics 2

2.4 Operating Environment 2

2.5 Design and Implementation Constraints 2

2.6 User Documentation 2

2.7 Assumptions and Dependencies 3

3. System Features 3

3.1 System Feature 1 3

3.2 System Feature 2 (and so on) 4

4. External Interface Requirements 4

4.1 User Interfaces 4

4.2 Hardware Interfaces 4

4.3 Software Interfaces 4

4.4 Communications Interfaces 4

5. Other Nonfunctional Requirements 5

5.1 Performance Requirements 5

5.2 Safety Requirements 5

5.3 Security Requirements 5

5.4 Software Quality Attributes 5

6. Other Requirements 5

Appendix A: Glossary 5

Appendix B: Analysis Models 6

Appendix C: Issues List 6

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason for Changes** | **Version** |
| Pawel Grimm | 9/26/20 | Began initial release | 0.1 |
|  |  |  |  |

# Introduction

## Purpose

This SRS describes the software functional and nonfunctional requirements for Pomi. Pomi will allow users to associate working sessions with Todoist tasks and generate reports corresponding to that data.

## Document Conventions

No document conventions are being used at this time.

## Intended Audience and Reading Suggestions

This document is intended to be used by members of the project team that will implement and verify the correct functioning of the system.

## Project Scope

Pomi is a web application that will allow users to manage and log their time using the Pomodoro technique (see *Pomodoro Technique*). The main goals are as follows:

1. Provide an intuitive and easy-to-use interface
2. Integrate with Todoist using their published Sync v8 specification *(see Todoist Developer API)*
3. Allow reports to be generated based on time tracking data

## References

1. F. Cocirillo. *Pomodoro Technique*. https://francescocirillo.com/pages/pomodoro-technique
2. Doist. *Todoist Developer API. Sync v8.* https://developer.todoist.com/sync/v8/

# Overall Description

## Product Perspective

Pomi is a new product meant to replace existing timer web applications. The key functionality is the ability to integrate with an existing task-tracking system (Todoist) and leverage task and time tracking data to generate reports. These reports will inform the user on the time spent on given tasks, leading to better time estimates. Additionally, the user may use this data to spend time more effectively in the future.

The application has three integration points, as shown in Figure 1.

1. Client (Web Application) to Server
   1. CRUD operations on time tracking data
   2. Read time tracking + task data
2. Server to Database
   1. CRUD operations time tracking data
3. Server to API
   1. Read task data

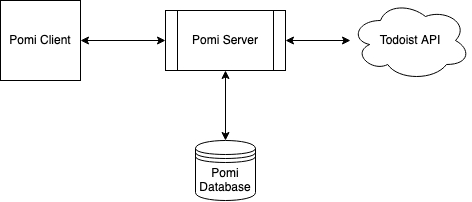


Figure . Application Architecture

## Product Features

The major features this program contains are as follows:

* **Pomodoro Timer**

Start a timer associated with a task

Adjust and save timer presets (work, short-break, long-break)

Trigger desktop notification when timer runs out

* **Dashboard**

View task + timer data in a chart, table, and calendar view

Apply filters such as task search, project, tag, time-frame

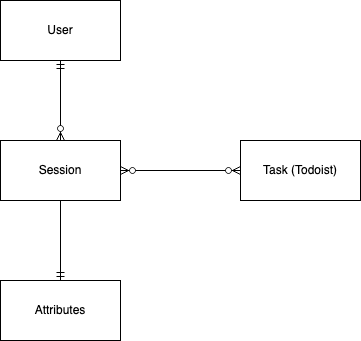


Figure . Class Diagram

## User Classes and Characteristics

This product is only concerned with the end-user, who has the following requirements:

* Configure a work session by setting a timer and optionally associating the sessions with a task
* Receive a notification when the timer has run out
* View completed sessions and associated tasks

## Operating Environment

|  |  |
| --- | --- |
| OE-1 | System is not dependent on geographical areas |
| OE-2 | The system shall operate in last 2 versions of all web browsers, as well as in Firefox ESR |
| OE-3 | There should be no constraint on users being to access the system at any time |
| OE-4 | Continuous service is preferred, but as long as there is no data loss, minor service interruptions can be tolerated. |
| OE-5 | Personal data will be stored in the database, so the database must be secure. |

## Design and Implementation Constraints

|  |  |
| --- | --- |
| CO-1 | UI shall be composed using the React JavaScript library |
| CO-2 | Where possible, all code should be written in TypeScript |
| CO-3 | TBD? shall be used as an authentication broker |

## User Documentation

A user quick-start guide will be delivered.

## Assumptions and Dependencies

|  |  |
| --- | --- |
|  |  |
|  |  |

# System Features

<This template illustrates organizing the functional requirements for the product by system features, the major services provided by the product. You may prefer to organize this section by use case, mode of operation, user class, object class, functional hierarchy, or combinations of these, whatever makes the most logical sense for your product.>

## System Feature 1

<Don’t really say “System Feature 1.” State the feature name in just a few words.>

3.1.1 Description and Priority

<Provide a short description of the feature and indicate whether it is of High, Medium, or Low priority. You could also include specific priority component ratings, such as benefit, penalty, cost, and risk (each rated on a relative scale from a low of 1 to a high of 9).>

3.1.2 Stimulus/Response Sequences

<List the sequences of user actions and system responses that stimulate the behavior defined for this feature. These will correspond to the dialog elements associated with use cases.>

3.1.3 Functional Requirements

<Itemize the detailed functional requirements associated with this feature. These are the software capabilities that must be present in order for the user to carry out the services provided by the feature, or to execute the use case. Include how the product should respond to anticipated error conditions or invalid inputs. Requirements should be concise, complete, unambiguous, verifiable, and necessary. Use “TBD” as a placeholder to indicate when necessary information is not yet available.>

<Each requirement should be uniquely identified with a sequence number or a meaningful tag of some kind.>

REQ-1:

REQ-2:

## System Feature 2 (and so on)

# External Interface Requirements

## User Interfaces

<Describe the logical characteristics of each interface between the software product and the users. This may include sample screen images, any GUI standards or product family style guides that are to be followed, screen layout constraints, standard buttons and functions (e.g., help) that will appear on every screen, keyboard shortcuts, error message display standards, and so on. Define the software components for which a user interface is needed. Details of the user interface design should be documented in a separate user interface specification.>

## Hardware Interfaces

<Describe the logical and physical characteristics of each interface between the software product and the hardware components of the system. This may include the supported device types, the nature of the data and control interactions between the software and the hardware, and communication protocols to be used.>

## Software Interfaces

<Describe the connections between this product and other specific software components (name and version), including databases, operating systems, tools, libraries, and integrated commercial components. Identify the data items or messages coming into the system and going out and describe the purpose of each. Describe the services needed and the nature of communications. Refer to documents that describe detailed application programming interface protocols. Identify data that will be shared across software components. If the data sharing mechanism must be implemented in a specific way (for example, use of a global data area in a multitasking operating system), specify this as an implementation constraint.>

## Communications Interfaces

<Describe the requirements associated with any communications functions required by this product, including e-mail, web browser, network server communications protocols, electronic forms, and so on. Define any pertinent message formatting. Identify any communication standards that will be used, such as FTP or HTTP. Specify any communication security or encryption issues, data transfer rates, and synchronization mechanisms.>

# Other Nonfunctional Requirements

## Performance Requirements

<If there are performance requirements for the product under various circumstances, state them here and explain their rationale, to help the developers understand the intent and make suitable design choices. Specify the timing relationships for real time systems. Make such requirements as specific as possible. You may need to state performance requirements for individual functional requirements or features.>

## Safety Requirements

<Specify those requirements that are concerned with possible loss, damage, or harm that could result from the use of the product. Define any safeguards or actions that must be taken, as well as actions that must be prevented. Refer to any external policies or regulations that state safety issues that affect the product’s design or use. Define any safety certifications that must be satisfied.>

## Security Requirements

<Specify any requirements regarding security or privacy issues surrounding use of the product or protection of the data used or created by the product. Define any user identity authentication requirements. Refer to any external policies or regulations containing security issues that affect the product. Define any security or privacy certifications that must be satisfied.>

## Software Quality Attributes

<Specify any additional quality characteristics for the product that will be important to either the customers or the developers. Some to consider are: adaptability, availability, correctness, flexibility, interoperability, maintainability, portability, reliability, reusability, robustness, testability, and usability. Write these to be specific, quantitative, and verifiable when possible. At the least, clarify the relative preferences for various attributes, such as ease of use over ease of learning.>

# Other Requirements

<Define any other requirements not covered elsewhere in the SRS. This might include database requirements, internationalization requirements, legal requirements, reuse objectives for the project, and so on. Add any new sections that are pertinent to the project.>

Appendix A: Glossary

<Define all the terms necessary to properly interpret the SRS, including acronyms and abbreviations. You may wish to build a separate glossary that spans multiple projects or the entire organization, and just include terms specific to a single project in each SRS.>

Appendix B: Analysis Models

<Optionally, include any pertinent analysis models, such as data flow diagrams, class diagrams, state-transition diagrams, or entity-relationship diagrams.>

Appendix C: Issues List

< This is a dynamic list of the open requirements issues that remain to be resolved, including TBDs, pending decisions, information that is needed, conflicts awaiting resolution, and the like.>