Pomodoro Timer App Documentation

# Technologies Used

## Core Technologies

1. **Python 3.7+**

 Main programming language

Cross-platform compatibility

 Rich standard library

1. **Tkinter**

 Python's standard GUI library

Built-in with Python (no extra installation)

Cross-platform (Windows, macOS, Linux)

 Lightweight and fast

1. **JSON**

 Data persistence format

Human-readable

 Easy to parse and save

**Python Standard Library Modules**

python

import tkinter as tk *# GUI framework* from tkinter import ttk *# Themed widgets* from tkinter import messagebox *# Dialog boxes* import json *# Data serialization*

import os *# File operations*

from datetime import datetime *# Timestamps*

from typing import Dict, List, Callable *# Type hints*

import math *# Mathematical operations*

# Architecture & Design Patterns

## MVC (Model-View-Controller) Architecture

┌─────────────────────────────────────────────────────────┐

│ Application Layer │

├─────────────────────────────────────────────────────────┤

└─────────────────────────────────────────────────────────┘

|  |  |  |  |
| --- | --- | --- | --- |
| │  │ | ┌──────────┐ | │  ┌──────────────┐ | ┌────────┐ │ |
| │ | │ MODEL │◄───────│ CONTROLLER │─────►│ VIEW │ │ | | |
| │ | │ │ │ │ │ │ │ | | |
| │ | │ Business │ │ Coordination │ │ UI │ │ | | |
| │ | │ Logic │ │ Logic │ │ Display│ │ | | |
| │ | └──────────┘ └──────────────┘ └────────┘ │ | | |
| │ | │ │ │ │ | | |
| │ | │ │ │ │ | | |
| │ | Data Layer Event Handling User Input │ | | |
| │ | Statistics Timer Management Rendering │ | | |
| │ | Persistence State Updates Themes │ | | |
| │ | │ | | |

**Why MVC?**

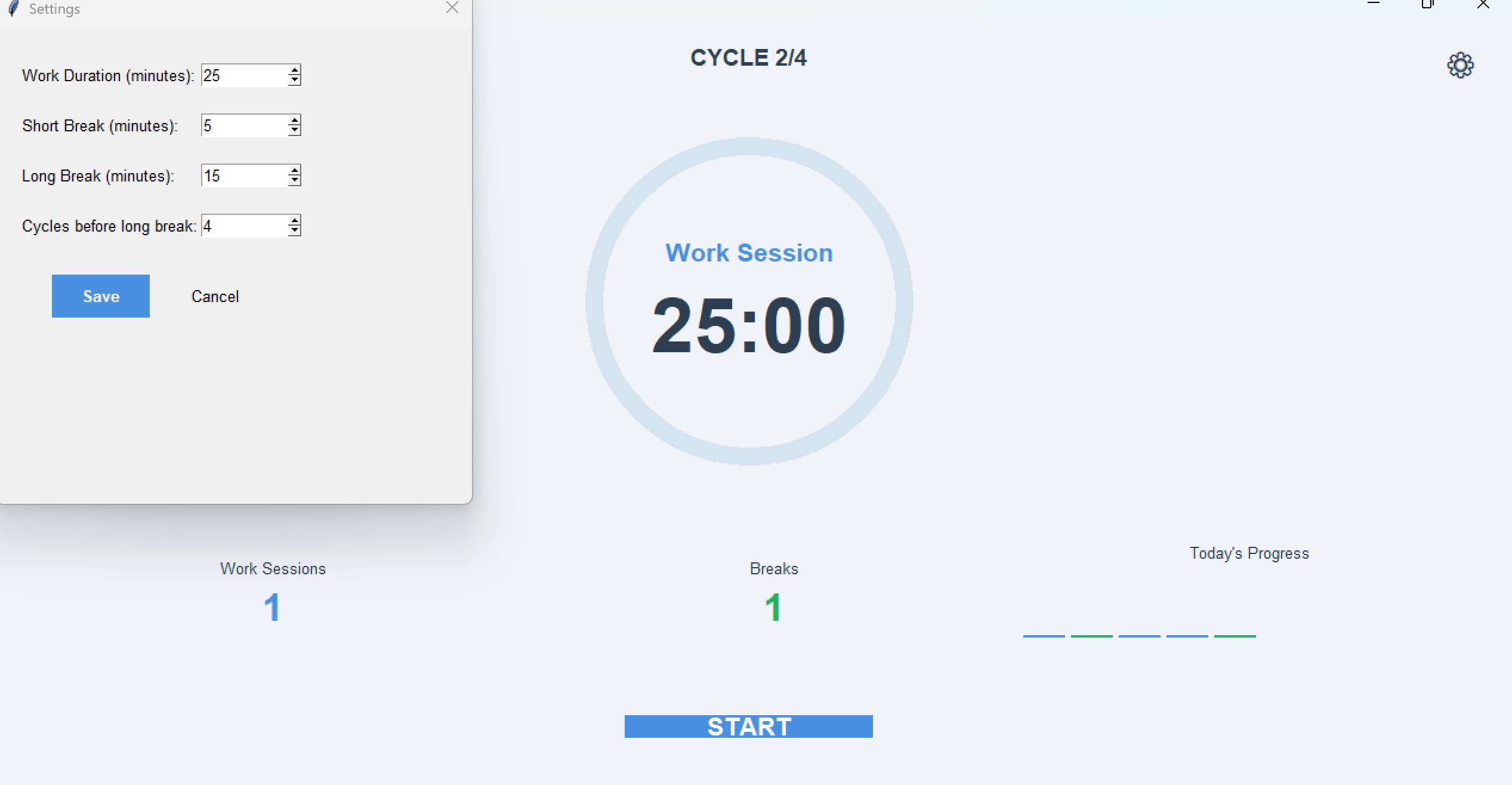
**Separation of Concerns**: Each component has one responsibility

**Maintainability**: Easy to modify one part without affecting others

**Testability**: Components can be tested independently

**Scalability**: Easy to add new features

**User Interface**



## Observer Pattern (Pub-Sub)

python

**Why Observer Pattern?**

*# Model notifies observers when state changes*

class PomodoroModel: def init (self):

self.observers = [] *# List of callbacks*

def add\_observer(self, callback):

self.observers.append(callback)

def notify\_observers(self):

for callback in self.observers: callback() *# Notify all observers*

*# Controller subscribes to model updates*

class PomodoroController: def init (self):

self.model.add\_observer(self.update\_view)

**Loose Coupling**: Model doesn't need to know about View

**Real-time Updates**: View updates automatically when data changes

**Extensibility**: Easy to add multiple observers

# File Structure & Responsibilities

## model.py - The Brain

**Purpose**: Manages all business logic and data

**Key Components**:

python

**Data Structures (Dictionaries)**:

class PomodoroModel:

*# State Management*

* session\_state: Dict *# Current timer state*
* config: Dict *# User settings*
* statistics: Dict *# Session history*

*# Business Logic Methods*

* start\_timer() *# Begin countdown*
* stop\_timer() *# Stop and reset*
* tick() *# Decrement timer (1 second)*
* \_complete\_session() *# Handle session end*
* \_next\_session() *# Transition to next phase*

*# Data Persistence*

* \_save\_statistics() *# Save to JSON*
* \_load\_statistics() *# Load from JSON*

*# Observer Pattern*

* add\_observer() *# Register observer*
* notify\_observers() *# Alert all observers*

python

*# Configuration dictionary*

config = {

'work\_duration': 1500, *# 25 minutes in seconds*

'short\_break\_duration': 300, *# 5 minutes*

'long\_break\_duration': 900, *# 15 minutes* 'cycles\_before\_long\_break': 4, *# Pomodoro cycles* 'auto\_start\_breaks': False, *# Auto-start setting* 'auto\_start\_work': False,

}

*# Session state dictionary*

session\_state = {

'current\_time': 1500, *# Remaining seconds* 'session\_type': 'work', *# work/short\_break/long\_break* 'state': 'idle', *# idle/running/paused* 'current\_cycle': 1, *# Current cycle number*

'total\_cycles': 4, *# Cycles before long break* 'completed\_work\_sessions': 0, *# Today's work count* 'completed\_break\_sessions': 0, *# Today's break count*

}

*# Statistics dictionary*

statistics = {

'total\_work\_time': 0, *# Cumulative work time* 'total\_break\_time': 0, *# Cumulative break time* 'sessions\_completed': 0, *# All-time sessions* 'today\_sessions': [], *# Array of session objects* 'last\_reset': '2025-10-29', *# Date tracking*

}

**Key Logic Flow**:

## view.py - The Face

User Clicks Start

↓

start\_timer() → state = 'running'

↓

notify\_observers() → Controller updates View

↓

Controller starts \_tick() (every 1 second)

↓

tick() → current\_time -= 1

↓

notify\_observers() → View shows updated time

↓

current\_time == 0?

↓

\_complete\_session()

↓

Update statistics, save to JSON

↓

\_next\_session() → Determine next phase

↓

Reset timer for next session

**Purpose**: Handles all UI rendering and user interactions

**Key Components**:

python

**Color Theme System**:

class PomodoroView:

*# UI Components*

* canvas *# Circular progress ring*
* time\_label *# MM:SS display*
* session\_label *# "Work Session" / "Break"*
* cycle\_label *# "CYCLE 3/4"*
* start\_stop\_btn *# Main control button*
* work\_progress *# Work session counter*
* break\_progress *# Break session counter*
* graph\_canvas *# Mini bar chart*

*# Color Themes*

* WORK\_COLORS: Dict *# Blue theme*
* BREAK\_COLORS: Dict *# Green theme*

*# Callback References*

* on\_start\_stop: Callable *# Controller method*
* on\_reset: Callable *# Controller method*
* on\_settings: Callable *# Controller method*

python

WORK\_COLORS = {

'primary': '#4A90E2', *# Professional blue*

'secondary': '#357ABD', *# Darker blue*

'bg': '#F0F4F8', *# Light blue-gray background*

'text': '#2C3E50', *# Dark text*

'progress': '#4A90E2', *# Progress ring color*

'progress\_bg': '#D6E4F0', *# Background ring*

}

BREAK\_COLORS = {

'primary': '#27AE60', *# Refreshing green*

'secondary': '#1E8449', *# Darker green*

'bg': '#E8F8F5', *# Light mint background* 'text': '#145A32', *# Dark green text* 'progress': '#27AE60', *# Progress ring color* 'progress\_bg': '#D5F4E6', *# Background ring*

}

**UI Layout Structure**:

|  |  |  |  |
| --- | --- | --- | --- |
| │ | ⚙ Settings |  | │ ← Top right button |
| │ |  | │ |  |
| │ | CYCLE 3/4 |  | │ ← Cycle indicator |
| │ |  | │ |  |
| │ | ┌───────────────────┐ │ | | |
| │ | │ ╱───────────╲ │ │ | | |
| │ | │ │ Work Session │ │ │ ← Circular timer | | |
| │ | │ │ 19:25 │ │ │ with labels | | |
| │ | │ ╲───────────╱ │ │ | | |
| │ | └───────────────────┘ │ | | |
| │ | │ | | |
| │ | Work Sessions: 3 Breaks: 2 │ ← Session counters | | |
| │ | │ | | |
| │ | Today's Progress │ ← Mini graph label | | |
| │ | ▮▮▮▮▯▮▮▯▮▮ │ ← Bar chart | | |
| │ | │ | | |
| │ | ┌─────────┐ │ | | |
| │ | │ START │ │ ← Main button | | |
| │ | └─────────┘ │ | | |
| │ | Reset │ ← Secondary button | | |

**Circular Progress Logic**:

┌─────────────────────────────────────┐

└─────────────────────────────────────┘

python

def \_update\_progress\_arc(self, state: Dict):

*# Calculate percentage complete*

progress = ((total\_time - current\_time) / total\_time) \* 100

*# Convert to arc extent (360 degrees total)*

extent = -359.99 \* (progress / 100)

*# Update the arc (starts at 90°, draws clockwise)*

self.canvas.itemconfig(self.progress\_arc, extent=extent)

**Theme Switching**:

python

## controller.py - The Coordinator

def \_apply\_theme(self, colors: Dict): *# Update all backgrounds* self.root.configure(bg=colors['bg'])

self.main\_frame.configure(bg=colors['bg'])

*# Update text colors*

self.cycle\_label.configure(fg=colors['text']) self.session\_label.configure(fg=colors['primary'])

*# Update progress ring colors*

self.canvas.itemconfig(self.progress\_arc, outline=colors['progress'])

**Purpose**: Connects Model and View, handles user actions

**Key Components**:

python

class PomodoroController: def init (self):

self.model = PomodoroModel() *# Create model*

self.view = PomodoroView(root) *# Create view*

*# Connect view callbacks to controller* self.view.on\_start\_stop = self.handle\_start\_stop self.view.on\_reset = self.handle\_reset self.view.on\_settings = self.handle\_settings

*# Subscribe to model changes*

self.model.add\_observer(self.update\_view)

*# Event Handlers*

* handle\_start\_stop() *# Start/stop button logic*
* handle\_reset() *# Reset button logic*
* handle\_settings() *# Settings dialog logic*

*# Timer Management*

* \_tick() *# Called every second*
* \_start\_tick() *# Begin timer loop*
* \_stop\_tick() *# Cancel timer loop*

*# Session Handling*

* \_handle\_session\_complete() *# Notification & sound*
* \_play\_notification\_sound() *# System beep*

*# View Updates*

* update\_view() *# Sync view with model*

**Event Flow Diagram**:

**Timer Tick System**:

User Action (Click START)

↓

View calls: self.on\_start\_stop()

↓

Controller: handle\_start\_stop()

↓

Check current state:

* If IDLE/PAUSED → Start timer
* If RUNNING → Stop timer

↓

Model: start\_timer() or stop\_timer()

↓

Model: notify\_observers()

↓

Controller: update\_view() (called automatically)

↓

View: update\_display(state)

↓

UI updates with new state

python

def \_tick(self):

*# 1. Update model (decrement 1 second)*

session\_complete = self.model.tick()

*# 2. Check if session finished*

if session\_complete: self.\_handle\_session\_complete()

*# 3. Schedule next tick if still running*

if self.model.session\_state['state'] == 'running':

*# Call \_tick() again after 1000ms*

self.timer\_job = self.root.after(1000, self.\_tick)

## main.py - The Entry Point

**Purpose**: Initialize and run the application

# Complete Application Flow

python

def main():

*# 1. Create root window*

root = tk.Tk()

*# 2. Initialize controller (creates model & view)*

controller = PomodoroController(root)

*# 3. Handle window close*

root.protocol("WM\_DELETE\_WINDOW", on\_closing)

*# 4. Start main event loop*

controller.run() *# Calls root.mainloop()*

## Startup Sequence

1. main.py runs

↓

1. Creates tk.Tk() root window

↓

1. Instantiates PomodoroController

↓

1. Controller creates PomodoroModel

↓

1. Model loads saved statistics from JSON

↓

1. Controller creates PomodoroView

↓

1. View builds all UI widgets

↓

1. Controller connects callbacks

↓

1. Controller subscribes to model updates

↓

1. Initial view update (displays 25:00)

↓

1. Application ready, waiting for user input

**User Starts Work Session**

User clicks START button

↓ View.\_handle\_start\_stop()

↓

Controller.handle\_start\_stop()

↓ Model.start\_timer()

- session\_state['state'] = 'running'

↓ Model.notify\_observers()

↓ Controller.update\_view()

↓ View.update\_display(state)

* Button text: "START" → "STOP"
* Button color: Blue → Red

↓ Controller.\_start\_tick()

↓

Every 1 second:

Model.tick() → current\_time -= 1 Model.notify\_observers()

View updates time display

↓

When current\_time == 0: Model.\_complete\_session()

* Update statistics
* Save to JSON
* Determine next session type Controller.\_handle\_session\_complete()
* Show notification dialog
* Play sound

View updates theme (Work → Break)

**Settings Configuration**

User clicks ⚙ Settings

↓ Controller.handle\_settings()

- Pause timer if running

↓

# Data Persistence

View.show\_settings\_dialog(current\_config)

* Opens modal dialog
* Shows current values

↓

User modifies settings and clicks Save

↓ SettingsDialog.\_save()

* Creates new config dictionary
* Closes dialog

↓

Controller receives new\_config

↓ Model.update\_config(new\_config)

* Updates config dictionary
* Resets current session time
* Notifies observers

↓

View updates with new times

↓

Resume timer if it was running

## JSON File Structure

**pomodoro\_stats.json**:

json

**Save/Load Logic**

{

"total\_work\_time": 7500,

"total\_break\_time": 1200,

"sessions\_completed": 5,

"last\_reset": "2025-10-29", "today\_sessions": [

{

"type": "work",

"timestamp": "2025-10-29T14:30:00",

"duration": 1500

},

{

"type": "break",

"timestamp": "2025-10-29T14:55:00",

"duration": 300

}

]

}

python

def \_save\_statistics(self): try:

with open('pomodoro\_stats.json', 'w') as f: json.dump(self.statistics, f, indent=2)

except Exception as e:

print(f"Error saving: {e}")

def \_load\_statistics(self):

if os.path.exists('pomodoro\_stats.json'): with open('pomodoro\_stats.json', 'r') as f:

saved\_stats = json.load(f)

*# Check if new day, reset daily stats*

today = datetime.now().strftime('%Y-%m-%d') if saved\_stats['last\_reset'] != today:

saved\_stats['today\_sessions'] = []

saved\_stats['last\_reset'] = today

self.statistics.update(saved\_stats)

# Visual Design Principles

## Color Psychology

 **Blue (Work)**: Professional, calm, focused

 **Green (Break)**: Relaxing, refreshing, restful

## Visual Hierarchy

Most Important (Largest):

↓

Time Display (48px bold)

↓

Circular Progress Ring

↓

Session Label (16px)

↓

START/STOP Button (16px bold)

↓

Session Counters (24px)

↓

Cycle Indicator (14px)

↓

Least Important (Smallest): Mini Graph

1. **Feedback Mechanisms**

 **Visual**: Color changes, progress ring fills

**Textual**: Time updates, session labels change

**Interactive**: Button text/color changes (START ↔ STOP)

**Auditory**: System beep on completion

**Modal**: Notification dialogs

# Key Design Decisions

## Why Dictionaries?

- **Flexible**: Easy to add new properties

- **Clear**: Self-documenting with keys

- **Serializable**: Direct JSON conversion

- **Passable**: Easy to send between components

## Why No Database?

 Lightweight application

Single-user usage

Simple data structure

 JSON is human-readable and editable

## Why Tkinter?

 Built-in with Python (no extra dependencies)

Cross-platform compatibility

Fast and lightweight

 Sufficient for this UI complexity

## Why MVC?

 Clean separation of concerns

Easy to test individual components

Scalable for future features

Industry-standard pattern

# Performance Optimizations

1. **Timer Precision**: Uses for accurate 1-second ticks

root.after()

1. **Lazy Updates**: Only redraws when state changes
2. **File I/O**: Saves statistics only on session complete (not every tick)
3. **Memory**: Limits today\_sessions to reasonable size
4. **Canvas Optimization**: Updates arc extent, doesn't redraw entire canvas

# Future Enhancement Ideas

1. **Sound Library**: Add custom notification sounds (pygame)
2. **Task Lists**: Integrate todo items with Pomodoros
3. **Analytics Dashboard**: Weekly/monthly productivity graphs
4. **Cloud Sync**: Save stats to cloud storage
5. **Desktop Notifications**: Native OS notifications
6. **Keyboard Shortcuts**: Space to start/stop, R to reset
7. **Themes**: Multiple color schemes (dark mode, etc.)
8. **Export**: CSV export of session history

This architecture ensures the app is maintainable, extensible, and follows best practices.