🕐 Employee Time Tracker - User Guide

Version 1.15 | Created by Philippe Addelia | NAKUPUNA CONSULTING

📋 Table of Contents

🚀 Getting Started

Initial Setup

1. **Employee Name**: Enter your name in the header section
2. **Daily Target**: Set your daily hour target (default: 8 hours)
3. **Period Dates**: Set your tracking period (start/end dates)
   - Default: Last 14 days
   - Tip: Use pay period dates for payroll tracking

Quick Start

1. Select a **Category** (Work, Overhead, Travel, etc.)
2. Enter a **Project** name (optional)

3. Click **Start** to begin tracking
4. Click **Stop** when finished
5. View your entries in the list below

⏱️ Timer Functionality

Starting a Timer

- Click the **Start** button in the timer section
- Timer displays in `HH:MM:SS` format
- Button changes to red **Stop** button
- Daily progress updates in real-time

Stopping a Timer

- Click the red **Stop** button
- Entry is automatically saved
- Timer resets to `00:00:00`
- Display refreshes with new entry

Real-Time Features

- **Live daily counter**: Shows current day progress
- **Progress bar**: Visual indicator of daily target
- **Session tracking**: Automatic saving every 60 seconds
- **Warnings**: Alerts at 6 PM and when target reached

📁 Categories & Projects

Available Categories

Category	Purpose	Color
Work	Regular work tasks	Blue
Overhead	Administrative tasks	Purple
Travel	Paid travel time	Green
PTO	Paid time off	Orange
Sick	Sick leave	Red
Holiday	Holiday pay	Green
Bereavement	Bereavement leave	Purple
Jury	Jury duty	Teal

Project Tracking

- **Optional field** for specific project identification
- **Free text entry** - enter any project name
- **Default**: "No Project" if left blank
- **Tip**: Use consistent naming for better reporting

📊 Data Management

Local Storage

- Data stored in your browser's local storage
- **Automatic saving** after each timer session
- **Persistent** across browser sessions
- **Private** - data stays on your device

Data Limits

- **Recommended**: Up to 500 entries for optimal performance
- **Warning threshold**: 100+ entries
- **Critical threshold**: 1000+ entries
- **Storage limit**: ~5-10MB (browser dependent)

📈 Views & Reporting

Detailed View

- **Complete entry list** with all information
- **Sortable** by date (newest first)
- **Editable entries** (click Edit button)
- **Individual actions** (Edit/Delete per entry)

Daily Summary View

- **Grouped by date** for easy overview
- **Category breakdown** per day
- **Total hours** per day
- **Compact format** for quick review

Period Statistics

Metric	Description
Period Hours	Total hours in selected date range
Total Period Days	Calendar days in range
Days Worked	Days with time entries
Days Elapsed	Days passed in current period

🔄 Session Recovery

Automatic Recovery

- **Detects incomplete sessions** on app reload
- **Shows recovery modal** with session details
- **Three options** for handling incomplete sessions

Recovery Options

1. 💾 **Save As-Is**
   - Uses estimated end time (current time)
   - Quick option for normal cases
2. ⏰ **Adjust Time**
   - Manually set the actual end time
   - Recalculates duration automatically
   - Best for accurate tracking

3. 🗑️ **Discard**
    - Deletes the incomplete session
    - Use when session was invalid

Session Information

- **Category** and **Project** from session
- **Start time** and **estimated duration**
- **Calculated end time** (adjustable)

📤 Data Export & Import

Export Options

Desktop Export

- 📄 **Download CSV**: Standard download
- 💾 **Save As**: Choose save location (Chrome/Edge)

Mobile Export

- 📧 **Email CSV Data**: Copy & paste approach
    - Copyable summary data
    - Full CSV data option
    - Gmail shortcut button
- 📱 **Try Mobile Download**: Web Share API or new tab

Export Data Format

```csv
Employee Name,Date,Category,Project,Start Time,End Time,Duration (Hours)
"John Doe","2025-06-29","WORK","Project Alpha","09:00","17:00",8.0
```

Import Features

Import Modes

- 📎 **Append**: Add to existing data (safe)
- 🔄 **Replace**: Delete all and replace (destructive)

Supported Formats

- **CSV files** from this app
- **Compatible CSV** with required columns:
  - Employee Name, Date, Category, Project, Start Time, End Time, Duration

---

🗑️ Data Cleanup

Clear Data Options

1. 📤 Export Then Clear All *(Recommended)*

- **Safest option** for regular maintenance
- Exports all data first
- Then clears everything
- **Perfect for**: Monthly/quarterly archiving

2. 📅 Keep Recent (30 days)

- **Maintains recent history**
- Deletes entries older than 30 days
- **Good for**: Regular cleanup without losing recent data

3. 📋 Clear Current Period Only

- **Selective cleaning**
- Clears only filtered date range
- Preserves other historical data
- **Useful for**: Specific period cleanup

4. 🚨 Clear All Data

- **Nuclear option** with multiple confirmations
- Requires typing "YES" to confirm
- **Use for**: Complete fresh starts

Performance Indicators

The Clear Data modal shows:

- **Total entries** and tracked hours

- **Data size** in KB
- **Date range** of your data
- **Performance warnings** for large datasets

📱 Mobile vs Desktop

Mobile Optimizations

- **Touch-friendly** buttons and interfaces
- **Larger tap targets** for easy interaction
- **Simplified export** with email options
- **Responsive design** for all screen sizes
- **Stack layout** for narrow screens

Desktop Features

- **Full export modal** with all options
- **File System Access API** (Save As)
- **Grid layouts** for efficient space use
- **Hover effects** and desktop interactions

Universal Features

- **Timer functionality** works identically
- **Data storage** consistent across platforms
- **Session recovery** available everywhere
- **Import/export** core functionality maintained

⚡ Performance & Limits

Optimal Performance

- **0-100 entries**: Smooth operation
- **100-500 entries**: Good performance, consider periodic cleanup
- **500+ entries**: Warnings shown, cleanup recommended
- **1000+ entries**: Performance degradation likely

Storage Considerations

- **Browser limit**: 5-10MB typical
- **Mobile impact**: More noticeable with large datasets
- **Recommendation**: Export and clear data monthly

Performance Tips

1. **Regular exports**: Monthly or per pay period
2. **Data cleanup**: Use "Keep Recent 30 days"
3. **Period filtering**: Focus on current timeframes
4. **Mobile awareness**: Smaller datasets work better

🔧 Troubleshooting

Common Issues

Timer Won't Start

- **Check**: No JavaScript errors in browser console
- **Solution**: Refresh the page
- **Prevention**: Keep browser updated

Data Not Saving

- **Check**: Browser storage permissions
- **Solution**: Clear browser cache and reload
- **Alternative**: Export data as backup

Export Not Working

- **Desktop**: Try different export option
- **Mobile**: Use email export method
- **Universal**: Check popup blockers

Import Failing

- **Check**: CSV format matches requirements
- **Solution**: Ensure proper column headers
- **Tip**: Export sample data first to see format

Session Recovery Not Appearing

- **Cause**: Clean browser close
- **Expected**: Only shows for interrupted sessions
- **Note**: Normal behavior if timer was stopped properly

Browser Compatibility

- **Recommended**: Chrome, Firefox, Safari, Edge
- **Mobile**: All modern mobile browsers
- **Features**: Some features (Save As) require newer browsers
- **Fallbacks**: Alternative methods provided for older browsers

---

💡 Best Practices

Daily Usage

1. **Start timer** when beginning work
2. **Switch categories** as tasks change
3. **Use descriptive project names**
4. **Stop timer** for breaks/lunch
5. **Review daily progress** before leaving

Weekly/Monthly Routine

1. **Review period statistics**
2. **Export data** for records
3. **Clear old data** if needed
4. **Adjust period dates** for new timeframes

Data Management

1. **Export before clearing** (always!)
2. **Use consistent project names**
3. **Set realistic daily targets**
4. **Monitor performance warnings**

Payroll Integration

1. **Set period to pay period dates**

2. **Export CSV** at period end
3. **Share with payroll/management**
4. **Clear period data** after processing
5. **Start fresh** for next period

Mobile Best Practices

1. **Use email export** for reliability
2. **Copy data to notes app** as backup
3. **Keep datasets smaller** for performance
4. **Use "Keep Recent"** cleanup option

📞 Support & Tips

Getting Help

- **Check this guide** for common questions
- **Use browser developer tools** for technical issues
- **Export data regularly** as precaution
- **Keep backups** of important time data

Pro Tips

- **Keyboard shortcuts**: Enter key submits forms
- **Project templates**: Use consistent project naming
- **Break tracking**: Stop timer for lunch breaks
- **Multiple projects**: Switch projects without stopping timer
- **Daily reviews**: Check progress throughout the day

Data Security

- **Local storage only** - data stays on your device
- **No cloud sync** - manually export for backups
- **Browser dependent** - clearing browser data removes app data
- **Export regularly** - only way to preserve data long-term

📄 Appendix

File Formats

CSV Export Structure

```
Employee Name,Date,Category,Project,Start Time,End Time,Duration (Hours)
[Data rows...]

SUMMARY
Period,"2025-06-01 to 2025-06-30"
Total Hours,160.5
Work Days,22
Average Daily Hours,7.30
Export Date,"2025-06-29"
```

Version History

- **v1.15**: Current version with email export and clear data features
- **Mobile optimizations**: Email export, improved touch interface
- **Data management**: Comprehensive clear options with safety features
- **Session recovery**: Automatic detection and recovery options

---

🔧 Appendix: Manual Data Management

For advanced users, IT administrators, and power users who need direct access to data

Data Storage Location

The Employee Time Tracker stores all data in **browser localStorage** under the key `timeTrackerData`. Here's where to find it:

Browser Developer Tools Access

Chrome/Edge:

1. Open Developer Tools (`F12` or `Ctrl+Shift+I`)
2. Go to **Application** tab
3. Expand **Local Storage** in left sidebar
4. Select your domain (e.g., `claude.ai` or `localhost`)
5. Find key: `timeTrackerData`

Firefox:

1. Open Developer Tools (`F12`)
2. Go to **Storage** tab
3. Expand **Local Storage**
4. Select your domain
5. Find key: `timeTrackerData`

Safari:

1. Enable Developer menu (Safari > Preferences > Advanced)
2. Develop > Show Web Inspector
3. Go to **Storage** tab
4. Select **Local Storage**

Data Structure

JSON Format

```json
{
  "employeeName": "John Doe",
  "dailyTarget": "8",
  "timeEntries": [
    {
      "id": 1640995200000,
      "date": "2025-06-29",
      "category": "work",
      "project": "Project Alpha",
      "startTime": "09:00",
      "endTime": "17:00",
      "duration": 8.0
    }
  ]
}
```

Session Recovery Data

Stored under key: `timeTrackerState`

```json
```

```
{
  "isTracking": true,
  "startTime": "2025-06-29T09:00:00.000Z",
  "category": "work",
  "project": "Project Alpha",
  "dailyTarget": "8",
  "lastSaved": "2025-06-29T09:15:00.000Z"
}
```

Browser Console Commands

Export Data Manually

```javascript
// Get all time tracker data
const data = localStorage.getItem('timeTrackerData');
console.log(JSON.parse(data));

// Export as downloadable file
const jsonData = localStorage.getItem('timeTrackerData');
const blob = new Blob([jsonData], {type: 'application/json'});
const url = URL.createObjectURL(blob);
const a = document.createElement('a');
a.href = url;
a.download = 'timetracker-backup.json';
a.click();
```

Import Data Manually

```javascript
// Import from JSON (DANGER: overwrites existing data)
const importData = {
  "employeeName": "Jane Doe",
  "dailyTarget": "8",
  "timeEntries": [...]
};
localStorage.setItem('timeTrackerData', JSON.stringify(importData));
location.reload(); // Refresh to see changes
```

Backup Current Data

```javascript
// Create timestamped backup
const data = localStorage.getItem('timeTrackerData');
const timestamp = new Date().toISOString().split('T')[0];
localStorage.setItem(`timeTrackerData_backup_${timestamp}`, data);
console.log(`Backup created: timeTrackerData_backup_${timestamp}`);
```

Query Data

```javascript
// Parse and analyze data
const data = JSON.parse(localStorage.getItem('timeTrackerData'));
const entries = data.timeEntries;

// Total hours
const totalHours = entries.reduce((sum, entry) => sum + entry.duration, 0);
console.log(`Total hours tracked: ${totalHours}`);

// Entries by category
const byCategory = entries.reduce((acc, entry) => {
  acc[entry.category] = (acc[entry.category] || 0) + entry.duration;
  return acc;
}, {});
console.log('Hours by category:', byCategory);

// Date range
const dates = entries.map(e => e.date).sort();
console.log(`Date range: ${dates[0]} to ${dates[dates.length-1]}`);
```

File System Locations

Chrome/Chromium

Windows:

```
%LOCALAPPDATA%\Google\Chrome\User Data\Default\Local Storage\leveldb\
```

macOS:

```
~/Library/Application Support/Google/Chrome/Default/Local Storage/leveldb/
```

Linux:

```
~/.config/google-chrome/Default/Local Storage/leveldb/
```

Firefox

Windows:

```
%APPDATA%\Mozilla\Firefox\Profiles\[profile]\storage\default\
```

macOS:

```
~/Library/Application Support/Firefox/Profiles/[profile]/storage/default/
```

Linux:

```
~/.mozilla/firefox/[profile]/storage/default/
```

Edge

Windows:

```
%LOCALAPPDATA%\Microsoft\Edge\User Data\Default\Local Storage\leveldb\
```

Command Line Tools

Using Node.js for Data Processing

Convert JSON to CSV:

```
javascript
```

```
// save as convert.js
const fs = require('fs');

const data = JSON.parse(fs.readFileSync('timetracker-backup.json', 'utf8'));
const entries = data.timeEntries;

// CSV header
let csv = 'Employee Name,Date,Category,Project,Start Time,End Time,Duration\n';

// Add data rows
entries.forEach(entry => {
  csv += `"${data.employeeName}","${entry.date}","${entry.category}","${entry.project}","${entry.startTime}","${entry.en
});

fs.writeFileSync('timetracker-export.csv', csv);
console.log('CSV exported to timetracker-export.csv');
```

Run with:

```bash
node convert.js
```

Using jq for JSON Processing

Extract specific date range:

```bash
# Linux/macOS with jq installed
cat timetracker-backup.json | jq '.timeEntries[] | select(.date >= "2025-06-01" and .date <= "2025-06-30")'
```

Calculate total hours:

```bash
cat timetracker-backup.json | jq '[.timeEntries[].duration] | add'
```

Group by category:

```bash
```

```
cat timetracker-backup.json | jq '.timeEntries | group_by(.category) | map({category: .[0].category, hours: [.[].duration] | a
```

Backup Strategies

Automated Browser Backup Script

PowerShell (Windows):

```powershell
# backup-timetracker.ps1
$timestamp = Get-Date -Format "yyyy-MM-dd_HH-mm-ss"
$backupDir = "$env:USERPROFILE\Documents\TimeTracker-Backups"

if (!(Test-Path $backupDir)) {
    New-Item -ItemType Directory -Path $backupDir
}

# Chrome localStorage path
$chromePath = "$env:LOCALAPPDATA\Google\Chrome\User Data\Default\Local Storage\leveldb"
$backupPath = "$backupDir\chrome-backup-$timestamp"

if (Test-Path $chromePath) {
    Copy-Item -Path $chromePath -Destination $backupPath -Recurse
    Write-Host "Backup created: $backupPath"
} else {
    Write-Host "Chrome localStorage not found"
}
```

Bash (Linux/macOS):

```bash
```

```
#!/bin/bash
# backup-timetracker.sh
TIMESTAMP=$(date +"%Y-%m-%d_%H-%M-%S")
BACKUP_DIR="$HOME/TimeTracker-Backups"

mkdir -p "$BACKUP_DIR"

# Chrome backup
CHROME_PATH="$HOME/.config/google-chrome/Default/Local Storage/leveldb"
if [ -d "$CHROME_PATH" ]; then
  cp -r "$CHROME_PATH" "$BACKUP_DIR/chrome-backup-$TIMESTAMP"
  echo "Chrome backup created: $BACKUP_DIR/chrome-backup-$TIMESTAMP"
fi

# Firefox backup
FIREFOX_PROFILE=$(find ~/.mozilla/firefox -name "*.default*" -type d | head -1)
if [ -d "$FIREFOX_PROFILE/storage" ]; then
  cp -r "$FIREFOX_PROFILE/storage" "$BACKUP_DIR/firefox-backup-$TIMESTAMP"
  echo "Firefox backup created: $BACKUP_DIR/firefox-backup-$TIMESTAMP"
fi
```

Data Migration

Between Browsers

```javascript
// Export from Browser A
const exportData = localStorage.getItem('timeTrackerData');
console.log('Copy this data:');
console.log(exportData);

// Import to Browser B
const importData = `[paste exported data here]`;
localStorage.setItem('timeTrackerData', importData);
location.reload();
```

Between Devices

1. **Export method**: Use browser console to export JSON
2. **Transfer**: Email, cloud storage, or USB
3. **Import method**: Use browser console to import JSON

Advanced Data Analysis

SQL-like Queries with JavaScript

```javascript
// Advanced data analysis functions
const data = JSON.parse(localStorage.getItem('timeTrackerData'));
const entries = data.timeEntries;

// Weekly summary
function getWeeklySummary() {
  const weekly = {};
  entries.forEach(entry => {
    const date = new Date(entry.date);
    const week = `${date.getFullYear()}-W${Math.ceil(date.getDate()/7)}`;
    if (!weekly[week]) weekly[week] = 0;
    weekly[week] += entry.duration;
  });
  return weekly;
}

// Productivity analysis
function getProductivityMetrics() {
  const workEntries = entries.filter(e => e.category === 'work');
  const avgHours = workEntries.reduce((sum, e) => sum + e.duration, 0) / workEntries.length;
  const maxDay = Math.max(...workEntries.map(e => e.duration));
  const minDay = Math.min(...workEntries.map(e => e.duration));

  return { avgHours, maxDay, minDay, totalDays: workEntries.length };
}

// Run analysis
console.log('Weekly Summary:', getWeeklySummary());
console.log('Productivity Metrics:', getProductivityMetrics());
```

Security Considerations

Data Protection

- **Local only**: Data never leaves your device
- **Browser dependent**: Clearing browser data removes app data
- **No encryption**: Data stored in plain text JSON

- **Access control**: Anyone with device access can view data

Best Practices

1. **Regular exports**: Don't rely solely on localStorage
2. **Secure backups**: Encrypt exported files if sensitive
3. **Access control**: Use device lock screens
4. **Clean up**: Remove data from shared/public computers

Recovery Procedures

Data Corruption Recovery

```javascript
```

```
// Check data integrity
function validateData() {
  try {
    const data = JSON.parse(localStorage.getItem('timeTrackerData'));
    if (!data.timeEntries || !Array.isArray(data.timeEntries)) {
      throw new Error('Invalid data structure');
    }
    console.log('Data validation passed');
    return true;
  } catch (error) {
    console.error('Data corruption detected:', error);
    return false;
  }
}

// Repair common issues
function repairData() {
  const data = JSON.parse(localStorage.getItem('timeTrackerData'));

  // Fix missing fields
  data.timeEntries = data.timeEntries.map(entry => ({
    id: entry.id || Date.now(),
    date: entry.date || new Date().toISOString().split('T')[0],
    category: entry.category || 'work',
    project: entry.project || 'Unknown',
    startTime: entry.startTime || '00:00',
    endTime: entry.endTime || '00:00',
    duration: entry.duration || 0
  }));

  localStorage.setItem('timeTrackerData', JSON.stringify(data));
  console.log('Data repair completed');
}
```

This manual data management appendix provides advanced users with direct access to the underlying data storage and manipulation capabilities of the Employee Time Tracker.