pace dev documentation

the pace authors

Contents

Chapter 1

Introduction

`pace` is a mindful productivity tool designed to help you keep track of your activities with ease and intention.

Born from the desire to blend simplicity with effectiveness, pace offers a command-line interface (CLI) that encourages focused work sessions, thoughtful reflection on task durations, and a harmonious balance between work and rest.

Whether you're a developer, a writer, or anyone who values structured time management, pace provides the framework to log activities, reflect on progress, and optimize how you spend your time. pace is designed to be a flexible and customizable tool that adapts to your unique workflow.

Target Audience

This documentation is aimed at developers, contributors and other technical audiences who are interested in learning more about the pace application. It provides an overview of the application, its features, and guidance on how to use and contribute to the project. If you are a user of the pace application, you can find user documentation at pace.cli.rs/docs.

Purpose of Documentation

The purpose of this documentation is to provide a comprehensive guide to the pace application. It is intended to help users understand the features and functionality of the application, as well as provide guidance on how to use and contribute to the project.

Contact

You can ask questions in the Discussions or have a look at the FAQ.

Contact	Where?
Issue Tracker	GitHub Issues
Discord	pace 4 members
Discussions	GitHub Discussions

Chapter 2

Contributing

Thank you for your interest in contributing to the `pace` ecosystem!

We appreciate your help in making this project better.

Table of Contents

Code of Conduct

Please review and abide by the general Rust Community Code of Conduct when contributing to this project. In the future, we might create our own Code of Conduct and supplement it at this location.

How to Contribute

Reporting Bugs

If you find a bug, please open an issue on GitHub and provide as much detail as possible. Include steps to reproduce the bug and the expected behavior.

Issue and Pull Request labels

Our Issues and Pull Request labels follow the official Rust style:

```
A - Area
C - Category
D - Diagnostic
E - Call for participation
F - Feature
I - Issue e.g. I-crash
M - Meta
O - Operating systems
P - priorities e.g. P-{low, medium, high, critical}
PG - Project Group
perf - Performance
S - Status e.g. S-{blocked, experimental, inactive}
T - Team relevancy
WG - Working group
```

Suggesting Enhancements

If you have an idea for an enhancement or a new feature, we'd love to hear it! Open an issue on GitHub and describe your suggestion in detail.

Code Style and Formatting

We follow the Rust community's best practices for code style and formatting. Before submitting code changes, please ensure your code adheres to these guidelines:

- Use `rustfmt` to format your code. You can run it with the following command:

  ```
  cargo fmt --all
  ```

- Use `dprint` to format text in markdown, toml, and json. You can install it with `cargo install dprint`/`scoop install dprint` and run it with the following command in the repository root:

  ```
  dprint fmt
  ```

- Use `Clippy` — a collection of lints to catch common mistakes. If you haven't installed your Rust toolchain with `rustup`, please install Clippy first. Then run Clippy with the following command in the crate root:

  ```
  cargo clippy --all -- -D warnings
  ```

- Write clear and concise code with meaningful, self-describing variable and function names. This tells the reader **what** the code does.

- Write clear and consise comments to tell the reader **why** you chose to implement it that way and **which** problem it solves.

Testing

We value code quality and maintainability. If you are adding new features or making changes, please include relevant unit tests. Run the test suite with:

```
cargo test --workspace
```

or check the testing guide for more information which tools we provide for making developing pace easier.

Make sure all tests pass before submitting your changes. PRs containing tests have a much higher probability of getting merged (fast).

We expect PRs especially ones that introduce new features to contain tests for the new code.

Besides that, we welcome PRs which increase the general test coverage of the project. You can check the testing guide for more information.

We appreciate tests in every form: be it *unit*, *doc* or *integration* tests (chose depending on your use case).

If you want to implement some *fuzzing* or *benchmarking*, that is also highly appreciated.

Submitting Pull Requests

To contribute code changes, follow these steps:

1. **Fork** the repository.
2. **Create** a new branch with a descriptive name:
   ```
   git checkout -b feature/your-feature-name
   ```
3. **Check** and **Commit** your changes:
   ```
   just pr
   git commit -m "Add your meaningful commit message here"
   ```
4. **Push** your branch to your forked repository:
   ```
   git push origin feature/your-feature-name
   ```
5. **Open** a Pull Request (PR) to our repository. Please include a detailed description of the changes and reference any related issues.

`Release early and often!` also applies to pull requests

Consider drafting a Pull request early in the development process, so we can follow your progress and can give early feedback.

Once your PR is submitted, it will be reviewed by the maintainers. We may suggest changes or ask for clarifications before merging.

IMPORTANT NOTE Please don't force push commits in your branch, in order to keep commit history and make it easier for us to see changes between reviews.

Make sure to Allow edits of maintainers (under the text box) in the PR so people can actually collaborate on things or fix smaller issues themselves.

Rebasing and other workflows

(taken from: openage on rebasing)

Rebasing is 'moving' your commits to a different parent commit.

In other words: *Cut off* your branch from its tree, and *attach it* somewhere else.

There's two main applications:

- If you based your work on a older main (so old that stuff can't be automatically merged), you can rebase to move your commits to the current upstream main:

```
# update the upstream remote to receive new commits
git fetch upstream

# be on your feature branch (you probably are)
git checkout my-awesome-feature

# make backup (you never know, you know?)
git branch my-awesome-feature-backup

# rebase: put your commits on top of upstream's main
git rebase -m upstream/main
```

- If you want to fix an older commit of yours, or merge several commits into a single one (**squash** them), rebase interactively. We ***don't*** want to have a commit history like this:
  - `add stuff`
  - `fix typo in stuff`
  - `fix compilation`
  - `change stuff a bit`
  - and so on...

rebase in practice `git log --graph --oneline` shows your commit history as graph. To make some changes in that graph, you do an **interactive rebase**:

```
git rebase -i -m upstream/main
```

With this command, your new ”base” is `upstream/main` and you can then change any of your branch's commits.

`-i` will open an interactive editor where you can choose actions for each individual commit:

- re-order commits
- drop commits by deleting their line
- squash/fixup (”meld”) your commits
- reword a commit message
- stop rebasing at a commit to edit (`--amend`) it manually

Just follow the messages on screen.

Changing commits with amend and `fixup` There's also `git commit --amend` which is a ”mini-rebase” that modifies just the last commit with your current changes by `git add`. It just skips the creation of a new commit and instead melds the changes into the last one you made.

If you want to update a single commit in the range `[upstream/main, current HEAD]` which is not the last commit:

- `edit stuff you wanna change in some previous commit`
- `git add changed_stuff`
- `git commit --fixup $hash_of_commit_to_be_fixed`
- `git rebase --autosquash -i -m upstream/main`

Pushing changes After you have rebased stuff (”rewritten history”) that had already been pushed, git will not accept your pushes because they're not simple fast-forwards:

- The commit contents and the parent commit have changed as you updated the commit, therefore the commit hash changed, too.

    - If somebody used those commits, they will keep a copy and have a hard time updating to your updated version (because they ”use” the old hashes).
    - Update your pull request branch with your re-written history!
- **force push** is the standard way of overwriting your development work with the fixed and mergeable version of your contribution!
    - Why? You changed the commits, so you want the old ones to be deleted! You can use any of:
    - `git push origin +my-awesome-feature`
    - `git push origin -f my-awesome-feature`
    - `git push origin --force my-awesome-feature`

Some extra tutorials on `git rebase`:

- Atlassian's Git Tutorial
- Pro Git book
- `man git-rebase`

Development Setup

If you want to set up a local development environment, follow the steps in the development guide file - which is currently being worked on.

License

By contributing to `pace` or any crates contained in this repository, you agree that your contributions will be licensed under:

- **AGPL-3.0-or-later**

Unless you explicitly state otherwise, any contribution intentionally submitted for inclusion in the work by you, as defined in the AGPL-3.0-or-later license, shall be licensed as above, without any additional terms or conditions.

Chapter 3

Getting Started

If you're new to pace, you should start with the user guide's getting started section, which provides an overview of the application and its features. This section also includes guidance on how to install and configure pace, as well as how to use the application to track your activities.

You can find the getting started section in the user guide.

Setting up your development environment

If you're interested in contributing to pace, you should start by setting up your development environment. This section provides an overview of the pace codebase, as well as guidance on how to build and test the application.

Chapter 4

Design Guide

This guide is intended to provide a high-level overview of the design of `pace`. It is intended to be a living document that evolves as the project evolves. It is not intended to be a comprehensive guide to the codebase, but rather a high-level overview of the design of the project.

Chapter 5

Requirements Analysis

Welcome to the foundational stage of the Pace project - our comprehensive requirements analysis document. This document is a cornerstone in the development of Pace, a tool designed with mindfulness at its core to enhance productivity and time management in a tech-savvy yet user-friendly manner. Our aim is to outline the necessary groundwork that will guide the subsequent phases of development, ensuring that Pace not only meets but exceeds the expectations of its users.

Purpose

The primary purpose of this requirements analysis is to identify and detail the specific needs and expectations that Pace intends to fulfill. By meticulously analyzing these requirements, we ensure that every feature and function of Pace is aligned with our goal of providing a seamless, intuitive, and effective productivity tool. This analysis serves as a roadmap, directing the development process towards creating a solution that is both impactful and meaningful to its users.

Scope

Within this document, we delve into the various aspects of the Pace project, covering the technical, functional, and user experience requirements. Our analysis encompasses:

- **User Interface and Experience**: Crafting an interface that is not only visually appealing but also intuitive and easy to navigate for users of varying technical expertise.
- **Functionality**: Outlining the key features and capabilities that Pace will offer, from activity tracking to insightful analytics, ensuring that each function serves to enhance user productivity.
- **Performance and Reliability**: Setting benchmarks for the performance of the Pace tool, ensuring that it remains responsive, stable, and reliable under various usage scenarios.
- **Security and Privacy**: Addressing the measures that will be implemented to protect user data, ensuring confidentiality, integrity, and availability.
- **Integration and Compatibility**: Considering the potential for integration with other tools and platforms, enhancing Pace's versatility and user utility.

Methodology

Our approach to requirements analysis involves a blend of user research, competitive analysis, and stakeholder feedback. By engaging with potential users, we gain insights into their needs, preferences, and pain points. A thorough examination of existing solutions allows us to identify gaps and opportunities for innovation. Stakeholder input further refines our understanding, aligning the project's direction with broader objectives and expectations.

Conclusion

This requirements analysis document is the first step in bringing the Pace project to life. It lays the groundwork for a tool that promises to redefine productivity through mindfulness, providing users with a means to track their activities, reflect on their work patterns, and ultimately, achieve greater focus and productivity. As we move forward, this document will evolve, reflecting new insights and adjustments to ensure that Pace remains at the forefront of mindful productivity tools.

We invite all stakeholders to engage with this document, providing feedback and suggestions that will enrich the Pace project. Together, we embark on a journey to create a tool that not only meets the needs of today's professionals but also inspires a more mindful approach to productivity and time management.

5.1 Personas

Introduction

A persona is a fictional character created to represent the different user types that might use a site, brand, or product in a similar way. Personas are used in UX research and design to represent the needs of a group of users, which helps guide decisions about product features, navigation, interactions, and even visual design.

Persona 1: The Freelancer

- **Name:** Alex Johnson
- **Age:** 29
- **Occupation:** Freelance Graphic Designer
- **Technical Proficiency:** Intermediate
- **Goals:**
- Track time spent on different projects to bill clients accurately.
- Improve productivity and time management.
- **Needs:**
- An easy way to switch between tasks/projects.
- Export detailed reports for billing purposes.
- **Frustrations:**
- Complex interfaces with a steep learning curve.
- Inaccuracies in time tracking that lead to billing disputes.

Persona 2: The Student

- **Name:** Priya Singh
- **Age:** 22
- **Occupation:** University Student (Computer Science Major)
- **Technical Proficiency:** Advanced
- **Goals:**
- Manage study time effectively across different subjects.
- Incorporate breaks and maintain a healthy study-life balance.
- **Needs:**
- Pomodoro timer to stay focused and take regular breaks.

- Customizable reports to analyze study patterns over time.
- **Frustrations:**
- Distractions leading to procrastination.
- Overwhelming workload and poor time allocation.

Persona 3: The Corporate Employee

- **Name:** Michael Thompson
- **Age:** 35
- **Occupation:** Software Developer at a large corporation
- **Technical Proficiency:** High
- **Goals:**
- Keep track of time spent on various tasks for productivity analysis.
- Balance work tasks with breaks to avoid burnout.
- **Needs:**
- Integration with corporate project management tools.
- Ability to generate and share reports with managers.
- **Frustrations:**
- Rigid corporate software that doesn't track what's needed.
- Lack of customization in reporting for personal analysis.

Persona 4: The Small Business Owner

- **Name:** Elena Rodriguez
- **Age:** 41
- **Occupation:** Owner of a small online retail business
- **Technical Proficiency:** Beginner to Intermediate
- **Goals:**
- Optimize the use of time across various business activities.
- Understand where most time is spent to delegate tasks effectively.
- **Needs:**
- Simple and intuitive interface for quick learning.
- Export data to calculate wages for employees based on time tracked.
- **Frustrations:**
- Complexity and time consumption in learning new software.
- Difficulty in tracking time across different business operations.

Persona 5: The Hobbyist

- **Name:** Jamal Edwards
- **Age:** 28
- **Occupation:** IT Technician / Part-time Game Developer
- **Technical Proficiency:** Moderate
- **Goals:**
- Allocate time effectively between full-time job, side projects, and leisure.
- Track progress on game development projects.
- **Needs:**
- Flexibility in starting and stopping the timer as work gets interrupted.
- Insights into time spent on hobbies for better work-life balance.
- **Frustrations:**
- Forgetting to track time due to getting deeply involved in tasks.
- Lack of motivation when progress isn't visually represented.

Persona 6: The Retired Professional

- **Name:** Susan Choi
- **Age:** 65
- **Occupation:** Retired School Principal, now a Volunteer and Hobbyist Painter
- **Technical Proficiency:** Low to Beginner
- **Goals:**
- Keep track of time spent on volunteering activities and painting to maintain a balanced and fulfilling retirement life.
- Learn to use technology to enhance daily life without feeling overwhelmed.
- **Needs:**
- A very simple and straightforward interface that doesn't require remembering complex commands or navigating through multiple options.
- Clear instructions and feedback from the application to reassure that the input has been correctly registered and the desired action is taking place.
- **Frustrations:**
- Feeling intimidated by applications that assume a lot of prior tech knowledge.
- Difficulty in reading small text or navigating cluttered interfaces, which are common in many applications designed for younger users.
- Forgetting to track activities due to distractions or because the process to start/stop tracking is too cumbersome.

Background: Susan is eager to make the most of her retirement by dedicating time to her passions and giving back to the community. While she is not very familiar with using command-line interfaces, she is open to using a simple tech solution that could help her manage her activities more efficiently. She prefers tools that require minimal setup and offer guidance through their features. Susan values clarity and ease of use over complex functionality and is looking for a solution that fits into her life without adding stress or confusion.

Additional Insights:

- Susan would benefit from a tutorial or guide that walks her through the initial setup and basic commands, with an emphasis on visual aids or step-by-step instructions.
- The application could offer voice commands or integration with more familiar interfaces (like a web interface) for users with varying levels of technical proficiency and physical abilities.
- Feedback and error messages should be clear, encouraging, and instructional, helping Susan correct mistakes without feeling frustrated or discouraged.

5.2 Use Cases and User Stories for Time-Tracking Application

Introduction

A time-tracking application like ”pace” should cater to a diverse user base, including freelancers, students, corporate employees, small business owners, hobbyists, and retired professionals. To ensure that the application meets the needs of these users, it's essential to define use cases and user stories that capture the various scenarios and requirements of each user persona.

Creating use cases and user stories helps in understanding how different users will interact with the time-tracking application, outlining their needs, goals, and the steps they take to achieve these goals. Below are use cases and user stories for each of the personas described earlier.

Persona 1: The Freelancer (Alex Johnson)

Use Case: Track Time for Multiple Freelance Projects
User Story: As a freelancer, Alex wants to be able to easily start and stop time tracking for different

projects so that he can accurately bill his clients based on the actual hours worked.

Persona 2: The Student (Priya Singh)

Use Case: Manage Study Sessions Using the Pomodoro Technique
User Story: As a student, Priya needs a way to manage her study sessions using the Pomodoro technique, enabling her to focus on her studies with regular breaks to enhance productivity and avoid burnout.

Persona 3: The Corporate Employee (Michael Thompson)

Use Case: Generate and Share Time Tracking Reports with Managers
User Story: As a corporate employee, Michael wants to generate detailed time tracking reports that he can share with his managers, showcasing his productivity and the distribution of his work hours across different tasks.

Persona 4: The Small Business Owner (Elena Rodriguez)

Use Case: Optimize Business Operations by Tracking Time
User Story: As a small business owner, Elena needs to track the time spent on various business activities by herself and her employees to identify areas where time can be saved or reallocated to improve efficiency and productivity.

Persona 5: The Hobbyist (Jamal Edwards)

Use Case: Balance Between Job, Side Projects, and Leisure
User Story: As a hobbyist working on side projects, Jamal wants to track the time he spends after work on his hobbies and side projects to ensure a healthy balance between work, personal projects, and leisure.

Persona 6: The Retired Professional (Susan Choi)

Use Case: Track Volunteering and Hobby Activities
User Story: As a retired professional, Susan wants a simple way to track the time she spends on her volunteering activities and her hobby of painting to manage her time effectively and ensure a fulfilling retirement life.

General Use Cases and User Stories

1. **Use Case:** Customizing Reports for Analysis
   User Story: As a user, I want to customize how my time tracking reports are generated and displayed, so I can focus on the information that's most important to me.
2. **Use Case:** Exporting Data for External Use
   User Story: As a user, I need to export my time tracking data in various formats like JSON and CSV, so I can analyze the data in other applications or share it with clients or supervisors.
3. **Use Case:** Learning to Use the Application
   User Story: As a user with lower technical proficiency, I want clear and simple instructions on how to use the application, so I can start tracking my time without feeling overwhelmed by technology.

These use cases and user stories cover a broad spectrum of functionalities and considerations necessary for designing a time-tracking application that meets the needs of a diverse user base. They highlight the importance of flexibility, customization, ease of use, and the ability to export data for various purposes.

5.3 Requirements

Functional Requirements

1. **Task Management:**

- Ability to define and categorize tasks (e.g., by project, subject, business activity).
- Support for starting, pausing, resuming, and stopping task timers with simple commands.

1. **Pomodoro Technique Integration:**

- Timer functionality that supports customizable focus periods and break intervals.
- Notifications or alerts to indicate the end of focus periods and breaks.

1. **Reporting and Data Visualization:**

- Generation of summary reports showing total time spent on tasks, with breakdowns by category.
- Customizable report formats to include specific data points (e.g., daily summaries, project-based breakdowns).
- Visual representations of time spent, such as pie charts or bar graphs, for easier analysis.

1. **Data Export and Integration:**

- Options to export data and reports in JSON and CSV formats for external analysis or record-keeping.
- Compatibility with statistical applications or spreadsheets for further data manipulation.

1. **User Preferences and Customization:**

- Interface customization options to accommodate different user preferences and accessibility needs.
- Settings for Pomodoro lengths, notification types, and report details.

1. **Simple and Intuitive Interface:**

- A clear and straightforward command-line interface with guidance for beginners.
- Error handling with informative feedback to guide users through correcting inputs.

1. **Learning and Support:**

- In-app tutorials or help commands that provide usage instructions and tips.
- Documentation accessible from within the application for easy reference.

Non-Functional Requirements

1. **Usability:**

- The application should be easy to use for individuals with varying levels of technical proficiency, including those who are not familiar with command-line interfaces.
- The interface must be clean, with legible text and straightforward navigation.

1. **Accessibility:**

- Consideration for users with different abilities, including options for text size, contrast settings, and potentially voice commands or text-to-speech functionality for users like Susan.

1. **Performance:**

- The application should have minimal lag in starting, stopping, and switching between tasks.
- Reports and exports should be generated quickly, without significant wait times.

1. **Scalability:**

- The design should allow for the easy addition of new features, such as integration with other tools or cloud-based tracking, without major overhauls.

1. **Security and Privacy:**

- User data should be stored securely, with consideration for privacy, especially if future versions of the app include cloud storage or synchronization.

1. **Portability:**

- The application should be compatible with multiple operating systems, ensuring users can migrate or sync data across different platforms.

1. **Reliability:**

- The application must be reliable, with minimal bugs or crashes, especially during time tracking and report generation.

This overview of requirements serves as the foundation for the next steps in the application development process, including system design, implementation, and testing. By addressing these requirements, the development team can create a time-tracking application that is versatile, user-friendly, and capable of meeting the diverse needs of its intended user base.

Chapter 6

Development guide

Work in progress ...

Justfile

We utilize `just` to provide additional functionality for the development process.

Installation

Install `just` with:

```
cargo install just --locked
```

or by using `scoop`:

```
scoop install just
```

Development dependencies

After you hav installed `just`, you can install the development dependencies with:

```
just install-dev-deps
```

Building the project

You can build the project with:

```
cargo build
```

Running the tests

You can run the tests with:

```
just test
```

Formatting the code

You can format the code with:

```
just fmt
```

Linting the code

You can lint the code with:

```
just lint
```

6.1 Domain Language

Introduction

A domain language is a set of terms and concepts that are specific to a particular domain or subject area. In the context of Pace, the domain language revolves around time tracking, productivity, and project management. By establishing a clear and consistent domain language, you can ensure that users understand the concepts and features of the tool and can use it effectively to manage their time and tasks.

1. Activity

- **Explanation:** An activity refers to the broad category of work or personal projects the user is engaged in. For example, ”Developing Feature X” or ”Writing Blog Post”. Activities encompass one or more tasks and represent a higher-level view of where time is being spent.

2. Task

- **Explanation:** A task is a specific action or piece of work to be completed as part of an activity. Tasks are the smallest unit of work and can be thought of as individual items on a to-do list, such as ”Fix Bug #123” or ”Outline Chapter 4”.

3. Project

- **Explanation:** A project is a collection of activities (and therefore tasks) that are grouped together under a common goal or theme. Projects can be personal (e.g., ”Learn Rust”) or professional (e.g., ”Website Redesign”).

4. Session

- **Explanation:** A session represents a block of time spent working on a specific task or activity. Sessions are started and stopped by the user and are used to track how much time is spent on different tasks or activities within a project.

5. Tag

- **Explanation:** Tags are keywords or labels used to categorize tasks and activities further. They provide a flexible way to organize and filter time entries based on criteria like urgency, context (e.g., ”Office”, ”Home”), type of work (e.g., ”Coding”, ”Research”), or any other user-defined category.

6. Time Entry

- **Explanation:** A time entry is a record of a completed session. It includes the start and end times, the task or activity worked on, and any tags associated with that work. Time entries are the fundamental data points used for generating reports and insights into how time is spent.

7. Report

- **Explanation:** A report is a summary or analysis of time spent across tasks, activities, projects, and tags over a specified period. Reports can show total time spent, time spent per project or activity, trends over time, and other metrics that help users understand their productivity patterns.

8. Goal

- **Explanation:** A goal is a target amount of time the user aims to spend on a task, activity, or project within a certain period (e.g., ”Spend 10 hours on Project X this week”). Goals help users manage their time effectively and stay focused on priorities.

9. Dashboard

- **Explanation:** The dashboard is a visual interface or summary view that displays key metrics, active sessions, recent time entries, and progress towards goals. It provides a quick overview of the user's time tracking status and achievements.

10. Integration

- **Explanation:** Integration refers to the ability of Pace to work alongside other tools or services, such as calendars, project management software, or invoicing tools. Integrations can help automate the tracking of time spent on meetings, tasks, or projects managed in other systems.

By defining these terms clearly, you establish a solid foundation for users to understand and engage with Pace. This domain language not only facilitates effective communication within the tool but also ensures users can leverage its features to their full potential.

More Advanced Concepts

Incorporating terms from GTD (Getting Things Done) and PARA (Projects, Areas, Resources, Archives) methodologies can enhance the flexibility and utility of Pace, making it a powerful tool for a wide range of time management and productivity approaches. Here's an expansion of the domain-specific language to include relevant GTD and PARA terms:

GTD-specific Terms

11. Inbox

- **Explanation:** The inbox in GTD is the initial collection point for all tasks, notes, ideas, and everything else that needs processing. In the context of Pace, an ”Inbox” could be used to quickly capture time-sensitive tasks or activities before they are categorized.

12. Next Actions

- **Explanation:** Next actions are the immediate next steps required to move a task or project forward. In Pace, users could tag sessions or tasks as ”Next Actions” to prioritize them in their daily or weekly workflow.

13. Projects (GTD Context)

- **Explanation:** In GTD, a project is defined as any outcome that requires more than one action step to achieve. This aligns with the general definition of projects in Pace but emphasizes the multi-step aspect of achieving specific goals.

14. Contexts

- **Explanation:** Contexts in GTD are tags or labels that group tasks based on the location, tools, or conditions required to complete them (e.g., @Office, @Home, @Online). In Pace, contexts can help users organize time entries and sessions based on these criteria.

15. Waiting For

- **Explanation:** This GTD term refers to tasks or actions that cannot progress until someone else completes a related action. In Pace, marking a session or task as ”Waiting For” could help users keep track of time-dependent actions involving collaborations or external dependencies.

PARA-specific Terms

16. Areas

- **Explanation:** Areas in the PARA method represent spheres of activity with a standard to be maintained over time (e.g., Health, Finances, Professional Development). In Pace, users could categorize their activities and projects under specific areas to maintain a balanced overview of their commitments.

17. Resources

- **Explanation:** Resources in PARA are thematic collections of information or tools for reference or inspiration (e.g., Articles, Courses, Tools). Though Pace primarily tracks time, it could allow users to link or reference resources associated with specific tasks, projects, or areas.

18. Archives

- **Explanation:** Archives in PARA are where completed, inactive, or irrelevant projects and tasks are stored. In Pace, completed projects and activities could be moved to an archive section, helping users keep their active workspace clutter-free while retaining access to past records for reflection or analysis.

Incorporating these terms from GTD and PARA into Pace not only broadens its applicability across different productivity systems but also provides users with a more nuanced set of tools for managing their time and tasks. This approach ensures that Pace can serve a wide array of users, from those seeking simple time tracking to those looking for a more comprehensive productivity and time management solution.

6.2 Release Process

CLI Releases: `pace-rs`

1. **Review and Merge the release-plz PR**: After the release-plz PR has been reviewed and all checks have passed, merge the release-plz PR. This will trigger the release workflow and create a release for the version(s) mentioned in the PR.

2. **Publishing to crates.io**: After merging the release-plz PR, the release workflow will publish the new version to crates.io. You can check the status of the release workflow in the GitHub Actions tab.
3. **Tag the release**: After the PR has been merged, and the version has been published to `crates.io`, tag the commit on the `main` branch with the version number and push the tag to GitHub. This should make the release workflow run and crate a release for the tag. It will also copy the changelog to the release notes and build the binaries for the release.

   You can tag the release for the latest version on `crates.io` using the following command:

   ```
   just tag-release
   ```
4. **Make latest release**: After the release workflow has finished, run the following command to make the latest release:

   ```
   just make-latest
   ```
5. **Update scoop manifest**: After the release has been made, update the `scoop` manifest to reflect the latest release. You can do this by running the following command:

   ```
   just update-scoop-manifest
   ```

   This will update the `scoop` manifest to the latest release version. Then go to the release page on GitHub and download the signatures for the `pace-rs` zip file and update the `sha256` in the `scoop` manifest.
6. **Update the website**: After the release has been made, update the website to reflect the latest release. You can do this by checking out the website repository and running the following command:

   ```
   just update-pace-version
   ```

   This will update the version of `pace` on the website to the latest release.
7. (Optional) **Write an announcement**: Write an announcement for the release and post it on the pace-rs/pace discussions.

Library Releases: `pace_core` / `pace_cli`

1. **Review and Merge the release-plz PR**: After the release PR has been reviewed and all checks have passed, merge the release-plz PR. This will trigger the release workflow and create a release for the version(s) mentioned in the PR and push the versions to `crates.io`. At this stage, you are done with the release process.

6.3 Testing strategy

As of now, we are using (or planning to use) the following types of tests, to ensure the quality of the `pace` project. These tests are run on every pull request and push to the main branch in our CI/CD pipeline. All tests are run using GitHub Actions and are mandatory to pass before merging a pull request.

Developers are encouraged to write tests for their code and to write tests for code that is not yet tested. We are aiming for a high test coverage and a high quality of tests.

Before a PR the test suite can be run locally using the following command (using `just`):

```
just pr
```

End-to-end tests

End-to-end tests are being used to test the CLI commands and their interaction with the `pace_core` library. They are used to ensure that the CLI commands are working as expected.

Fuzz tests

Currently no fuzz tests are being used. We should consider using them to test functions that parse user input or other untrusted input.

Integration tests

Integration tests are being used to test the service layer of `pace_core`. `ActivityStore` and `ActivityTracker` are the main components being tested.

We initialize the `ActivityStore` with an `InMemoryStore` and the `ActivityTracker` with the `ActivityStore`. We then use the `ActivityTracker` to perform operations on the `ActivityStore` and assert that the operations are working as expected.

Journey tests

Journey tests are being used to test that the workflow of important user stories are working as expected. One example of a journey test is `test_hold_resume_journey_for_activities_passes`.

Mutation tests

Currently no mutation tests are being used. We should consider using them to test the quality of our tests.

Snapshot tests

Snapshot tests are being used to test the output of the CLI commands (see Visual Regression Tests). They are used to ensure that the output of the CLI commands does not change unexpectedly.

Property-based tests

Currently no property-based tests are being used.

Unit tests

Unit tests are being used to test the individual components of `pace_core`. Especially regarding the `Activity` related traits and structs. But also the `time` module is being tested extensively.

Visual regression tests

Visual regression tests are being used to test the output of the CLI commands. We use `insta_cmd` to take snapshots of the output of the CLI commands and compare them to the expected output. This is used to ensure that the output of the CLI commands does not change unexpectedly.

Code coverage

VS-Code/VS-Codium Extension

You can live preview the coverage on your code with this extension:

```
Name: Coverage Gutters
Id: ryanluker.vscode-coverage-gutters
Description: Display test coverage generated by lcov or xml - works with many
 ↪  languages
Version: 2.11.0 (at the time of writing)
Publisher: ryanluker
VS Marketplace Link:
 ↪  <https://marketplace.visualstudio.com/items?itemName=ryanluker.vscode-
 ↪  coverage-gutters>
```

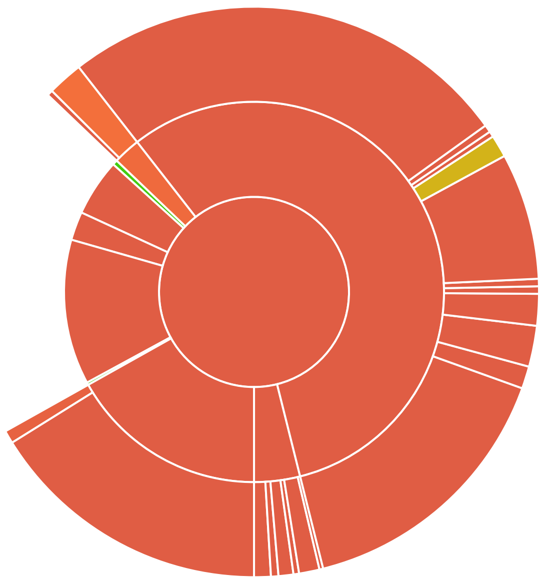
Generate with **`cargo-tarpaulin`**

Install cargo tarpaulin (with `cargo install cargo-tarpaulin`) and run in the repository root:

```
cargo tarpaulin --all-features -p pace_core --output-dir coverage/ -o Lcov
```

Codecov.io

We use `codecov.io` to track the code coverage of the `pace` project. The coverage is updated on every push to the main branch and on every pull request.



You can view the coverage report on the `codecov.io` website.

Chapter 7

Glossary

Activity

A unit of work tracked by Pace, representing a task, project, or period of time.

Activity Session

An activity and all its associated tasks and intermissions, forming a complete work session.

Intermission

A pause or break within an activity, used to track interruptions or non-productive time.

Task

A specific action or assignment within an activity, providing granularity in tracking work.

Activity Log

A record of all activities tracked by Pace, including details such as start/end times, durations, and descriptions.

Time Tracking

The process of recording and monitoring the time spent on various activities or tasks.

CLI (Command Line Interface)

A text-based user interface for interacting with Pace through commands and arguments.

Reflection / Insights

A summary or report generated by Pace, providing insights into time usage, productivity, and activity trends.

Time Zone

A region of the globe that observes a uniform standard time, affecting the representation and calculation of time in Pace.

Data Conversion

The process of transferring data into or out of Pace, facilitating backup, migration, and integration with other systems. Usually involves importing and exporting data.

Configuration

Settings and options that determine the behavior and appearance of Pace, including storage locations, time formats, and default values.

UTC (Coordinated Universal Time)

The primary time standard by which the world regulates clocks and time, often used as the internal time representation in Pace.

Pomodoro Technique

A time management method that involves breaking work into intervals (typically 25 minutes) separated by short breaks.

Database Integration

The incorporation of a database system (e.g., SQLite) into Pace for storing and managing activity data.

Team Collaboration

Features in Pace enabling multiple users to work together, share data, and collaborate on projects or tasks.

Billing and Invoicing

Functionality in Pace for tracking billable hours, calculating costs, and generating invoices for clients or projects.

Chapter 8

Appendix

This section contains additional resources and frequently asked technical questions (FATQs) that are not covered in the main documentation.

8.1 Additional Resources

This section provides a list of additional resources that can be helpful for learning more about Pace and related topics.

Official Website

The official website for Pace provides the latest information, downloads, and documentation for the application.

Pace Website

GitHub Repository

The source code for Pace is hosted on GitHub, where you can find the latest releases, source code, and contribute to the project.

Pace GitHub Repository

Documentation

The official documentation for Pace provides detailed information on installation, usage, and development of the application.

Pace Documentation

Community Discord

The official Discord server for Pace provides a place to ask questions, get support, and discuss development and usage of the application.

Pace Discord Server

Github Discussions

The official GitHub Discussions for Pace provides a place to ask questions, get support, and discuss development and usage of the application.

Pace GitHub Discussions

8.2 Frequently asked technical questions

How does pace store my data?

Pace stores your data in a TOML file for now. This is a simple and human-readable format that is easy to parse and write - and to backup. Read more about how pace utilizes the storage.

How can I import my data from other time tracking tools?

Pace currently does not support data import, but it is planned for the near future. The data is stored in a human-readable format, so you can always convert it manually. Or write a script to do it for you. The storage format is documented.

Can users integrate pace with other tools?

Pace currently does not support integrations, but it is planned for the near future. The data is stored in a human-readable format, so you can always write a script to do it for you. This is a bit cumbersome, also when it comes to syncing data. The idea is to have a plugin system, where you can write your own plugins to integrate with other tools.

How can I export my data to other time tracking tools?

Pace currently does not support data export, but it is planned for the near future. The data is stored in a human-readable format, so you can always convert it manually if you need to. Or write a script to do it for you.

Will users be able to use pace on mobile devices?

Pace is a command-line tool and is not designed to be used on mobile devices. However, you can use it on a server and access it via a web interface. This is planned for the near future. Also different frontends are planned for the future, for Desktop and Mobile.