1. Team Working Agreement

Last Revised: 02/08/26 [Time Period]

1.1. Memori

- Cainan Enneking - Developer
- Surendra Jammishetti - Product Owner, Developer
- Kenric Tee - Scrum Master
- Preston Clayton - Developer
- Julian Montano - Developer

2. Definition of Done

2.1. User Stories

2.1.1. Functionality implemented fully

- All tasks for a user story are completed and functionality is implemented as needed by requirements.
- Code integrated in correct place, and is shown to work for primary use case.
- Code is merged with the main branch.

2.1.2. Additional components removed

- Anything that isn't necessary to functionality of code must be removed (i.e excess comments, files that aren't used, anything unaccessed)
- Testing console.logs removed

2.2. Sprints

- Demo shown and approved of by group (i.e. no objections or suggestions)
- All planned user stories in the sprint backlog have been completed and meet DoD for user stories
- Any incomplete stories are moved to next sprint backlog
- Sprint retrospective is held to discuss what went well, what didn't, and areas for improvement
- Sprint artifacts (burndown chart, backlog) are updated to reflect progress

3. Style Guide

3.1. Our Styles

- Primarily focused on:
  - Rust code meets compiler based style guidelines (snake case for vars, camel for types)
  - Following Google's JavaScript Style Guide
  - Encapsulate components in separate files for reuse/readability
  - Created helper functions and stores files for organization
  - Formatting enforced by formatter runs (imports, spacing, etc)
- Documentation and commenting approach
  - No need to say who wrote what, important functions should be outlined with comments descsribing functionality and idiosyncrasies
  - Splitting user stories into smaller coding tasks has helped us allocate areas of code to be worked on in a non-overlapping fashion
- Code formatting and spacing approach
  - Rust code is formatted using rustfmt

- Code organization and reusability approach
  - As visible in file structure, most modular functionality is broken into separate rust crates.

4. Folder Structure

- `doc/` - Holds all of our sprint plans / reports / team working agreements / release plan.
  - `releaseplan` - self explanatory.
  - `sprint1` - Documents for sprint 3.
  - `sprint2` - Documents for sprint 3.
  - `sprint3` - Documents for sprint 3.
  - `sprint4` - Documents for sprint 4.
  - `templates` - pdf templates from canvas.
- `memori-app` - App code.
  - `src` - source code for web application.
    - `routes` - contains app layout and page files
    - `lib` - shared utils/components for app
      - `tauri` - ipc layer (Rust TypeScript)
  - `src-tauri` - source code for the tauri rust backend.
  - `static` - collection of sattic assets.
- `memori-dev` - Device level code
  - `memori-esp32c3` - Firmware for the ESP32-C3 microcontroller.
  - `simulator` - Code for simulating the device.
- `memori-transport` - Transport layer implementation for communicating between host and device.
  - `ble-device` - Device side implementation of communication over BLE.
  - `ble-host` - Host side implementation of communication over BLE.
  - `memor-tcp` - TCP implementation for communicating between host and simulator device.
  - `transport` - High level type defs and shared traits for communication.
- `memori-ui` - The UI for the Memori application.
  - `widgets` - Holds the UI for various widgets.