Code </> Studios – SRS DOC

"What are the Essentials Here?"

- System functional:
  - Administration
    - Admin account(s) needed for system management. Will have permissions to see database, ad, and make appropriate edits to the same
  - Transactions
    - HTTP requests: GET, PUT, UPDATE, etc.
  - Regulatory and compliance
    - Functional domain
    - Privacy policy
    - Cookie consent
    - Check for copyright/plagiarism
    - Terms and conditions?
  - Performance requirements
    - Fast load time
    - Feedback if buffering
  - Page operations
    - Entry page
      - Send to login page
    - Login page
      - Send request for validation to OAuth server
      - Receive confirmation/denial
      - Show error or redirect to main page
    - Main page
      - Calendar display
      - Button/link to reservation page
    - Reservation page
      - Entry of information to make CRUD requests of API
    - Desktop Pi
      - Make GET requests to the API
- External interface:

    - Server
        - Webserver
            - CRUD requests to the API
            - Display reservations in webpage
                - Datetime
                - Reserver
                - Room
                - Display forms for managing reservations
            - Display reservations desktop application (Raspberry Pi)
                - Datetime
                - Room
            - Authentication
                - Redirect to AAD login page.
        - API
            - Service requests from Webserver and Raspberry PI
            - Data validation
            - Security (refuse erroneous or unauthorized requests)
        - Database
            - Service queries from API
            - Store data
    - Physical
        - Raspberry PI
        - Screen
    - Azure Active Directory
        - Azure SSO SAML, service requests from the Webserver
- Non-functional
    - Security: Software will protect any PII (Personally Identifiable Information) at rest and in motion. PII will be encrypted at all points in the system workflow. Additionally, room display systems will be locked down to only being able to read relevant information to displaying the room reservation status. The display system will not have any access to PII but will be locked down to the best of our ability so that equipment cannot be stolen or tampered with. Code

bases will not be accessible by the public and restricted access to specific users.

- Capacity: This system is designed to be locally installed on location, it is thus as scalable as the number hard drives the server can hold. A single server is not likely to need more than 100 GB of storage (which includes software and database). 100 GB is an overestimate but should allow for system longevity.
- Compatibility Server:
  - **OS:**
    - Modern Linux distro such as Ubuntu v24.04.1 LTS
    - Nginx for reverse proxy of API and Webserver
  - **Software:**
    - Webserver: Node.js v22.22.0 LTS
      - React v18.2.0 LTS
    - API: GO v1.23.3 LTS
      - Echo v4.12
    - DB: Postgres or MariaDB (version doesn't matter much)

- Compatibility Pi:
  - **OS:** Most recent version of Raspberry Pi OS: released 2024-10-22, Linux kernel 6.6.51, Chromium 130.0.6723.58
  - **Desktop Application SDK:** Electron version v33.1.0, or Flutter 3.24.0
- Reliability and availability: *How often do you expect users to be using your software and what critical failure time is under normal usage.*
  - Only a few users are expected at any given time. General use times are very minimal. Most of the time, users will just sign on, check or make a reservation and log off.
  - Although very little harm will result from a critical failure our system would still hope to have a mean-time-between-failure that better than once a month.
- Scalability: (target maximums)
  - 3000 users viewing reservations

        - 500 simultaneous (quick succession) reservation creation/update requests
        - 50 physical interfaces
    - Maintainability Web Application:
        - About self-hosted runners - GitHub Docs
            - We may end up manually uploading changes with FTP and then rebuilding GO and Node.JS over SSH.
            - However, this can become tedious and is not really a continuous integration. A self-hosted runner is probably our best option for continuous integration.
            - GitHub actions can pull from the repository and rebuild the application and API. Alternatively, we could write a shell script to do this, utilizing Git CLI.
    - Maintainability Desktop (Pi) Application:
        - Electron or Flutter:
            - Open-source application enabling authors to contribute issues or PRs to the project that is then manually updated and then downloaded via GitHub Releases to the Pi.
    - Usability Web Application:
        - Web usability is a priority, if it is not easy to use nobody will use it. Simple straight forward forms to quickly add reservations and view current reservations. Users should be able to create reservations
    - Usability Desktop (Pi) Application:
        - The Raspberry Pi interface will be simple and non-interactive. Will concisely display current and upcoming reservations for the day.