Doctor OnDemand

Author: Jinal Bangur

Team Members: Ayana, Jinal, John and Tiahao

Date: 2/4/2019

Subsequent issue dates listed earliest to latest

INTRODUCTION

DoD is being developed to allow patients to talk to a Doctor whenever, regardless of whether you have medical insurance coverage or not. Medical Care 24/7 is the future for treating patients.

The concept of this application Doctor OnDemand came from end users, customers and/or patients who expressed other alternatives to Urgent Medical Care facilities, Emergency Rooms and Hospital appointments for minor concerns and questions.

The Doctor on Demand App has been developed by a team who wanted to make an impact on the medical industry as we know it. The DoD App goal is to reach anyone who does not want to wait days to visit a doctor or must book an in-person appointment just to speak with their primary care physician.

The focus for now is on the national level with the intent to expand internationally in the future. So, come and sign up and enjoy a new fresh idea to modern medicine as we know it.

# ROLES AND RESPONSIBILITIES

These vary for each type of product and, for small projects, folks may serve multiple roles.

This is a list of common roles we have used for software development:

**Development Lead** - John

**Buildmeister** - John

**Architect** - Jini, Ayana

**Developers** - John, Ayana, Tony

**Test Lead** - Jini, Ayana

**Testers** - Jini, Ayana, Tony, John

**Documentation** - Jini, Ayana, Tony, John

**Documentation Editor** - Jini

**Designer** - Tony

**User advocate** - Jini

**Risk Management** - Tony

**System Administrator** - Ayana

**Modification Request Board** - Jini, Ayana, Tony, John

**Requirements Resource** - Jini

**Customer Representative** - Jini

**Customer responsible for acceptance testing** - Ayana

# METHOD (required)

These are unique to software development, although there may be some overlap.

* **Software:**
  + Google Flutter for App Development (Using WebRTC plugin for Flutter <https://pub.dartlang.org/packages/flutter_webrtc>)

Example: https://github.com/cloudwebrtc/flutter-webrtc

* + Android OS 9.x
  + Google Cloud Platform for hosting
  + Travis for CI/CD tasks
  + Draw.io for diagrams / wireframe
  + Ubuntu Linux for hosting
  + Containerize (later)
  + MySQL 11.x Database (managed)
  + Go Lang 1.11.x Web Services
  + Text Chat (Rocket.chat <https://rocket.chat/docs/developer-guides/livechat-api/>)
    - Server: 2.1
    - SDK: 0.0.38
  + VS Code for Web Service work
  + Android Studio for App Development
  + Go Lang unit testing for Web Services
  + Google O-Authentication for securing API
  + Google Pay API for payments
  + Google Maps API in DoD application
  + Investigating Flutter Unit Testing (<https://flutter.io/docs/testing> - includes unit testing, widge and other tests) (Tony TODO)
  + Investigating Video Chat options (Tony TODO)
* **Environments:**
  + UAT Web Service stack
  + Dev Web Service Stack
* **Backup plan (individual and project)** 
  + We have everything uploaded on GitHub as a backup
  + Also, local copies on our desktop
* Review Process:
  + Will you do architecture, usability, design, security, privacy or code reviews? – Yes
  + What approach will you use for the reviews (formal, informal, corporate standard)?
    - Formal approach for architecture, usability, design, security, privacy or code reviews
  + Who handles the reviews and resolving any issues uncovered by the reviews?
    - Ayana and John for code reviews backend
    - Tony for code reviews frontend
    - Jini for architecture, usability, design and documentation
    - Ayana and John for security and privacy
  + Code readings? Yes
* **Build Plan:**

**Revision control system and repository used:**

* + Github <https://github.com/aperry567/SSW690B>
  + Travis CI for CI/CD only running against Development branch
  + CI/CD builds kicked off by merges to Development branch
  + Branching Strategy
    - Master Branch (Production code, ready for demo)
      * No direct commits
    - Developer Branch (Issues completed and merged, tested)
      * No direct commits
      * Only branch into directly for quick bug fixes
      * Pull requests will go into this branch from separate branches
    - Issues (branch for issue working on)
      * Named after ticket with short title
    - Pull Requests requires 2 approvers for development branch to master branch.
    - Critical Bug issues
      * Can be a branch that has a pull request to master also requires 2 approvers
  + Regularity of the builds – daily
  + Deadlines for the builds – Every week Monday 6pm
  + Multiplicity of builds – Frequent (Almost Daily)
  + Regression test process – TODO (Jini, Tony)
* Modification Request Process:
  + MR tool
  + Ticket Process
    - Labels for sprints
    - Kanban board represents active sprint
  + Decision process (board – if more than paragraph should point to alternate description)
  + State whether there will be two process streams one during development and one after development: Yes, we have Dev and UAT environments

# **Virtual and Real Work Space**

Real work space specifies meeting every week in class on Tuesdays also virtually on google hangouts every Fridays as well as Saturdays. Using google docs, google drive as well as GitHub for document repository system and is accessible only to the project team members.

# COMMUNICATION PLAN

## “Heartbeat/Sprint” meetings

Tuesdays at 6:30pm (In class)

Fridays at 7:30pm (Google Hangout)

Saturdays at 8:00pm (Google Hangout)

Google Hangouts for meeting when not in person.

WhatsApp for team communication

## Status meetings

Within typical class meeting time

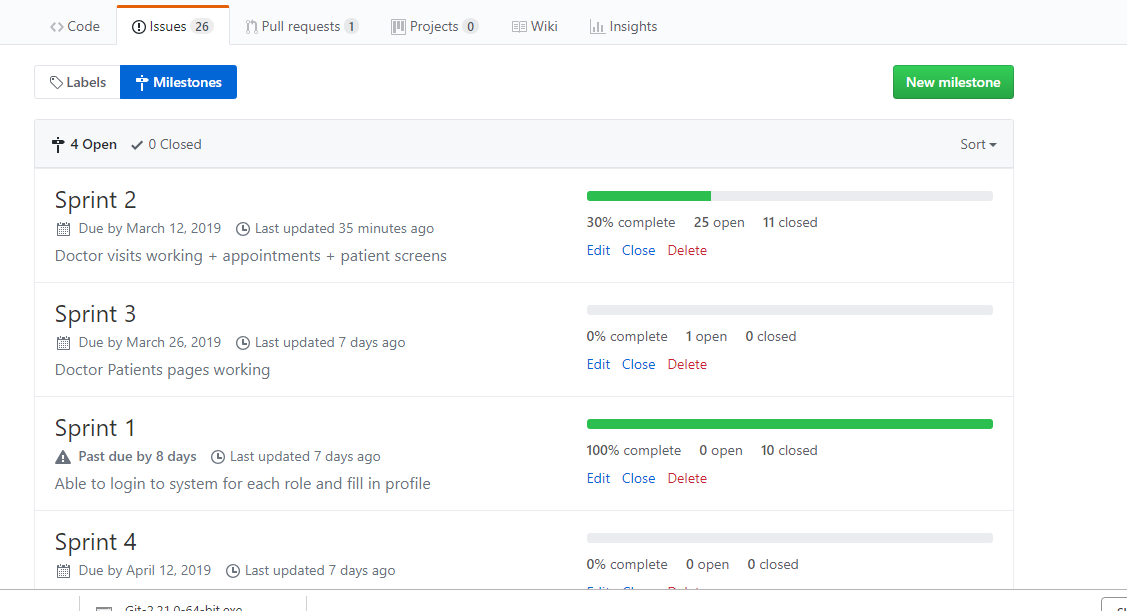
WhatsApp for team communication

## Issues meetings

If a problem does arise, never surprise your manager. Schedule a meeting at his or her earliest convenience.

* WhatsApp for team communication
* Schedule a meeting remotely at a convenient time
* Google Hangouts for the meeting

# TIMELINE AND MILESTONES



# TESTING POLICY/PLAN (optional–software relevant)

This should probably point to a plan or the document would get unwieldy. At the very least it should describe when testing begins.

# RISKS (required)

1. Schedule / Time-Related / Delivery Related Planning Risks
2. Budget / Financial Risks
3. Operational / Procedural Risks
4. Technical / Functional / Performance Risks
5. Other Unavoidable Risks

Link : https://www.getzephyr.com/insights/how-identify-and-manage-software-testing-risks

# ASSUMPTIONS (required)

It may be clear to the project insiders what assumptions are being made about staffing, hardware, vacations, rewards, … but make it clear to everyone else and to the other half of the project that cannot read your thoughts.

# DISTRIBUTION LIST

Professor

DoD Team

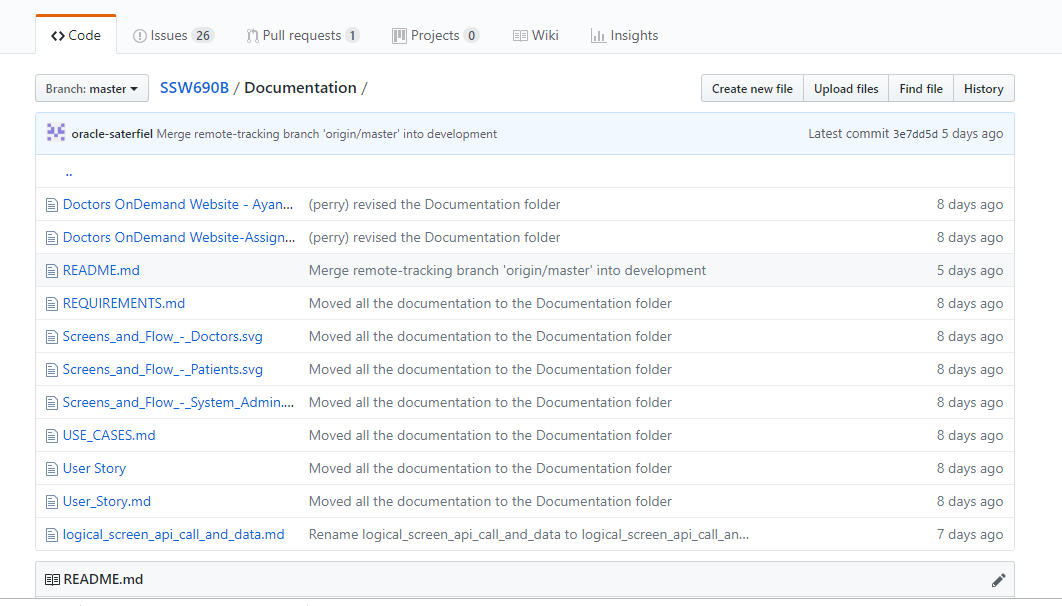
# MORE OPTIONAL SECTIONS:

These should be self-explanatory.

Worry beads

I add this section to describe the things as manager I am most worried about at the time of latest document issue. This section is useful because it helps you to focus on the parts most likely to fail. Sometimes, I segment the worries by time scale: day, week, month, quarter … lifetime.

Documentation Plan



Build Plan

When builds and testing become complex, this might be a separate section or point to a separate document.