Problem
- Medical doctors in Malawi need a system that allows them to triage, track, store, and provide metrics on patients they assess in rural areas of Africa.
- My specific problem space is:
  1. Design and implement a logical workflow that governs the interaction between the client and server, thus enabling all devices in the system a methodology to keep and restore synchronization
  2. Creating a server application to handle data from clients and cloud applications

Current System
- Cloud Application
  - Design and implement a logical workflow that governs the interaction between the client and server, thus enabling all devices in the system a methodology to keep and restore synchronization
  - Creating a server application to handle data from clients and cloud applications

Object Design
- Communication is handled by ServerCore
  - TCP/IP via Bonjour
  - Done over Ad-Hoc or Wi-Fi
- System uses Command, Factory, Façade, Singleton and Delegate design patterns
- System Data Objects Abstract the tables in the database while BaseObject receives and sends data

Implementation
- Objects that need to be saved on server are marked as “Dirty” to identify which objects need to be synced with the Cloud
- Locked Objects can be opened by the user who locked it or through the Server Application
- If an error occurs, the server will save what it can and report what it could not save

Requirements
- US-20: As an application administrator I want the system to gracefully fail and give a proper error message so that action can be taken quickly to rectify the issue
- US-21: As an application administrator I want to verify when the local system backs up to the Cloud so that I can verify the data’s status
- US-24: As an application administrator I want easily backup and restore information to local and remote servers so that data loss can be prevented
- US-103: As an application administrator I want the server to connect with the physical Device so that information can be persistent throughout the system

Verification
- Test & Code followed the Behavior Driven Development pattern
- Testing frameworks include:
  - GHUnit
  - OCMockito
  - OCUnit

Summary
- This solution was built using Xcode, and employed 3rd party tools such as GHUnit & OCMocito for testing and GCDAsync for networking
- In a worst case scenario the system will store data on the client, even if the server is down.
- The workflow allows users to make changes confidently without overwriting each other’s information
- The Server application offers an easy to use interface that offer users the ability to see the status of the system and its data

Acknowledgement
The material presented in this poster is based upon the work supported by Orant Charities, FIU and my Mentor Steven Luis. I am thankful to the help that I received from my group members Carlos Corvia, Rigoberto Hernandez, Steven Berlanga, and Sebastian Zanlongo.