# Mobile Health EMA
# Solution Approach

**Senior Design I (Cpt S 421)**

Erin Mullen | Micah Jenkins | Kily Nhan | Kim Nguyen | Edwin Zheng

**Mentors:**
Porismita Borah
Nicole O'Donnell

**Instructor:**
Behrooz Shirazi

## Concepts, Algorithms, or Other Formal Solutions Generated

***Web Application (admins and clinicians)***[2]

The web-app will consist of four main classes[2]: Program, User, Study, and Message. Each class will be encrypted to maintain security.

**Program Class**

- In charge of current user
- Interacts with UI
- Handles login and logout procedures

**User Class**

- Abstract base class
    - Three types of users (administrators, clinicians, participants) inherit from User
- Admins and clinicians can start studies, create and schedule surveys and messages, download data from the database, control permissions, and remove Users from a Study
- Admins are the only User type with the ability to delete data and add/remove clinicians

**Study Class**

- Contains a list of messages/surveys, participants, and clinicians/admins
- If an instance of a Study has no administrators or clinicians attached to it, it will be removed

**Message Class**

- Contains timestamps and contents (text, image, or both)
- Admins and Clinicians may create, modify, and schedule instances of Message through a Study
- Surveys inherit from Message
    - Stores answer and question type (ex: a survey question might have a text attribute with the value “What’s your favorite color?” with a textbox answer type)

(Note: classes will be similar for the mobile application).

***Model View Controller for Mobile Application (study participants):***

**View[1]:**

- The view subscribes to the controller when there is any update information in the controller
- Display the information to the participants
- This should be the app itself to display any notification/survey questions

**Participants[1]:**

- Can only see the update from the View
- Send requests to the controller to respond to the events in the View
- This “participant” component in the MVC design pattern could also be the Admin(web portal) side as well

**Controller[1]:**

- Responsible for any updates/changes
- Should somehow link to the Admin side
- Receive requests from the Admin
- If no response from the participants, resend the notification

**Model[1]:**

- Notify the update to the controller
- Is the server side that responds upon the controller's request
- PostgreSQL
- Encrypted data when database is being transfer

(Note: the mobile app MVC architecture will be similar for the web application).

***Database***

For the database we will work with PostgreSQL. We will be able to store data securely and return that data in response to requests from applications. PostgreSQL can handle workloads ranging from small single-machine applications to large Internet-facing applications with many concurrent users.

The data being stored will contain user information, research topics, surveys, and survey answers. Therefore, we can use the ER Model and Relational Model to visualize the relationship between entities. An ER diagram consists of an entity, its attributes, and the association among entities.

ER diagrams can be also mapped to relational schema. Even though we cannot import all the ER constraints into relational model, an approximate schema can be generated. There are several processes and algorithms available to convert ER Diagrams into Relational Schema, with the following primary steps:

1. Create table for each entity
2. Entity's attributes should become fields of tables with their respective data types
3. Declare primary key

Since the nature of our client's research does not require a large server to maintain the best connection, we do not need to focus much on the internet trafficking problem. The server will be provided from our client so they can continue to access after we deploy the application.

Roles will either be a user (a role that can log in), or a group (a role of which other roles are members). Permissions can be granted or revoked on any object down to the column level, and can also allow/prevent the creation of new objects at the database, schema or table levels. And considering the size/scale of data that our database need to sort isn't very big, PostgreSQL will be easily handle it. The only tricky part will be connecting the PostgreSQL on the server with the applications, depending on how secure the server that our client provided is, and how much access we have to the provided server. The configuration of connecting database with the applications may take some time and outside sources to figure out during the actual implementation.

## Solution Selection Process—Discussion of Design Choices

Breaking up the project into three separate entities- web application, mobile application, and database, is not only logical but helpful in order to understand the flow of data as well as permissions and functionality.

For the mobile application, by using the model-view-controller design pattern we can better control the flow of data and understand how the application communicates with the users and database. We can also restrict the access a user (the study participant) has in the app and protect their data from being accessed or altered.

By using PostgreSQL, security will be easier to manage since PostgreSQL automatically manages its internal security on a per-role basis. It is the default database for MacOS, and is also available for Windows and Linux.

When it comes to the user interface of the web application, we decided our best option would be to utilize Bootstrap. The framework for responsive HTML/CSS and JavaScript is provided with Bootstrap, and since it is open source and free to use we can edit the JavaScript as necessary to connect the logic discussed above. Not only is the framework already in place, Bootstrap is also easy to use and web pages created using the framework are clean, mobile responsive, and simple to understand. Our team decided to use a web application instead of a desktop application because it allowed us to easily deploy cross-platform and simplified the process by allowing us to use HTML/CSS and JavaScript for the UI instead of C#.

## Theoretical or Formal Description of Selected Solution, Including Considerations for Software Testing (both Alpha and Beta Versions)

Software testing for the Alpha and Beta versions can be done piece by piece along each step of the way. As discussed in previous documents, by using Xamarin we will have access to the Xamarin Test Cloud, which allows us to easily test the app on hundreds of mobile devices.

Mobile app (survey participant) testing:

- Ensure messages are received
    - Includes checking that messages have intended format/content and are received at the correct time
- Filling out surveys
    - Answers is recorded accurately
    - Starting and ending timestamps are accurate
    - Test online and offline mode
- Testing will be done mainly using Xamarin's test cloud

Web app (admin & clinician) testing:

- Creating basic surveys & messages
- Scheduling of messages and surveys
    - Confirm that the messages and surveys are sent at correct times
- Creating an admin/clinician account
    - Testing login/logout functionalities
    - Test data/survey access
- Creating basic studies
    - Studies with small numbers of participants
    - Testing access to studies (ensuring other users cannot access study information and dashboards)

Database Testing:

- Manually query the database to ensure that data is formatted and stored correctly
- Test communication between mobile app and server as well as web app and server

Alpha testing will be similar to Beta testing, but the mobile app will be tested on our own devices, as well as on desktop. We will also run detailed real-world scenarios using actual operations and mock study participants. This stage will also be testing the UI- including its functionality, layout, and visual feedback.

## Summary of the State of This Project

Where this project currently stands we have a detailed list of what's needed by the stakeholders, and what tools we have at our disposal. The project has been broken down into three main portions: a web application, a mobile application, and a database. Each of these portions have been further explored and described through diagrams (see appendices).

## Future Work for This Semester

Going forward this semester our main goal is to have a basic working program. The basic program (also known as the Beta Program) will allow the web app (admins and clinicians) to send messages and make surveys, while the mobile app (study participants) should be able to receive those messages and take take surveys. The database will be able to store surveys and messages. We will start this process by setting up a basic messaging system between the web application, database, and mobile app.

We will not be prioritizing UI design this semester.

## Appendices

Figure 1:

Model-View-Controller (MVC) for the participants

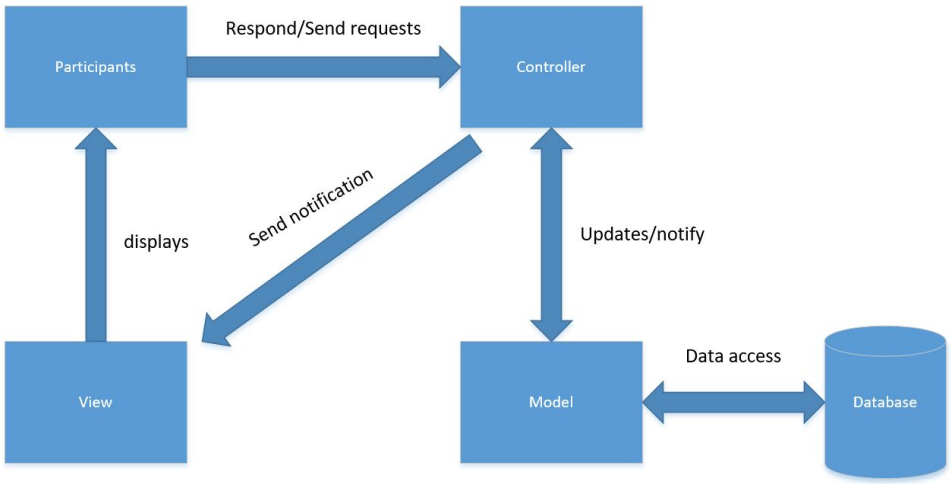


Figure 2: