Getting the project set up:

- Install android studio
    - I would suggest build a few simple projects to ensure you have everything set up properly before diving right
- Fork/clone my GitHub repository sdunn94/SoccerManagement
- Make a firebase account and set up an app (don't add any data yet, just set it up)
- Open up the project
- Copy the URL to your firebase app (this is the URL from the page that you go to when you click manage app on the app you made)
- In the project, find the resources folder, open up the strings.xml and paste your URL into the FirebaseURL string
- Uncomment the portion of the MainActivity OnCreate method that will load some data into firebase for you
- Run the app, so the test data is loaded into firebase (DO NOT DO THIS UNTIL YOU HAVE CHANGED THE URL'S TO YOUR OWN ACCOUNT!!!!!!)
    - Note: the app is meant for tablets so whatever emulator you use, make sure it is a tablet
- Read through the overview of functionality document

Code Documentation: (by class in alphabetical order)

- DeletePlayer (associated with activity_delete_player.xml layout)
    - OnCreate sets up the OnItemClickListener for the listView so that when an item is clicked it will ask for confirmation for the delete then delete the selected player. It also sets up the adapter for the listView that pulls directly from firebase to populate the list.
- HyperlinksActivity (associated with activity_hyperlinks.xml layout)
    - Does virtually nothing!!! Fun stuff is in the layout file
- ItemAdapter
    - Extends the ArrayAdapter for a Player
    - This is what fills each row in the list views in the practice set up page and the practice field page
    - It takes the player_profile_row_layout.xml as a template for each row and fills in the appropriate information about each player supplied from an ArrayList
- MainActivity (associated with activity_main.xml)
    - OnCreate sets up the button's click listeners, which simply navigates to their respective activities
    - The commented part below this is the code that will load some test data into firebase
- NewProfileForm (associated with activity_new_profile_form.xml)
    - Two ways to get here, one starting a brand new profile and two editing an existing profile
    - OnCreate determines if there was information passed from the Profiles activity indicating that they want to edit a profile

        - If there is information passed along, it fills in all the fields with that information
    - submitNewProfile gathers up the information (with some safe guards), creates a new Player with that information, and pushes it to firebase
    - the bottom two methods are for retrieving a picture from the gallery on the device
- Player
    - This is the class that represents a Player object
    - Note: the data stored inside a player is the same data that is stored on firebase. Also notice the names of the data members are the same and they are on firebase. These names must be consistent.
- PlayerLists
    - This class is in place to help pass data between the Profile activity and the NewProfileForm activity
    - When the information is sent from one activity to the other only the name of the player is passed
    - This list of players is filled in the Profile Activity and then searched through in the NewProfileForm activity to find the player with the matching name so all the fields on the NewProfileForm activity can be filled
    - Note: this is probably not the best solution, but I did not have time to go back and re-design it
- PracticeFieldActivity (associated with activity_practice_field.xml)
    - OnCreate sets up the listener for when the radio button selection changes, and the listener for any changes in firebase, as well as initializing the adapters for the four group arrays
        - ChildEventListener: OnChildAdded is activated with the activity is loaded and basically sorts the players where they need to go based on their group number and whether or not they are in play
        - ChildEventListener: OnChildChanged is activated whenever something in firebase changes
            - First there is another listener set up inside here that deals specifically with the timer functionality i.e. when it is started and stopped
            - Under this it tries to determine what scenario prompted the change in firebase and each scenario is handled (see comments in code for more details)
    - isInList with a player determines if the given player is in the given list
    - removeFromList removes the given player from whatever list they happen to be in
    - listItemClickListener listens for a long click on an item in the list view, gathers the necessary data, and starts the drag event
    - fieldItemClickListener listens for a long click on an item in the field, gathers the necessary data, and starts the drag event
    - MyDragShadowBuilder creates the shadow that appears when drag event begins

    - MyDragEventListener listens and handles all the different things that happen during a drag event
        - ACTION_DROP determines what drag scenario happened i.e. what was dragged and where it was dropped and handles each scenario accordingly
    - generateImageViewForPlayer generates an image to represent the given player on the field at the provided position
    - startTimerListener starts timer for all in-play players
    - stopTimerListener stops timer for all in-play players
    - clearTimersListener asks for confirmation then clears out all the timers for all the players (in-play and in the lists)
    - clearAllListener asks for confirmation then clears all the timers, resets all the group numbers, moves all players off the field, and redirects to set up activity
    - clearFieldListener makes sure the timers still aren't running, asks for confirmation then moves all players on the field back to the lists
- PracticeSetUpActivity (associated with activity_practice_set_up.xml)
    - OnCreate sets up the listener that gets activated when any change is made in firebase
        - ChildEventListener OnChildAdded activates when the activity loads and sorts all the players from firebase into the proper list views based on what group they are in
        - ChildEventListener OnChildChanged activates when any change to data in firebase takes place and handles each scenario that could have prompted the change in data
        - Finally oncreate sets up all the adapters and listeners so that you can drag and drop between any combination of lists
    - isInList and removeFromList are basically the same as the ones in practice field activity
    - clearClickListener sets all the player's group number back to zero which puts them back in the master list
    - startPracticeListener just navigates to the practice field activity
    - listItemClickListener listens for a long click on an item in a list view, gathers up the necessary information, and starts the drag event
    - CustomShadowBuilder creates the shadow that appears when drag event begins
    - CustomDragEventListener handles all the different events that are associated with a drag event
        - ACTION_DROP resets the player's group number to be the group that they were dropped into
- ProfileActivity (associated with activity_profile_activity.xml)
    - OnCreate sets up click listeners for the items in the list and a firebase adapter that populates the list directly from firebase
    - startNewProfile navigates to newProfileForm acitivity
    - deleteProfile navigates to deletePlayer activity
- Profiler (Singleton)

    - Handles all the timer start, stop, clear, and duration functionality
    - Note: this code may look familiar if you have taken OOP class with Mark Mahoney because it is taken from a project in that class ☺
- Timer
    - Class that defines what is contained in a timer object
    - Used in the Profiler class

Sample Scenario to demonstrate how things connect:

1. On practice field, drag player from list to field
    a. When long click action is done on a player in one of the listView the listItemClickListener is activated which collects the name of the player being dragged and starts the drag event
    b. This activates the MyDragEventListener ACTION_DRAG_STARTED event which vibrates the device
    c. Next when you drop the player onto the field the ACTION_DROP event is activated and the name that was sent with the drag is extracted and the correct player found
    d. It then updates that players inPlay value to true in firebase
        i. This activates the listener that was created in oncreate specifically the onChildChanged event
        ii. It extracts the player that was changed from the snapshot of firebase
        iii. At this point the player's inPlay value is true but it doesn't have a position so nothing happens
    e. Then it updates the player's X position in firebase
        i. This again activates the listener's onChildChanged event
        ii. At this point the player is in play and has an X position so still nothing happens in the listener
    f. Finally it updates the player's Y position in firebase
        i. This again activates the listener's onChildChanged event
        ii. Now that all three values have been updated it checks to see if the player is already in play because this listener also get activated at other points in the program and the player will already be in play i.e. moving a player to a new position
        iii. Finding that the player is not in the inPlayPlayers list it will add it, then remove it from which ever list it came from and notify the item adapter for the list this player came from that the data set had changed and the list will be re-populated without the player that was moved
        iv. Finally it will generate the image that will appear on the field

Other useful information

- Apk file (executable) is located in your_project_folder\app\build\outputs\apk
    - this file is what I email to the coach so he can install it on his tablet

- My email address for questions/clarification sarah.dunn000@gmail.com
- There are no partially implemented features in this app, I finished everything I started so who ever works on this after me can simply add new features or modify the existing features to fit whatever new requirements there may be