fSER421 Fall 2018 Lab: Android Hybrid Apps
Assigned 11/19/18, due 11/30/18 at 11:59pm via online submission to Blackboard

Objectives:

1. Demonstrate proficiency in constructing a hybrid Android app
2. Understand the Android build process and toolchain

Overview:

For this lab you will adapt a pre-existing client-side DOM manipulation web application to a hybrid app on Android using both native and WebView activities. ***This lab is mandatory, it may not be dropped as a lowest grade!***

Activity 1 (30) : Port a game (as is) to an Android WebView

This task is creating an Android app with a single Activity that is a WebView, and loading a game into the WebView as a local asset. Enable the proper features on the WebView to have the game function properly. The game is the game of Clue (text version), which is available on Blackboard. Start by running the game as an SPA in your web browser to get a feel for what it does.

The requirements for the Clue game are on the next page, but they are already implemented in the given file.

Activity 2 (50 points): Move the username start to a native Activity

Your app, after completing Activity 1, should be a single Activity in a WebView. Modify this solution to make a native Activity be the new starting point for the app. This new native Activity should:

R1. Get the username. That is, take the "Enter your name" textbox and Submit button out of the WebView and put it in a native Activity.
R2. Make sure this new native Activity is the initial Activity for the app.
R3. Upon clicking Submit in the native Activity, make the WebView the current Activity. However the WebView should start with the screen that says "Welcome <name>, You hold the cards…"; that is, skip the old initial screen. <name> should be bound in the WebView to the native-side entered textbox.
R4. Currently, when a game completes, the "Continue" button resets the game by clearing the History and setting the Record. Now, the Continue button should still clear the History and set the Record, but it should also make the native Activity the top of the Activity stack.
R5. Your app now has a native Activity and a WebView Activity. Provide the most appropriate behavior for the "back" button in each part of the app.

Activity 3 (20 points): Modify the storage of the app

Review functional requirements #7 and #8 and non-functional requirement #3 on the next page. It indicates that the app saves the state of the game's history and record within the DOM itself using hidden elements (see the code). To do:

R1. Change the game's history to use the proper (meaning best design) of Cookies, SessionStorage, or LocalStorage.
R2. Change the game's record to use the proper (meaning best design) of Cookies, SessionStorage, or LocalStorage.
R3. After a game has completed, pass history data back out to the native Activity before the WebView Activity is destroyed (see Activity 2). Have the native activity display a message saying "In the last game, <name> won after <XXX> rounds by guessing <suspect> in the <room> with a <weapon>", where <name> is the previous player's name, <XXX> is the number of guesses, and <room>, <suspect>, and <weapon> are self-explanatory. All of these values must be passed from the Javascript back to the native side. Note that this shows in the native Activity from Activity 2 below the textbox and Submit button after clicking "Continue" (Activity 2 R4).

Extra Credit (30 points):

1. (15 points) The app does not look all that great. Add responsive CSS to make the game display in an aesthetically pleasing way, formatted for various sized devices. You may use "off-the-shelf" CSS for this (CSS you did not write) but you 1) MUST package it locally in the app (no external CSS loading), and 2) it cannot be used to implement the show/hide functionality, or replace any other form of DOM manipulation from the code I gave you.
2. (15 points) Make it so the WebView part of the app dynamically sets the array of suspects, rooms, and weapons via an AJAX/Fetch call to an API that you construct. You design the API and explain it in your README.txt. Make the necessary changes to the clue.html file to initialize these values from the API.

Submission:

1. ***You may do this app by yourself or with a single partner***. If you do it with a partner YOU MUST ADHERE TO THE PARTNER REQUIREMENTS STATED BELOW! If you do not you risk receiving ½ your score!
2. Name your submission <asurite>_[<asurite2>_]labClue.zip. In the submission file should be 3 distinct Android projects, named activity1.zip, activity2.zip, and activity3. Zip. Yes, make sure you make 3 distinct Android Studio project copies from the root of your Android project directory. RUN A GRADLE CLEAN BEFORE MAKING YOUR ZIPFILE, OTHERWISE YOUR SOLUTION RISKS REJECTION BY BLACKBOARD!
3. Provide a README.txt in the root directory of the zipfile. In addition to what you normally indicate in this file, also explain your design approaches to 1) Activity 1 where it says to decide what features to enable on the WebView – indicate what you enabled and why; 2) Activity 2 R5 is a design decision, explain your approach and justify it; and 3) explain your storage design for Activity 3 requirements 1 and 2.
4. Your app must work on Android API level 23 except API. Please test appropriately.

Partnering Requirements:

1. You must use a single private Github repository for the two of you, and add ser421asu as a collaborator.
2. Your work, evidenced by Github commits and comments, and in how it reads, must demonstrate that you and your partner had a true intellectual collaboration. This form of "pair programming" does NOT necessitate synchronous coding activities, but does necessitate extensive design collaboration, code reviews, and conversations that result in code that has a cohesive, "looks as if it was written by one person" representation.
3. Choose partners at your own risk. I will not get involved with "I am unhappy with my partner" grade appeals.
4. These constraints are inflexible – if they are not adhered I will not accept the lab in its entirety.

Background: The game of Clue

Clue is a popular board game where guests are invited to a dinner party, only to have one guest murdered by a secret guest, in a secret room, with a secret weapon. It is also the subject of an awful movie (www.imdb.com/title/tt0088930) from 1985. The real game of Clue is a multiplayer game (en.wikipedia.org/wiki/Cluedo). The environment consists of 6 guests and a butler, 8 rooms, and 6 weapons. There are assorted rules for moving around the board and guessing suspects and circumstances in a process of elimination until a player declares "whodunit", in what room, and with what weapon.

You are given a drastically simplified version of the game with two players, you and the computer. The suspects, weapons, and rooms in your environment are pre-set as separate global arrays. The human player and the computer are given an equal number of cards with 3 cards held in reserve as the triplet with the secret to guess. The player and the computer take turns guessing triples until one arrives at the solution (matches the secret).

Functional requirements:

1. Display the full set of suspects, rooms, and weapons at the top of an HTML page for the user to see.
2. Underneath that, provide a textbox prompting for the user's name with a Submit button. When the user's enters her/his name, replace the form with a dynamic message saying "Welcome <name>".
3. Display the set of "cards" the human user "holds in her/his hand".
4. Display an HTML form that allows you to select 1 suspect, 1 weapon, and 1 room. I suggest you use a choice box for this but in fact it does not matter what you use as long as it is an HTML form.
5. The submit button for your form should cause Javascript to read the form values, and compare them to the secret.
   a. If the guess matches the secret, dynamically display a "win" message
   b. If the guess does not match the secret, dynamically display a message stating 1) the guess did not match the secret, and 2) Reveal ONE AND ONLY ONE incorrect component of your guess.
   c. Provide a "Continue" button that either resets the game to the beginning (if the user won), or allows the Computer to move (next step).
6. When the user's (incorrect) move is complete, have the Computer make a guess. A simple random guess will do, though the program should know not to guess cards that it itself holds. Like #4, display a message indicating whether the Computer's guess was correct or not, and a Continue button at the end. CAVEAT: The distinction here is that you are told the Computer's guess, but should not be told the incorrect component of the guess, which is different than what happens for the user (5.b.2).
7. Provide a button (outside the form) named "Show History" that dynamically shows the guesses you and the Computer have made so far in the current game. When shown, the "Show History" button should become "Hide History" (you are essentially creating a toggle button).
8. Provide a button named "Show Record" that when pressed, dynamically displays the won-loss record for the Computer, and a history of who the Computer played, the date, and the outcome.

Nonfunctional Requirements:

1. Do not use alerts. "Dynamic display" in the functional requirements means you are to dynamically manipulate the DOM to substitute new content as needed.
2. Do not hardcode a set number of suspects, rooms, or weapons. This data will be initialized as global variable arrays named, aptly enough, *suspects*, *weapons*, and *rooms*.
3. Functional requirements #7 and #8 should be implemented by hiding the state (data) in the DOM. Do not use other techniques like Cookies, SQLite, or Session/Local storage.