Start Application Workflow –-------------------------------------------

This is the code that gets run when the app starts, before any user input.

* Initialize Firebase, picking appropriate db environment
* Set player status to logged off and render game.

Register For Game Workflow –-------------------------------------------

* Get Game status from db
* If game status is completed, reject Registration request, else continue
* Build a random 8 digit number to use as the id
* Check if id is already in use, if it is, reject Registration and force user to click Register again. Else continue. \*\*\* Should fix this to generate the IDs in a loop until a unique one is found \*\*\*
* Create the player in the db with the generated id
* Render game with status Registered.

Login Workflow –-------------------------------------------

* Read screen data, player id
* Create reference to player doc in players database (May or may not exist at this pont)
* Retrieve player data with .get
* If doc exists
  + Update screen with player data
  + Set global vars – this player data
  + Create listener on my player object in db (Subscribe) with onSnapShot
  + Check my status:
    - Active
      * Get a reference to my link in the chain
      * Get a reference to my target’s player record
        + Save targets name, target’s picture link,
        + Get a reference to my target’s picture
        + Get picture
      * Create listener on my link object in db (Subscribe)
    - Not Active – Just render game
  + Get and display game data
  + Create listener on game data
* Else Login error
  + Display error

// ----- 3 Listeners (Subscribe to data changes) ----------------

Player document listener onSnapShot Workflow (Subscribe)

* If doc exists
  + Check if basic data changed
    - Owed – updated global variable and screen
  + Status change logic
    - Game Over – If my local status is set to Game Over, then update screen and exit. This is to handle an extra call to the listener, with Active status from DB.
    - Assassinated – Current (Active) to new status (Inactive)
    - Activated – From Waiting to Active
    - Registered – Only care the first time this is set, ignore afterwards
    - Waiting to Waiting - ? – Put in code for a reason, but didn’t document well enough. Not sure when or why this is called. Candidate to remove.
    - Just Waiting – Could have been added by admin or approved by admin.
* Else – No doc – Player was removed from game
  + Clean out data
  + Log out and return to login screen

Chain link document listener onSnapShot Workflow (Subscribe)

* Does doc exist?
  + Yes – creation or change to link
    - Check if Paused scenario – my target is me?
      * Set game status to paused in db – Let the Game Data listener handle the rest of this.
    - Create reference to my target’s player doc
    - Update screen data
    - Create reference to my target’s picture
    - Retrieve picture
  + No – link was deleted

Game Data document listener onSnapShot Workflow (Subscribe)

* Update screen data with game status
* If volunteer needed = true, make volunteer button visible, update screen message
* Game Status check
  + Paused
    - If I’m active, move myself to waiting queue
    - Update my status in the db
    - Set waiting queue to only this player. Otherwise, wouldn’t be paused status if any other players exist.
  + Game Not Started – unlikely status, resetting game?
    - Log off user
  + Completed – update screen. Still allow user to log in and out and view bounties, but nothing else.

Confirm Assassination Workflow

* Read screen data to get your target’s target’s name (name1)
* Retrieve my link from the chain – db call
* Retrieve my target’s id from link
* Retrieve my target’s link from the chain – db call
* Retrieve my target’s target from the link
* Retrieve my target’s target’s name from the players db (name2) – db call
* Compare name1 vs. name 2, if match, confirm assassination
  + Increment owed in db
  + Check the waiting queue for players
  + If no waiting queue
    - Check if I’m the only player remaining.
      * If yes,
        + Write game status to paused in db – let the subscriber handle the rest of the paused updates
      * If no,
        + Assign my target to my target’s target
  + If waiting queue
    - Shuffle queue
    - Assign my target to the first person in the shuffled queue
    - Loop through queue assigning i to i+1. Stop 1 iteration short.
    - Set each player status to active along the way
    - Outside loop, set target of last player in queue to my original target’s target.
    - Delete queue.
  + Delete the link for the assassinated target
  + Set assassinated player’s status to inactive

Re-Buy Workflow

* Get a reference to my Player object
* If my status is inactive and I have 1 or more bounties
  + Update my status record to Waiting
  + Decrement my bounties owed in the DB
* Else reject buy back in request
* Add player to Waiting queue
* Render game with status = Waiting

Take Break Workflow

* Get a reference to my Player object
* If my status is active or waiting
  + Update my status record to On Break and set break timestamp
  + **Update Chain to bypass me**: (re-used in quit game workflow)
    - Get reference to my link in the chain
    - Store my target’s id
    - Delete my chain link
    - Determine which player has me as their target:
      * Query the chain collection for links where target = my id. There should only be one matching object.
      * Save the id of my assassin.
      * If my saved target id = id of my assassin, we had each other and this is a paused scenario.
        + Set game status in db to paused
      * Else – not paused scenario
        + Get the link in the chain for my assassin (The person that had me as their target when I went on break)
        + Update Chain:

If waiting queue

Shuffle queue

Assign my assassin’s target to the first person in the shuffled queue

Loop through queue assigning i to i+1. Stop 1 iteration short.

Set each player status to active along the way

Outside loop, set target of last player in queue to my original target.

Delete queue.

Else, nobody in waiting queue

Set my assassins target to my original target id

* + Render screen with status = on break
* Else reject buy back in request

Return From Break Workflow

* Get a reference to my Player object
* If my status is on break
  + Check if I’ve waited long enough. If OK to return:
    - Set player status to waiting
    - Add player to queue
    - Render game, status = waiting
  + Else – alert to player its too early to return
* Else – not on break

Upload Picture Workflow

* Retrieve file data from the screen
* If the number of files selected is 1 then proceed
  + Get a reference to my Player object
  + Update player picture name field
  + Create the full path variable using the players id.
  + Upload the file to cloud storage
  + Monitor the upload and log the status
* Else – report error that only 1 file can/needs to be selected

View My Picture Workflow

* Create the full path variable for the file download using the players id
* Attempt to download picture file. If success:
  + Update picture on screen
* Else failure or missing picture – Display correct error.

Log Out Workflow

* Blank out screen data
* Unsubscribe from player record listener
* Unsubscribe from game data listener
* If the player was active, unsubscribe from chain link listener
* Clear out global vars
* Render game with status = logged off

Quit Game Workflow

* Get a reference to my Player object
* Switch on my status:
  + Active
    - See the same steps for the “Update Chain to bypass me” step in Take a Break above.
  + Waiting
    - Get the waiting queue
    - Create a temp queue with all players except me
    - Update Waiting queue in the db using the temp queue.
  + All other status – No action required
* Delete player document

Volunteer Button Game Workflow

* If button is visible, still searching for a volunteer. May need error checking here.
* Flip volunteer needed field in db to false
* Process Volunteer – Similar to bypass me in the chain function, but I couldn’t get it just right so I duplicated it and make a few tweaks for this scenario.
  + Get reference to my link in the chain
  + Store my target’s id
  + Determine which player has me as their target:
    - Query the chain collection for links where target = my id. There should only be one matching object.
    - Save the id of my assassin.
    - If my saved target id = id of my assassin, we had each other and this is a paused scenario.
      * Set game status in db to paused
    - Else – not paused scenario
      * Get the link in the chain for my assassin (The person that had me as their target when I went on break)
      * Update Chain:
        + Check Schedule queue first. If queue exists with players count > 0

Save players queue to temp array

If waiting queue exists with players count > 0

Push all waiting players onto temp queue

Shuffle temp queue

Assign my assassin’s target to the first person in the shuffled temp queue

Loop through temp queue assigning i to i+1. Stop 1 iteration short.

Set each player status to active along the way

Outside loop, set target of last player in temp queue to my original target.

Delete scheduled and waiting queue.

Delete my link in the chain.

* Update player status to inactive and increase bounty by 1