//Function to initialize the database, must be called immediately and only once

initializeDatabase();

**All following functions are Future asynchronous and will need to be called with ‘await’:**

//Function for creating an Admin user, only able to be accessed by admins

createAdminUser(*String username, String password, int uin, String firstName, String lastName, String email, int phoneNumber*);

* If the username already exists return ‘Invalid Username’
* If the email is already in use it returns ‘Invalid Email’
* If the UIN is already in use it returns ‘Invalid UIN’
* If the creation works it returns ‘Created’

//Function for creating a Student user, available on the create user page

createStudentUser(*String username, String password, int uin, String firstName, String lastName, String email, int phoneNumber, int teamNumber, int courseNumber*);

* If the username already exists return ‘Invalid Username’
* If the email is already in use it returns ‘Invalid Email’
* If the UIN is already in use it returns ‘Invalid UIN’
* If the creation works it returns ‘Created’

//Function for adding a new equipment item to the inventory, Admin only

newEquipment(*String name, int totalAmount, String category, String storageLocation*);

* If the equipment already exists it returns ‘Creation Failed’
* If the totalAmount is lower than 0 it returns ‘Invalid Amount’
* If the equipment is created it returns ‘Created’

//Function for adding a new individual student form to the system

newForm(*String formName, String user, String info*);

* If the user already has a form with this name it returns ‘User Already Has Form’
* If the form is created returns ‘Form Created’

//Function for Admins to verify form completion

verifyForm(*String user, String formName*);

* If a form with that name isn’t found to be associated with the user it returns ‘Not Verified’
* If the form is located and verified it returns ‘Verified’

//Function for adding form requirements for an equipment, admin only function

addFormRequirement(*String formName, String equipmentName*);

* If the form is already required returns ‘Form Already Required’
* If the form is added as a requirement it returns ‘Form Required’
* If the equipment isn’t found it returns ‘Invalid Equipment’

//Function for Student accounts to generate an equipment request

generateNewRequest(*String user, String equipment, int requestedAmount*);

* If the student does not have the forms required for the checkout it returns ‘Invalid Permissions’
* If the amount requested is more than what is available or negative it returns ‘Invalid Amount’
* If the equipment is not found it returns ‘Invalid Equipment’
* If the username isn’t found it returns ‘Invalid Username’
* If the request is generated it returns ‘Generated’

//Function to approve request, admin only

approveRequest(*String requestID, int studentUIN, String adminUser*);

* If the approval is valid and goes through it returns ‘Approved’
* If the UIN does not match the request it returns ‘Invalid UIN’
* If the request is not found it returns ‘Invalid Request’

//Function to deny a request, admin only

denyRequest(*String requestID*);

* If the deny goes through it returns ‘Denied’
* If the request isn’t found it returns ‘Invalid Request’

//Function to verify a check in of equipment

verifyCheckIn*(String historyID, int uin, int amountIn, String adminUser*);

* If the check-in is a success it returns ‘Checked In’
* If amountIn is negative or more than the checked out amount it returns ‘Invalid Amount’
* If the UIN is doesn’t match the check out user it returns ‘Invalid UIN’
* If the user isn’t found on the database it returns ‘Invalid Username’
* If the equipment isn’t found it returns ‘Invalid Equipment’
* If the history entry can’t be found it returns ‘Invalid Checkout History’

//Function to list the active requests that a student user has created

findRequests(*String user*);

* If no requests are found for the user it returns ‘No Requests’
* If any requests are found it returns a list of strings of the request IDs found

//Function to Authenticate login credentials

authenticate(*String username, String password*);

* If the username does not exist, returns the string ‘Invalid Username’
* If the password does not match an existing account username it returns ‘Invalid Password’
* If the admin status check fails it returns ‘Failed Login’
* If the username and password match a student account it returns ‘Valid Student’
* If the username and password match an admin account it returns ‘Valid Admin’

//Function to get the login information for a forgot password

forgotPassword(*String email*);

* If the email isn’t found as having an associated account, it will return ‘Invalid Email’
* If the email is found to be associated with an account it will return a list of strings with the format [username, password]

//Function to return a users information

getUserInfo(*String username*);

* If the username isn’t found in the database it returns ‘Invalid Username’
* If the username is found, it will return a different list of strings depending on the admin status of the username.
  + For an Admin user it will output a list of strings in the following format [username, uin, firstName, lastName, email, phoneNumber]
  + If the user is a Student the format is as follows [username, uin, firstName, lastName, email, phoneNumber, courseNumber, teamNumber, **forms**]
    - The format for forms is that all of the forms associated with the student will have their names appended together in a comma separated list and this list will be stored as a string that is considered to be a single list entry. The forms are included only for display so that they are aware of the name of the forms associated with the student as in the backend I have a function that searches formIDs based on student usernames

//Function to return equipment information

getEquipmentInfo(*String equipment*);

* If the passed equipment name isn’t found in the database it returns ‘Invalid Equipment’
* If the equipment is found the function returns a list of strings in the following format [equipmentName, category, storageLocation, amountAvailable, totalAmount, **forms**]
  + Like the previous function the forms section is a single string of every form that is associated with the equipment by name in a comma separated list

//Function to return request information

getRequestInfo(*String requestID*);

* If the request ID isn’t found in the database it will return ‘Invalid Request’
* If the request is found it will return a list of strings with the format [requestID, username, equipmentName, requestedAmount, timestamp]

//Function to find formIDs by username

formIDbyUser(*String username*);

* If no forms are found associated with the username it returns ‘User Has No Forms’
* If the user has forms associated that are found, it returns a list of strings of form IDs

//Function to find formIDs by equipment name

formIDbyEquipment(*String equipment*);

* If the equipment isn’t found to have any associated forms it returns ‘Equipment Has No Forms’
* If the equipment is found to have associated forms it returns a list of strings of form IDs

//Function to find formIDs by form name

formIDbyName(*String formName*);

* If no forms are found with this name it returns ‘No Forms With This Name’
* If forms are found with the given name it returns a list of strings of form IDs

//Function to return form information

getFormInfo(*String formID*);

* If the form ID is not found in the database it returns ‘Invalid Form’
* If the ID is found, it returns a list of strings in the following format [formID, formName, username, formInfo, verificationStatus]

//Function to return checkout history for a specific user

historyIDbyUser(*String username*);

* If the username isn’t associated with any checkouts it returns ‘No Checkout History’
* If the username is found to be associated with any checkouts it returns a list of strings of all of the checkout history IDs.

//Function to return checkout history for a specific equipment

historyIDbyEquipment(*String equipment*);

* If the equipment isn’t associated with any checkouts it returns ‘No Checkout History’
* If the equipment is found to be associated with any checkouts it returns a list of strings of all of the checkout history IDs.

//Function to return the details of a specific checkout history

getHistoryInfo(*String historyID*);

* If the history ID isn’t found on the database it returns ‘Invalid History ID’
* If the history ID is found and **none** of the checked out equipment has been checked in it returns a list of strings of the format [historyID, username, equipmentName, amountCheckedout, adminThatAprovedCheckout, timestampOut]
* If the history ID is found and **some** of the checked out equipment has been checked in it returns a list of strings of the format [historyID, username, equipmentName, amountCheckedout, adminThatAprovedCheckout, timestampOut, amountCheckedIn, adminThatAprovedMostRecentCheckIn, MostRecentCheckInTimestamp]

//Function to return all student usernames

allStudentUsers();

* If no student users are found on the database it returns ‘No Student Users’
* If any student users are on the database it returns a list of strings of usernames for all student users

//Function to return all Admin usernames

allAdminUsers();

* If no admin users are found on the database it returns ‘No Admin Users’
* If any admin users are on the database it returns a list of strings of usernames for all admin users

//Function to return all equipment names

allEquipment();

* If no equipment is found on the database it returns ‘No Equipment’
* If any equipment is on the database it returns a list of strings of equipment names for all equipment in the database

//Function to return all active request ids

allRequestIDs();

* If no requests are found on the database it returns ‘No Requests’
* If any requests are found it returns all request IDs on the database

//Function to return all form IDs

allFormIDs();

* If no forms are found on the database it returns ‘No Forms’
* If any forms are on the database it returns a list of strings of form IDs for all forms on the database

//Function to return all checkout history IDs

allHistoryIDs();

* If no checkout history is on the database it returns ‘No Checkout History’
* If any history is found it returns a list of strings of all of the checkout history IDs on the database

//Function to edit a user account

editUserAccount(*String username, int uin, String firstName, String lastName, String email, int phoneNumber, int teamNumber, int courseNumber*);

* If another account is using the same email it returns ‘Invalid Email’
* If any other account is using the uin it returns ‘Invalid UIN’
* If the username does not exist it returns ‘Invalid Username’
* If it updates it returns ‘Updated’

//Function for editing an equipment item in the inventory, Admin only

editEquipment(*String name, int totalAmount, int amountAvailable, String category, String storageLocation*);

* If the equipment isn’t found it returns ‘Invalid Equipment’
* If the input amounts cause errors it returns ‘Invalid Amount’
* If it works it returns ‘Updated’

//Function for editing a form, Admin only

editForm(*String formID, String formName, String info, bool verification*);

* If the associated user can’t be found it returns ‘Invalid User’
* If the Form ID isn’t found it returns ‘Invalid Form ID’
* If it works it returns ‘Updated’

//Function for editing a checkout history, Admin only

editHistory(*String historyID, int amountOut, String adminOut, String timestampOut, int amountIn, String adminIn, String timestampIn,*);

* If the in or out amounts cause errors ‘Invalid Amounts’
* If the historyID isn’t found it returns ‘Invalid History ID’
* If it works it returns ‘Updated’

//Function for removing a user, Admin only

removeUser(*String user*);

* If the user is removed it returns ‘Removed’
* If the user isn’t found it returns ‘Invalid Username’

//Function for removing equipment, Admin only

removeEquipment(*String equipment*);

* If the equipment is removed it returns ‘Removed’
* If the equipment isn’t found it returns ‘Invalid Equipment’

//Function for remove a form, Admin only

removeForm(*String formID*);

* If the associated user can’t be found it returns ‘Invalid User’
* If the Form ID isn’t found it returns ‘Invalid Form ID’
* If it works it returns ‘Removed’’

//Function for removing equipment, Admin only

removeHistory(*String historyID*);

* If the history is removed it returns ‘Removed’
* If the history isn’t found it returns ‘Invalid History ID’

//Function for removing a form requirement from an item in the inventory, Admin only

removeFormRequirement(*String formName, String equipmentName*);

* If the equipment isn’t found it returns ‘Invalid Equipment’
* If the form isn’t found in the equipment it returns ‘Form Not Required’
* If it works it returns ‘Form Removed’