• **Unit tests (25 points):** Include a file/directory named ‘Testing’ in your Git Repository. There should be details (can be in a separate file in the directory) provided by each team member about the module and the functional testing they have done. Each team member picks a module or module and lists the equivalence classes and the test cases selected to cover all equivalence classes.

Enes:

- ActionBar item sorting (from RecipeScreenComponent)
    - Sort by number of valid ingredients
        - Description: Recipes are sorted by how many of their total required ingredients are in the user’s ingredient list.
        - Test case: List is in percentage order. Press sort button. List changes to percentage order.
    - Sort by percentage of valid ingredients
        - Description: Recipes are sorted by percentage of ingredients that the user has in their list to the total number ingredients required by the recipe.
        - Test case: List is in number order. Press sort button. List changes to number order.
- FireBase add ingredient from database (from DataBase)
    - Return a user’s ingredients list from Firebase database
        - Test case: User is not yet logged in. Log in to a user’s account. If the user has an ingredients list, it will appear in ingredients screen, otherwise ingredients screen will be empty.
- FireBase add ingredient to database (from DataBase)
    - Send a user’s ingredients list update to Firebase database
        - Test case: User is logged in. Add or update an ingredient. Log out, then log in again. The update should be visible and reflected in the ingredients list.

Danilo’s modules:

- IngredientInfo Item Property Values (from IngredientScreenComponent)
    - Equivalence classes:
    - Value for any ingredient property that is numeric (like protein, carbs, etc) = negative
        - Test cases: set property value to 0, and press the “-” button. The button keeps the value at 0, does not let it become negative
    - Value for any ingredient property that is numeric (like protein, carbs, etc) = 0

        - Test case: set property value to positive, press the "-" button. The button will decrease the value to 0 if pressed enough times, does not let the value go below 0.
        - Test case: press value in order to change it to a specific value. If 0 is entered, 0 will show up and update the value of the property for the item.
    - Value for any ingredient property that is numeric (like protein, carbs, etc) = positive
        - Test case: set property value = 0 manually, press "+" button. The button will increase the value by 1.
        - Test case: the property value is 0 by default when the ingredient is added/initialized. Press "+" button. The button will increase the value by 1.
        - Test case: property value will overflow if too large
- List Quantity (from IngredientScreenComponent)
    - Equivalence classes:
    - Increase/decrease item quantity
        - Test case: Set property value to positive, press the "-" button. The button will decrease the value to 0 if pressed enough times.
        - Test case: Press value in order to change it to a specific positive value.
        - Test case: Set property value to any positive value. Pressing the "-" button will decrease the value by 1, pressing the "+" button will increase the value by 1.
        - Test case: property value will overflow if too large
    - Delete item
        - Test case: The user sets the item quantity to 0. The user decreases the item quantity by pressing the "-" button. A red "X" shows up in place of the "-". The user presses the "X". The ingredient is removed from the inventory list.
        - Test case: The user sets the item quantity to 0. The user decreases the item quantity by pressing the "-" button. A red "X" shows up. The user does not press the "X" but instead increases the quantity again, as the user changed their mind. The red "X" disappears and the "-" is in its place. The quantity is back to 1.
        - The user edits the item name and completely clears it. The item disappears from the inventory list.
- List Info button (from IngredientScreenComponent)
    - Press info button

        - Test case: The user presses the info button for one ingredient. The ingredient property information is shown. The user presses the button for the same ingredient again. It closes the ingredient property information.
        - Test case: The user presses the info button for one ingredient. The ingredient property information is shown. The user presses the info button for a different ingredient. This closes the ingredient property information for the current ingredient and opens it for the ingredient just pressed.
- ActionBar item adding/searching (from IngredientScreenComponent)
    - Equivalence classes:
    - Add empty item
        - Test case: User presses text input area to add a new ingredient. No characters are entered, only "return" is pressed. No ingredient is added to inventory and the keyboard disappears.
    - Add item already in inventory
        - Test case: User presses text input area to add a new ingredient. The text entered matches the name of an ingredient already in inventory. The ingredient is listed below the text input area and can be interacted with as normal. If the user presses the "+" icon, the item's quantity is incremented by 1, the screen goes back to the full inventory list. If the user clears the text input area, the screen goes back to the full inventory list.
    - Add item not already in inventory
        - Test case: User presses text input area to add a new ingredient. The text entered does not match the name of an ingredient already in inventory. The user presses the "+" icon, and so the screen goes back to the full inventory list, which now includes the new item with quantity set to 1 but all other ingredient properties set to 0 or "none set".
        - Test case: User presses text input area to add a new ingredient. The text entered does not match the name of an ingredient already in inventory. If the user clears the text input area, the screen goes back to the full inventory list.
- ActionBar camera (from IngredientScreenComponent)
    - Open camera/close camera
        - Test case: Camera is closed. Press camera button. The camera never opened before, so the app asks if the user wants to give the app camera permissions. User does not give permissions. Line of text pops up that says "no access to camera". User can press the camera button again, and that line of text disappears.

Test case: Camera is closed. Press camera button. User already did not grant the app camera permissions. Line of text pops up that says “no access to camera”. User can press the camera button again, and that line of text disappears.

        - Test case: Camera is closed. Press camera button. If camera never opened before, the app asks if the user wants to give the app camera permissions. User gives permissions. Camera opens.
        - Test case: Camera is closed. Press camera button. User already gave permissions. Camera opens. If camera button pressed again, camera closes.

- ActionBar item sorting (from IngredientScreenComponent)
    - Sort A-Z
        - Test case: List is in Z-A order. Press sort button. List changes to A-Z order.
    - Sort Z-A
        - Test case: List is in A-Z order. Press sort button. List changes to Z-A order.

Lance:

- JSON file database
    - Test case: user can upload JSON file to database successfully. The JSON file format can be used in firebase. The JSON file can be imported into database successfully.
- Recipe file database
    - Test case: user can load Recipes from database in the correct format. The app can output the recipe data from database with the format we set. There is no error caused by no data under a specific key in database.
- User database
    - Test case: user can load their data of ingredients in the database. The app can output the data from database under the user directory. The data can be changed by adding ingredients, changing quantity and updating the information of ingredients.
- Log in/Sign up authentication in firebase
    - Test case: user can sign up and log in by using log in and sign up button. The database can create the correct user directory by using the user id. The firebase can arrange the user by deleting and adding users. User can use email address and password to sign up for a new user in firebase.
- Multiple devices log in

    - Test case: user can have multiple devices to login in the same account at the same time. The data between devices can be changed in real time. There is no error, when two devices both edited the ingredients.

Simon:

- Barcodes JSON file
    - Read a CSV file containing barcodes (both upc12 and upc14 formats).
    - Take each row of the CSV and convert it to an object containing the name of the product, the brand, and both barcode formats.
    - Output each formatted object into a JSON file for use in Firebase. Since indexing by a barcode leads to faster lookups, 2 JSON files are created: one indexed by the upc12 value, and the other by the upc14 value.
- Recipe Scraping
    - Access the base url we are using for scraping (basically.com/simply)
    - Run a Scrapy Spider on said page, which extracts the href values of each recipe link on the page. The recipe links are identified by a CSS class, so we make sure to only grab those, not just any page link.
    - The spider clicks the next button and runs the same script on the next page. This repeats until there are no more pages.
    - Collect all the urls into a JSON file for output
    - A separate script file, recipes_to_json.py, reads in the urls from the JSON file and accesses them using lxml and requests. lxml is used to quickly read the HTML from the page that is retrieved with requests, and key characteristics of each recipe are extracted, such as a name, steps, ingredients, and equipment needed for the recipe.
    - Each extracted portion requires formatting to make sure that it is properly Unicode formatted to display well in the app.
    - The ingredients take special attention, as they are parsed in order to be displayed for matching in the main app. This is done by breaking them into cases:
        - Has units and name. Example: “12 tbsp. Butter”. This is parsed with regex by checking if there is a match for an integer value before any sort of unit string. In this case, “12” is found before “tbsp”, which tells us that this is an ingredient with a set unit and name.
        - Has name, but no units. Example: “3 large onions”. If no match for a unit is found in the previous step, the string is split at every space and the first index is used as the number of the object. The object is considered the rest of the string.

        - If the string fails the previous two cases, then the string is taken alone and is considered an anomaly that can't be properly sorted. Something like "a pinch of salt" would be here.

Sean's modules:

- Loading Screen
    - Function: Prevents user from modifying local data before all of their firebase data is fetched
    - Function: Provides visual indication that their firebase data is currently being fetched for devices with slower internet speeds

    - Equivalence classes: New users, existing users with no data in the database, existing users with data in the database
        - Test case: A new user creates an account for the app. They do not currently have a directory in the database. The loading screen waits for the new directory to be made before letting the new user use the app.
        - Test case: An existing user logs in to the app. They do not currently have any data stored in the database. The loading screen waits to synchronize the app with the database, then lets the user use the app.
        - Test case: An existing user logs in to the app. They have some data already stored in the database. The loading screen waits to fetch this data before letting the user use the app.

- Fetching barcode data
    - Function: When the user scans a barcode, this attempts to search the barcode database for the corresponding item

    - Equivalence classes: Barcodes that have corresponding grocery items in the database, barcodes without corresponding grocery items in the database, invalid codes (ex. QR codes)
        - Test case: A user scans a barcode that does have a corresponding item in the database. They get a pop-up telling them the name that was found for the barcode they scanned, and a new item is added to their inventory with that name.
        - Test case: A user scans a barcode that does not have a corresponding item in the database. They get a pop-up telling them that the barcode isn't found in the database.
        - Test case: A user scans a code that is not a barcode (ex. a QR code) and they get a pop-up telling them that the barcode is invalid.

    - Tested on both Android and iOS devices. iOS devices add an extra digit to the beginning of the barcode data, and this is trimmed in the app.

- Editing Grocery Names
    - Function: Allows user to change the name of a grocery item in their inventory as they see fit.

    - Equivalence classes: Changing the name using alphanumeric characters, changing the name to another name that has non-alphanumeric characters, changing the name to be equal to another name in the list, changing the name to an empty string, changing the name to an absurdly long string.
        - Test case: An item name is set using alphanumeric characters. The name of the item changes successfully, with no side effects.
        - Test case: An item name is set using non-alphanumeric characters, such as symbols. The name of the item changes successfully, with no side effects.
        - Test case: An item name is set to the name of an existing item in the list. The current item is removed and the existing item's quantity is incremented.
        - Test case: An item name is set to an empty string. The current item is deleted.
        - Test case: An item name is set to an absurdly long string. The name of the item changes successfully, with no side effects, but is truncated. The rest of the name can be seen by scrolling.

- Adding groceries to list
    - Function: Allows user to add their own groceries to their inventory.

    - Equivalence classes: Adding a grocery using alphanumeric characters, adding a grocery that already exists in the list, adding a grocery with an absurdly long string.
        - Test case: An item is added using alphanumeric characters. The item is created and added to the database successfully, with no side effects.
        - Test case: An item is added using non-alphanumeric characters, such as symbols. The item is created and added to the database successfully, with no side effects.
        - Test case: The user attempts to add an item that already exists in the inventory. No new items are created and the existing item's quantity is incremented.

        - Test case: An item is added using an absurdly long string name. The item is created and added to the database successfully, with no side effects, but is truncated. The rest of the name can be seen by scrolling.

- Live reloading
    - Function: Propagates changes from a user's inventory to all devices logged in with the same user's information.

    - Equivalence classes: Sending a change to firebase, receiving a change from firebase
        - Test case: A user adds, edits or removes an item in their inventory. Other devices logged in to the same account and connected to the internet will be able to see this change.
        - Test case: The user will be able to see changes made from other devices logged into their account.

- Changing quantity in list
    - Function: Allows user to quickly and easily edit quantities in their inventory.
    - Editing the text
        - Inputting a positive numeric value
        - Inputting a negative numeric value
        - Inputting an extremely large value
        - Inputting invalid characters
    - Pressing the plus/minus buttons
        - Tested decrementing when the value is zero
        - Tested decrementing with the value is greater than zero
        - Tested incrementing an extremely large number

- Sort ingredients by name
    - Test case: List is sorted alphabetically. Press sort button. List is sorted in reverse (Z-A).
    - Test case: List is sorted in reverse alphabetical order. Press sort button. List is sorted alphabetically (A-Z).

- Searching for groceries in inventory
    - Test Case: Search for groceries in inventory. You get a list of your groceries whose first letters match your query.

- Searching for recipes by title

    - Test Case: Search for recipes in database. You get a list of your recipes whose title includes your query.

- Searching for ingredients in recipes
    - Test Case: Search for ingredients in recipes. The ingredients are matched with some false positives (ex. When searching for “salt”, recipes with “un*salt*ed butter” appear)

- Calculating amount of matching ingredients in the list and show them in the recipe information expansion
    - Test Case: Show matching ingredients from user’s current inventory. Current user changes inventory. Matching ingredients change accordingly.

- Switching Screens
    - Test Case: User switches from recipe screen to groceries screen. Any expanded recipe is collapsed.
    - Test Case: User switches from groceries screen to recipes screen. Any expanded grocery is collapsed, and the camera collapses.
    - Test Case: User switches from login screen to groceries screen. The app pauses to wait for the data to be fetched, then lets the user use the app.
    - Test Case: User switches from steps screen back to recipes screen. Progress starts from the beginning whenever the user opens the steps screen.

- Manual matching of ingredients
    - Tested validating the same item repeatedly
    - Tested excluding the same item repeatedly
    - Tested to see if the information would be erased after switching screens

- Steps Screen - incrementing/decrementing current step
    - Function: Allow the user to easily traverse through the steps of a recipe

    - Equivalence Classes: Decrementing when at the start of the steps, decrementing/incrementing when in the middle of steps, incrementing when at the end of the steps
        - Test Case: When a user decrements the current step at the beginning, no current step is set.
        - Test Case: When a user increments/decrements the current step in the middle of the list, the current step increments/decrements with no errors.

- Test Case: When a user increments the current step at the end, no current step is set, and all previous steps are faded out.