Detailed Design

Control Your Figures Recipe Nutrition Facts Web Application

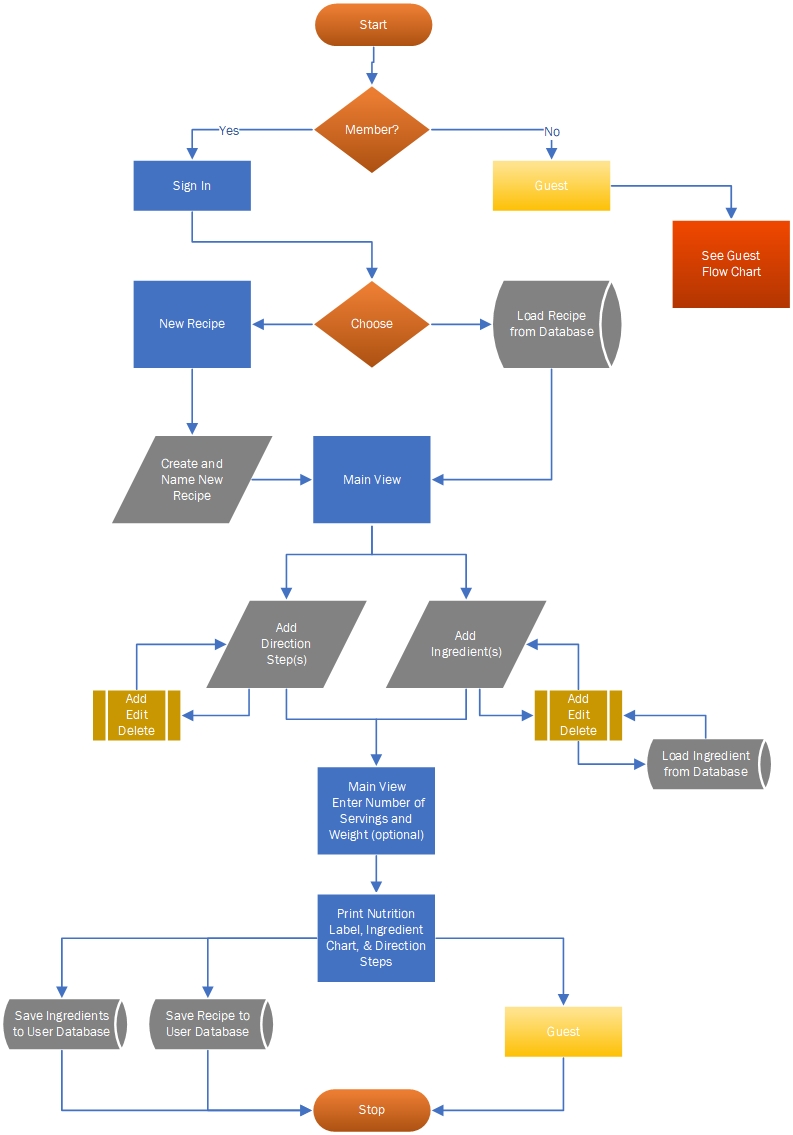
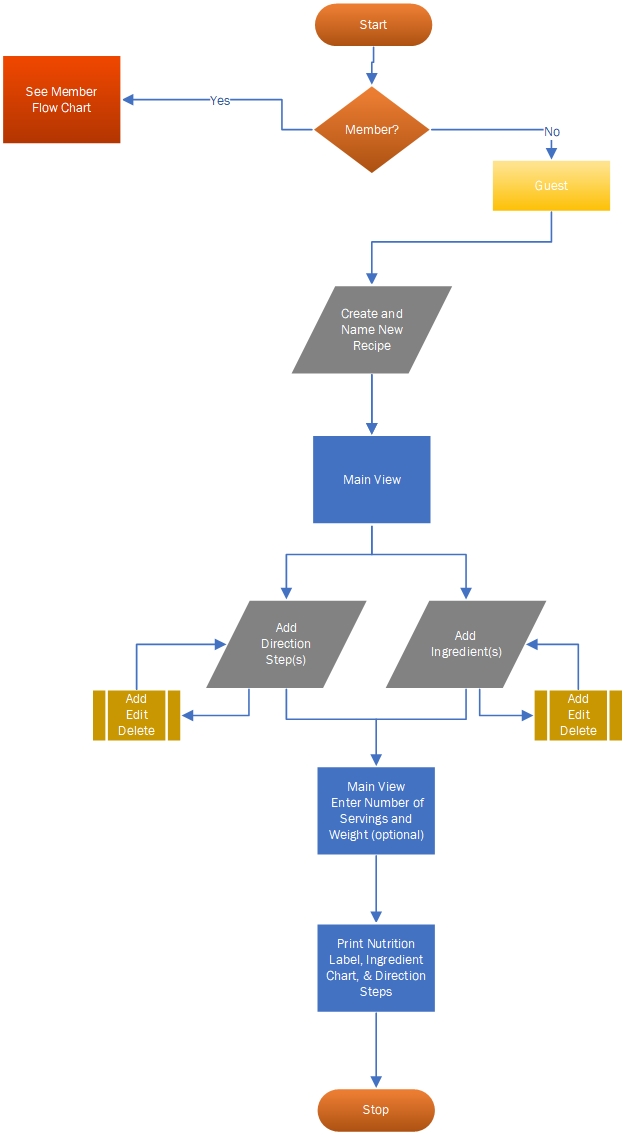
# Overview

Included in this document is: User Flowchart, Wireframe Diagram, UML Diagram, and Pseudocode.

# User Flowchart

## The Flow

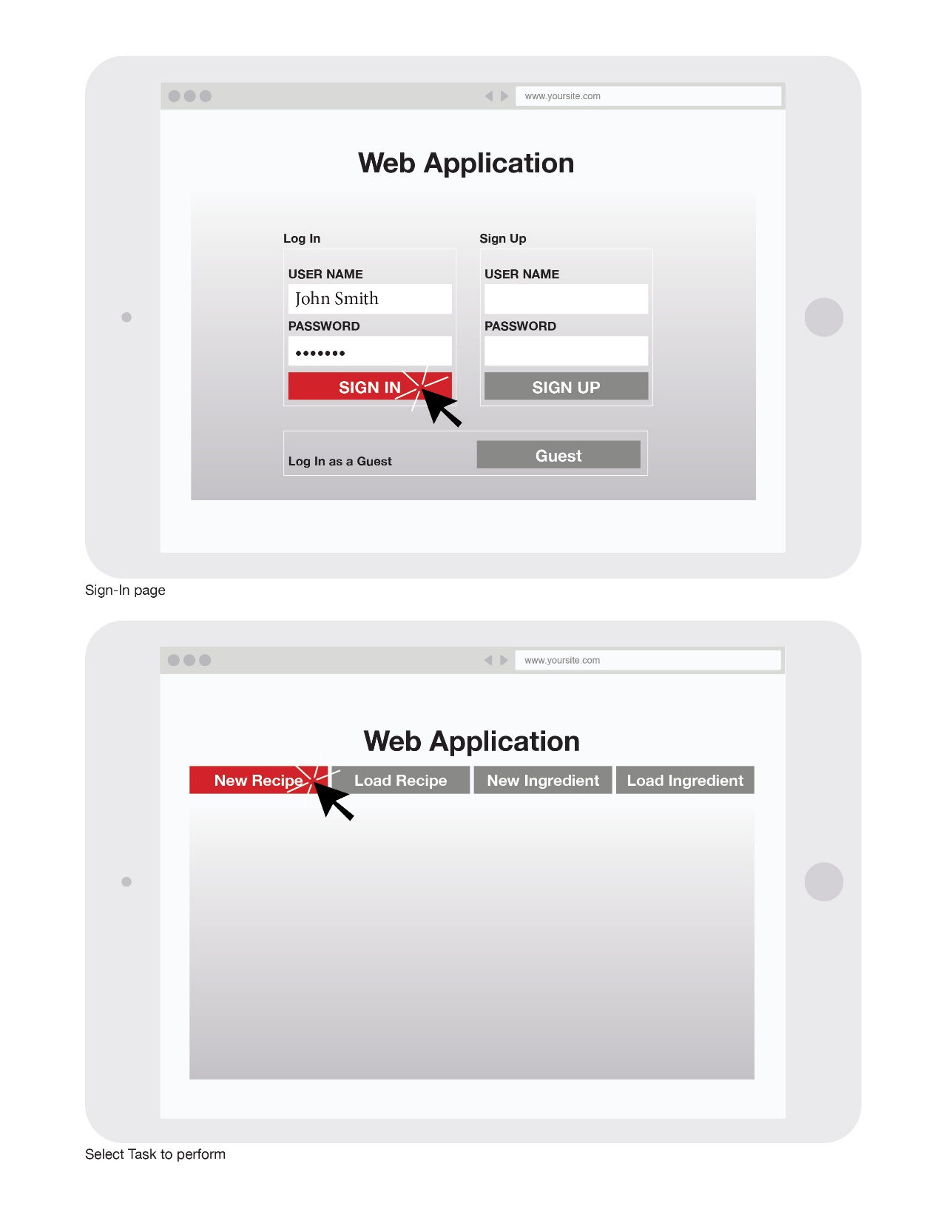
The Flowchart represents the perspective from two points of view; first the member is represented, then the guest is represented. The user does not have to have a membership to use it, but if they are a member, then the recipe entered can be saved to their database.

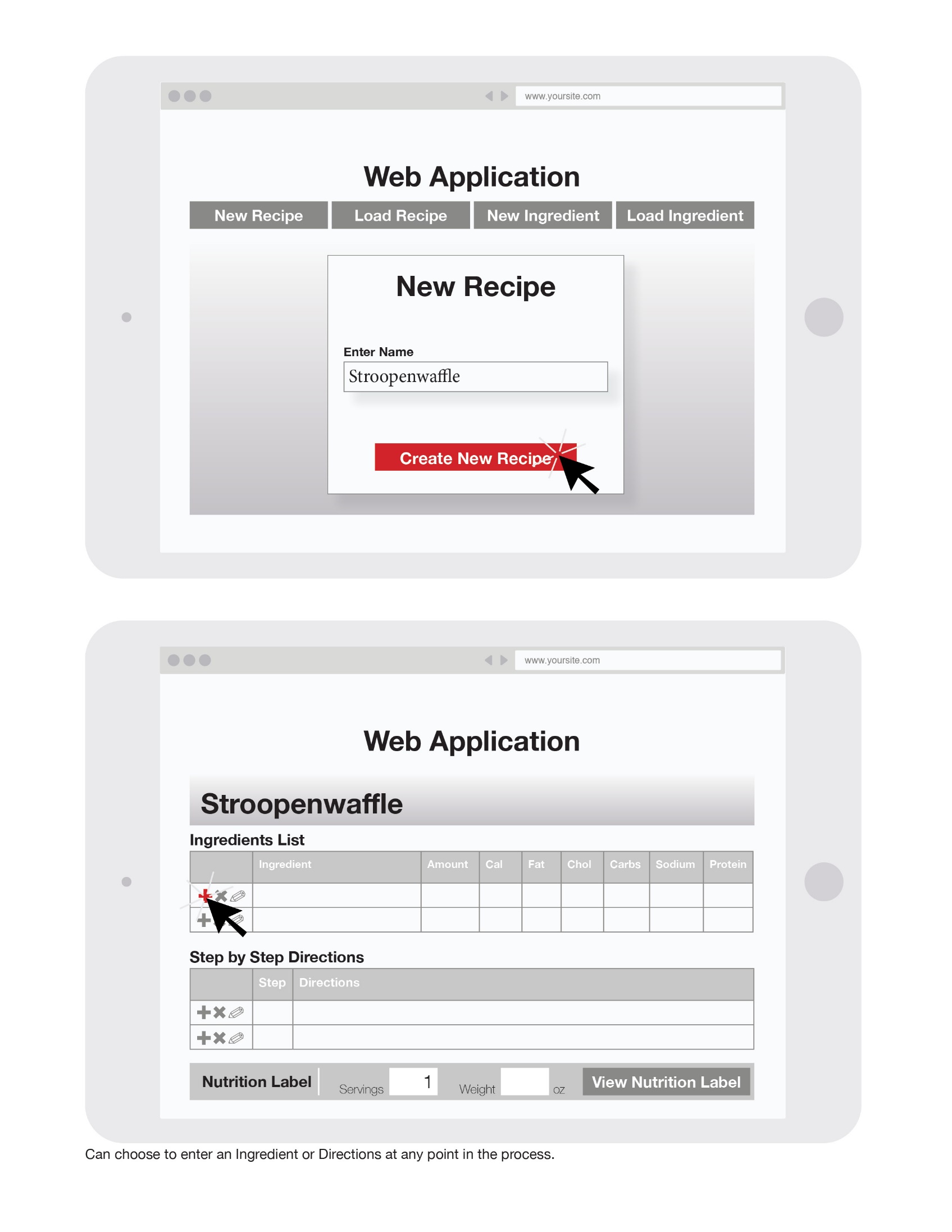
****

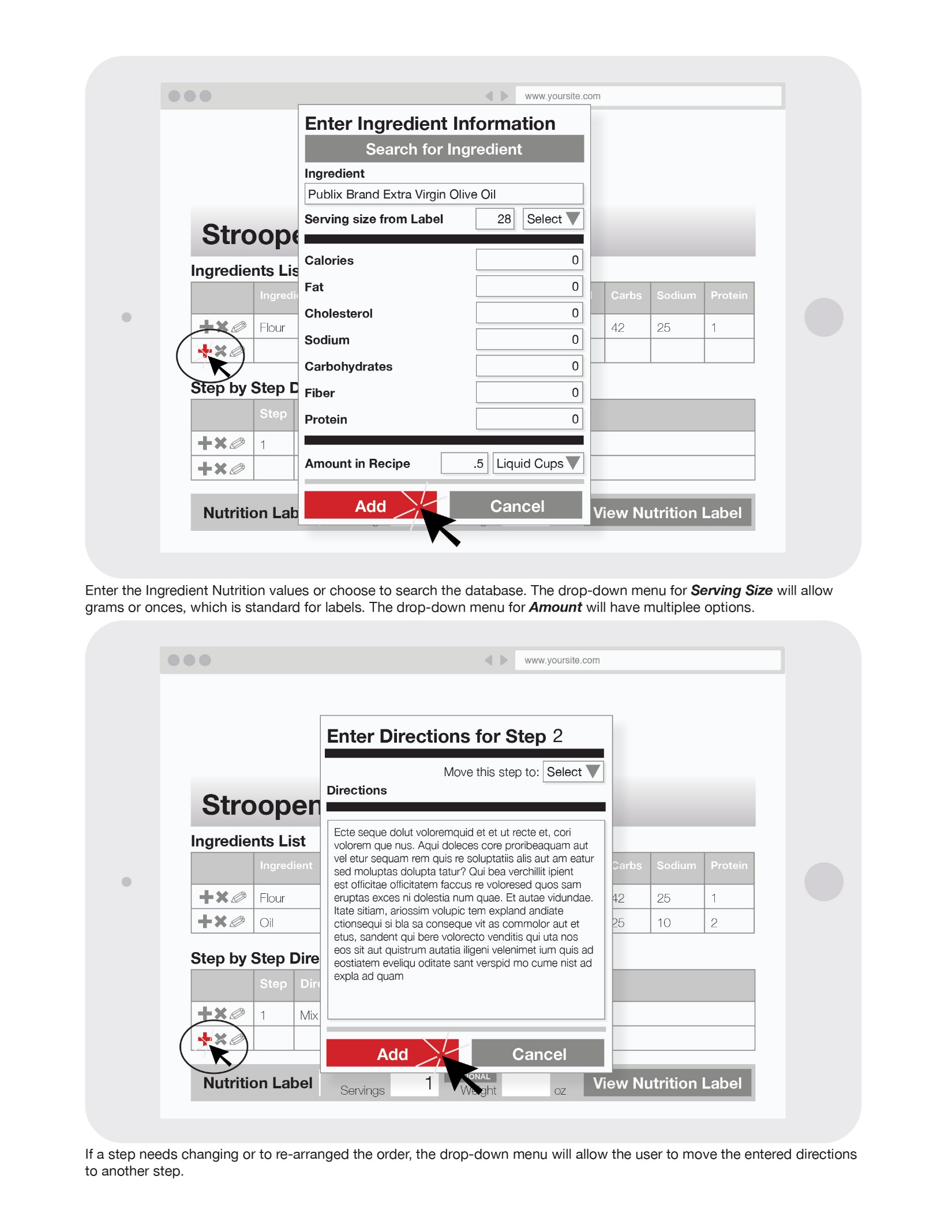
# Wireframe

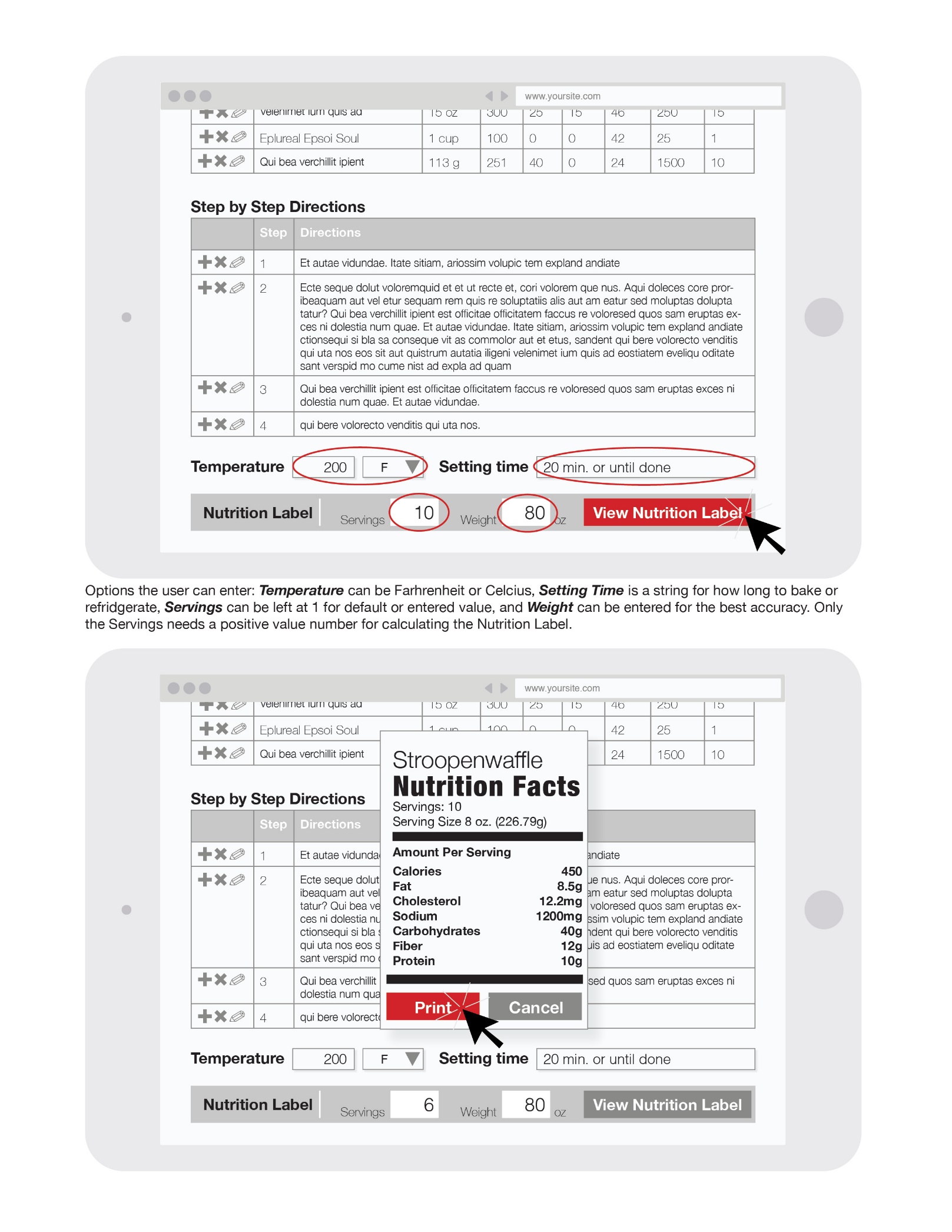
## Summary

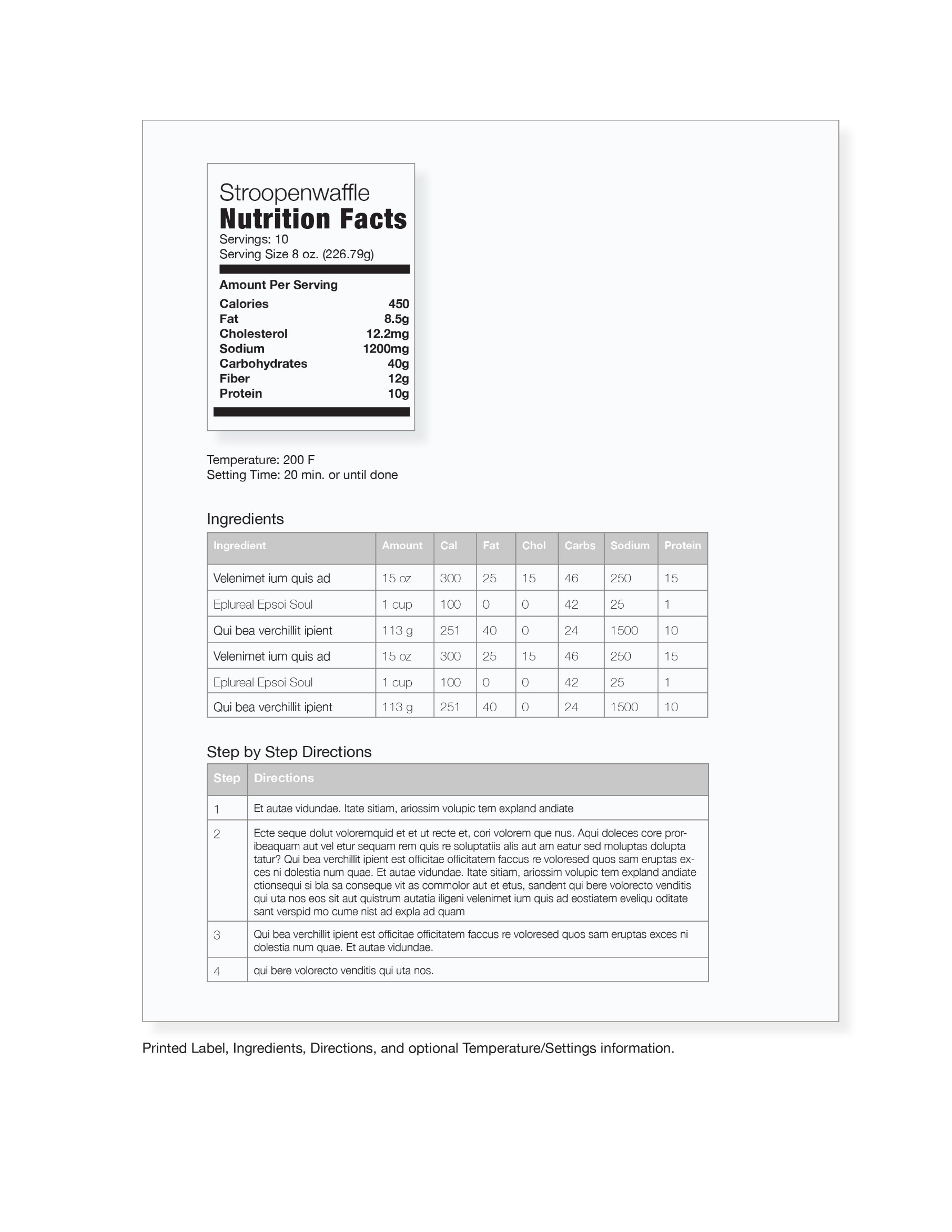
The wireframe demonstrates its’ intuitive nature with minimal directions. The Ingredients and Steps are editable, the values for each nutrient is displayed in a chart, and the user can load ingredients or previously saved recipes. The last image in the wireframe shows a printed sample.



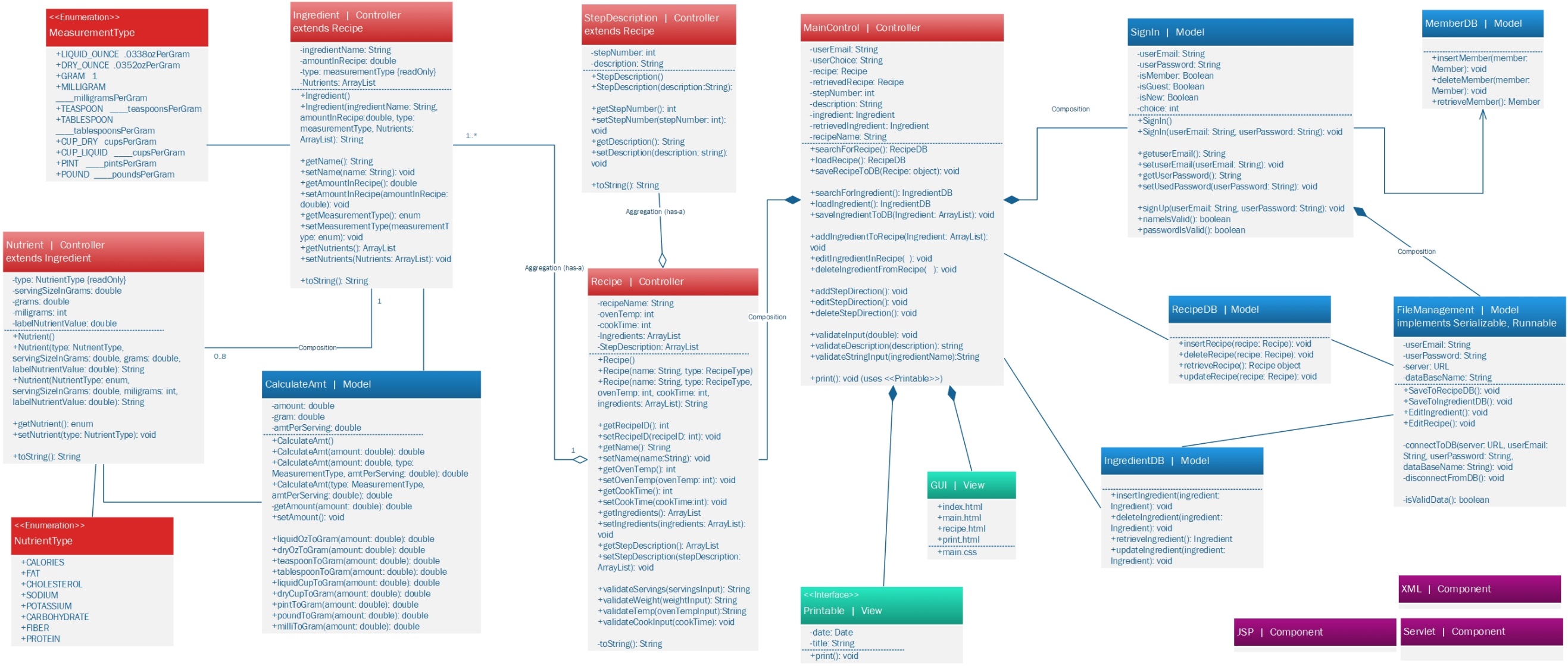


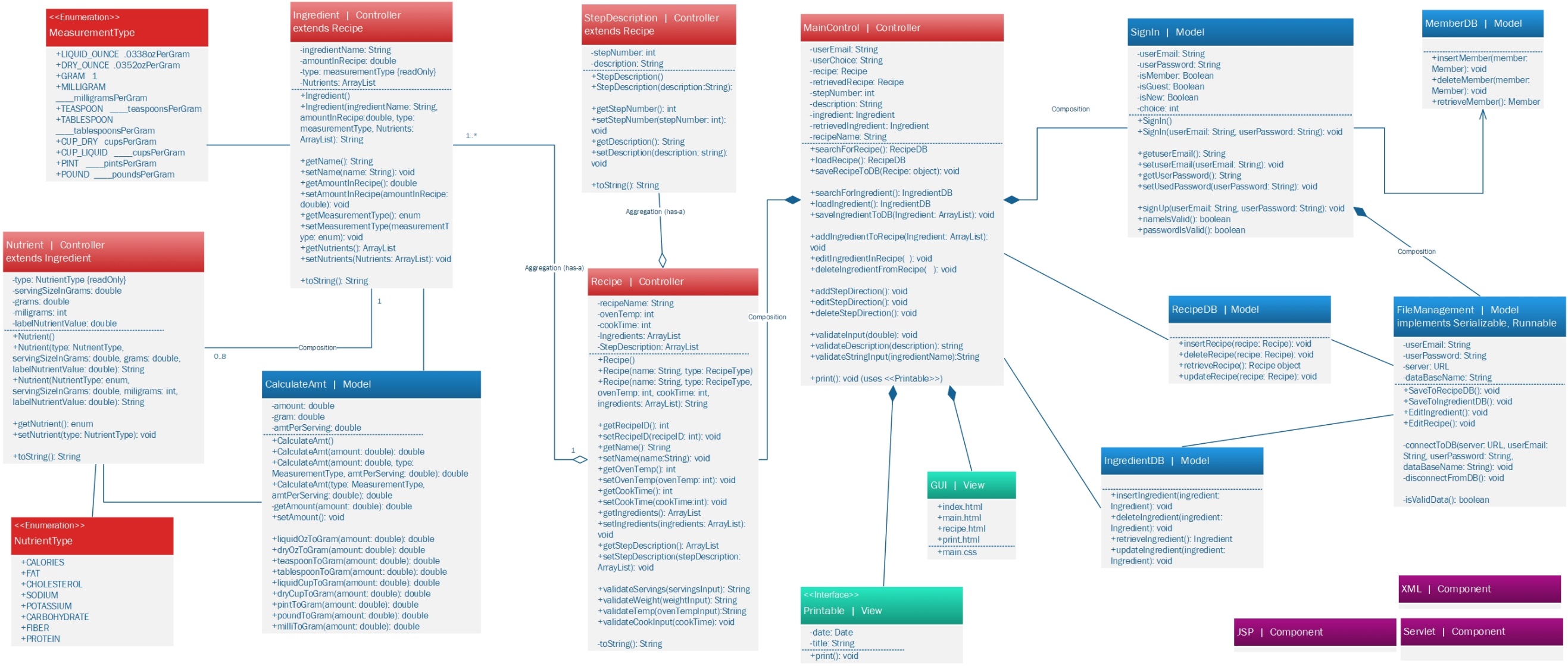


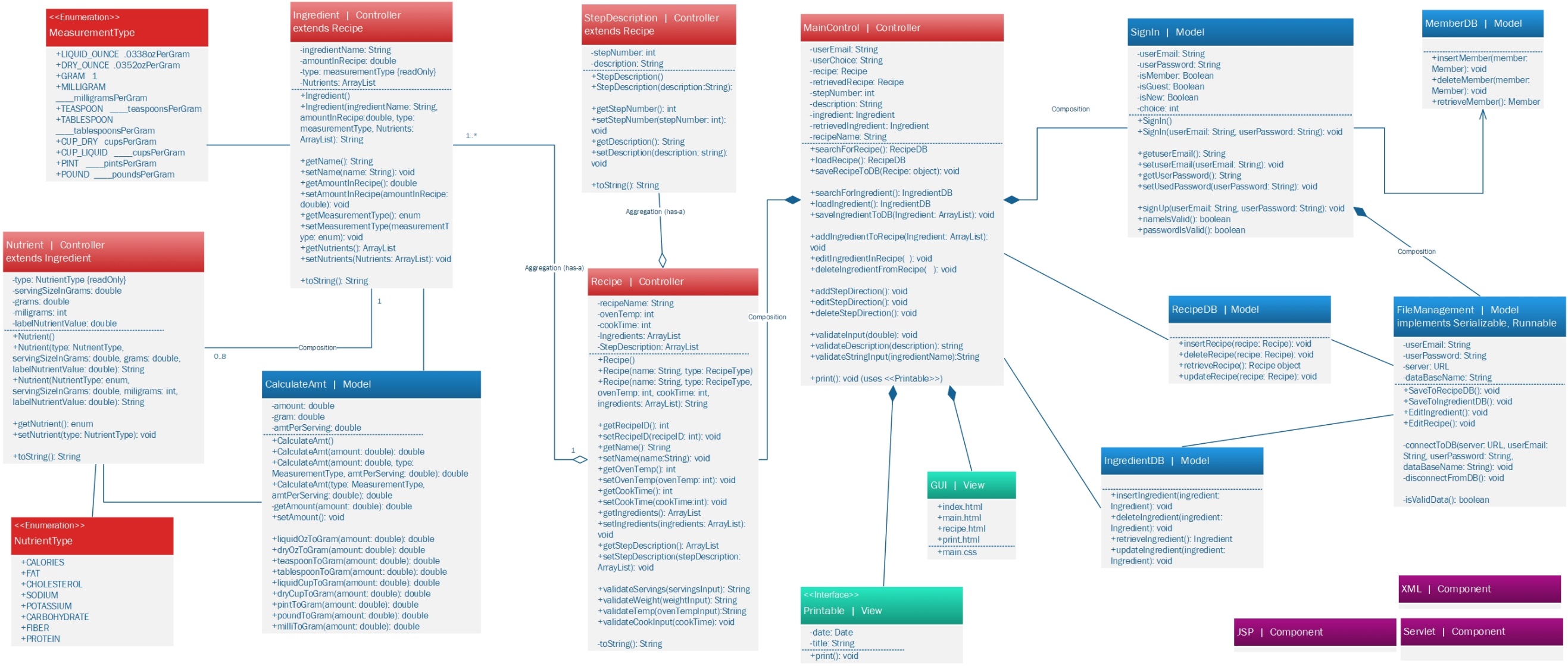




# UML Diagram







# Pseudocode

The Model View Controller will be the framework of the coding.

/\*

SignIn class

initial user entry point

controls "on-click" events for index.html

\*/

START

// Variables

private String userEmail

private String userPassword

private String loginFailer

private int choice

private boolean isMember = false

private boolean isGuest = false

private boolean isNew = false

// default constructor

public void SignIn() {

// list default values

}

// constructor

public void SignIn(userEmail, userPassword) {

}

// getters and setters

getUserEmail()

setUserEmail(userEmail)

getUserPassword()

setUserPassword(userPassword)

getChoice()

setChoice(choice)

// Get user email and password

userEmail = validateInput(userEmail)

userPassword = validateInput(userPassword)

// create sign-in object

SignIn user = new SignIn(userEmail, userPassword)

/\* SIGN-IN, SIGN-UP, OR GUEST EVENT CLICK EVENT \*/

// set boolean variable to true if it matches

if choice == signin

isMember = true

break

else if choice == guest

isGuest = true

break

else if choice == signup

isNew = true

break

endif

// login type is determined by choice variable

if isGuest

userEmail = "Guest"

loadGui(main.html)

else if isMember

// validate membership (a little sketchy on how this part works)

if user == MemberDB(user)

FileManagement.connectToDB(userEmail, userPassword)

loadGui(main.html)

else

display loginFailer message

endif

else if isNew

// check to see if email matches email in database

// Register new member

if emailIsValid(userEmail) & passwordIsValid(userPassword)

createNewMember(user)

// reload page for new member to sign in

loadGui(index.html)

else

Display errorMessage

endif

else

Display error

endif

STOP

/\*

MainControl class

mainControl controls "on-click" events for main.html and recipe.html

\*/

START

// Variables

private String userEmail

private String userChoice

private Ingredient ingredient

private Recipe newRecipe

private Recipe savedRecipe

private String recipeName

private Array String recipeList

private int stepNumber

private String description

/\* USER CHOICE SELECTION FOR ON-CLICK EVENT - MAIN.HTML \*/

// get user choice of new recipe or load from DB

if userChoice == "new recipe"

// display pop-up window for Recipe entry

GUI(recipe entry window)

// get user input and validate )

recipeName = validateStringInput(recipeName)

// Create new Recipe object and load the new GUI

newRecipe = new Recipe(recipeName)

loadGui(recipe.html)

else if userChoice == "load recipe"

// load saved recipes from DB and display list

searchForRecipe()

if savedRecipes == 0

display error

else

// load the new GUI and populate all the fields/attributes

getSelection()

loadGui(recipe.html)

loadRecipe()

endif

else

display error

endif

/\* INGREDIENT SELECTION FOR ON-CLICK EVENT - RECIPE.HTML \*/

// add, edit, or delete ingredient

if ingredientSelection == "add"

// display pop-up window for ingredient entry

GUI(ingredient entry window)

// get user input and validate (measurementType is a drop-down menu item)

ingredientName = validateStringInput(ingredientName)

amountInRecipe = validateInput(amountInRecipe)

measurementType = measurementTypeSelection(measurementType) // enum

//// not sure how to work with nutrient enums here

Nutrients.CALORIES = validateInput(calories)

Nutrients.FAT = validateInput(fat)

Nutrients.CHOLESTEROL = validateInput(cholesterol)

Nutrients.SODIUM = validateInput(sodium)

Nutrients.POTASSIUM = validateInput(potassium)

Nutrients.CARBOHYDRATE = validateInput(carbohydrate)

Nutrients.FIBER = validateInput(fiber)

Nutrients.PROTEIN = validateInput(protein)

// creat Ingredient object

ingredient = new Ingredient(ingredientName, amountInRecipe,

ingredientType, Nutrients<arrayList>)

else if ingredientSelection == edit

// display pop-up window for ingredient entry

GUI(ingredient entry window)

loadIngredient(ingredientRowSelection)

editIngredientInRecipe(ingredientRowSelection)

else if ingredientSelection == delete

deleteIngredientFromRecipe(rowSelection)

else

display error

endif

/\* STEP DESCRIPTION FOR ON-CLICK EVENT - RECIPE.HTML \*/

// add, edit, or delete a step

if stepSelection == add

// display pop-up window for Step Description

GUI(step description window)

// get user input and validate (stepNumberSelection is a drop-down menu item)

stepNumber = stepSelection(stepNumberSelection)

description = validateDescription(description)

StepDescription(stepNumber, description)

else if stepSelection = edit

// display pop-up window for Step Description

GUI(step description window)

loadStep(stepRowSelection)

editStepDescription(stepRowSelection)

else if stepSelection == delete

deleteStepDescription(rowSelection)

else

display error

endif

/\* PRINTING FOR ON-CLICK EVENT - RECIPE.HTML \*/

// print recipe, nutrition label, and steps

if printSelection == nutrition lable

loadGui(print.html)

printLabel()

else if printSelection == ingredients and directions

loadGui(print.html)

printRecipe()

else if printSelection == all

loadGui(print.html)

printAll()

/\* SAVING TO DATABASE FOR ON-CLICK EVENT - RECIPE.HTML \*/

// Save all data to user's database on closing

if status = logOut

RecipeDB.saveRecipeToDB()

foreach (Ingredient in ingredient) {

IngredientDB.saveIngredientsToDB()

}

logOut()

endif

/\* METHODS \*/

private String validateDescription() {

// validates against sql injections

// validates for string input with length < 500 chars

}

private double validateDoubleInput() {

// validates against sql injections,

// validates for double type input

// validates for chars

if value == null

set value = 0

}

private String validateStringInput() {

// validates against sql injections,

// validates for string input with length < 25 chars

}

STOP

/\*

Recipe class

Aggregate

\*/

START

// variables

private String recipeName

private int ovenTemp = 0

private int cookTime = 0

private int servings = 1

private int weight = null

private ArrayList Ingredient

private ArrayList StepDescription

// default constructor

public Recipe(){

// set default values

}

// constructor basic

public Recipe(recipeName) {

}

// constructor with everything filled out

public Recipe(recipeName, ovenTemp, cookTime, servings, weight, Ingredient<ArrayList>) {

}

// get each Ingredient

foreach ingredient in Ingredient

display row in table

endforeach

/\* OPTIONAL DATA ENTERED \*/

// get oven temp, cook temp, servings and weight

servings = validateServings(servingsInput)

ovenTemp = validateTemp(ovenTempInput)

cookTime = validateCookInput(cookTimeInput)

weight = validateWeight(weightInput)

public int validateServings(servingsInput) {

// validate range (cannot be < 1) with default = 0

}

public int validateTemp(ovenTempInput) {

// validate range 0 - 500 with default = 0

}

public string validateCookInput(cookTimeInput) {

// validates against sql injections,

// validates for string input with length < 25 chars with default = 0

}

public int validateWeight(weightInput) {

// in grams with default = 0

}

// getters and setters

getOvenTemp()

setOvenTemp(ovenTemp)

getCookTime()

setCookTime(cookTime)

getServings()

setServings(servings)

getWeight()

setWeight(weight)

@override

toString()

STOP

/\*

Ingredient class extends Recipe

This class holds ingredient and gets the nutrient

values from the nutrient class

\*/

START

// variables

private String ingredientName

private double amountInRecipe

private enum MeasurementType

private ArrayList Nutrients

// default constructor

public Ingredient() {

// set default values

}

// constructor (not sure how to handle the Nutrient arrayList and enums)

public Ingredient(ingredientName, amountInRecipe, measurementType, Nutrient<arrayList>)

// get each nutrient value for the ingredient

foreach nutient in Nutrients

getNutrients()

endforeach

// getters and setters

getIngredientID()

setIngredientID(ingredientId)

getIngredientName()

setIngredientName(ingredientName)

getAmountInRecipe()

setAmountInRecipe(amountInRecipe)

getMeasurementType()

setMeasurementType(measurementType)

getNutrients()

setNutrients(Nutrients<ArrayList) // getters/setters for each ingredient?

@override

toString()

STOP

/\* MEASUREMENT TYPE ENUMERATION -- USED BY INGREDIENT CLASS \*/

START

LIQUID\_OUNCE

DRY\_OUNCE

GRAM

MILLIGRAM

TEASPOON

TABLESPOON

LIQUID\_CUP

DRY\_CUP

PINT

POUND

STOP

/\*

Nutrient class extends Ingredient

This class holds the nutrient names and values for each ingredient

\*/

START

// variables

private enum NutrientType

private double servingSizeInGrams

private double grams

private int milligrams

private double labelNutrientValue

// default constructor

public Nutrient() {

// set default values

}

// constructor for grams

public Nutrient(nutrientType, servingSizeInGrams, grams, labelNutrientValue) {

}

// constructor for milligrams

public Nutrient(nutrientType, servingSizeInGrams, milligrams, labelNutrientValue) {

}

// getters/setters for each nutrient

// nutrients are in an enum, so not sure how to handle this

getNutrient() enum

setNutrient(CALORIES) // ???

@override

toString()

STOP

/\* NUTRIENT TYPE ENUMERATION -- USED BY NUTRIENT CLASS \*/

START

CALORIES

FAT

CHOLESTEROL

SODIUM

POTASSIUM

CARBOHYDRATE

FIBER

PROTEIN

STOP

/\*

StepDescription class extends Recipe

This class holds the steps and directions for the recipe

\*/

START

// variables

private int stepNumber

private String description

// default constructor

public StepDescription() {

// set default values

}

// constructor for grams

public StepDescription(stepNumber, description) {

}

// getters/setters for each step

getStepNumber()

setStepNumber(stepNumber)

getDescription()

setDescription(description)

@override

toString()

STOP

/\*

CalculateAmt class

This class holds all the methods for math

in the Nutrient, Ingredient, and Recipe classes

\*/

START

// variables

private double amount

private double grams

private double amtPerServing

// default constructor

public CalculateAmt(){

// set default values

}

// constructor

public CalculateAmt(amount, grams, amtPerServing) {

//

}

public CalculateAmt() {

//

}

// methods for converting weight and volume

public liquidOzToGram(amount) {

// 1 liquid oz. = 28.3 grams

}

public dryOzToGram(amount) {

// 1 dry oz. = \_\_\_ grams

}

public teaspoonToGram(amount) {

// 1 dry teas. = \_\_ grams

}

public tablespoonToGram(amount) {

// 1 dry TBSP. = \_\_\_ grams

}

public liquidCupToGram(amount) {

// 1 liquid cup = \_\_\_ grams

}

public dryCupToGram(amount) {

// 1 dry cup = \_\_\_ grams

}

public pintToGram(amount) {

// 1 liquid pint = \_\_\_ grams

}

public poundToGram(amount) {

// 1 dry lb = \_\_\_\_ grams

}

public milliToGram(amount) {

// 1 milligram = \_\_\_ grams

}

STOP

/\*

MemberDB class implements Serializable

This class connects to database

\*/

START

public insertMember(Member member) {

// insert new Member into DB

}

public retrieveMember(Member member) {

// save data to persistant data

}

STOP

/\*

RecipeDB class implements Serializable

This class holds all the recipe and Step/Description values

\*/

START

public insertRecipe(Recipe recipe) {

// insert new recipe into member's database

}

public retrieveRecipe(Recipe recipe) {

// save stored member's recipe to persistant file

}

public deleteRecipe(Recipe recipe) {

// delete stored recipe from member's database

}

public updateRecipe(Recipe recipe) {

// replace stored recipe in database with new values

}

STOP

/\*

IngredientDB class implements Serializable

This class holds all the ingredient and its' nutrient values

\*/

START

public insertIngredient(Ingredient ingredient) {

// insert new ingredient into database

}

public retrieveIngredient(Ingredient ingredient) {

// save stored ingredient object to persistant file

}

public deleteIngredient(Ingredient ingredient) {

// delete stored ingredient from member's database

}

public updateIngredient(Ingredient ingredient) {

// replace stored ingredient in member's database with new values

}

STOP

/\*

FileManagement class implements Serializable, Runnable

This class connects to the Database

\*/

START

private server url

private String databaseName

private String userEmail

private String userPassword

// use try-catch block for connection

public void connectToDB(url, userEmail, userPassword, databaseName) {

// connectToDB

}

STOP