System Design Document V.2.0
11-6-2023

Waste Watcher Built for the Senator George J. Mitchell Center for Sustainability Solutions
by:

Sustainable Waste Solutions (Current Team)

- Christian Silva
- Kevin Bretthauer
- Caiden Emerson
- Jackson Cyr
- Callen Shaeffer

Kayak Development Solutions (Past Team)

- Declan Brinn
- Gavin Palazzo
- Levi Sturtevant
- Chase Pisone
- Finn Jacobs

Table of Contents

1 Introduction

The Food Waste Tracking & Measuring software application is a University of Maine Capstone project for Susanne Lee, in partial fulfillment of the Computer Science BS degree for the University of Maine. Susanne Lee is a Faculty Fellow at the Mitchell Center for Sustainability Solutions at the University of Maine. Susanne leads a student/faculty team working towards developing solutions to end food waste in Maine. Due to the massive amounts of food that goes to waste each year the need for our project arose. The goal of our Food Waste Tracking & Measuring app is to make the user aware of how much food they waste and what they can do to minimize food waste in their household. This project is being developed to provide a cost-free product that will assist in the tracking of food waste among schools, businesses, and households in Maine.

"In 2015, a Mitchell Center multidisciplinary team identified eliminating food waste as the single most important issue to ensure a more sustainable waste system in Maine." [1]

1.1 Purpose of This Document

The purpose of this document is to provide a technical description of the system that we are designing for the app Waste Watcher. The document will include sections for the system architecture, database design, and the requirements for each component from our SRS. This will act as our reference when we begin developing these systems.

1.2 References

[1] "Home." *Food Rescue MAINE*, 4 Nov. 2023, umaine.edu/foodrescuemaine/.

[2] "Waste Management." *Waste Management, Maine Department of Environmental Protection*, www.maine.gov/dep/waste/index.html. Accessed 15 Nov. 2023.

[3] "Recycling Drop off Locations." *Ecomaine*, www.ecomaine.org/what-can-be-recycled/recycling-drop-off-locations/. Accessed 15 Nov. 2023.

UML Distilled, 2nd edition, Martin Fowler and Kendall Scott, 2000

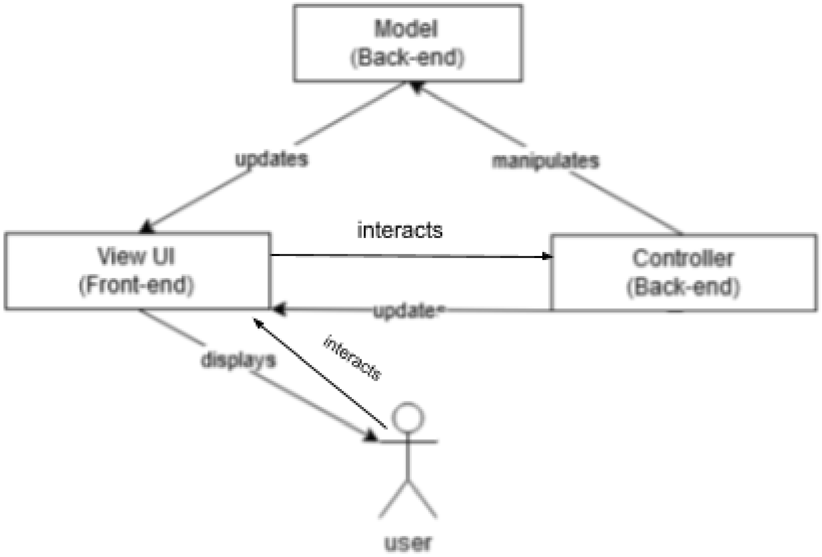
It should be noted that this is version 2.0 of the Waste Watchers System Design Document, and it is building off of the work done by the Kayak Development team. There have been some significant changes made to the design of the backend data storage system that will be used for the project, as well as the login system implementation details. We have gone through and updated the design for the system to account for these changes, but much of the design remains constant.

2 System Architecture

This section will outline the ways that we will design and build each system within Waste Watcher. This includes the interactions between front-end and back-end components as well as the hardware and software requirements. The requirements will be supplemented by illustrations that will correspond to each of the requirements.

2.1 Architectural Design

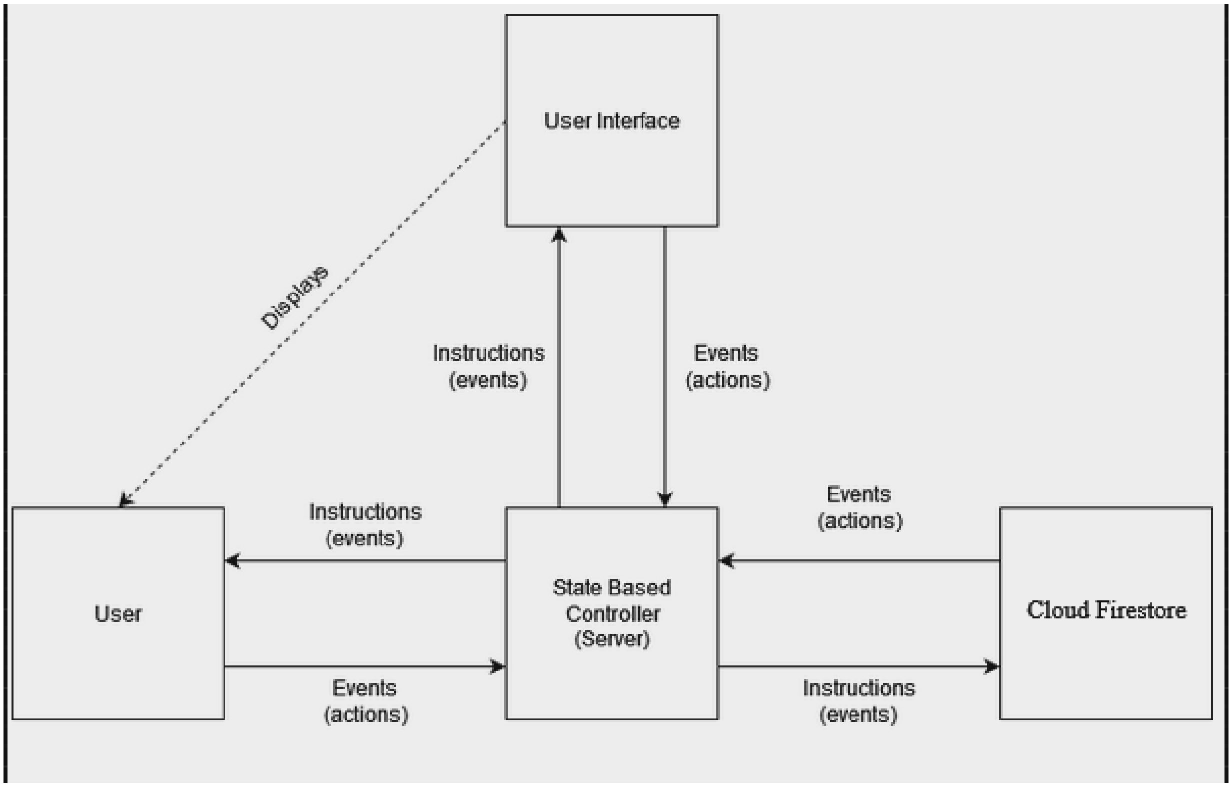
<u>Model-View-Controller (MVC) Architecture Diagram:</u> Figure 2.1.1



Description: 2.1

The mobile application must be able to be downloaded for both Android and Apple products. To do this the application architecture will consist of two main components, a back-end and a front-end. The Front-end, or the UI, will be created using React Native which is an open-source JavaScript UI software framework for mobile application development. This will allow the application to have a displayed interface that the user can interact with. This includes buttons, pictures, graphs, text boxes for inputs, and various other visuals/interactive elements, which will then give these inputs to the controller to process and provide the product functionality. This back-end is the database/server side. This is where user information and data will be stored and analyzed. To accomplish this, a Firestore database management system will be employed. All of these features will allow the mobile app to be downloadable cross-platform from both Android and mobile apps.

High-Level Event Driven Component Interaction Diagram: Figure 2.1.2

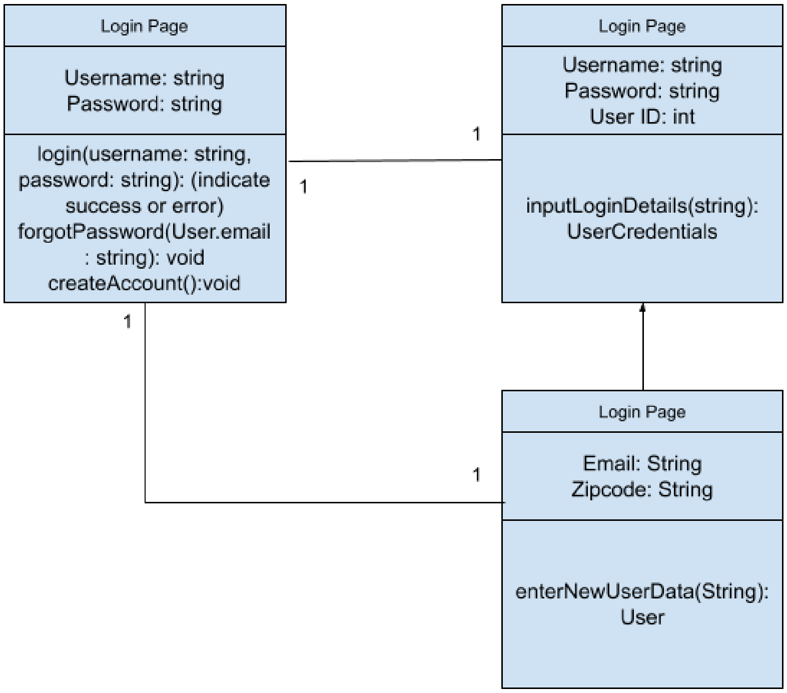


Description: 2.1.2

This is an alternate view of diagram 2.1 which illustrates the base interaction between each of the main systems for the application. This diagram shows the flow of the technology involved in creating the architecture of the program as well as an overview of the interactions between one another. Overall, there are three components: the User, the UI, and the Database all of which take in instructions and respond to events. There is a Controller(Server) as well which takes in events from the components and responds with instructions. Events are only generated by components, an example of which would be the UI taking in an input, from which the event is then sent to the Controller. From here, the Controller receives and interprets the information it was given and formulates a set of instructions in response to the event. The Controller then sends those instructions to the specific components that need to be updated via the given event. Continuing from the previous example, the Server would take in the UI-generated event and send an instruction to the Firestore database component telling it to store the input given from the UI. The database stores events from the controller and sends back an instruction for the server to execute. The server would also send a similar instruction back to the UI telling it to update its display accordingly which is then shown to the user.

2.2 Decomposition Description

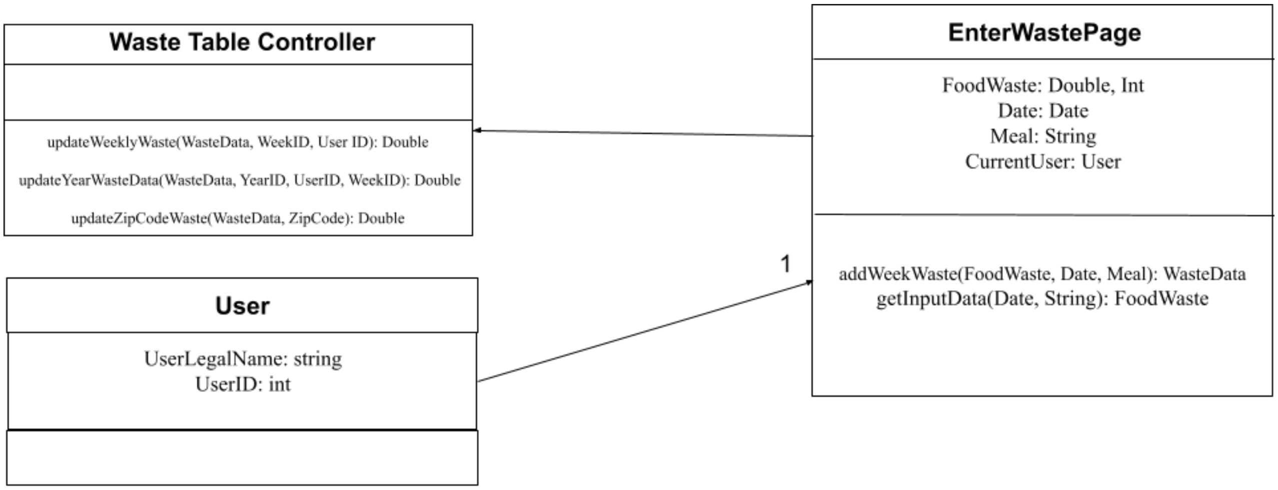
Login Functionality Class Diagram: Figure 2.2.1


Login Page
Username: string
Password: string
login(username: string, password: string): (indicate success or error)
forgotPassword(User.email : string): void
createAccount():void
Login Page
Username: string
Password: string
User ID: int
inputLoginDetails(string): UserCredentials
Login Page
Email: String
Zipcode: String
enterNewUserData(String): User
1
1
1
1

Description: 2.2.1

The separate pages of the application will be object-oriented as each individual page will have its own specific functionalities/classes associated with it. Most pages will inherit class data identified in the login page above with the exception of the input food waste page. Thus, a detailed class diagram illustrating user account login and creation has been provided. Each box represents a class in which it contains its associated variables with their correlated data types (int, string, double). Each class also contains the functions/methods that it will be involved with. Any parameters passed to the function are found within the parenthesis and the functions outputs are found after the colon character. Outputs can be any associated data type based on the function's intended purpose. For example, in the New User class, the *"enterNewUserData(String):User"* method would take in a string for Username, Password, and Email and return a new instance of the User class. links without arrows between classes imply that there is a one-to-one relationship between the two. Essentially what this means is any singular instance of a class will have exactly one instance of a connected class associated with it. So, in the case of the Login class, there is only one User that can be associated with it.

Waste Entry Diagram: Figure 2.2.2



Waste Table Controller
updateWeeklyWaste(WasteData, WeekID, User ID): Double
updateYearWasteData(WasteData, YearID, UserID, WeekID): Double
updateZipCodeWaste(WasteData, ZipCode): Double
EnterWastePage
FoodWaste: Double, Int
Date: Date
Meal: String
CurrentUser: User
addWeekWaste(FoodWaste, Date, Meal): WasteData
getInputData(Date, String): FoodWaste
1
User
UserLegalName: string
UserID: int

Description: 2.2.2

This class diagram illustrates the food waste input functionality. The user interacts with the application, entering food waste for each meal on a specified day into the UI. The waste table controller class then takes this data and updates the weekly database table and yearly database table, adding the new data entered by the user to the respective totals. This data will be used to model food waste statistics at a later point in app development.

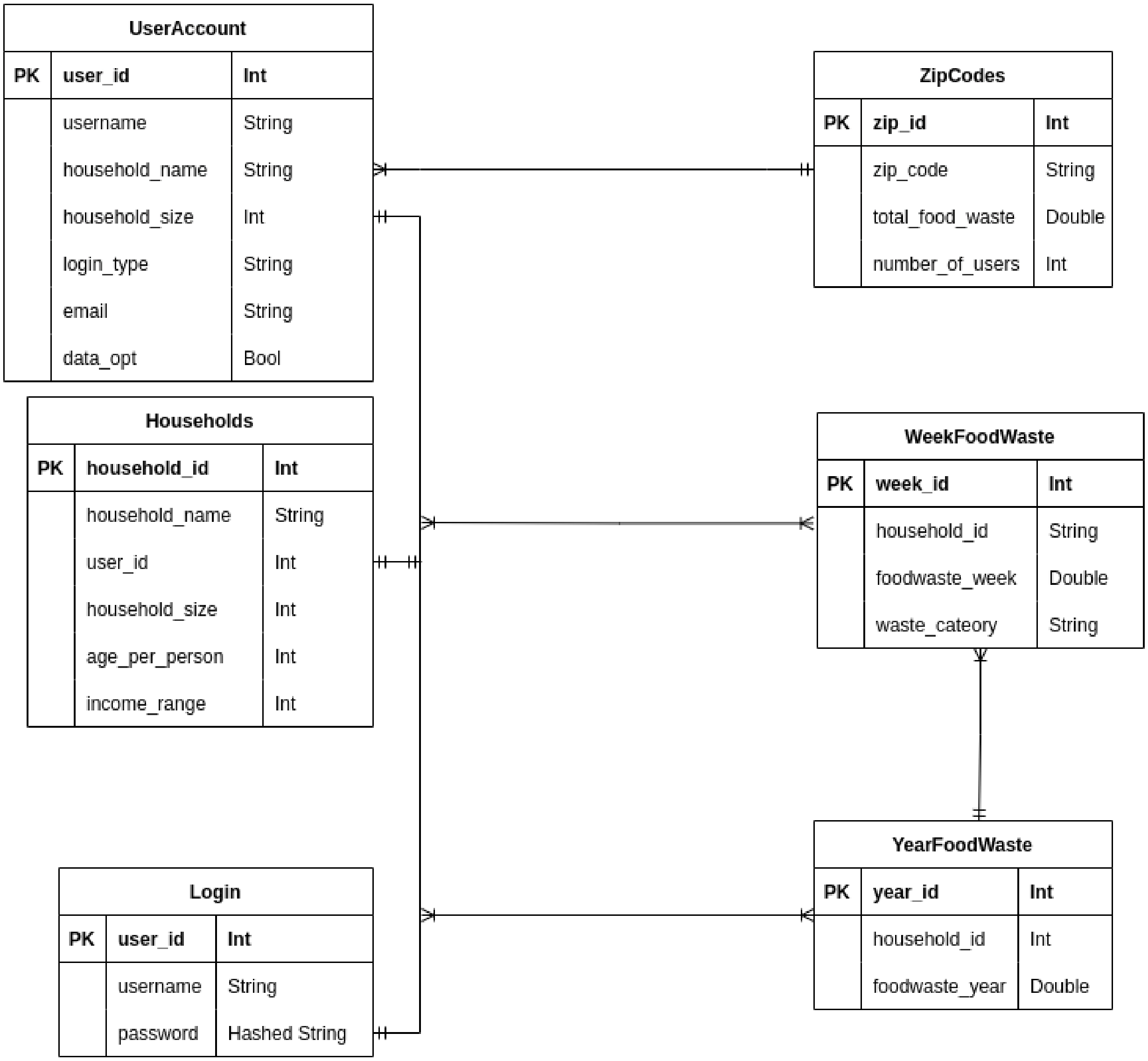
3 Persistent Data Design

This section will serve as our reference for the system database and will include information such as the file structure and database type. Section 3.1 will illustrate the system database and its interactions.

3.1 Database Descriptions

Each of the tables in the diagram below represents a collection (effectively a JSON format) within our Firestore database. The UserAccount table will store information given to the UI when a user registers for an account on the application. The Household table stores more granular household information, including the age of each person in the household and the income range of the household. As stated above, information that is user/household specific is split into multiple tables, linked with the user_id data in order to improve the security of the database.The ZipCode table will compile user waste data input by users to provide a breakdown of food waste data within a specific geographic area.

Firestore Database Description: Figure 3.1.1



UserAccount		
PK	user_id	Int
	username	String
	household_name	String
	household_size	Int
	login_type	String
	email	String
	data_opt	Bool

ZipCodes		
PK	zip_id	Int
	zip_code	String
	total_food_waste	Double
	number_of_users	Int

Households		
PK	household_id	Int
	household_name	String
	user_id	Int
	household_size	Int
	age_per_person	Int
	income_range	Int

WeekFoodWaste		
PK	week_id	Int
	household_id	String
	foodwaste_week	Double
	waste_cateory	String

YearFoodWaste		
PK	year_id	Int
	household_id	Int
	foodwaste_year	Double

Login		
PK	user_id	Int
	username	String
	password	Hashed String

The WeekFoodWaste table will be a compilation of all food waste submitted by users each week. Entries will be added to the table on a weekly basis, and the total food waste will be updated once users start inputting data. Meal-specific food waste data (i.e. how much food a household wastes at dinner on a specific day) will be collected and stored locally on the user's device, and deleted once it is added to the broader database table. Lastly, the YearFoodWaste table takes data from the WeekFoodWaste table and will create entries on a yearly basis, tabulating total waste figures when weekly data is collected. All entries in these tables will be given a respective identifier which will be used to link data within different tables.

3.2 File Descriptions

No files are required for this system.

4 Requirements Matrix

The following table outlines the relationship between each of our use cases and the functions/methods that will be used to perform them.

Use Case No.	Name	Function/Method
FR-01	Input food waste	TrackWaste()
FR-02	Create an account	CreateAccount()
FR-03	Display food waste hierarchy	ShowFoodChart()
FR-04	Add friend	AddFriend()
FR-05	Send referral message	SendReferral()
FR-06	Opt out of data collection	OptFlag()
FR-07	Leaderboard management	Leaderboard()
FR-08	Badge system, and achievements	Achievements()
FR-09	Tips for users	Tips()
FR-10	Food waste history tracking	WasteGraph()
FR-11	Track cost of wasted food	Cost()
FR-12	Meal location history tracker	TrackMealLocations()
FR-13	Display donation map	DonationMap()
FR-14	Account management	ManageAccount()

Appendix A – Agreement Between Customer and Contractor

By signing this document, Susanne Lee and Waste Management Solutions agree upon the basic system design for this project. These requirements are susceptible to change based on if new more important requirements come up or if things become obsolete. In the future, if these changes need to be made, they will be discussed with our customer first for clarification and then updated here once an agreement has been made. Changes to this document should be known by whoever it is relevant to.

Client Signature:

Name: Susanne Lee	Date: 11/10/23
Signature: Susanne Lee	
Comments:	

Team Signatures:

Name: Kevin Bretthauer	Date:11/06/23
Signature: Kevin	
Comments:	

Name: Callen Shaeffer	Date: 11/06/23
Signature: Callen Shaeffer	
Comments:	

Name: Christian Silva	Date: 11/06/23
Signature:	
Comments:	

Name: Caiden Emerson	Date: 11/06/23
Signature:	
Comments:	

Name: Jackson Cyr	Date: 11/06/23
Signature: Jackson Cyr	
Comments:	

Appendix B – Team Review Sign-off

By signing below members have indicated that they agree to the information mentioned above in this document. You agree that this document may see changes in the future.

Team Signatures:

Name: Kevin Bretthauer	Date:11/06/23
Signature:	
Comments:	

Name: Callen Shaeffer	Date: 11/06/23
Signature:	
Comments:	

Name: Christian Silva	Date: 11/06/23
Signature:	
Comments:	

Name: Caiden Emerson	Date: 11/06/23
Signature:	
Comments:	

Name: Jackson Cyr	Date: 11/06/23
Signature: Jackson Cyr	

Appendix C – Document Contributions

Christian Silva -

- Did 1.2 and 2.2.1
- Did appendices

Caiden Emerson

- Contributed to sections 1 and 3

Jackson Cyr

- Contributed to section 3.1
- Contributed to section 4

Callen Shaffer

- Contributed to section 1
- Contributed to section 2.2

Kevin Bretthauer

- Contributed to section 2
- Helped with functionality diagram