REQUIREMENTS ANALYSIS

HUNTERGATHER

ALEC BARRAN
GRIFFIN GERRY
WILLIAM HEINECKE

MARCH 8, 2024

INTRODUCTION

This Use Case document outlines the development of HunterGather, a revolutionary web platform that integrates recipe browsing with ingredient procurement. HunterGather addresses the inefficiencies of traditional cooking methods by offering a unified solution for discovering recipes, creating shopping lists, and engaging with the culinary community. The site also has a retail-facing interface for local vendors to create and update their inventory, so that customers can check out with ease. HunterGather aims to become a valuable asset for culinary enthusiasts, recipe creators, and merchants in contributing to a seamless and enjoyable cooking experience.

ACTORS AND ROLES

Actors and their roles for this application include:

- Visitor - an individual who accesses HunterGather without logging in. They can browse recipes and view content but cannot save recipes or make orders.
- Registered User - an individual who has created a HunterGather account. They have access to additional features such as saving recipes, adding ingredients to the cart, and interacting with other users through comments and ratings.
- Store - suppliers or vendors from which users can purchase ingredients. They provide an inventory of available products and process user orders.
- Administrator - a user with elevated privileges responsible for moderating HunterGather. They have access to backend functionalities such as user management, content moderation, and analytics.
- System - represents the application itself, including software, databases, and servers. It handles user interactions, processes data, and executes business logic to provide functionality to users.
- Payment Processor - a third-party service for handling online transactions between users and stores. Payment processors securely process payment information and facilitate the transfer of funds between users and stores.

USE CASES

Name of Use Case:	User Registration
Description:	Enables users to create an account on the platform to grant them access to features such as saving recipes and checkout. The expected outcome is for new users to register an account with the necessary information like name and email address.
Actors:	• Primary Actor: Visitor • Secondary Actor: System
Preconditions:	• User has access to HunterGather

	• User does not already have an account
Postconditions:	• The user successfully creates a HunterGather account • The system saves the user account information
Flow:	1. New user navigates from the homepage to the registration page 2. System serves the registration form with fields including name, email address, and password 3. User completes the form with required information 4. User agrees to the terms and conditions, and submits the form 5. System validates the information and confirms that the user does not already have an account 6. System creates a new account in the database according to form 7. System sends a verification email to the user's email address 8. User receives verification email and clicks link to verify account 9. System confirms the account and enables user login 10. User logins into their account with email address and password
Alternative Flows:	In step 5 of the normal flow, if the user already has an account: 1. User is notified that the email address is already in use 2. System displays a redirect link to the login screen
Exceptions:	In step 5 of the normal flow, if the information fails to validate: 1. User is notified of the error (e.g., invalid email, weak password) 2. Erroneous fields are reset, and user is given chance to correct
Requirements:	• Password is 8 character or more and contains at least one capital letter, number, and special character

Name of Use Case:	User Login
Description:	Enables users with a HunterGather account to access their account and use the features reserved for registered users. The expected outcome is for the user to be logged into their account.
Actors:	• Primary Actor: Visitor • Secondary Actor: System
Preconditions:	• User has a verified HunterGather account
Postconditions:	• User is logged into their account with cookie

Flow:	1. User navigates to any page with the login button on the navigation bar 2. System serves login button which displays a login form with CSRF token when clicked 3. User clicks login button and enters their email address and password in the displayed form 4. User submits login form 5. System receives and validates login request 6. System responds to user with validated login and basic account details 7. User is redirected to the homepage with logged in account
Alternative Flows:	In step 5 of the normal flow, if the email address does not have an account: 1. System warns user that no account was found, and offers link to user registration page In step 5 of the normal flow, if the password is incorrect 1. System warns user that the password is incorrect
Exceptions:	In step 5 of the normal flow, if the CSRF token fails to validate: 1. System warns user that the CSRF token does not match 2. User login page reloads
Requirements:	• CSRF token is served securely

Name of Use Case:	Browsing Recipes
Description:	Allows all users to browse and search recipes available on HunterGather. The expected outcome is that the user finds a recipe based on their search criteria.
Actors:	• Primary Actor: Any user • Secondary Actor: System
Preconditions:	• User has access to HunterGather • System has database of recipes
Postconditions:	• System serves recipes matching search criteria
Flow:	1. User navigates to the Browse page 2. System serves a list of top-rated recipes and a form with queries like keyword search and time constraint 3. User fills out and submits form according to their preferences 4. System receives form and queries database for recipes matching criteria

	5. System serves and displays search results to user 6. User selects recipe to view
Alternative Flows:	In step 6 of the normal flow, if the user does not like any of the returned options: 1. Return to step 3
Exceptions:	In step 4 of the normal flow, if the database returns no data: 1. System displays error message to user that no recipes are found
Requirements:	• Recipes all have keyword and time attributes that can be queried

Name of Use Case:	Saving Recipes
Description:	Allows registered users to save recipes to their account so that they may easily find them again in the future. The expected outcome is that the recipe is successfully saved to the user saved recipe database.
Actors:	• Primary Actor: Registered user • Secondary Actor: System
Preconditions:	• User is logged in with verified account • Recipe exists
Postconditions:	• Recipe is saved to the user's saved recipes
Flow:	1. User navigates to a recipe page 2. System displays all the recipe content, as well as forms such as adding a comment and a button to save the recipe 3. User clicks the save recipe button 4. System receives and validates request to save recipe 5. System creates new entry in saved recipe database associated with logged in user 6. System displays that the recipe is successfully saved
Alternative Flows:	In step 2 of the normal flow, if the user already has the recipe saved: 1. The system instead displays an unsave recipe button 2. User clicks unsave recipe button 3. System receives and validates request to unsave recipe 4. System removed entry in saved recipe database 5. System displays that the recipe is successfully unsaved
Exceptions:	In step 4 of the normal flow, if the user requests to unsave a recipe they do not have saved and vice versa: 1. System displays and error message

Requirements:	• Saved user lists are only visible to current logged in user or admin • User is only able to save to their own list

Name of Use Case:	Create Recipe
Description:	Allows registered users or stores to create recipes, including a title, description, ingredients, time, and image. The ingredients should be associated with known ingredient objects which can be added to the cart. The expected outcome is that the recipe is successfully created and publicly visible.
Actors:	• Primary Actor: Registered user or store • Secondary Actor: System
Preconditions:	• User is logged in with verified account
Postconditions:	• Recipe is saved to the database and publicly visible
Flow:	1. User navigates to the create recipe page 2. System serves form with attributes like title, description, ingredients, time, and image 3. User fills out form with the required information and submits 4. System receives form and validates data 5. System creates recipe database entry using the form data 6. System displays message to user that recipe is successfully created, and redirects to new recipe page
Alternative Flows:	In step 4 of the normal flow, if the recipe title already exists for that user: 1. Numeric identifier (e.g. (1), (2) ...) is appended to the title
Exceptions:	In step 4 of the normal flow, if the form does not have all the required data: 1. System highlights the erroneous fields 2. Return to step 3
Requirements:	• Support for rich content like images and videos

Name of Use Case:	Add recipe ingredients to cart
Description:	Allows registered users to add all or individual ingredients from a recipe to their cart. The ingredients may be available at zero, one, or more local stores participating in HunterGather. The expected outcome is that the ingredients in the recipe are all successfully added to the cart.
Actors:	• Primary Actor: Registered user

	• Secondary Actor: System
Preconditions:	• User is logged in with verified account • Recipe ingredients are associated with ingredient listings
Postconditions:	• Added ingredients are visible in cart
Flow:	1. User navigates to a recipe page 2. System displays all the recipe content, as well as forms and select boxes for each ingredient 3. User toggles select boxes for ingredients to add to cart, and clicks button to submit 4. System receives and validates request for selected ingredients 5. System queries ingredient database for each ingredient and adds results to the user cart 6. System notifies user that ingredients were successfully added to cart with optional link to cart and checkout
Alternative Flows:	In step 5 of the normal flow, if the ingredients are not available at the user's preferred store: 1. System notifies user that the ingredient is available elsewhere, with an option to buy from another store 2. User responds with new store preferences and system proceeds accordingly at step 6
Exceptions:	In step 4 of the normal flow, if the user requests ingredients which do not exist (malformed or malicious request): 1. System notifies user that the requested ingredient does not exist 2. Return to step 3
Requirements:	• Ingredient quantities are adjusted for the size available in-store • User is only able to add to their own cart

Name of Use Case:	Checkout Cart
Description:	Allows registered users with items in their cart to checkout and complete an order with a local store. The expected outcome is that the order is successfully placed with the store and the user will be able to pick it up.
Actors:	• Primary Actor: Registered user • Secondary Actors: ○ System

	○ Payment processor ○ Registered store
Preconditions:	• User is logged in with verified account • User has items in their cart
Postconditions:	• Store receives • Store inventory is updated with decremented ingredients after purchase • Store receives payment
Flow:	1. User navigates to cart page 2. System displays ingredients in user's cart with store availability 3. User clicks button to checkout 4. System queries ingredient database and compiles items and total 5. System redirects user to third-party payment processor with calculated total 6. Payment processor confirms user payment 7. System receives payment confirmation 8. System forwards user order to one or more stores
Alternative Flows:	In step 6 of the normal flow, if the user fails payment processing: 1. Payment processor notifies system 2. System warns user that their payment was not confirmed, and the order has not been placed 3. Return to step 2
Exceptions:	In step 2 of the normal flow, if some ingredients are not in stock: 1. System warns user that the item is not in stock 2. User may continue the checkout process without the missing items
Requirements:	• Items in user cart must be in stock or ignored before checkout process may be started • User is only able to checkout their own cart

Name of Use Case:	Update Inventory
Description:	Allows stores to add or change items and their corresponding stocks. This ensures that users are only purchasing items which are in stock and will not arrive at the store with a surprise that one or more items are missing. The expected outcome is that the database is

	successfully updated, and the public will be able to view items in stock.
Actors:	• Primary Actor: Registered store • Secondary Actor: System
Preconditions:	• Store account is logged in with verified account
Postconditions:	• Store items are available for purchase
Flow:	1. Store navigates to inventory page 2. System displays current items in store inventory as well as form for new item 3. Store inputs new item information including item title, description, size, and image, and submits form 4. System receives form and validates data 5. System creates new item in item database with the requested information 6. System notifies user that new item was successfully added 7. User returns to step 2
Alternative Flows:	In step 2 of the normal flow, if the store chooses to update existing items: 1. Store inputs new stock number 2. System receives form and validates data 3. System updates item in item database with the new value 4. System notifies user that the item was updated 5. User returns to step 2 of the normal flow
Exceptions:	In step 4 of the normal flow, if the item already exists: 1. System warns user that the item already exists 2. User returns to step 2 of the normal flow
Requirements:	• Store is only able to modify its own inventory

Name of Use Case:	Delete Comment
Description:	Allows administrators to moderate the site by deleting abusive or malicious comments. It is up to the administrator to determine what content qualifies as abusive or malicious. The expected outcome is for the comment to be removed and the offending user to be notified of their offense.
Actors:	• Primary Actor: Administrator • Secondary Actors: ○ System ○ User

Preconditions:	• Administrator is logged in • Comment exists
Postconditions:	• Comment no longer exists • Comment's owner is notified of removal
Flow:	1. Administrator navigates to recipe page with comments 2. System serves recipe and comment content where each button has a button to edit or delete 3. Administrator clicks button to delete comment 4. System receives and validates request to delete comment 5. System removes comment from the comment database 6. System notifies administrator that the comment was removed 7. System sends email to comment owner that their comment was removed
Alternative Flows:	In step 3 of the normal flow, if the administrator chooses to edit the comment: 1. Editable form appears with current comment content 2. Administrator submits comment form 3. System receives and validates request to edit comment 4. System updates comment in the comment database 5. System notifies administrator that the comment was successfully edited
Exceptions:	In step 4 of the normal flow, if the comment does not exist (e.g., malformed query, already deleted) 1. System warns administrator that the requested comment does not exist
Requirements:	• Unprivileged users cannot remove comments

Name of Use Case:	Delete Account
Description:	Allows administrators to moderate the site by deleting abusive or malicious accounts. It is up to the administrator to determine what content qualifies as abusive or malicious. The expected outcome is for the account to be removed and the offending user to be notified of their ban.
Actors:	• Primary Actor: Administrator • Secondary Actors: ○ System ○ User

Preconditions:	• Administrator is logged in • User account exists
Postconditions:	• Account no longer exists • Account owner is notified of ban
Flow:	1. Administrator navigates to user account page 2. System serves user account details with button to delete account 3. Administrator clicks button to delete account, and clicks again to confirm 4. System receives and validates the request to delete account 5. System removes account from the account database and adds it to the deleted accounts database 6. System notifies administrator that the account was successfully removed 7. System sends email to account owner that their account was removed
Alternative Flows:	In step 3 of the normal flow, if the administrator does not confirm account deletion: 1. System takes no action and return to step 2
Exceptions:	In step 4 of the normal flow, if the account does not exist (e.g., malformed query, already deleted) 1. System warns administrator that the requested account does not exist
Requirements:	• Unprivileged users cannot remove accounts • Deleted account information should never be completely removed, so that banned users cannot sign up with the same email again