CookUps Design Specifications

by

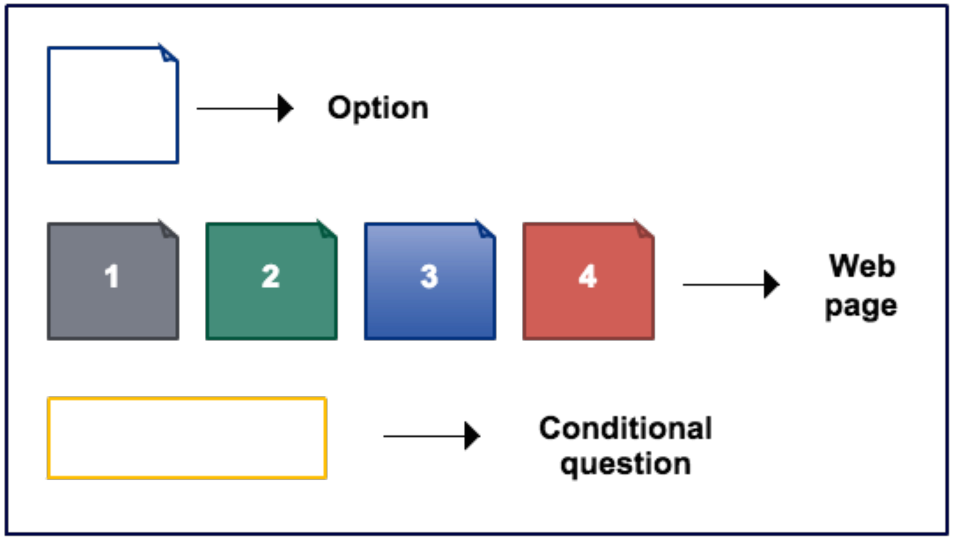
Team Cinnamon Crickets

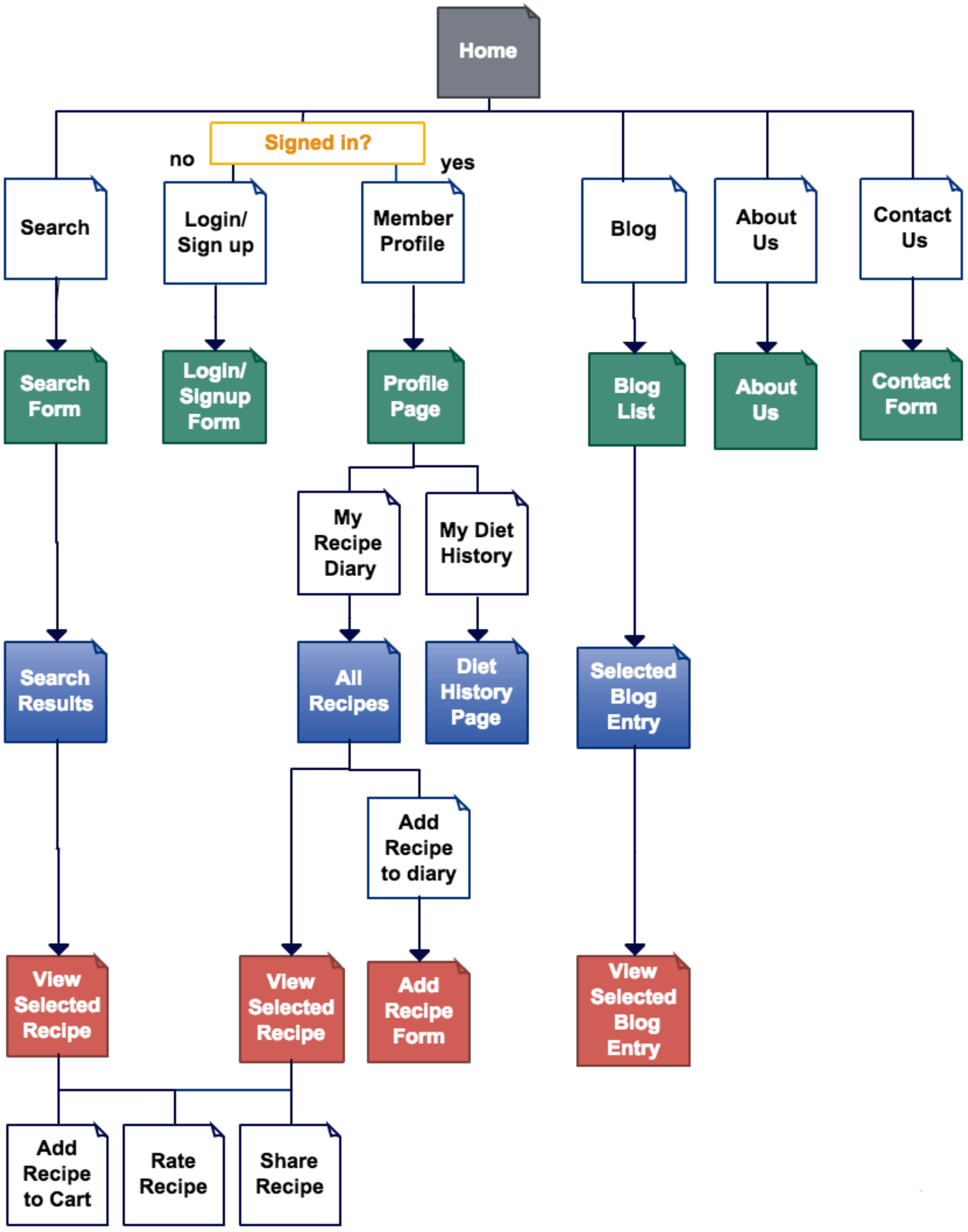
Summary

Think of Cookups as the "Google for Recipes"! Cookups allows you to search for recipes based on your available ingredients. CookUps generates recipes for your ingredients and other preferences, such as calorie count, preparation time, preferred cuisine, dietary restrictions and food allergies. Once a recipe is found CookUps will make a shopping cart for you of needed ingredients. If you have everything, great; if not, our shopping cart makes it easy to run to the store and pick up missing ingredients. We give our users convenience, tasty food and fitness under one roof.

CookUps also allows members to add their recipes to CookUps database or to their private recipe diaries so that members may maintain their own cookbooks online and also search for recipes that CookUps members have uploaded. Cookups makes cooking fun by adding a social component to cooking as our members may add, share, rate and recommend recipes on Cookups and talk about cooking tips on our forums. Cookups' membership makes CookUps highly personalized as members can keep track of food they have cooked in past and bookmark their favourite recipes.

Site Flow


Option
1
2
3
4
Web page
Conditional question


Home
Signed in?
no
yes
Search
Login/ Sign up
Member Profile
Blog
About Us
Contact Us
Search Form
Login/ Signup Form
Profile Page
Blog List
About Us
Contact Form
My Recipe Diary
My Diet History
Search Results
All Recipes
Diet History Page
Selected Blog Entry
Add Recipe to diary
View Selected Recipe
View Selected Recipe
Add Recipe Form
View Selected Blog Entry
Add Recipe to Cart
Rate Recipe
Share Recipe

Libraries, Frameworks, APIs and Modules

1. Node Modules:

* body parser - Node.js body parsing middleware
* connect-flash - Allow for an action/event to occur without having to reload a page
* cookie-parser - Parse cookie header and populate req.cookies with an object keyed by the cookie names
* debug - Node's core debugging technique
* ejs - Embedded JavaScript template
* express-session - Allow for sessions to occur / saved on cookie
* morgan - Http request logger middleware for node
* request - Designed to be the simplest way possible to make http calls. It supports HTTPS follows redirects by default.
* server-favicon - Node.js middleware for serving a favicon
* stylus - Robust, expressive, and feature rich CSS superset
* postgres - Node.js module for postgreSQL access

2. PostgreSQL Library

* To store information about user information,
* To store recipes that are being uploaded by users that have signed up on the cookups website,
* To search for recipes,
* Our database contains four tables, which are used to handle user accounts, user recipes, and user pantries.

3. Express web development framework

4. Bootstrap (HTML, CSS and JS framework)

5. jQuery - javascript library

6. Yummly API

7. AJAX

Individual components

Views and Routes

* Home page, depending on whether or not the user is a member, will use the DBModule.GetUserById. Routes can be /index.html brings to the Cookups. It can go to /search. Will be rendered with index.ejs and use home.js. This will be the base layout view for the rest of the routes, most routes will be able to route to the same places this one routes too, show functionality.

* /Search (Possible) Will have all the routes and views of home page (if we add), with collaborate with the search.ejs and use the search.js and the DBModule.searchIngredient.

* /viewrecipe Will display a recipe with all available details, including instructions, recipe photo, attributes, and createdbyuser.

* /loginpage works so that people without an account can make one and renders with the view of login.ejs.

Views and Routes

* /aboutus is rendered with the view aboutus.ejs, and contains information regarding the incredible team of Cinnamon Crickets, how we formed, 326 class, o on and so forth, similar to /index.

* /viewprofile Renders a public view of a users profile, accessed from a user profile link on a recipe page.

* /privateprofile Renders the users private editable profile. It makes use of the privateprofile.ejs and privateprofile.js files, as well as DBModule.getPersonalProfile and DBModule.AddUserInformation if a user makes changes to their profile inforation, including password, picture, and bio info.

* /register so those without a membership can create one, and collaborates with register.ejs and uses the DatabaseModule.addUser method, has a similar view to /login and will be similarly simple.

Views and Routes

* /addrecipe the page can be accessed by members only and uses the DatabaseModule.addRecipe method, if not a member you will be redirected to /login and this will render the view of addrecipe.ejs

* /logout just deletes session and redirect to /index and will be with the view.

* /blog with render blog.ejs and will have not a lot of js to it, mostly be posts from time to time about updates.

* /contactus (possible), or this may be combined with the /aboutus section, anything in here would be similar to aboutus.

* /Mydiet and /Myhistory are possible pages to be made, or we could dynamically create them, but they would be rendered with mydiet.ejs and myhistory.ejs respectivly.

Database

* Users: As people register to use CookUps.org their userdata is stored in the CookUps.org's local database. This information is used whenever a user attempts to login into Cookup.org. It is also used to link individual users to their favorite and added recipes.
* Recipes: The recipes that users add to CookUps.org are stored in the local database and modeled after the recipe objects provided by the Yummly api. These recipes are accessed via Cookup.org's search function as well as user's recipe diary.
* Favorites: User's are able to flag their favorite recipes that they've found on CookUps. These recipes are mapped to the user within the database and are directly accessable to that user.
* User's Recipes: Users have the option to store recipes in the local database. These are automatically flagged as favorite recipes for that user and are accessible to every user on Cookups.org via the search function.
* Pantry: Each user can store ingredients within the local database that they wish to use frequently in the searches they make.

CookupsDB module:
Handling all database access, this module makes SQL requests through the pg module. Alongside the Yummly Module, It is used within the MergedAccess Module.

* AddUser: Used in a new user page, the passed in userInfo JSON will provide the information needed to store a new user. Required fields are userName, email, and password.

* LoginUser: A request and callback are passed into the function. The user's ID and password taken from the request will be used to query the database. On a successful return the callback will validate the user's session.

* GetUserById: A database Lookup for a users public profile information. This call is used when a user profile page is generated. The userId comes from the link to the users profile.

* GetPersonalProfile: A database lookup for a users private profile. This information will be used to generate a users own profile page, where they can modify user information

CookupsDB module:
* AddUserInformation: Called when a user wants to commit a change to profile, including email or password change. Will replace existing info in db, if exists.

* AddUserFavorite: A recipe Id, taken from recipe page, is added to users favorites array

* RemoveUserFavorite: A recipe Id, taken from favorites list page, is removed from users favorites array

* SearchRecipes: Each search paramater in the object is optional. This allows searches to range from general by name to specific by ingredients, taste, etc.

* GetRecipeById: Looks up a single Recipe By Id. Used to generate a recipe page

* AddRecipe: With an associated User id, a new recipe is added to the database. Required fields are UserId, recipeName, Instructions, ImagePath.

CookupsDB module:

* DeleteRecipe: All Recipe Data related to Id, including entries in users favorites, is removed from database

* GetUserRecipesById: Using a UserId, all recipes created by a user are returned. The object returned contains an array of objects identical to GetRecipeById

* GetUserFavorites: Using a UserId, a list of users favorite recipes is returned. Return objects are modeled as in GetUserRecipesById

* GetUserPantryById: Using a UserId, an array of ingredients is returned. Thse are the users pantry list

* AddToUserPantry: An object containing a UserId and an array of ingredients to add is inserted into the users pantry in database

Yummly module:

Using a request object from the MergedAccess Module, a RESTful request is built and sent to yummly. The response object will return yummly's search results back to the MergedAccessModule.

* SearchRecipes : this will build a RESTful call for the yummly api. The resulting object comes from yummly

* GetRecipeById : By using a Yummly Recipe Id, we use the Yummly API to request that recipe object

Yummly API

We use a serialized form to get search based on the following parameters:
* preferred ingredients, excluded ingredients, allergies, dietary restrictions, cuisine types, allowed course, excluded course, allowed holiday, flavor, nutritional value and number of calories.

MergedAccess Module:
Accessing both the CookupsDB module and the Yummly module, this custom module returns user information and recipe information. Recipe data responding to search requests are returned from both the CookupsDB module and the Yummly Module are combined for use within appropriate routes.
* Login User
* AddUser
* GetUserById
* GetPersonalProfile
* AddUserInformation
* AddUserFavorite
* RemoveUserFavorite
* GetUserPantryById
* AddToUserPantry
* AddRecipe
* DeleteRecipe
* SearchRecipes
* GetRecipeById
* GetUserRecipesById
* GetUserFavorites

Connection and communication between parts of the system

Cookups.org has multiple backend modules that it works with, MergedAccessModule, YummlyModule and CookupsDBModule.

The general design of Cookups.org is that our front end pages will communicate directly to with the MergedAccessModule which then uses both YummlyModule and CookupsDBModule to simultaneously work with our database and the Yummly API. The modules then return the data to the web pages and present them to a

Relationship to functional specification

Our sitemap and pages are carefully drafted to fulfill all the functional specifications listed in functional specifications in assignment 3

Implementation challenges

We have already started the implementation and the main challenges we see are in simultaneously querying our internal database and Yummly API, and then ranking the search results from the two databases. We also need to maintain member accounts, member's recipes and member's diet history. Also, we shall have blog posts, so we shall need our back-end and database to be fairly well designed and effective.

In terms of front-end, we want to have a premium design and user-friedly interface, so we'll have to try a number of different design considerations.

Task Allocation

Front End:

Albert & Evan:
* Create the base product for each of the web pages. Create and handle log in and sign up page to the site.

Mina:
* Implement bootstrap theming. Design form for searching recipes and to display web pages for individual recipes and recipe search results.

Patrick:
* Formatting established web pages
* Handling fine styling to give a polished look and feel of the application

Back End:

Jon & Isaac:
* Designed the database Database Backup, Database Access Module (CookupsDB module)
* Synchronising search between yummly and our database (MergedAccess Module)

Mina:
* Send query parameters to yummly api and handle the search results.Design the portion of the backend regarding yummly calls(YummlyModule)
* Handle the front-end and back-end portion concerned with recipe search in Yummly API and display the search form and results on the front-end.

**Server Configuration: **
Gabriel:
* Handling the domain name and the requests that come into the domain

System Overview and architecture

The application a node application using the express framework. Our database is using postgreSQL. We will be using our database for storing recipes added by users and user information. For non-stored recipes we are using the Yummly API. Our front end pages will make one call to the MergedAccessModule and that will simultaneously search our database and the Yummly Database returning results.

End