KetoHub - Design Doc

v1.0 - The Hubbening

Overview

KetoHub is a web app that aggregates keto recipes.

Goals

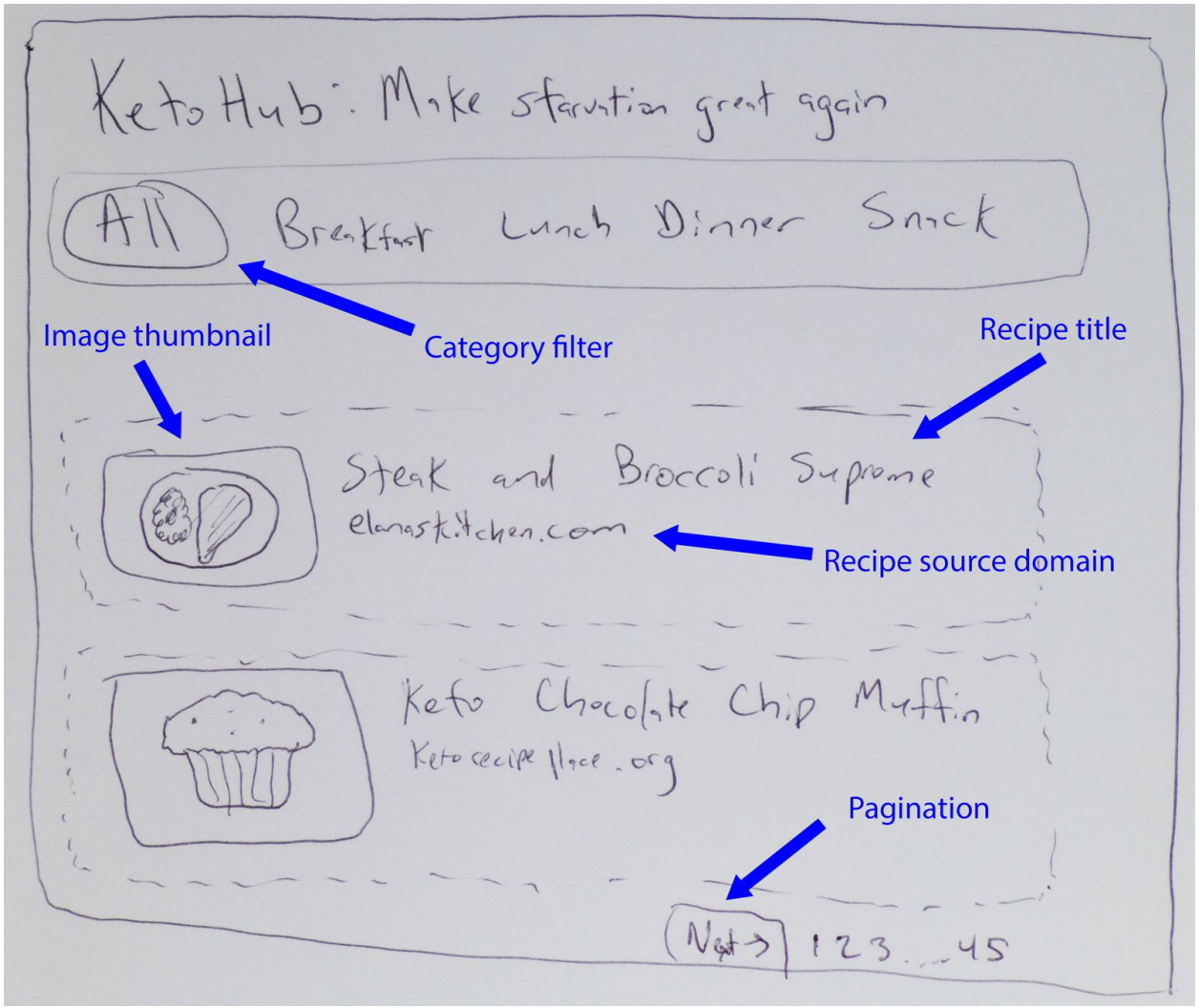
- Aggregate two keto recipe sites
- Support recipe titles
- Support recipe image thumbnails
- Support filtering by meal type (e.g. breakfast, lunch, dinner, snack)
- Web app renders properly on desktop and mobile on all major browsers

Non-Goals

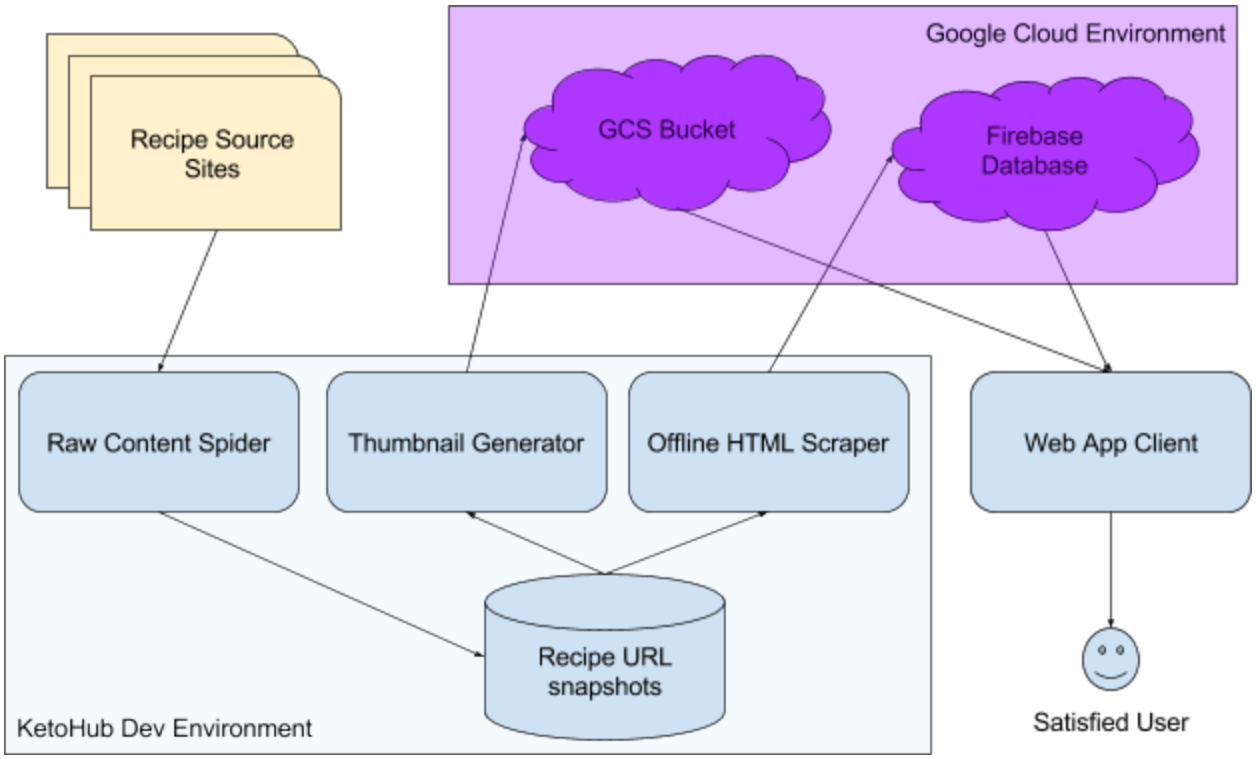
- Support macronutrient information (e.g. carbs, calories)
- Support recipe prep time
- Support recipe steps (the contents of the recipe itself)
- Cloud-based scraping

Wireframe

This is a mockup that provides a rough idea of what the KetoHub web app will look like:


Keto Hub: Make starvation great again
All
Breakfast
Lunch
Dinner
Snack
Image thumbnail
Category filter
Recipe title
Steak and Broccoli Supreme
elanaskitchen.com
Recipe source domain
Keto Chocolate Chip Muffin
Keto recipe place .org
Pagination
Next
1 2 3 ... 45

Architecture



Recipe Scrapers

The recipe scrapers are the backend of KetoHub. They download and parse information from the keto recipe source sites and insert them into the database.

The scrapers consist of three separate binaries that the operator can use in sequence as follows:

1. Run the **raw content spider** to spider the source sites and download snapshots, consisting of the rendered HTML and the recipe image for each recipe.
2. Run the **thumbnail processor** to generate thumbnails from the raw recipe images and upload them to Firebase image storage.
3. Run the **offline HTML scraper** to parse the recipe metadata from the rendered HTML for each recipe and upload it to the Firebase Realtime Database.

Recipe Sources

KetoHub v1 will only scrape recipes from the following two recipe sites:

- https://www.ruled.me/keto-recipes/
- https://www.ketoconnect.net/recipes/

Scalability

For v1, it is not that important to design for scalability to many recipe sources.

Where it's cheap to do so, avoid implementation choices that will make it difficult to scale beyond the initial two recipe sources.

Framework

The scrapers will use the scrapy framework for spidering sites, downloading rendered HTML, and scraping HTML.

Raw Content Spider

The raw content spider crawls each target site to identify the pages that contain recipes, then downloads snapshots of the recipe pages to the local filesystem.

Folder name schema

The spider should download raw content to the local filesystem according to the following schema:

- `[download_root]/YYYYMMDD/hhmmssZ/[source_domain]/[relative_url]/`

Key:

- `YYYYMMDD/hhmmss` - Time of scrape start, in UTC
- `download_root` - Root directory of scrape output (provided as command-line flag)
- `source_domain` - Domain name of scrape source (e.g. ruled.me)
- `relative_url` - Relative URL of the recipe page (e.g. `/recipes/keto-cupcakes/`)

Example:

- `ketohub-downloads/20170821/215304Z/ruled.me/chicken-pesto-roulade/`

Snapshot Contents

The recipe page snapshots contains two files from each recipe source page:

1. Rendered HTML

2. Main recipe image

`index.html`
The rendered HTML should be saved as `index.html`.

"Rendered HTML" here refers to the HTML as viewed within a browser after the page completes loading. This is not to be confused with the *raw* HTML that a tool like `wget` would produce, which contains the HTML before any JavaScript within the page has modified it.

`main.[jpg|png|gif]`
The main recipe image should be the top image shown for the recipe. Ideally, this should be the full-size version of whatever image the recipe source site chooses for the recipe thumbnail in recipe index views.

Thumbnail Generator

For each recipe, the thumbnail generator should:

1. Resize the recipe image to 340px in width.
    a. If resizing to 340px in width makes the height longer than 230px, crop from the center to reduce height to 230px (115px up, 115px down from the center).
2. Convert the recipe image to JPEG format.
3. Upload the file to the KetoHub GCS bucket.

The thumbnails should be uploaded to the following GCS path:

- `gs://ketohub/[key]_thumbnail.jpg`

Example:

- `gs://ketohub/ruled-me_chicken-pesto-roulade_thumbnail.jpg`

Recipe Keys

Each recipe has a unique key of the following schema:

- `sanitized(strip_prefix('www.', domain))_sanitized(title)`[1]

Where `sanitized` means:

- Convert all characters to lowercase

[1] We want to eliminate any characters that are prohibited in Firebase keys (`$ # . [ ] /`) or on UNIX/Windows filenames.

- Replace all non `a-z0-9` characters with `-`

Example, for https://www.ruled.me/chicken-pesto-roulade/:

1. `sanitized(strip_prefix('www.', 'www.ruled.me'))_sanitized('Chicken Pesto Roulade')`
2. `sanitized('ruled.me')_sanitized('Chicken Pesto Roulade')`
3. `ruled-me_chicken-pesto-roulade`

Offline HTML Scraper

The offline HTML scraper reads the latest data the raw content scraper downloaded and parses it to generate metadata about the recipes. The scraper then inserts this information into the KetoHub database.

The scraper parses two pieces of information: the recipe title and recipe category: **title** and **category**.

Recipe Title

The recipe title is the title of the recipe without any flavor text. For example, the title of "Oven Baked Fish | Ginger Lime Whiting!" is just "Oven Baked Fish."

Recipe Category

Different recipe sites use different categories. KetoHub will map the site-specific category to a canonical KetoHub category with the following mapping:

KetoHub Canonical Category	**ruled.me Category**	**KetoConnect Category**
breakfast	Breakfast	Breakfasts
entree	Lunch	
entree	Dinner	Main Dishes
dessert	Dessert	Desserts
snack	Snacks	Snacks
side	Side items	Side Dishes
condiment	Condiments	
beverage		Beverages

Note: We are deliberately ignoring multi-category items for now. For example, milkshakes are both a dessert and a beverage, but KetoHub will use whatever single category the source site uses.

Image Storage

KetoHub stores image thumbnails in Firebase Cloud Storage.

Database

KetoHub stores recipe metadata in the Firebase Realtime Database.

Schema

KetoHub's schema is based on recommendations in "Structure Your Database."

- `recipes`: Contains all recipes, keyed by original URL (without http/https protocol prefix)
  - Recipe key: See Recipe Keys section.
  - `title`: (string) Original recipe title
  - `url`: (string) Original recipe URL
  - `category`: (string) Recipe category
- `categories`: Contains categories as keys and matching recipe keys as key-value pairs, where the value is always `true`.

Example

```
{
  "ruled-me_chicken-pesto-roulade": {
    "title": "Chicken Pesto Roulade",
    "url": "https://www.ruled.me/chicken-pesto-roulade/",
    "title": "Chicken Pesto Roulade",
    "category": "entree"
  },
  "ruled-me_keto-bacon-burger-bombs": {
    "title": "Keto Bacon Burger Bombs",
    "url": "https://www.ruled.me/keto-bacon-burger-bombs/",
    "category": "snack"
  },
  "ketoconnect-net_oven-baked-fish": {
    "title": "Oven Baked Fish",
    "url": "https://www.ketoconnect.net/recipe/oven-baked-fish/",
    "category": "entree"
  },
  "ruled-me_keto-gelatin-rum-shots": {
    "title": "Keto Gelatin Rum Shots",
```

```
    "url": "https://www.ruled.me/keto-gelatin-rum-shots/",
    "category": "dessert"
  },
  "ketoconnect-net_irish-coffee": {
    "title": "Keto Irish Coffee",
    "url": "https://www.ketoconnect.net/recipe/irish-coffee/",
    "category": "beverage"
  }
}
```

Web App

The web app will be an AngularJS app, based on the angularfire package to use Firebase APIs.

For each recipe, the app will display a panel containing the following:

- Recipe thumbnail image
- Recipe title
- Recipe domain name

Clicking on the panel will open the recipe URL in a new tab.

Filters

KetoHub allows the user to filter recipes by category. The user can only filter by one category at a time or view all recipes (no filter).

Data Retrieval

The web app will retrieve image thumbnails from Firebase Cloud Storage. The web app will retrieve all other recipe metadata from the Firebase Realtime Database.

Github Repositories

- Raw Content Spider
- Thumbnail Processor
- Offline Scraper
- Web App

Environment

Backend code must be compatible with Ubuntu 16.04 Server LTS.

Frontend code must render properly in the latest versions of the following browsers:

- Chrome
- Firefox
- Edge
- Safari

Frontend code must render properly on the latest versions of the following operating systems:

- Windows
- Mac OS X
- Android
- iOS

Testing

Every component should have unit test coverage. Degree of coverage is at the developer's discretion.

Code Style

All code uses the Google style guides:

- Python Style Guide
- JavaScript Style Guide

Continuous Integration

Each Python project should include a `build` file that does the following:

1. Run unit tests and calculate code coverage
2. Run the latest version of YAPF with the Google style guide.
3. Run pyflakes on the source.
4. Run DocStringChecker on the source.

Each Python project should include a `.travis.yml` file that installs all necessary dependencies for the project and runs the `build` script.

See https://github.com/m-lab/ndt-e2e-clientworker as an example.