# **Final Project – Proposal**

## **Overview**

The project I will be making will be a calculator with conversion methods. I chose this project because I am always converting cooking measurements while cooking, and sometimes I don’t know the conversion or the equivalence between the measurements, so this web application solves a real-world problem I have, and it will go well with the website project I created for the previous class: a recipe sharing website called “Cookbook Corner.”

## Target Audience

The target audience will be chefs, amateur cooks, and housewives, or househusbands who cook for their families.

## Major Functions

1. Unit Conversion Engine:
   * This function will handle the conversions between different units: length, weight, volume; for construction: feet to meters, pounds to kilograms; for cooking: cups to milliliters, ounces to grams. It will take input values, source units, and target units, and then apply the conversion factors.
2. Monetary Conversion Handler:
   * This function will perform currency conversion using real-time exchange rates. It will accept input amounts, source currency, and target currency, fetching rates from an external API, and return the converted amount with proper rounding.
3. Calculation History Manager:
   * This function will store and display the user’s calculation history. It will save input, conversion types, and results. Possibly allow the user to reuse past calculations. It will store and retrieve through local storage.
4. Input Validation and Parsing:
   * This function will validate and parse user inputs (numerical values from unit types or currency codes). It will ensure inputs are valid (no negative measurements for construction, valid currency codes).
5. User Interface Renderer:
   * This function will manage and render the web interface, including the input fields, dropdowns for unit/currency selection, and display results.
6. Error Handling and Feedback:
   * This function will catch errors like invalid inputs, API failures and provide user-friendly feedback. It will ensure the application continues to work and communicate issues clearly to users.
7. Settings and Preferences Manager:
   * This function will handle the user preferences like default units (metric vs. imperial), preferred currencies, or display formats (decimal places). It will store it locally using local storage.
8. Category-Specific Formula Library:
   * This function will manage specialized calculations for each domain: construction, cooking, and monetary.

## Wireframe

Desktop view: https://app.moqups.com/4EKXYGdwp8KxTsLXUdDTY9OD2Grce5Xb/view/page/ad64222d5

Mobile view: https://app.moqups.com/4EKXYGdwp8KxTsLXUdDTY9OD2Grce5Xb/view/page/ab2b56d63External Data

Unit converter: https://rapidapi.com/goforamit16/api/unitconversion

* Will store mathematical unit conversions: area, data, length, mass, pressure, temperature, time, and volume.

Currency exchange rate: <https://fixer.io/>

* Will store exchange rates: USD, EUR, BRL, and so forth

Food Unit of Measurement Converter: <https://rapidapi.com/smilebot/api/food-unit-of-measurement-converter>

* Will store food unit measurements: cups, milliliters, pounds, tablespoons, teaspoons, and so forth.

## Module List

1. Conversion Core Module:
   * This module will handle the core logic for all conversions, being the backbone for the Unit Conversion Engine and Monetary Conversion Handler functions.
   * The constructor will be called Unit Converter.
   * This constructor will convert currency, construction, and cooking measurements.
2. Input Processing Module:
   * This module will validate and parse the user inputs to ensure they are valid and secure, aligning with the Input Validation and Parsing function.
   * The Constructor will be called Input Validator.
   * This constructor will parse user input and maintain a list of supported units and currencies from the API provided.
3. User Interface Module:
   * This module will manage the front-end rendering and user interactions, corresponding to the User Interface Renderer function.
   * The constructor will be called Conversion Form.
   * The Constructor will dynamically inject a form with input fields for values, and dropdowns for units/currencies. It will display results and allow the user to switch between modes.
4. History Management Module:
   * This module will store and retrieve past calculations, implementing the Calculation History Manager function.
   * The constructor will be called History Manager.
   * The constructor will save conversion details using local storage, fetch and display past calculations, and allow the user to clear or filter history entries.
5. Formula Library Module:
   * This module will manage domain-specific calculations, supporting the Category-Specific Formula Library.
   * The constructor will be called Formula Module.
   * The constructor will be responsible for handling the construction calculations, cooking formulas, and monetary formulas.
6. Error Handling Module:
   * This module will manage errors and provide user feedback, aligning with the Error Handling and Feedback function.
   * The constructor will be called Error Handling.
   * The constructor will catch and process API errors and show user-friendly messages for invalid inputs.
7. Settings Module:
   * This module will manage the user preferences, implementing the Settings and Preferences Manager function.
   * The constructor will be called Preferences UI.
   * The constructor will save the user's settings using local storage, provide a settings panel for customization, and apply settings to conversions.
8. Utilities Module:
   * This module will provide shared utilities to support other modules and ensure maintainability.
   * There won’t be any constructors for this module. This module will format numbers and units for display, manage communication between modules, and cache API responses and local conversions to reduce redundant calls.

## Graphic Identity

Typography:

1. Primary Font: Roboto
   1. Will be used for body text (form labels, result displays, history entries).
2. Secondary Font: Open Sans
   1. Will be used for the headings (“History” or “Settings”), category tabs (“Calculator,” “Currency,” and “Construction”).
3. Accent Font: Montserrat
   1. Will be used for the App Title in the header and footer credits.

Color Scheme:

1. Primary Color: Warm Gray (#D3D3D3)
   1. Will be used as a background color for the entire page, form fields, and non-interactive areas.
2. Secondary Color: Deep Teal (#006666)
   1. Will be used for the Text color for the body text (labels, results), inactive category tabs, and footer links.
3. Accent Color: Burnt Orange (#CC5500)
   1. Will be used for buttons (“Convert,” “Clear History”), active category tab, and hover effects.

Application icon:

Since the application is about calculating and converting, I will find a simple, but clean and modern calculator icon.

## Timeline

By the end of week 5, I will have the HTML finished, with most CSS applied to it. I will have dynamically injected the normal calculator semantics using JS, and all functionalities of this calculator will be working.

By the end of week 6, I will have the following API’s working: cooking measurements, currency exchange rate, along with all of their functionalities.

By the end of week 7, I will deliver the project with the last API fully functional (construction measurements), along with all the error-handling feedback, history management, and all the remaining functions that will support the application.

## Trello Board

https://trello.com/b/3D2RI1d3/calculator-master

## Challenges

Some of the challenges I foresee in this project will be to implement the API data into my application and make sure the website is responsive to the user's inputs.