

Making Recipes Readable Again

DESIGN DOCUMENT

Team Number: 12

Client: Mat Wymore

Team Leader: John Paton

Meeting Scribe: Bret Knous

Design Facilitator: Vismay Gehlot

Test Facilitator: Luke Knous

Report Facilitator: Rithwik Gokhale

E-mail: sddec21-12@iastate.edu

Website: <https://sddec21-12.sd.ece.iastate.edu>

Revised: 12/8/2021

Development Standards & Practices Used

There are two standards that have been considered for this project; engineering development standards and engineering coding standards. We shall be following the Agile method using two week sprints for the engineering development standards and we will be following the coding standards outlined by the below link. In addition, we shall be adhering to the COVID-19 safety guidelines of Iowa State University.

Coding Standards:

<https://www.sos.state.co.us/pubs/elections/VotingSystems/DVS-DemocracySuite511/documentation/SD-DVSJavaScriptCodingStandards-5-11-CO.pdf>

Summary of Requirements

Requirements:

- Must work on a typical mobile device/smartphone
- May be platform specific (e.g. Android), though cross-platform is ideal
- Interface must be user-friendly (typical user is expected to be viewing device from roughly one meter away and using one non-primary finger to navigate)
- Capable of loading/upgrading a recipe from an arbitrary URL
- "Upgraded" recipes should load in 5s or less
- Solution should not interfere with the source's revenue stream (e.g. ads should still display)
- Should not require a user account (may be optional if it would help with desired features)
- Should not require backend infrastructure (server)

Applicable Courses from Iowa State University Curriculum

1. COM S 309 - Software Development Practices
2. COM S 363 - Introduction to database management
3. COM S 227 - Introduction to Object oriented programming
4. COM S 228 - Data Structures
5. COM S 319 - Construction of User Interfaces
6. CPR E 388 - Android development
7. COM S 311 - Introduction to Algorithms

New Skills/Knowledge acquired that was not taught in courses

1. Javascript
2. React Native Development
3. Jest Javascript testing frameworks
4. Project Management and team-work through remote interactions
5. Maintaining COVID-19 health standards
6. Expo

Table of Contents

1 Introduction	6
1.1 Acknowledgement	6
1.2 Problem and Project Statement	6
1.3 Operational Environment	6
1.4 Requirements	7
1.5 Intended Users and Uses	7
1.6 Assumptions and Limitations	8
1.7 End Product and Deliverables	8
2 Project Plan	9
2.1 Risks And Risk Management/Mitigation	9
2.2 Project Tracking Procedures	10
3 Design	10
3.1 Previous Work And Literature	10
3.2 Design Thinking	12
3.3 Proposed Design	12
3.4 Technology Considerations	14
3.5 Development Process	15
3.6 Actual Design	15
3.7 How the design changed	17
4 Testing	17
4.1 Unit Testing	22
4.2 Interface Testing	22
4.3 Acceptance Testing	22
4.4 Results	23

5 Implementation	23
5.1 Frontend	23
5.2 Backend	24
5.3 Web Scraping	24
6 Closing Material	25
6.1 Conclusion	25
6.2 References	27
7 Appendix	28
7.1 Appendix I	28
7.2 Appendix III	36

List of figures/tables/symbols/definitions

- Image 2.1 - Dependencies
- Image 2.2 - Project Plan (Spring)
- Image 2.3 - Project plan (Fall)
- Table 2.4 - Time Allocation
- Image 3.1 - screen mockups
- Image 3.6 - Actual app design
- Image 3.6.1 - Actual block diagram
- Image 3.7 - Proposed block diagram
- Image 4.1 - Jest test example
- Image 4.2 - Coverage from failed test
- Image 4.3 - Failed test received vs expected
- Image 4.4 - Html coverage from failed test
- Image 4.5 - Unreached lines and error from failed test
- Image 4.6 - Coverage from successful test
- Image 4.7 - Html coverage from successful test
- Image 4.8 - Fixed lines from successful test

1 Introduction

1.1 ACKNOWLEDGEMENT

Our client and advisor, Mat Wymore, is acknowledged for their contributions towards the design of this project and remaining available to provide assistance as needed.

1.2 PROBLEM AND PROJECT STATEMENT

Online recipe blogs and websites are often confusing to understand and take a long time to navigate with miscellaneous stories from the author and information other than the recipe itself. This project aims to reformat and modify what the user sees to make the recipes more comprehensible and easier to navigate.

For our project, we have developed a mobile app that makes these changes to a website based on a URL provided by the user. We were able to accomplish this by building a web scraping service which takes all the relevant information from a recipe website, reorganizes and displays it in a concise manner on our custom mobile application. This allows users to clearly understand the ingredients and instructions present in a recipe website.

This app was designed with the aim to simplify the process of following recipes and make the overall cooking experience more enjoyable. Additionally, we hope that recipes that once might have been difficult to understand can become more straightforward, allowing more users to utilize them.

1.3 OPERATIONAL ENVIRONMENT

This project was developed with the intention of online use only and is not expected to function without access to the internet. Because of this, users may have different experiences with the application depending on their internet connectivity. The constraint of needing the internet, and a decent internet connection, may have an effect on the location our application is used and limit some potential users. In terms of environmental constraints, this app was designed to work in the context of an active kitchen which has had a significant impact on a number of UI related decisions. UI elements and page layouts were decided with the idea to ensure limited interaction with the app when the user is in the kitchen and should be in a position to read the updated content clearly.

1.4 REQUIREMENTS

Requirements:

- Must work on a typical mobile device/smartphone
- May be platform specific (e.g. Android), though cross-platform is ideal
- Interface must be user-friendly (typical user is expected to be viewing device from roughly one meter away and using one non-primary finger to navigate)
- Capable of loading/upgrading a recipe from an arbitrary URL
- "Upgraded" recipes should load in 5s or less
- Solution should not interfere with the source's revenue stream (e.g. ads should still display)
- Should not require a user account (may be optional if it would help with desired features)
- Should not require backend infrastructure (server)

Specialized Features:

- Hyperlink to images of different ingredients for visual assistance
- Easy unit conversion for the different quantities of ingredients
- Relevant substitutes/alternatives for specific ingredients in select recipes
- Hyperlinks to online/nearby stores where rare/specific ingredients can be purchased
- Social and online sharing abilities for the recipes enhanced through the app
- Ability to bookmark and save the recipes which have been enhanced by the app for easy access in the future
- Ability to scale recipes, as in multiply or divide ingredients proportionally.
- Listing quantities of the different ingredients in the instructions part of the webpage

1.5 INTENDED USERS AND USES

Based on the base functionality and the benefits of the application that have been described above, this app is geared towards a wide range of end users. The primary objective of the product is to provide enhanced recipes to individuals who would benefit from any of the features which have been listed above. Therefore, it is safe to conclude that a large number of the end users for this app will be individuals who are just getting into culinary practices or beginner cooks. However, easy access to only the ingredients and instructions can also help save time for professional cooks when they are referring to

recipes. And the specialized UI scheme allows for the application to be used in the kitchen may encourage even professional culinary experts to benefit from the enhanced recipe app.

1.6 ASSUMPTIONS AND LIMITATIONS

Assumptions:

- 1) The end product can be used globally with no restriction
- 2) The make and model of a preferred smartphone will not affect app compatibility
- 3) There will be no limitations with access to the required software and APIs during development
- 4) The user puts in a URL address to one of the supported recipe websites

Limitations

- 1) This app will require constant internet connection since the websites are 'enhanced in real time'
- 2) Specific features in the app will also require constant GPS data to provide shopping information for the specialized ingredients
- 3) A new instance of the app will be loaded at each use due to the lack of external database (one of the functional requirements put forward by the client)
- 4) The end product will only work on mobile devices. Thus a web version of the product will not be available.
- 5) The application will only work on english websites
- 6) The application will only work on a list of supported sites found below. The app supports mainstream recipe websites which are used by 80% of the target users such as allrecipes.com

Note that assumption 4 and limitation 6 are caused because of the way our web scraping is handled. The number of websites that can be used is limited since each one has to be handled individually. The goal would be to eventually remove this limitation. See appendix below for more details.

1.7 END PRODUCT AND DELIVERABLES

Project Deliverables

- 1) The final product delivered to the client is a cross platform mobile application which successfully completes the base functionality of the application along with some partially implemented special features.
- 2) Design and product mockups were delivered to the client on a bi-weekly basis since the start of development.

2 Project Plan

2.1 RISKS AND RISK MANAGEMENT/MITIGATION

Planning - While we might have missed something in our planning process it will most likely be able to be fixed at a later time. Rating: 0.1

Testing - It is possible that bugs and other issues may escape our testing when our application is used on a wide variety of websites since they have a range of different build styles. 0.7

Mitigation Plan: The best way for us to deal with this is to test our application on as many different websites as possible, making sure it is tested on the largest and most commonly used websites. As long as it works on the websites that get the majority of the traffic it will not be as big of an issue if it doesn't work well on small obscure websites that few people use.

UI - The risks here are that the app might not be suitable for use in the kitchen or to people with disabilities like poor eyesight or color blindness. Rating: 0.5

Mitigation Plan: We will do what we can to make the app as readable as possible by using colorblind safe colors and making the font as large and readable as possible. Furthermore, the UI elements have been selected in a manner to ensure that the app can be navigated with the user's non-dominant hand/finger since it will largely be used in kitchen environments.

Feature Functionality - While it is possible for some of the libraries we use to depreciate otherwise have issues, this is unlikely and as long as our application is coded well the risks here should be minimized. Rating: 0.3

URL Safety - It is possible that a URL is not a valid URL, could redirect the user to an unintended site, or contain hidden JavaScript. Our project is a local application and our assumption is the user knows what is on the site they are trying to scrape. Therefore, this

should not be a large issue. This is categorized as a user decision from a risk perspective.
Rating: 0.3

2.2 PROJECT TRACKING PROCEDURES

As detailed below in the development processes, we employed an Agile form of software development and management. Development was completed in two week sprint periods and a MVP presentation at the end of each cycle. The complete project implementation was broken down into smaller sections such as front or backend and further into specific tasks which needed to be completed. These tasks were then picked by group members from a sprint backlog. The following technologies and project tracking processes were adopted for our project:

- 1) Discord - This is our IM channel for the project. Since all the team members and our client are more used to the environment and features of Discord, we have been using this as our primary mode of communication with the team and the client (as an alternative to Slack).
- 2) Trello - Our initial objective was to use Trello to manage the tasks that need to be completed in the two week sprint cycles. During the development cycle, our team kept regular notes on Google docs and were able to effectively handle tasks (tickets) directly from the document and hence depreciated the Trello use for this project. While task management through a document did inhibit the team from benefiting from the easy to use UI provided by Trello, we were able to meet the scope of the project through this form of tasks management as well.
- 3) GIT - Github is being used as our primary project code repository. This is a code management software which will allow multiple developers to collaborate on the project simultaneously. Github also provides the team and the client a method to keep track of code commits made by different team members to get a better understanding of the overall project progress.

3 Design

3.1 PREVIOUS WORK AND LITERATURE

Detailed research was conducted by the team members during the design phase of the project specifically into pre-existing solutions/products in the market which work with the same topic as our project. While we were not able to find any applications which provide the same service as our product, there are a number of applications in the market

which focus on improving the culinary experience with regards to finding and saving recipes from across the internet.

Therefore, while addressing the previous works and literature for this project, a direct comparison will not be made with another app which already exists in the market. Alternatively, the functionality, the strengths and weaknesses of the different products will be analyzed so that appropriate design decisions can be made during the implementation of our application.

1) Paprika

The main objective of this app is to provide users with the opportunity to save recipes from different web sources in one location for easy access. This app provides users with a built in web browser and the ability to bookmark any favorite recipes which they come across. The main strength of this application is that it provides users with the option to save all of their favorite recipes in one easy to access location. But beyond this main feature, the app does not provide users with any other options to further enhance the recipes which they are viewing. Our proposed solution will provide users the option to save recipes on the app while also providing other features to ‘enhance’ the recipes which they find on the web.

2) BBC Good Food

The main selling point for this app is the wide variety of recipes available which have been created by industry professionals and reviewed by thousands of users from across the world. This provides the end user of this app with a well curated range of recipes with feedback from other users to improve the overall cooking experience. Furthermore, users are able to bookmark/save their preferred recipes on the phone for easy access in the future. But on the other hand, this app does not specifically offer any other features for the users to customize the recipes to their needs and view only the relevant information. This would be the main difference between BBC Good Food and our proposed application.

3) Tasty

Tasty uses a crowdsourcing approach to its recipes where users are able to actively comment on the different sections of the recipe and offer suggestions and tweaks which can be adopted by others who view the same recipe in the future. This allows the app to provide a wide range of recipes with essential feedback and reviews provided by other users of the app. While this app focuses on improving the recipes which are currently available on the web, the approach taken by Tasty is very different from our proposed

solution which will not offer users the ability to rate and review recipes but provide them with machine generated improvements such as easy unit conversion, improved information layout and relevant visual aid.

3.2 DESIGN THINKING

During the design phase of the project, as a team, we focused on determining the most effective product as the solution for the problem presented by our client. Over multiple brainstorming sessions, we considered different options such as websites plugins extensions. After thorough evaluation, we came to the conclusion that a mobile application would be the best option. This option would provide the service to the largest population of users since smartphones have become a major part of modern day living. Furthermore, smartphones also come with preinstalled settings which are designed to improve the user's app experiences. The next step in the define phase was to brainstorm the different features which could be included in our app to provide the users with an 'enhanced' recipe. During this phase, the team conducted research on different existing apps in the market and discussed different services which can be provided by the app and can be translated into features. And the final step of the define phase was to start coming up with different UI mockups for the app and create a draft of the system design for the application. During this process, mockups for the home screen and the recipe screen were developed by the team to get a better idea of the placement of the different buttons and features on the app screens.

During the ideate phase of the project, our main goal was to finalize the project requirements (the features for our application), discuss the non-functional requirements of the project and conduct sufficient technology research so that we do not face many roadblocks during the actual implementation of the app. And the last step of the ideate phase was to finalize the design of the product in terms of system diagrams and UI mockups. This would adequately prepare us to begin development of the application in the next phase of the project.

3.3 PROPOSED DESIGN

During the design phase of the project, our team had the following objectives to complete:

- Identify all the functional and nonfunctional requirements
- Develop mockups of app UI

- Conduct research on technologies which will be used for this project

Before creating the design document for this project, the team finalized on the requirements (seen in chapter 1). Mockups were also developed for the home page and the recipe page of the mobile application. The objective of these mockups were to provide the team with an idea of how the app will look from a design perspective and to spark conversations between the team members and the client on where the different components would go on each of the screens and the overall structure and page layouts of the app. The following mockups are for the home screen and the recipe screens:

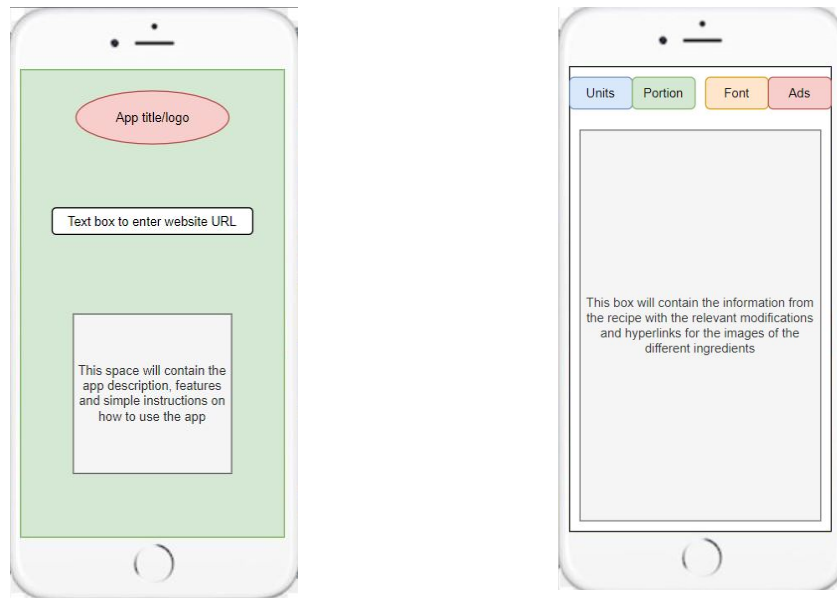


Image 3.1 - screen mockups

As seen above, the first mockup is of the home screen/URL screen. This will be the first page which opens up when the app is launched. Since our app prioritizes ease of use, the home screen will have very easy to understand structure. This page will contain the title/logo, a text box for the user to enter the recipe website URL and we had initially intended to put in app usage instructions but due to the super easy UI scheme employed for this application, the team decided to forgo this section to reduce in-screen clutter. The second screen mockup is of the actual recipe page itself. This page will contain the enhanced recipe once the app has processed all the information from the original website. A majority of the screen will be taken up to display the recipe information. This screen

also contains a number of button components at the top to use the different features provided by our app. These include the option to change units, change portion sizes, enable/disable advertisements and change the font to make it easy to read from long distances. As mentioned earlier, these mockups were designed to meet all the feature requirements of the app which were finalized with our client during the initial brainstorming sessions of this project. One of the major design changes which were made during the development phase of the project was to relocate the feature buttons to the bottom of the screen since this is a more easy to access location on the phone screen especially when considered from a kitchen scenario.

The design proposed above was developed after careful consideration on how they could help meet the functional requirements of this project which were provided by the client. As it can be seen in the above images, this program is designed to run on a standard iOS and Android smartphone. Therefore, it follows the basic design principles of mobile applications. Furthermore, another major requirement listed in chapter 1 is ease of use with regards to the UI. Hence, the different components on the screen have been labeled clearly to ensure that users are able to easily navigate through the app while cooking or in the kitchen.

3.4 TECHNOLOGY CONSIDERATIONS

Since it was determined that we will be developing a cross-platform mobile application which can function on both iOS and Android devices, we believe that using the React Native Framework is the most appropriate option. Developing the app through React ensures a smoother development process and reduces the additional effort of optimizing the app for iOS and Android devices. While independent development for the two OS would provide users with more features and an overall better app experience, this would have doubled the work for the team and increased the overall implementation time since we would have to learn different software and libraries which are native for Android and iOS development.

Testing was also a major part of this project. Before launching the application it is important to conduct a number of different tests to ensure that there are no logical errors or UI bugs in the code. Since the app is being developed with React, we were able to use a native Javascript testing framework called Jest. The use of this library provided us with the option to conduct unit tests and system integration tests on the app and ensure that the overall app experience remained smooth.

Another big decision made by the team was how to store our data. Many things were considered to implement local data storage. The biggest though had to be MongoDB. We considered this largely because it seemed simple and inexpensive to use and even had libraries to connect easily to React Native. However, this was also partially because our team thought JSON files would not work and were wrong on this assumption. In the end, we went with JSON files because they did in fact work, were easier to use, and much cheaper.

Figuring out how to save and read JSON files in React Native was also another challenge. Initially this was done in node js using the fs library. However this library was not compatible with React Native and so another compatible one needed to be found. Many were found and tried but in the end the library expo-file-system was used because we were using Expo and it was the one that worked best with it.

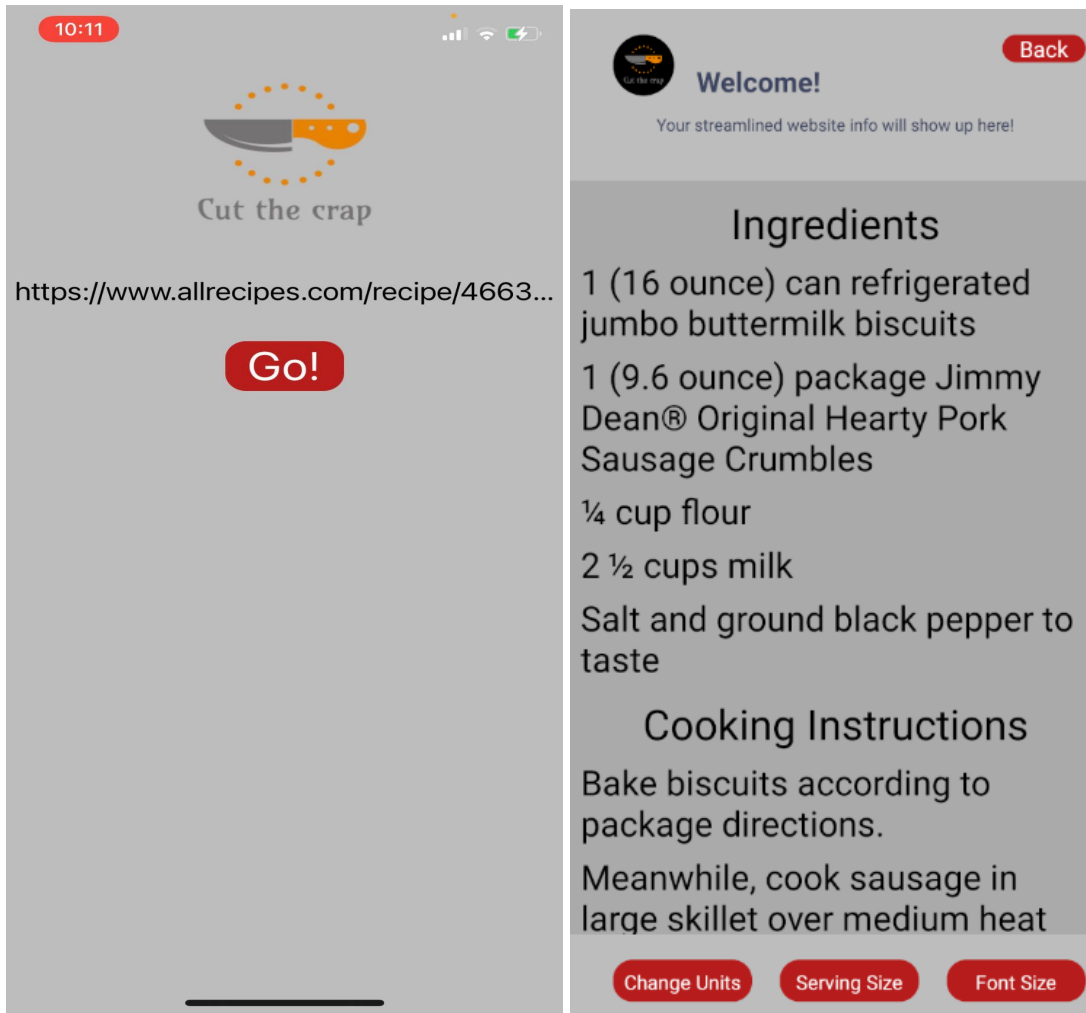
We decided to use Expo because it allowed for hot reloading, which meant that we could make any changes to the code and save it, and the app would show the changes in live time. In addition to this, it also incorporated an intuitive debugging environment and was overall easier to use due to the option of various emulator devices than other competitors.

3.5 DEVELOPMENT PROCESS

For the development process, we employed an Agile form of software development process. We had 2 week sprints throughout the development period and at each of these MVP meetings, we discussed the project progress and discussed the net tasks that would be picked and worked on from the sprint backlog. A minimum value product (MVP) was presented to the client and the rest of the team at the end of the two week sprints where items from the backlog were completed. This development process selected for this project ensured that the client and the team members had a realistic idea of the project progress and therefore were able to better plan out the next steps which need to be completed in the upcoming sprint cycle.

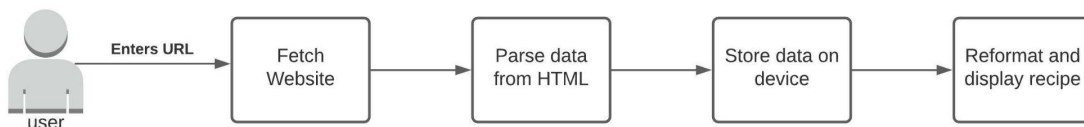
3.6 ACTUAL DESIGN

The initial design (seen in 3.3) was not changed too much in the final iteration, most if not all elements are still present in the final version. The only significant difference is that it was made to look more professional using Styled Components. This can be seen below.



3.6 - Actual app design

In terms of overall structure the project essentially utilizes three components. A web scraping function to pull and reorganize data from recipe websites, a backend to take the scraped data and store it, and a front end to retrieve data from storage and display it to the user. The block diagram below describes the path the data takes from beginning to end, from the request to being displayed properly.

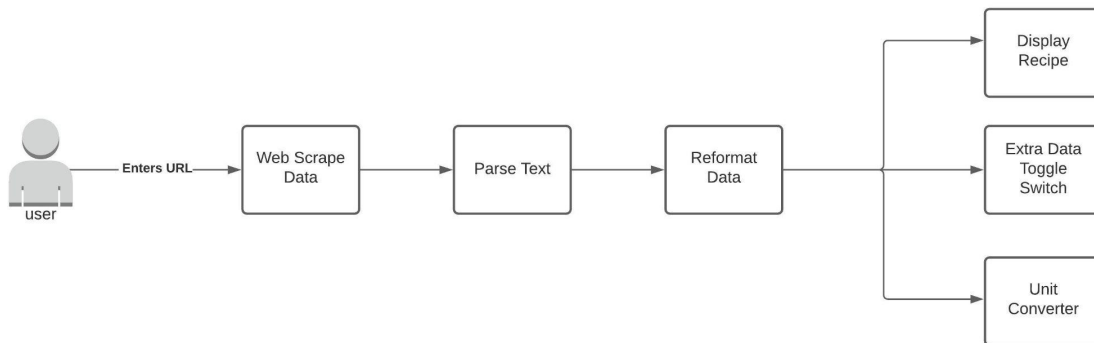


3.6.2 - final block diagram

The actions taken in each section will be discussed more thoroughly in their respective sections.

3.7 HOW THE DESIGN CHANGED

Our design had to go through a few changes in order for the critical path to be completed. The first change was with the front end design itself. We decided to change the color scheme used to be more readable and in order to create a theme for the app. This can be seen in the change from the original white/green background and light color buttons to the gray and orange theme. The second big change was with the features included in the final deliverable. We needed to drop some of the planned features, such as unit conversion and scaling, in order to ensure quality main functionality. Overall though, the original design for how each of the pieces remained largely the same. The big difference was that originally we thought each piece of the app, such as the unit converter and toggle switch, would interact directly with the reformatted data and display it. This changed in the final version where the frontend of the app interacts with each of the pieces and is the only part that actually displays the code. The below proposed block diagram and actual block diagram above can be used to see this difference.



3.7 - proposed block diagram

4 Testing

1. Our project needed unit tests for modules, integrity tests for interfaces, acceptance tests for all functional and nonfunctional requirements, black and white box testing for end functionality and performance, and security testing.
2. The following lists items were tested and the types of tests we used throughout the project:

- a. Units: Unit Conversion for measurements, Retrieving Data from URL, Scaling Measurements, Ensuring URL is safe.
- b. Interfaces: Between UI and Retrieving Data from URL, Between Retrieving URL and Ensuring URL is safe, Between UI and Scaling Measurements/Unit Conversions.
- c. Acceptance: All the functional and non-functional requirements and any features that have been implemented. This includes, but is not limited to, loading a recipe from an arbitrary URL, solution should not require a user account, and solution should be user-friendly.
- d. Black Box: Ensure end functionality is working properly. Such as our functions for retrieving the website data from the URL are doing so.
- e. White Box: Ensure the flow of input and output of the code inside our functions is correct and to find improvements in design.
- f. Security: URL retrieved should be valid, preventing redirect URLs, and checking for hidden javascript in URLs.

3 & 4. Below is an example of a test case we have defined, designed, and developed. The test below is part of a test suite designed to make sure the function we wrote to convert from units in the metric system to the imperial system is working as intended. This test has the goal to make sure the function converts properly between mL, specifically its abbreviation, to an appropriate unit in the imperial system. These tests were written in JavaScript in Visual Studio and tested using Jest, a native testing framework for JavaScript. The example below has eight different test cases. The first test case demonstrates converting 10 mL to imperial. This is done by passing a value of 10 and a string of mL to our function. Our expectation is that the result will be 2 tsp.

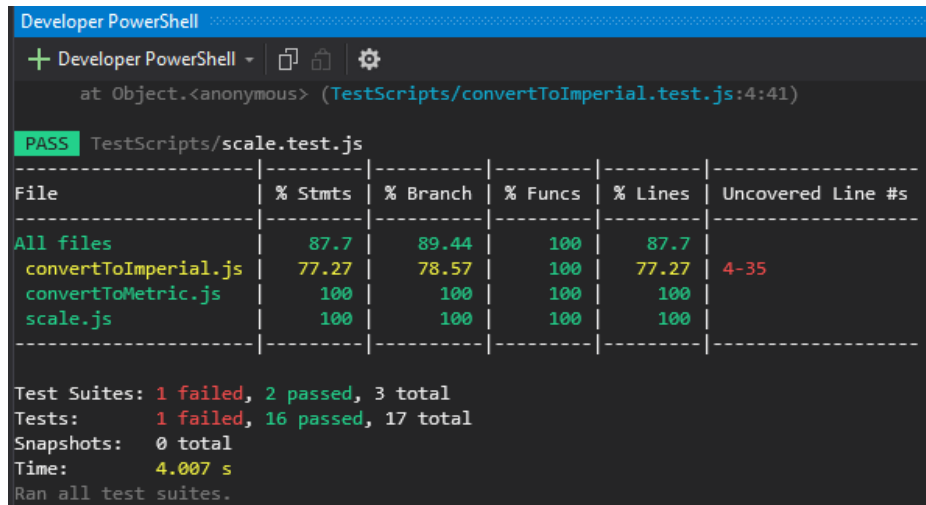
Image 4.1

```
test('properly converts mL to imperial', () => {  
  expect(convertToImperial(10, 'mL')).toStrictEqual([2, 'tsp'])  
  expect(convertToImperial(22.5, 'mL')).toStrictEqual([1.5, 'tbsp'])  
  expect(convertToImperial(135, 'mL')).toStrictEqual([4.5, 'fl oz'])  
  expect(convertToImperial(240, 'mL')).toStrictEqual([1, 'cup'])  
  expect(convertToImperial(357.5, 'mL')).toStrictEqual([(357.5 / 240), 'cups'])  
  expect(convertToImperial(712.5, 'mL')).toStrictEqual([1.5, 'pt'])  
  expect(convertToImperial(2375, 'mL')).toStrictEqual([2.5, 'qt'])  
  expect(convertToImperial(3800, 'mL')).toStrictEqual([1, 'gal'])  
})
```

5. We ran the tests in the Jest framework by running our suites. We ran our suites with the `--coverage` tag to provide more insight to how our functions are behaving, such as what percentage of lines are being reached by our tests.

6. Below are the results we found on our initial run of our test suites. We discovered that one of the tests was failing in the convert to imperial suite, the failed test was part of the properly converts mL to imperial test that we outlined above, and the convert to imperial suite did not have full statement, branch, and line coverage. Jest provided an html which showed us the specific lines which were never reached by our code by highlighting them red, which was helpful in determining where our problem existed.

Image 4.2



```
Developer PowerShell
+ Developer PowerShell
at Object.<anonymous> (TestScripts/convertToImperial.test.js:4:41)

PASS TestScripts/scale.test.js
-----|-----|-----|-----|-----|-----
File    | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-----|-----|-----|-----|-----|-----
All files | 87.7    | 89.44    | 100     | 87.7    |
convertToImperial.js | 77.27   | 78.57    | 100     | 77.27   | 4-35
convertToMetric.js  | 100     | 100      | 100     | 100     |
scale.js    | 100     | 100      | 100     | 100     |
-----|-----|-----|-----|-----|-----

Test Suites: 1 failed, 2 passed, 3 total
Tests:       1 failed, 16 passed, 17 total
Snapshots:  0 total
Time:        4.007 s
Ran all test suites.
```

Image 4.3

```

FAIL TestScripts/convertToImperial.test.js
  • properly converts mL to imperial

    expect(received).toStrictEqual(expected) // deep equality

    - Expected  - 2
    + Received  + 2

    Array [
      - 2,
      - "tsp",
      + 10,
      + "no conversion found",
    ]

    2 |
    3 | test('properly converts mL to imperial', () => {
    > 4 |   expect(convertToImperial(10, 'mL')).toStrictEqual([2, 'tsp'])
      |                                     ^
    5 |   expect(convertToImperial(22.5, 'mL')).toStrictEqual([1.5, 'tbsp'])
    6 |   expect(convertToImperial(135, 'mL')).toStrictEqual([4.5, 'fl oz'])
    7 |   expect(convertToImperial(240, 'mL')).toStrictEqual([1, 'cup'])
  
```

Image 4.4

87.7% Statements 107/122 89.44% Branches 127/142 100% Functions 3/3 87.7% Lines 107/122

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File	Statements	Branches	Functions	Lines
convertToImperial.js	77.27% 51/66	78.57% 55/70	100% 1/1	77.27% 51/66
convertToMetric.js	100% 54/54	100% 72/72	100% 1/1	100% 54/54
scale.js	100% 2/2	100% 0/0	100% 1/1	100% 2/2

7. After some digging, we found that we were converting the measurement we passed to the function to lower case but trying to compare it to mL with a capital L. To fix this we changed that L to be lower case. The picture below shows where this was in our code.

Image 4.5

```

1 function convertToImperial(currentValue, currentMeasurement) {
2   //gal to L
3   if (currentMeasurement.toLowerCase().normalize() === 'mL' normalize()) {
4     if (currentValue <= 10) {
5       //to teaspoon
6       return [currentValue / 5, 'tsp']
7     }
  
```

8. After making the changes and re-running our tests, we found the failed test now passes and we have complete coverage for the statements, branches, and lines. In addition, the html no longer has red lines indicating part of the code was never reached.

Image 4.6

```

Developer PowerShell
+ Developer PowerShell
PASS TestScripts/scale.test.js
PASS TestScripts/convertToImperial.test.js
PASS TestScripts/convertToMetric.test.js
-----|-----|-----|-----|-----|-----
File      | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-----|-----|-----|-----|-----|-----
All files |    100   |    100   |    100   |    100   |
convertToImperial.js |    100   |    100   |    100   |    100   |
convertToMetric.js   |    100   |    100   |    100   |    100   |
scale.js             |    100   |    100   |    100   |    100   |
-----|-----|-----|-----|-----|-----
Test Suites: 3 passed, 3 total
Tests:       17 passed, 17 total
Snapshots:   0 total
Time:        2.904 s
Ran all test suites.
  
```

Image 4.7

100% Statements 122/122 100% Branches 142/142 100% Functions 3/3 100% Lines 122/122

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File	Statements	Branches	Functions	Lines
convertToImperial.js	100% 66/66	100% 70/70	100% 1/1	100% 66/66
convertToMetric.js	100% 54/54	100% 72/72	100% 1/1	100% 54/54
scale.js	100% 2/2	100% 0/0	100% 1/1	100% 2/2

Image 4.8

```

1 function convertToImperial(currentValue, currentMeasurement) {
2   //gal to L
3   39x if (currentMeasurement.toLowerCase().normalize() === 'ml'.normalize()) {
4     8x   if (currentValue <= 10) {
5       //to teaspoon
6     1x     return [currentValue / 5, 'tsp']
7   }
  
```

9. The above process illustrates one example of how we plan to test our team's JavaScript code for this project. Our team continued to use Jest to help ensure our code is fully tested. This is just an example for our unit testing but other types of testing might use a different framework or program other than Jest.

4.1 UNIT TESTING

The following units will be tested in isolation; unit conversion for measurements, retrieving data from a URL, scaling recipe measurements, ensuring the URL is safe, and displaying UI elements.

Unit Conversion for measurements: Our plan was to have a function that has the ability to convert recipe amounts between metric and imperial units and vice versa.

Retrieving data from a URL: Our plan was to have a function that takes an URL as input and returns back the data from the website the URL points to.

Scaling Measurements: Our plan was to have a function that can scale the amounts needed for each ingredient based on how many batches the user plans to make.

Ensure the URL is safe: We made a function that examines the URL and detects whether it is safe or not.

4.2 INTERFACE TESTING

Our compositions of two or more interfaces were tested through UI observations resulting from user input. Since we used the React Native framework to develop our application, we had to use an emulator technology which could emulate both iOS and Android performances. Therefore, for interface testing, we used the Expo CLI Quickstart testing software which has been detailed in the previous chapters of this report.

4.3 ACCEPTANCE TESTING

Once all of our tests passed, we created a presentation for our client that demonstrates and explains how all the requirements are being met. After the presentation, the client will be provided with a user guide and the application itself to get some hands-on interaction and the ability to test out the base functionality of the application.

4.4 RESULTS

We found a failure when trying to compare a string we had converted to lowercase with one that we had set to have an upper case letter. After this problem was fixed, we successfully created functions to scale batches and covert ingredient measurements between the metric and imperial systems.

Using the Jest frameworks allowed the team to test our individual outputs and returns from different code sections and immediately verify new code which was being written. One of the major advantages of using EXPO CLI for testing was being able to immediately see any code updates directly on the app itself especially for the frontend development of the application since the UI changes could be seen clearly. This played a significant role in successful testing since the team and the clients were able to immediately see the updates of the application.

5 Implementation

5.1 FRONTEND

The frontend of our application had to be designed with simplicity in mind, as we wanted it to be easily usable by someone who might have their hands full in the kitchen. In order to successfully implement this, we only had two pages in our frontend; the home page and the recipe page.

The home page consists of the app logo, a textbox to enter the recipe website's URL in, and a "Go" button that will send the URL to backend for parsing and then move to the second page to display the information.

The second page receives the formatted information from the backend and then displays it, making it easier to interpret instead of outputting a block of text. The output is in a two-part format, with the first part being required ingredients and the second being cooking instructions. In addition to this, there is a back button that allows the user to go back to the previous page. There are another 3 buttons on the bottom of the screen that were meant to adjust font size, scale the recipe, and switch any units from metric to imperial and vice versa, but we were unable to finish those features at this time.

These pages were made in JS using React Native with the help of some VS Code extensions such as Styled Components. These extensions allowed for a streamlined

customization of certain components such as buttons, textboxes, wrappers, sliders, etc. to give the application a professional look and feel to it.

5.2 BACKEND

The backend for our project covers two main parts of the project. The first part consists of any logic calculations needed for the features of the app. The second part is the storage implementation for our project.

The logic calculations are done in JavaScript. JavaScript was chosen for the coding language because of its ease of use with app development and therefore React Native. There are 3 functions associated with the logic calculation for the project. The first is one to convert metric units to imperial units, the second is to convert imperial units to metric units, and the third is to scale the size of a given input by a given amount. The first two functions are used to complete the unit conversion feature of the project. The third function is used to complete the scaling requirement of the project. The three functions have been tested, as shown above with Jest, to be successful and have complete code coverage. We were unable to complete the features that use these functions.

JSON files used to implement the data storage for our project. After considerations for different options, highlighted in section 3.4 Technology Considerations, JSON files were determined to be the best for their simplicity, lightweight nature, and lack of an external interaction. Three functions and the expo-file-system library for React Native were used to make this possible. The file ReadJSON is responsible for the call the frontend makes to read a JSON file. The file WriteJSON is similar but for writing files instead. Together the two create the data interaction loop the frontend and web scraper both use. The third file, useAsync, is a custom function that provides the frontend with a way to wait for the information to be loaded. This is needed because the library call being used is asynchronous, therefore the rest of the code continues without waiting on that step, there is a chance the data is not loaded right away. Lastly, the library provides the app with a way to access the file system on the mobile device running the application and provides a location for use to save the files.

5.3 WEB SCRAPING

In order to take in data from recipe websites to display to the end user it first has to be retrieved from the website and then transformed into a usable format. First the HTML is retrieved from the website using fetch. It comes in a very raw form, so the strings then need to be reformatted using various JavaScript string methods to shape it correctly to

then be parsed into JSON objects. Extra syntax is added and excess data is removed from the data string. A JSON object is then created for the ingredients and a separate object is made for the instructions. They are then combined into a single object and written to the backend infrastructure.

6 Closing Material

6.1 CONCLUSION

The introductory phase of this project consisted of us identifying the problem statement and brainstorming an initial plan of action that we would use as a solution for the issue at hand. We identified our target audience as well and decided to choose a platform that would allow the widest range of users to access our final product. We chose to go with a mobile application for the end product as we believe this would allow people to quickly and easily get the information they need without taking too much space in the kitchen. We identified that the end users would most likely be someone who was interested in cooking, whether they're beginners or experienced in the culinary field already.

The design phase of this project was one of the more time intensive parts of the project since the team had several discussions on the structure and UI designs for the app. By the end of this phase, we were able to create realistic mockups of the URL screen and the recipe screen which provided the client and everyone in the team with an idea on how the end product would look after the completion of the development phase. In this phase of the project, we also looked at pre-existing apps in the market which provide similar services to the culinary industry. Research into 'competing' products gave us an opportunity to evaluate the features which the app plans on offering to the users and how they would be more beneficial compared to what is already available on the app stores. And finally, the team also had a chance to finalize the design plan for the development period. Since the app requirements, features and UI mockups have now been completed, a majority of the time during the second semester of this project was spent on the implementation of the app itself. The project team was divided up into smaller sections where members were simultaneously working on the frontend, backend and the web scraping service of the application.

For testing the application, we used Expo because of the numerous advantages it offered. Expo has an intuitive debugging environment as well as hot reloading, which would allow us to know that there was an error in the code right away, and then the debugging software could be used directly afterward to fix any issues.

For the testing part of the project we started with unit tests for some JavaScript functions including scaling ingredient quantities and converting between metric and imperial units. After performing research, our team settled on using Jest to test our JavaScript code and we plan to continue using the framework for the remainder of the project. Jest allows us to write tests for our functions and see if our code passes them all. In addition, Jest allows us to view an html form with additional information regarding code coverage. Using Jest, our team will be able to ensure our code passes all the required tests and all of the code we write is being covered by our tests. For other forms of testing such as interface testing, our team will research ways to perform such tests as our project draws closer to needing them.

In conclusion, over the course of the two semesters, the project team was able to map out features, design a mobile application, conduct technology research and build out a cross platform application which is able to complete all the base functionality of the app which was listed out by our client. As expected in any project, the team experienced some challenges with technology changes, implementation limitations and the development of the additional specialized features. This was also a great learning opportunity for the team members since everyone was able to learn a new form of application development and associated technologies and relevant strategies which should be employed for successful software project management as well. With a fully functional application with successfully implemented base functionality, we are confident that this application can be used to solve the client's original problem of dealing with large amounts of irrelevant information in modern recipe websites. Furthermore, with the application's ability to function on a wide range of smartphones and operating systems, the app can now be used by a wide range of users from beginners to advanced culinary experts and everyone can benefit from a fully streamlined modern day recipe website.

6.2 REFERENCES

- [1] Amit Thinks. *How to run JavaScript on Visual Studio Code*. (Oct. 13, 2020). [Online Video]. Available: https://www.youtube.com/watch?v=Z_G86SKXP3s. Accessed: Apr. 24, 2021.
- [2] Ben Awad. *Running Create React Native App on Your Phone*. (Dec. 31, 2017). [Online Video]. Available: <https://www.youtube.com/watch?v=mhoiwfShSnE>. Accessed: Apr. 24, 2021.
- [3] Jordan Walke. “React Native Documentation” React Native <https://reactnative.dev/>
Accessed: Apr. 25, 2021
- [4] LevelUpTuts. *React Native For Everyone Preview*. (June 5, 2017). [Online Video Playlist]. Available: <https://www.youtube.com/playlist?list=PLLnPHn493BHG30qU2Rw403x6w7XP8hHB5>. Accessed: Apr. 24, 2021.
- [5] “Setting up the development environment · React Native,” *React Native*, Mar. 12, 2021. [Online]. Available: <https://reactnative.dev/docs/environment-setup>. Accessed: Apr. 24, 2021.
- [6] Web Dev Simplified. *Introduction To Testing In JavaScript With Jest*. (Sep. 24, 2019). [Online Video]. Available: <https://www.youtube.com/watch?v=FgnxcUQ5vho>. Accessed: Apr. 24, 2021.
- [7] “Installation.” *Expo Documentation*. [Online]. Available: <https://docs.expo.dev/get-started/installation/>. Accessed: Dec. 8, 2021.

7 Appendix

7.1 APPENDIX I

To start the application, you will need to first have Expo. Details can be found [here](#). After you have downloaded Expo, locate your working directory through your command prompt.

```
11/09/2021  08:27 PM    <DIR>        Backend
11/10/2021  07:05 PM    <DIR>        Frontend
11/14/2021  08:06 PM    <DIR>        Working_Demo
              0 File(s)            0 bytes
              5 Dir(s)  54,706,798,592 bytes free

C:\Users\visma\Desktop\sddec21-12>cd Working_Demo

C:\Users\visma\Desktop\sddec21-12\Working_Demo>dir
Volume in drive C has no label.
Volume Serial Number is 6A6A-8E5F

Directory of C:\Users\visma\Desktop\sddec21-12\Working_Demo

11/14/2021  08:06 PM    <DIR>        .
11/14/2021  08:06 PM    <DIR>        ..
11/14/2021  08:06 PM    <DIR>        .expo-shared
11/14/2021  08:06 PM                126 .gitignore
11/14/2021  08:06 PM            8,970 App.js
11/14/2021  08:06 PM            690 app.json
11/14/2021  08:06 PM    <DIR>        assets
11/14/2021  08:06 PM            113 babel.config.js
11/14/2021  08:06 PM    <DIR>        components
11/14/2021  08:06 PM        799,261 package-lock.json
11/14/2021  08:06 PM            786 package.json
11/14/2021  08:06 PM        263,278 yarn.lock
              7 File(s)        1,073,224 bytes
              5 Dir(s)  54,706,794,496 bytes free

C:\Users\visma\Desktop\sddec21-12\Working_Demo>expo start
```

After the directory is found, run “npm install” and then “yarn install”.

```
Directory of C:\Users\visma\Desktop\sddec21-12\Working_Demo
12/08/2021 10:06 PM <DIR> .
12/08/2021 10:06 PM <DIR> ..
12/08/2021 10:06 PM <DIR> .expo
11/14/2021 08:06 PM <DIR> .expo-shared
11/14/2021 08:06 PM          126 .gitignore
11/14/2021 08:06 PM      8,970 App.js
11/14/2021 08:06 PM        690 app.json
11/14/2021 08:06 PM <DIR> assets
11/14/2021 08:06 PM       113 babel.config.js
11/14/2021 08:06 PM <DIR> components
11/14/2021 08:06 PM   799,261 package-lock.json
11/14/2021 08:06 PM        786 package.json
11/14/2021 08:06 PM   263,278 yarn.lock
          7 File(s)    1,073,224 bytes
          6 Dir(s)    54,713,851,904 bytes free

C:\Users\visma\Desktop\sddec21-12\Working_Demo>npm install
npm WARN read-shrinkwrap This version of npm is compatible with lockfileVersion@1, but package-lock.json was generated f
or lockfileVersion@2. I'll try to do my best with it!
[.....] / extract:react-devtools-core: sill extract react-devtools-core@4.20.1 extracted to C:\Users\visma
```

Once these steps have been taken, run “Expo start” which should redirect you to a webpage.

```
C:\Users\visma\Desktop\sddec21-12\Working_Demo>expo start
```

```
There is a new version of expo-cli available (4.13.0).  
You are currently using expo-cli 4.12.4  
Install expo-cli globally using the package manager of your choice;  
for example: `npm install -g expo-cli` to get the latest version
```

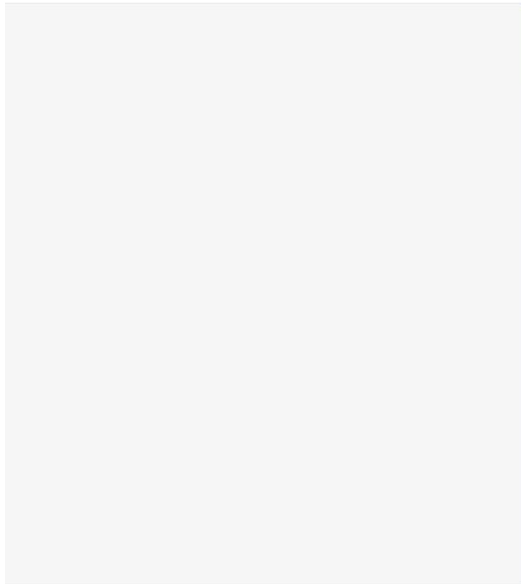
```
> Port 19000 is  
√ Use port 19001 instead? ... yes  
Starting project at C:\Users\visma\Desktop\sddec21-12\Working_Demo  
Developer tools running on http://localhost:19003  
Opening developer tools in the browser...  
Starting Metro Bundler
```



```
> Metro waiting on exp://10.8.103.108:19001  
> Scan the QR code above with Expo Go (Android) or the Camera app (iOS)  
  
> Press a | open Android  
> Press w | open web  
  
> Press r | reload app  
> Press m | toggle menu  
> Press d | show developer tools  
> shift+d | toggle auto opening developer tools on startup (enabled)  
  
> Press ? | show all commands  
  
Logs for your project will appear below. Press Ctrl+C to exit.
```

Metro Bundler

PROCESS (1) - 10:08:01 PM



Run on Android device/emulator

Run on iOS simulator

Run in web browser

Send link with email...

Publish or republish project... [↗](#)

PRODUCTION MODE



CONNECTION

Tunnel

LAN

Local

<exp://10.8.103.108:19001>



LOGGED IN AS

METRO BUNDLER

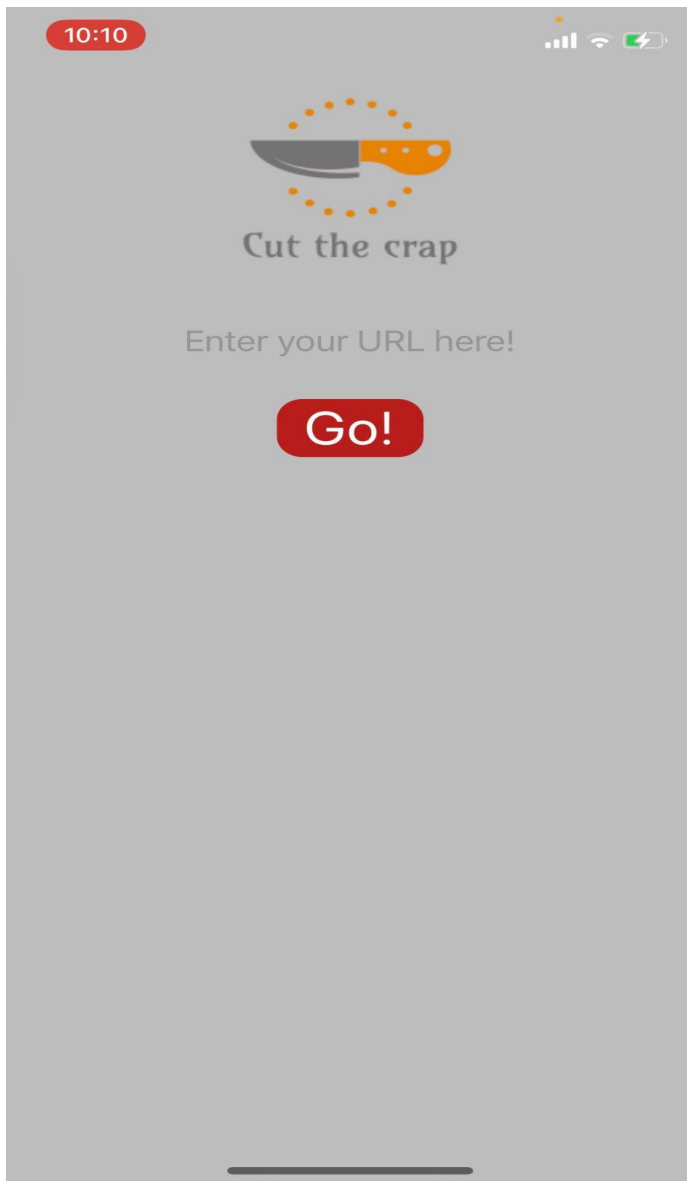
INFO Starting Metro Bundler

22:08

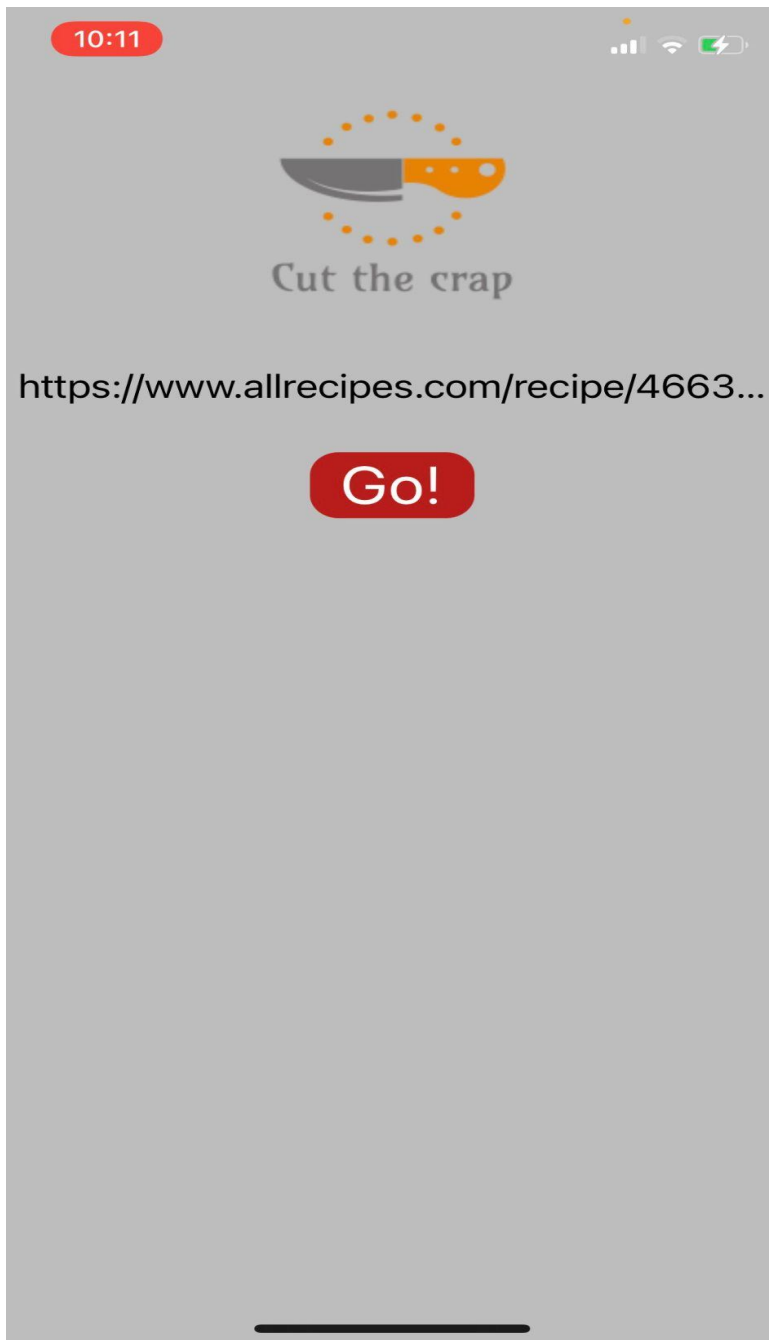
Once this webpage appears, download the Expo GO application on your mobile device and then scan the QR Code. It should redirect you directly to the application.



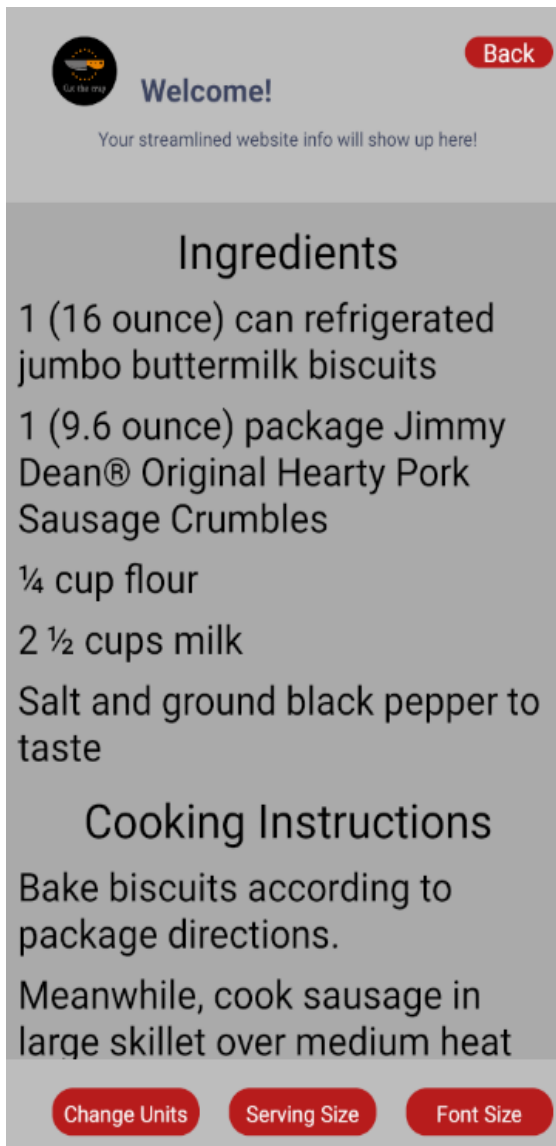
This is the loading screen that will pop up while the application is starting up.



Pictured above is the home page of the application, this is where you would input the URL for a recipe that you are interested in.



Once the URL has been pasted in, simply hit the “Go!” button.



Once on this page, you can scroll through the ingredients and instructions, and tap the back button on the top right of the page once you want to return to the home page or input a different URL.

7.2 APPENDIX III

Challenges & Lessons Learned:

- Training / Knowledge
- Over promising
- Focus main functionality and then feature update
- Time is limited, this is not our only class
- In person meetings are nice
- Better task manager

Training and knowledge were a big part of our project. Almost everything we worked with to develop this app was new to our team. Learning something new takes quite a bit of time. The lesson we learned here is that we could have spent more time training at the beginning to avoid over searching towards the end of the project. Also, having to search for the libraries, frameworks, etc. we are going to use and learn them to see if they work takes a lot of time as well.

Over promising is a big lesson our team learned this semester. Our client asked for a pretty straightforward app at the beginning. Our team then came up with extra features to add. Instead of having those be additional features we could work on should extra time be available, we promised to have them done by the end of the semester. It is ok to plan extra features but it is better to only promise the base functionality for the project and not everything.

Another lesson that ties into the previous one, is to focus on the main functionality of the project first and then release updates with new features. Our team started by just working on everything in one go including the extra features. This caused us to not have as much time to devote to the base functionality and made it take much longer than needed to complete. Our team would have benefitted larger from getting that completed first and then trying to get the features done second.

Time is a big lesson our team learned this semester. It is a limited resource and always runs out faster than you think.

In person meetings are very valuable and that has been something we have missed since the start of COVID. There are so many benefits to having in person meetings such as building team energy, reducing distractions, and clearer communications to name a few. Our team learned that if you can have in person meetings, strongly consider doing so.

Having a good task manager is another lesson we learned. Our plan was to use trello to manage our tasks and stay focused. Our team ended up using google docs instead. This worked out well for keeping track of who was doing each task but not as well as an actual

manager. In addition, using google docs we lost out on the many features a task manager, like Trello, provides that make things easy to manage.

Future Expansions:

- Loading from arbitrary URL
- Scaling and unit conversions
- Including measurements/amounts in instructions
- Font size adjustments
- The revenue stream

The above bullets are requirements that were not fully complete due to limited time. The biggest problem with the first three bullets is being able to parse large amounts of data that could come in a large variety effectively. One solution that could solve that is to look into a machine learning/natural language processing solution.

As for the revenue stream requirement, the solution our team was going to use was to include a toggle switch that would hide/show the ads and author's original content.