CHITKARA UNIVERSITY INSTITUTE OF ENGINEERING AND TECHNOLOGY

Project Report on Recipe Wizard (Recipe Maker)

Submitted by
Prabhdeep singh (2110992605)
Nitin Kumar (2110992596)
Pranav Jain (2110992606)

Department of Computer Applications

INDEX

b. **Remaining areas of concern**
c. **Technical and managerial lessons learnt**
d. **Future recommendations**

12. Bibliography

13. Source Code

[To be shared as a compressed file comprising following folders
(i) Project Report (ii) Source Code (iii) Database backup file(s) (iv) readme.txt]

Abstract

The RecipeWizard Application Project Report presents a comprehensive overview of the development and implementation of the RecipeWizard mobile application. This report details the application's objectives, functionalities, technological aspects, and the process of bringing this innovative culinary tool to fruition. The RecipeWizard app is designed to simplify the cooking experience by allowing users to find recipes based on available ingredients, dietary preferences, and cooking skills. Through a user-friendly interface and advanced algorithms, RecipeWizard offers a vast recipe database, personalized recommendations, meal planning capabilities, and a community platform for users to connect, share, and explore their culinary passions. This project report covers the app's development lifecycle, including design, coding, testing, and user feedback. It also delves into the technology stack, database structure, and the challenges and solutions encountered during development. With the RecipeWizard app, we aim to provide an exceptional user experience, inspiring and assisting home cooks in their culinary endeavours while fostering a community of food enthusiasts.

Profile of Problems Assigned

Here are some profile of problems assigned that our application can tackle-

1. Ingredient-Based Recipe Generation:
   - Developing algorithms to generate recipes based on available ingredients provided by the user.
   - Ensure that generated recipes are not only practical but also delicious and well-balanced.

2. Dietary Restriction and Allergen Considerations:
   - Creating a system that generates recipes tailored to specific dietary restrictions, such as gluten-free, vegan, keto, or low-sodium diets.

3.. Recipe Scaling and Portion Control:
   - Develop features that allow users to scale recipes up or down to meet their desired portion sizes.
   - Ensure accurate ingredient adjustments and cooking times for different serving sizes.

4. Cooking Techniques and Skill Levels:
   - Offer step-by-step guidance and video tutorials for cooking techniques.

5. Seasonal and Local Ingredient Integration:
   - Incorporate local and seasonal ingredients into recipe suggestions to support sustainable and fresh cooking.
   - Notify users about seasonal produce and suggest recipes that make the best use of them.

6. Recipe Search and Filtering:
   - Creating a robust search and filtering system that allows users to find recipes based on various criteria, such as cuisine, cooking time, ingredients on hand, and more.

7. Nutritional Analysis and Health Goals-
   - Suggest recipes that align with users' health objectives and provide detailed nutrition facts.

8. Community and User-Generated Content:
   - Encourage users to share their own recipes and cooking tips within the app.
   - Implement features for rating, reviewing, and sharing recipes with other users.

These profiles of problems highlight the diverse challenges that Recipe Wizard may address, ultimately providing users with a valuable and enjoyable cooking experience. The app's success depends on its ability to solve these problems effectively and meet the needs of its users.

Study of Existing Systems

1. **AllRecipes:** AllRecipes is a well-established online platform where users can find a vast collection of user-submitted recipes. It offers ingredient-based recipe searches, user reviews, and ratings.
2. **Cookpad:** Cookpad is a global platform that allows users to share and discover recipes. It offers features for creating and sharing recipes, as well as searching for recipes by ingredients.
3. **BigOven:** BigOven is a recipe app that provides meal planning, grocery list creation, and a recipe search engine based on available ingredients. It also includes a feature for saving and organizing recipes.
4. **Tasty (BuzzFeed's Tasty):** Tasty offers a wide range of recipe videos and written recipes with a focus on quick and visually appealing dishes. Users can search for recipes and save them for later use.
5. **Paprika Recipe Manager:** Paprika is a recipe management app that allows users to save and organize their own recipes, create grocery lists, and plan meals. It also offers a feature for importing recipes from websites.
6. **MyFridgeFood:** MyFridgeFood is a web-based platform that helps users find recipes based on the ingredients they have in their kitchen. It provides simple and quick recipe suggestions.
7. **ChefTap:** ChefTap is a recipe management app that allows users to import and organize recipes from various websites. It includes features for creating shopping lists and meal planning.
8. **Yummly:** Yummly is a popular recipe and cooking app that offers personalized recipe recommendations, shopping lists, and the ability to search for recipes by ingredients, dietary preferences, and cuisines.

System Requirements

Product Definition-

RecipeWizard is a culinary platform designed to inspire and simplify the cooking experience for users of all skill levels. It empowers users to create delicious meals by generating personalized recipes based on the ingredients they have on hand. Whether users are seasoned home chefs or beginners in the kitchen, RecipeWizard offers a user-friendly and feature-rich platform that encourages culinary exploration and creativity.

a) Problem Statement- The problem is that there is a significant need for a user-friendly and innovative culinary platform that empowers individuals to create delicious and resourceful meals using the ingredients they already have, while also catering to their dietary preferences and cooking skill levels.

b) Functions to be Provided-

1. Ingredient Search: Users can input the ingredients they currently have in their kitchen.

2. Recipe Suggestions: RecipeWizard then generates a list of recipes that can be prepared using the specified ingredients.

3. Filtering Options: Users can often filter recipes based on various criteria such as cuisine, course, or dietary restrictions.

4. Recipe Details: When a user selects a specific recipe, RecipeWizard typically provides detailed instructions, including ingredients, preparation steps, and sometimes nutritional information.

5. Mobile App Integration: RecipeWizard might offer a mobile app for users to access the service on the go.

c) Processing Environment: H/W, S/W-

Client Side

- Operating system supported – Android / IOS
- Memory - Minimum 4 GB RAM, and application size is 60.05 MB
- Processor - Android 4.1 (API 16) or newer / iOS 9.0 or newer
- Software Version – Compatible with modern web browsers (Chrome, Google) to open Allrecipes.com

Service Side

Software versions
React native, Expo, Jquery as Frontend language

d) <u>Solution purposed -</u>

All above problems are solved in our application, our application is not any kind of AI or any filtering engine, its just a search engine that will search for the dishes contains the ingredients selected by the user.

The second problem is that having larger database makes system heavy, we will not store all the data of dishes origin or steps of making the recipe in the database. Our application will redirect the founded recipes to the AllRecipes (containing more than 100,000 recipes from all over the world) this will help us managing our database as light as possible for less size of mobile application and smooth running of machine.

e) <u>Acceptance Criteria-</u>

After discussing this project with our coordinators, they agreed to accept the project of
'Recipe Generator! According to the requirements we developed framework for our application such as ingredients page, ingredients list, results page, search option etc. We used one of the most trending programming languages React Native, Jquery. The project is a service-based application developed to help an individuals as much as we can.

Feasibility Analysis

- **Feasibility Study**

The feasibility study for RecipeWizard encompasses a comprehensive assessment of its viability across various dimensions. This analysis begins with a clear project description, outlining the purpose, goals, and target audience of RecipeWizard, along with defining its scope and key features. A thorough market analysis examines the target market and competitive landscape, ensuring a solid understanding of user demographics and trends. Technical feasibility assesses the readiness of required technology and infrastructure, while financial feasibility estimates initial investments, revenue streams, and break-even points. Operational feasibility evaluates the day-to-day functioning of RecipeWizard, considering resource availability and potential challenges. Legal and regulatory considerations ensure compliance with relevant laws, and user acceptance studies gauge interest and preferences. A detailed timeline, risk analysis, and examination of social and environmental impact round out the study. In conclusion, the feasibility study provides a holistic view of RecipeWizard's potential, offering informed recommendations for its development and launch.

- **Economic Feasibility**

The economic feasibility of RecipeWizard involves looking at whether it makes good financial sense. First, we need to figure out how much it will cost to create and run RecipeWizard, including making the app, marketing it, and keeping it running. Then, we think about how RecipeWizard can make money. This might be through people paying for extra features, teaming up with kitchen brands, showing ads, or working with grocery delivery services. We also need to find out when RecipeWizard will start making more money than it's costing to run – this is called the "break-even point." We predict how much money RecipeWizard could make over time, taking into account things like how many people will use it and how much they might pay. We also look at the risks involved, like changes in costs or user numbers. By understanding all these things, we can see if RecipeWizard is likely to be a good investment and start making a profit.

- **Technical Feasibility**

The technical feasibility of RecipeWizard focuses on assessing whether the proposed project can be successfully developed and implemented from a technological perspective. Here's an explanation in simple language:

Technical feasibility is about checking if we can actually build RecipeWizard. We look at the technology and tools needed to create the app. This includes the computer code, databases to store information, and how everything will work together. We need to see if the technology we plan to use is available, and if we have the skills to make it happen. We also check if there are any challenges or roadblocks that might make it difficult to build RecipeWizard. It's like making sure we have the right ingredients and recipe to cook a delicious meal - in this case, a digital one called RecipeWizard!

- **<u>Operational Feasibility</u>**

Without a doubt, the suggested system is totally GUI-based, very user-friendly, and all the inputs to be taken include explanations for even lay-people. In addition, appropriate training has been provided to users so that they are familiar with the basics of the system and feel at ease using it. As far as our research is concerned, the clients are comfortable and happy as a method for reducing loads and doing.

Project Plan

a) <u>Team Structure and Responsibilities-</u>

The team adopted a democratic team structure, fostering a collaborative and flexible work environment. Each team member autonomously undertook tasks based on personal interests and dedication to the project. This approach allowed for a dynamic distribution of responsibilities, ensuring that everyone was engaged in areas that resonated with their skills and passions. Team members willingly supported one another, creating a supportive atmosphere. In instances where a member encountered challenges, a collective effort was made to provide assistance. The team's cooperation and shared commitment were pivotal in overcoming obstacles and driving progress. The collective aspiration is for the RecipeWizard app to be a valuable and user-friendly resource, reflecting the collaborative spirit and dedication invested by the team.

b) <u>Development Schedule -</u>

Timeline	Task
15 – 07 – 2023	Building Tasks (distributing work)
28 – 08 – 2023	Frontend design implementation
18 – 09 – 2023	Frontend and Backend implementation
2 – 11 – 2023	Completion of Backend and Frontend
7 – 11 – 2023	Midway of creating project report
24 – 11 – 2023	Submission of Project Report

We divided our total done work into 6 Phases-

<u>Phase 1: Planning and Design (Weeks 1-2)</u>

<u>1. Project Kickoff:</u>

- The scope of making this project is to reduce the food wastage as much as we can, by providing recipes with the ingredients you have in your pantry.

- Here is the Team working on parts of the project-

* Prabhdeep singh (Project Manager)

* Pranav Jain (Data Collector/ organizer)

* Nitin Kumar (Frontend Developer)

<u>2. Market Research:</u>

- We analysed some already existing projects in the market, like MyFridgeFood, Yummly as mentioned before, and got the features in our project which they don’t have.

- We started collecting data from the market like the ingredients, database with full of recipes, and other engines to optimize our application.

- The unique selling point of our project is that it is in the form of application and don’t uses AI to make recipes rather matches the ingredients with the database of dishes, and connecting it to a external website allrecipes.com for showing recipes details and other steps to continue, which makes our project (application) really lightweight.

3. Design Wireframes:

- Created basic wireframes for our app's user interface like-

* Home Screen [Which contains (a) Search Button

(b) Ingredients List

(c) Add Ingredients]

* Ingredients [Which contains (a) Dairy

(b) Vegetables

(c) Fruits

(d) Baking and Grains

(e) Condiments

(f) Meats

(g) SeaFoods

(h) Liquids

(I) Nuts

and they contain further more ingredients under their sub category]

- Gathered feedback from the teams and other users to check whether they are right or not.

4. Technology Stack:

- We have choosen React Native, Expo , Json for creating our application, because react native application can be used on any device, IOS / Android

- Settled up the the development environment (Android Studio/Visual Studio) to create our application.

Phase 2: Frontend Development (Weeks 3-11)

1. Core UI Development:

- Started building the basic user interface according to the wireframes.

- Around (25 – 09 – 2023) we successfully created frontend added all the ingredients and Navigation to visit the other parts of application.

3. Recipe Search Functionality:

- Developed the core functionality for searching Recipes and Ingredients in the database.

4. Ingredient Database Integration:

- Integrated a API for ingredient information with AllRecipes.com for fetching their recipes according to the users ingredients.

Phase 4: Testing (Weeks 11-14)

1. Unit Testing:

- Conducted unit tests for individual components / Ingredients contained in our application to check whether they are working right or not.

2. Integration Testing:

- Tested the interaction between different modules, that navigation is working properly or not.

3. User Acceptance Testing (UAT):

- Afterwards we included users outside of our team to test our applications working, and collected their feedback and solves all the errors bugs coming on our way.

Phase 5: Deployment and Launch (Weeks 14-18)

1. Deployment:

- In last, Around (10 – 11- 2023) Deployed the app to a staging environment for final testing.

2. Bug Fixes:

- Addressed all the issues identified during testing and then fixed them.

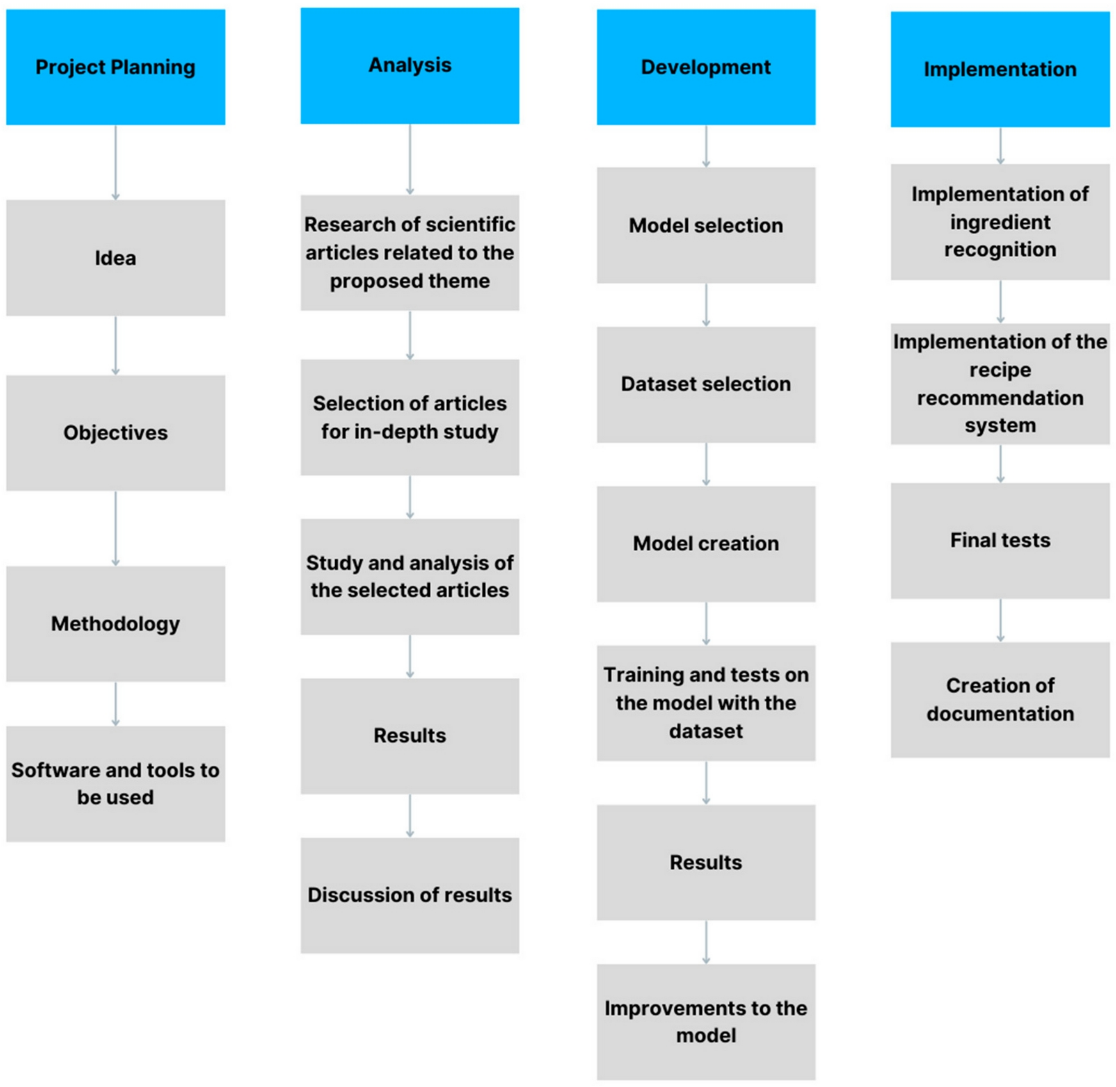
Phase 6: Post-Launch (Ongoing)

1. Monitoring and Maintenance:

- Monitored the app's performance and address any post-launch issues.

2. User Feedback:

- Collected user feedback and consider updates or new features.


Project Planning
Idea
Objectives
Methodology
Software and tools to be used
Analysis
Research of scientific articles related to the proposed theme
Selection of articles for in-depth study
Study and analysis of the selected articles
Results
Discussion of results
Development
Model selection
Dataset selection
Model creation
Training and tests on the model with the dataset
Results
Improvements to the model
Implementation
Implementation of ingredient recognition
Implementation of the recipe recommendation system
Final tests
Creation of documentation

c) Programming Languages And Development Tools -

Here are the Programming Languages we used -

1. React Native:

RecipeWizard utilizes React Native, an open-source framework developed by Facebook, to build cross-platform mobile applications. By leveraging the power of React, a JavaScript library for constructing user interfaces, RecipeWizard allows developers to create native mobile apps for both iOS and Android platforms using a single, shared codebase. The component-based architecture of React Native enhances code reusability, making it efficient for developers to maintain and update the app across different platforms simultaneously.

In the context of RecipeWizard, React Native empowers the development team to craft a seamless and performant user interface that feels native to both iOS and Android users. This not only streamlines the development process but also ensures a consistent and engaging user experience across a variety of devices.

2. Expo:

Expo serves as a valuable set of tools and services built around React Native, specifically designed to simplify and accelerate the mobile app development process for RecipeWizard. It provides a development environment, a command-line interface, and a variety of pre-built components and libraries. Expo eliminates the need for complex native modules setup, making it an excellent choice for rapid prototyping and development.

For RecipeWizard, Expo enhances the development workflow by providing tools for building, testing, and deploying the app. The Expo client app facilitates easy testing on physical devices during development, and Expo's build service simplifies the process of generating standalone app binaries for distribution.

3. JSON (JavaScript Object Notation):

JSON, a lightweight data interchange format, plays a crucial role in RecipeWizard by structuring and exchanging data between the frontend and backend of the application. Commonly used for representing recipe information, user profiles, and other relevant data, JSON ensures a standardized and interoperable data format. This facilitates seamless communication between RecipeWizard's frontend, backend, and any external APIs integrated into the system.

Certainly! Let's incorporate Python and machine learning (ML) into the development stack for RecipeWizard:

4. Python for Machine Learning:

Python, a versatile and widely-used programming language, is integrated into the RecipeWizard development stack to leverage machine learning capabilities. The Python programming language, along with popular ML libraries such as TensorFlow, PyTorch, or scikit-learn, allows RecipeWizard to incorporate intelligent features into the app. For example, machine learning algorithms can be applied to provide personalized recipe recommendations based on user preferences, dietary restrictions, and past cooking history.

Advantages of Python ML in RecipeWizard:

- Enhanced User Experience: Personalized recommendations and intelligent features enhance user engagement and satisfaction.

- Adaptability: Python's extensive ML ecosystem allows for the integration of cutting-edge algorithms and models to continually improve and adapt RecipeWizard's capabilities.

- Data-Driven Insights: Machine learning algorithms can provide valuable insights into user behaviour, helping RecipeWizard refine its features and offerings over time.

Integration of Python ML with React Native and Expo:

- Python-based ML models can be seamlessly integrated into the React Native codebase using suitable libraries or APIs. This integration ensures a cohesive user experience, where ML-driven features seamlessly blend with the app's overall functionality.

- Expo's flexibility supports the integration of additional functionalities, including those driven by machine learning, allowing RecipeWizard to harness the benefits of both React Native and Python ML in a unified development environment.

And the Development tools we used to make our project-

Android Studio:

Android Studio, the official integrated development environment (IDE) for Android app development, is utilized in the RecipeWizard project for fine-tuning the Android-specific aspects of the app. This includes optimizing the user interface for Android devices, managing resource files, and ensuring compatibility with various screen sizes and **resolutions. Android** Studio is also instrumental in testing and debugging the app on Android emulators or physical devices.

In summary, the combination of React Native, Expo, JSON, and Android Studio provides a robust and efficient development environment for creating the RecipeWizard app. This tech stack enables a cross-platform approach, accelerates development cycles, and ensures smooth communication and data exchange within the RecipeWizard application ecosystem.

System Requirement Specifications

a) Developing / Operating / Maintenance Environments -

1. Development Environment:

The development environment is where we written, test, and debug code. For RecipeWizard, a robust development environment we include is:

- Version Control System (VCS): Utilized a VCS Git to track changes in the codebase, allowing multiple our whole team to collaborate seamlessly (This steps makes our communication much easier and efficient.)

- Integrated Development Environment (IDE): We used popular IDEs such as Visual Studio Code IDEs (WebStorm for JavaScript/React).

- Node Package Manager (NPM): As RecipeWizard is built using React Native, npm is essential for managing packages and dependencies.

- Emulators and Simulators: We used mobile emulators (Android Emulator) to test the RecipeWizard app on various devices during development.

- Code Linting and Formatting: We Integrated Prettier like tools to maintain code consistency and quality.

2. Maintenance Environment:

The maintenance environment involves ongoing tasks related to monitoring, updates, and addressing potential issues for maintaining our RecipeWizard app-

- Monitoring Tools: We can implement monitoring tools (like New Relic, Datadog) to track performance, detect anomalies, and troubleshoot issues in real-time.

- Regular Backups: After collecting feedback we can creates regular backups of the database to prevent data loss and facilitate recovery in case of unexpected events.

- Patch and Update Management: Can regularly update dependencies, frameworks, and libraries to address security vulnerabilities and benefit from new features.

b) External Interface and Data Flows -

1. Data Aggregation from External Recipe Websites:

RecipeWizard aims to provide users with an extensive and diverse recipe catalog. To achieve this, the application connects to various external recipe websites using web scraping techniques. The process involves fetching data from these websites and integrating it into RecipeWizard's centralized recipe database.

2. BeautifulSoup for Web Scraping:

BeautifulSoup, a Python library, is utilized for web scraping tasks. It allows RecipeWizard to parse HTML and XML documents, extract relevant data, and navigate the structure of web pages. With BeautifulSoup, the application can systematically gather information such as recipe titles, ingredients, instructions, and images from external websites.

3. Automated Data Extraction:

RecipeWizard employs automated scripts that utilize BeautifulSoup to visit selected recipe websites, simulate user interactions, and extract relevant data. This data extraction process occurs in a systematic and non-disruptive manner, ensuring adherence to website policies and ethical scraping practices.

4. Integration with RecipeWizard Recipe Database:

The extracted data is then integrated into RecipeWizard's recipe database, enriching the app's content. The database is designed to accommodate diverse recipe structures from different websites, ensuring uniformity and consistency in presentation for RecipeWizard users.

5. Regular Data Updates:

To keep the recipe catalog current and comprehensive, RecipeWizard regularly updates its database by re-scraping external websites. This ensures that users have access to the latest and widest variety of recipes from across the web.

6. Enhancing Recipe Information:

BeautifulSoup allows RecipeWizard to not only collect basic recipe details but also to enhance the information by pulling additional metadata. This may include nutritional information, cooking times, difficulty levels, and user ratings, providing users with a more detailed and informed recipe selection.

User display and report format, user command summary

1. User Display:

- Dashboard: Upon logging in, users are greeted with a personalized dashboard. The dashboard may feature recommended recipes.

- Search Interface: The primary interaction point for users is a user-friendly search interface. Users can enter ingredients, cuisine preferences, or dietary restrictions to discover recipes.

- Recipe Cards: Search results are presented as visually appealing recipe cards. Each card displays the recipe title, image, key ingredients, and a brief description.

- Detailed Recipe View: Clicking on a recipe card opens a detailed view. This view includes the full recipe, step-by-step instructions, nutritional information, and user ratings.

2. User Commands and Interactions:

- Search Commands:

  - Text-Based Search: Users can enter ingredients, dish names, or cuisines into the search bar.

  - Ingredient Scaling: Users can adjust ingredient quantities based on the number of servings they desire.

- Reporting and Analytics:

  - Recipe History: Users can view their cooking history, including recipes they've tried and their success ratings.

  - Nutritional Insights: The app may provide nutritional insights based on users' cooking preferences and history.

- Reporting and Analytics:

  - `nutritional insights`: Prov1ides insights into the user's nutritional preferences.

4. Reporting Format:

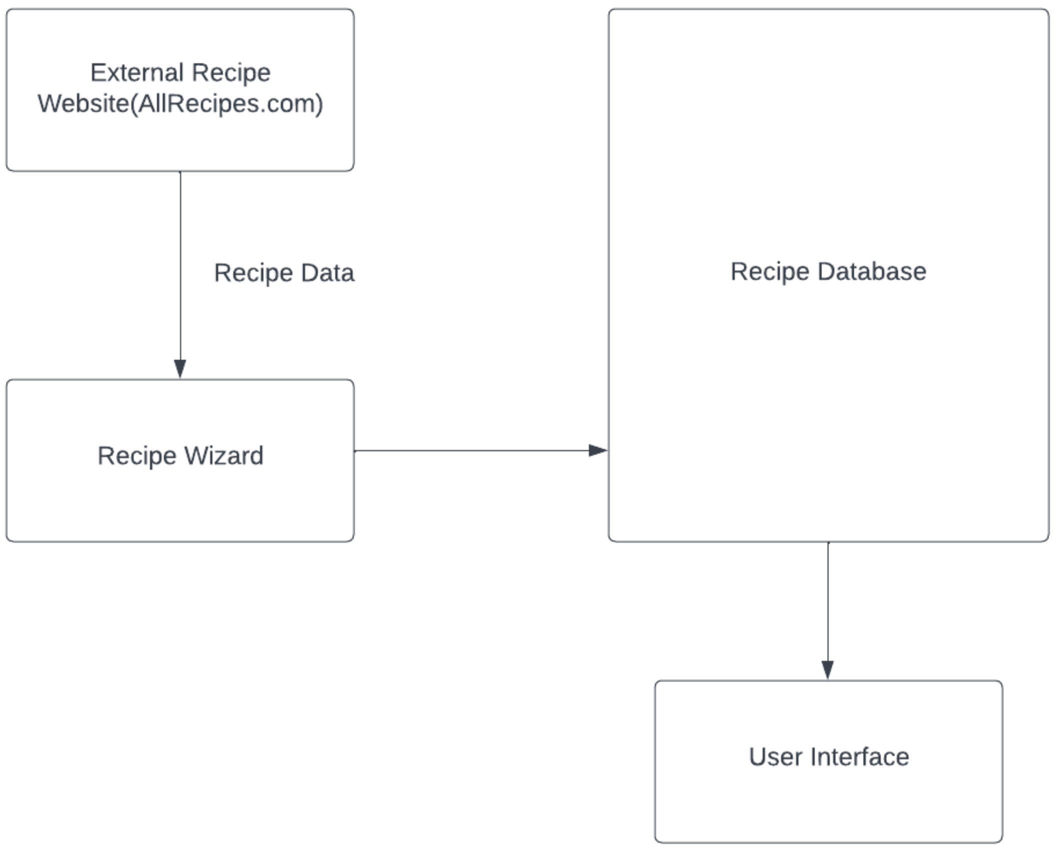
- Nutritional Insights Report:

  - Highlights nutritional patterns based on the user's cooking and ingredient preferences.

  - Provides suggestions for a more balanced diet.

In summary, RecipeWizard's user display focuses on a visually appealing and intuitive interface, while user commands cover various interactions from searching and filtering to profile management and recipe interaction. The reporting format provides users with valuable insights into their cooking habits and preferences. This combination enhances the overall user experience and engagement with the RecipeWizard app.

a) High Level DFD and data dictionary -



Data Dictionary:

Entities:

1. External Recipe Websites:
    - Attributes: ALLRecipes.com
2. RecipeWizard:
    - Attributes: Search Queries, Ingredients List, Recipe Data
3. Recipe Database:
    - Attributes: Recipe ID, Title, Ingredients, Instructions
4. User Interface:
    - Attributes: User Commands, Displayed Recipes, Add Ingredients

Processes:

1. Extract Recipe Data:
    - Description: Automated scripts using BeautifulSoup to scrape data from external recipe websites.
    - Inputs: URLs of external recipe websites.
    - Outputs: Extracted recipe data.
2. Store in Database:
    - Description: Stores the extracted recipe data into the Recipe Database.
    - Inputs: Recipe data.
    - Outputs: Stored recipe data.
3. Retrieve for Display:
    - Description: Retrieves recipe data from the database for user display.
    - Inputs: User requests for recipes.
    - Outputs: Displayed recipe data.
4. User Interaction:
    - Description: Handles user commands and interactions with the RecipeWizard interface.
    - Inputs: User commands, displayed recipes.
    - Outputs: Updates to user profiles, saved recipes, and user interface.

b) Functional and performance specifications-

Functional Specifications:

1. Recipe Search and Filtering:

- Requirement: Users can search for recipes based on ingredients, cuisine, or dietary preferences.

- Feature: Advanced search functionalities, including filters for dietary restrictions.

2. Recipe Display and Details:

- Requirement: Detailed recipe information, including ingredients, instructions, and nutritional facts.

- Feature: Visually appealing recipe cards, step-by-step instructions, and optional nutritional information.

3. User Interaction:
   - Requirement: Users can rate recipes, leave reviews, and share their cooking experiences.
   - Feature: Ratings, reviews, and social sharing options integrated into the user interface.

4. Cross-Platform Compatibility:
   - Requirement: RecipeWizard works seamlessly on both iOS and Android devices.
   - Feature: Native experiences on both platforms using technologies like React Native.

5. Offline Access:
   -Requirement: Users can access saved recipes and some functionalities offline.
   -Feature: Offline caching of favorite recipes and recent searches.

6. Social Integration:
   - Requirement: Integration with social media for easy sharing and interaction.
   - Feature: Share recipes, achievements, and cooking experiences on social platforms.

7. Accessibility:
   - Requirement: RecipeWizard is accessible to users with disabilities.

- Feature: Compliance with accessibility standards, including screen reader compatibility.

<u>Performance Specifications:</u>

1. Response Time:
   - Requirement: Quick response times for search queries and interactions.
   - Specification: 95% of search queries respond within 2 seconds.

2. Scalability:
   - Requirement: RecipeWizard should handle an increasing number of users and recipes.

- Specification: Scalability testing to ensure performance under high user loads.

3. Reliability and Uptime:

   - Requirement: The platform should be highly reliable with minimal downtime.
   - Specification: 99.9% uptime over a specified period.

4. Data Security:

   - Requirement: User data, including personal information, should be securely stored.
   - Specification: Implementation of encryption protocols and regular security audits.

5. Device Compatibility:

   - Requirement: RecipeWizard works well on various devices with different screen sizes.
   - Specification: Responsive design ensuring usability on smartphones, tablets, and desktops.

6. Update and Maintenance:

   - Requirement: Regular updates and maintenance should not disrupt user experience.
   - Specification: Scheduled maintenance windows with minimal impact on user accessibility.

7. Third-Party Integrations:

   - Requirement: Smooth integration with external APIs and services.

- Specification: Test performance during interactions with external databases and APIs.

Design

a) Detailed DFD's and structure diagrams -

Level 0 DFD overview :

Level 1 DFD – User interface Module:

Component Diagram - RecipeWizard Application:

b) Data Structures, database and file specifications -

Certainly! To design the data structures, database, and file specifications for RecipeWizard, we'll consider the entities involved in the system and the relationships between them.

Data Structures:

1. Recipe Data Structure:
   - Attributes:
     - Recipe ID (Unique identifier)
     - Title
     - Ingredients (List or structured format)
     - Instructions (List or structured format)
     - Cooking Time
     - Difficulty Level
     - Nutritional Information

- User Ratings
- Reviews (List or structured format)
- Cuisine Type

- Dietary Tags (List)

2. Collection Data Structure:
   - Attributes:
     - Collection ID (Unique identifier)
     - Title
     - Description

- Recipes (List of Recipe IDs)

File Specifications:

1. Image Storage:
   - Storing recipe images in a dedicated file storage system or cloud storage (e.g., AWS S3).
   - Implementing a structured storage system, categorizing images by recipe or user.

2. File Formats:
   - Using common file formats such as JPEG or PNG for recipe images.
   - Text-based formats (JSON, XML) for exporting or importing data.

3. Backup and Restore:
   - Regularly back up the database and associated files to prevent data loss.
   - Implementing a process for restoring data from backups in case of emergencies.

4. File Upload Handling:

- Implementing secure file upload mechanisms with proper validation to prevent common security issues.

Security Considerations:

1. Data Encryption:

   - Encrypting sensitive data, especially user passwords, using strong encryption algorithms (e.g., bcrypt).

2. Access Control:

   - Implementing proper access controls to restrict unauthorized access to sensitive data.

3. Audit Trails:

   - Maintaining audit trails to log changes to critical data (e.g., user profile modifications, recipe updates).

4. Backup Encryption:

- Encrypting database backups to protect sensitive information during storage.

These specifications provide a foundation for building a scalable, secure, and efficient RecipeWizard system.

<u>c) Pseudo code -</u>

Certainly! Below is a high-level pseudo-code representation of the steps involved in building RecipeWizard using React Native, Expo, and JSON. Please note that this is a conceptual overview, and actual implementation details may vary.

Pseudo-Code Overview:

<u>1. Initialize Expo Project:</u>

```
# Terminal
#after opening file in terminal
expo init RecipeWizardApp
cd RecipeWizardApp
```

2. Create Navigation Structure:

```
# Install React Navigation
npm install @react-navigation/native @react-navigation/stack

# App.js
import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';
const Stack = createStackNavigator();
function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator>
        {/* Define screens */}
      </Stack.Navigator>
    </NavigationContainer>
  );
}
```

3. Create Screens:

```
# Create various screens using React components
# Example: HomeScreen, RecipeDetailScreen, UserProfileScreen
```

4. Fetch Recipes from JSON API:

```
# Install Axios for API requests
npm install axios
# HomeScreen.js
import axios from 'axios';

function HomeScreen() {
  useEffect(() => {
    # Fetch recipes from JSON API
    axios.get('https://api.recipewizard.com/recipes')
      .then(response => {
        # Handle the data
      })
      .catch(error => {
        # Handle errors
      });
  },
[]);

  return (
    # Display recipe data
  );
}
```

<u>5. Implement Recipe Search:</u>

# Add a search bar to HomeScreen

# Fetch recipes based on search input

<u>6. User Authentication:</u>

# Install Firebase or Auth0 for authentication

npm install firebase

# Implement user authentication in UserProfileScreen

# Handle user login, registration, and logout

<u>7. Integrate Machine Learning for Recommendations</u>

# Use a pre-trained ML model or train a model with user preferences

# Provide personalized recipe recommendations

<u>8. Store User Data in JSON Format:</u>

# Store user data in JSON format or use a NoSQL database

# Serialize and deserialize data as needed

<u>9. Optimize and Test:</u>

# Optimize app performance and test on various devices

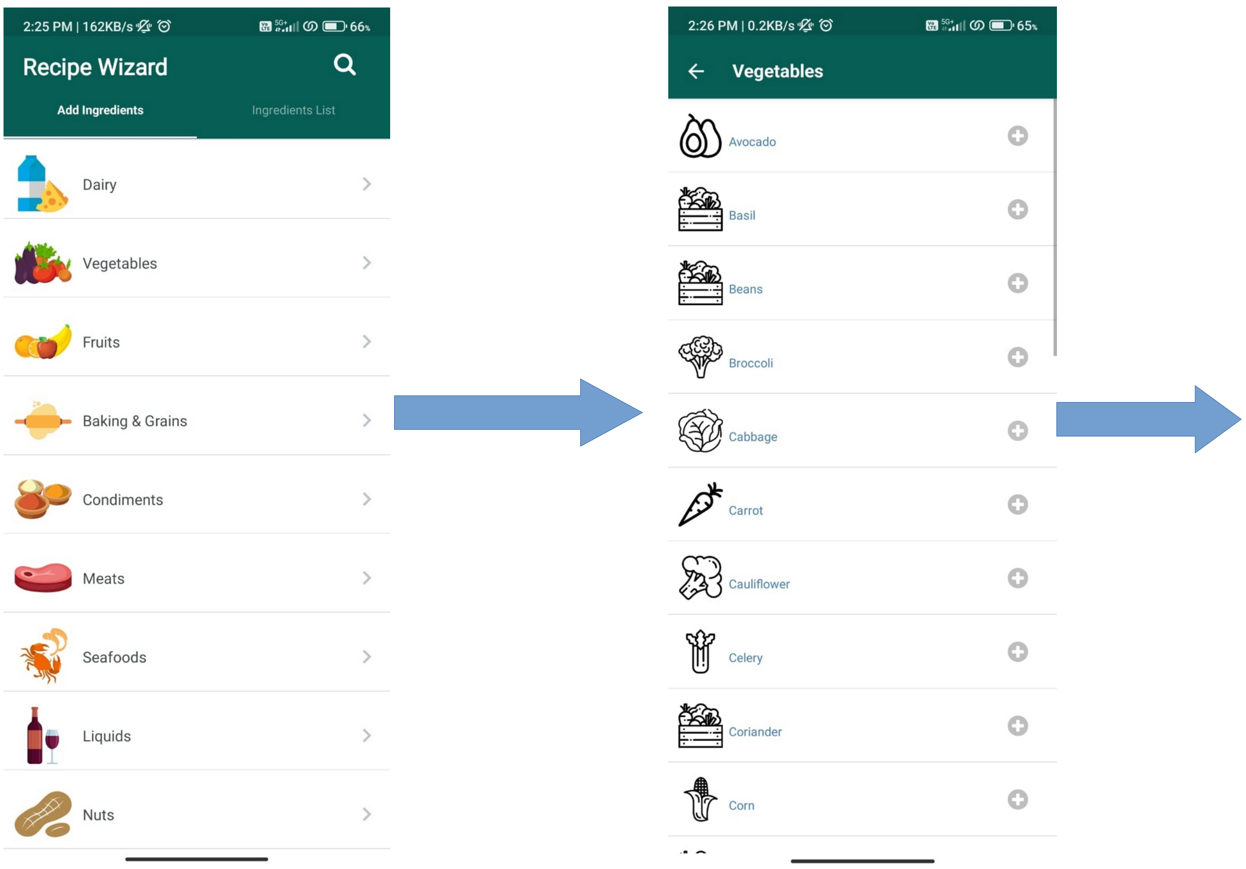
# Address any bugs or issues

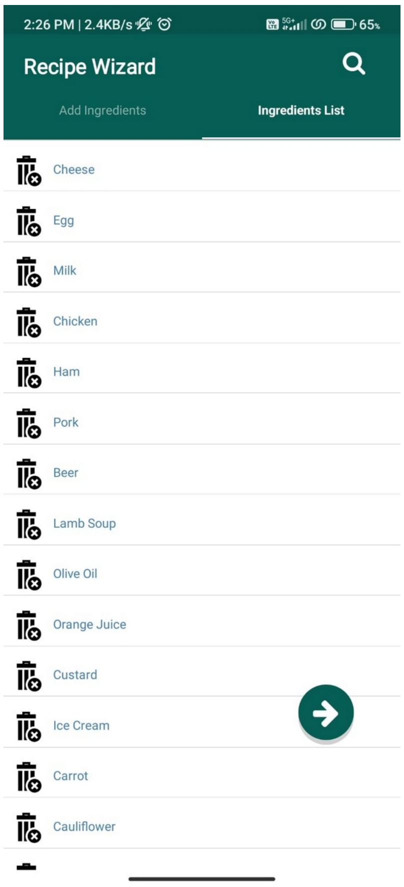
10. Build and Deploy:

```
# Build the app for iOS and Android
expo build:ios
expo build:android
# Deploy to app stores or distribute through Expo
```

This pseudo-code provides a basic outline of the steps involved in building RecipeWizard using React Native, Expo, and JSON. The actual implementation will require detailed coding, UI design, and integration with external services as needed.

User Inter Face Design-



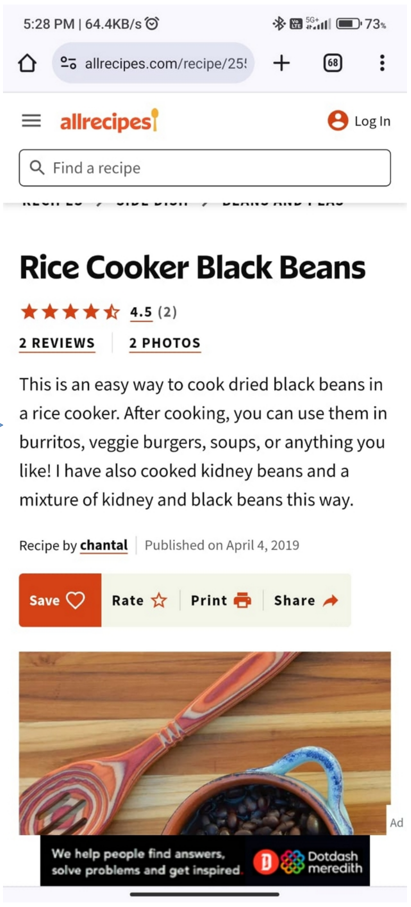


2:26 PM | 2.4KB/s 65%

Recipe Wizard

Add Ingredients
Ingredients List

Cheese
Egg
Milk
Chicken
Ham
Pork
Beer
Lamb Soup
Olive Oil
Orange Juice
Custard
Ice Cream
Carrot
Cauliflower



5:28 PM | 64.4KB/s 73%

allrecipes.com/recipe/25

allrecipes

Log In

Find a recipe

Rice Cooker Black Beans

4.5 (2)

2 REVIEWS | 2 PHOTOS

This is an easy way to cook dried black beans in a rice cooker. After cooking, you can use them in burritos, veggie burgers, soups, or anything you like! I have also cooked kidney beans and a mixture of kidney and black beans this way.

Recipe by chantal | Published on April 4, 2019

Save | Rate | Print | Share

Ad

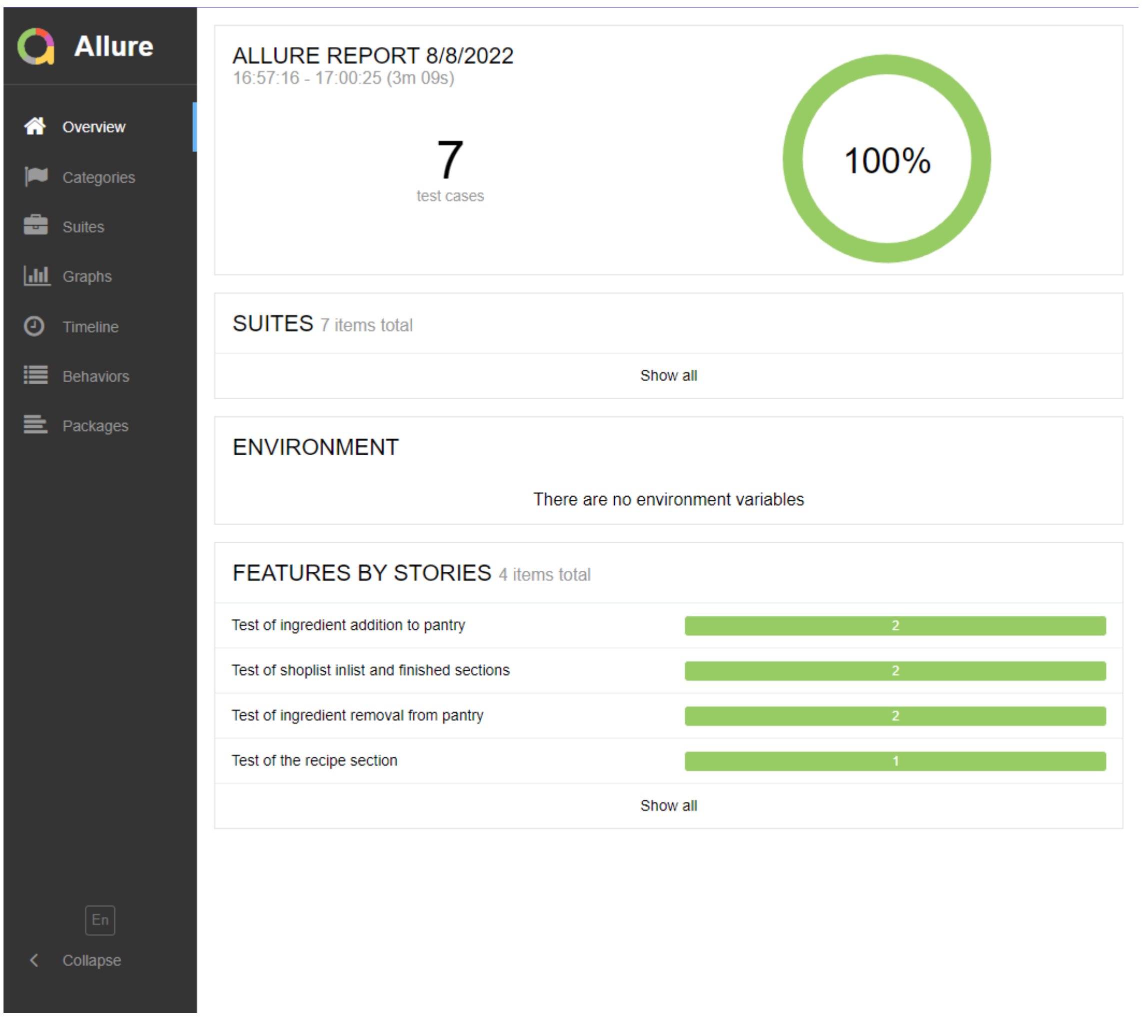
We help people find answers, solve problems and get inspired Dotdash meredith

Test Plan

Functional, Performance, Stress Tests -

Automated testing-

Automated testing refers to the use of specialized tools and scripts to execute predetermined test cases on a software application. This approach is more efficient than manual testing, as it allows for quick and repeatable test executions. Automated tests ensure consistency by performing the same steps in the same way every time, reducing the chances of human errors associated with manual testing. They play a crucial role in regression testing, where the goal is to confirm that new code changes do not introduce unintended side effects.
The need for automated testing arises from its ability to offer efficiency, repeatability, and cost-effectiveness in the software development process. Automated tests can be swiftly and consistently executed, making them ideal for projects with frequent releases or complex functionalities. This methodology enhances test coverage, enabling the examination of a broad range of scenarios and data combinations that might be impractical with manual testing. Additionally, automated testing supports Continuous Integration and Continuous Deployment (C/CD) practices, facilitating the rapid and reliable testing of code changes, which is vital for delivering high-quality software.
In summary, automated testing is a valuable practice that ensures the efficiency, repeatability, and cost-effectiveness of the software testing process. Its ability to provide consistent results, cover a wide range of scenarios, and seamlessly integrate with modern development practices makes it an indispensable component in ensuring the reliability and quality of software applications.



ALLURE REPORT 8/8/2022
16:57:16 - 17:00:25 (3m 09s)

7
test cases

100%

SUITES 7 items total

Show all

ENVIRONMENT

There are no environment variables

FEATURES BY STORIES 4 items total

Test of ingredient addition to pantry	2
Test of shoplist inlist and finished sections	2
Test of ingredient removal from pantry	2
Test of the recipe section	1

Show all

Implementation / Conversion Plan

1. Project Kickoff:
   - Conducting a project kickoff meeting to align the development team on goals, requirements, and timelines.

2. Requirement Analysis:
   - Gathering detailed requirements for RecipeWizard, including core features, user stories, and any specific functionalities.
   - Collaborating with stakeholders, including designers and product owners, to finalize the feature set.

3. Technology Stack:
   - Finalizing the technology stack, considering factors like React Native, Expo, JSON, Android Studio, and any additional tools or libraries required.

4. Design Wireframes:
   - Creating detailed wireframes and design mockups for RecipeWizard's user interface.
   - Gathering feedback from stakeholders and make necessary adjustments.

5. Divide Work Among Team Members:
   - Dividing the development work among team members based on their expertise and the different components of RecipeWizard.

6. Backend Development:
   - Developing the backend infrastructure for RecipeWizard, including the recipe database, user authentication, and any server-side functionalities.
   - Implementing data structures and set up the database schema.

7. Frontend Development:
   - Started developing the frontend of RecipeWizard using React Native and Expo.
   - Implementing core features such as recipe search, display, user profiles,

8. Integration of Machine Learning:
   - Integrated machine learning models for personalized recipe recommendations.
   - Test and optimize the recommendation system.

9. Testing:
   - Conducted thorough testing, including functional, performance, and stress testing.
   - Addressing any bugs or issues identified during the testing phase.

10. User Acceptance Testing (UAT):
   - Involving stakeholders and end-users in UAT to validate that RecipeWizard meets their expectations.
   - Gathering feedback for any final adjustments.

11. Implementation of Design:
   - Implementing the final design based on approved wireframes.
   - Ensuring a consistent and visually appealing user interface.

12. Documentation:
- Creating comprehensive documentation, including user guides, API documentation, and any technical documentation needed for future maintenance.

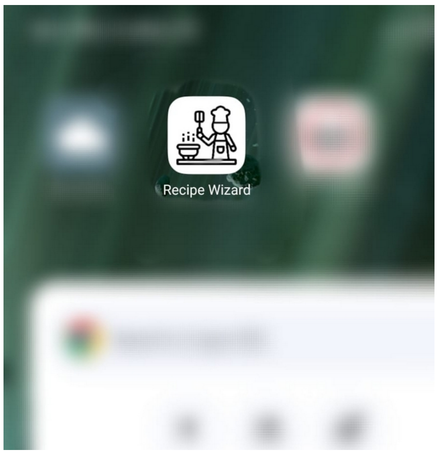
13. Deployment:
- Deploying RecipeWizard to production or relevant app stores.
- Monitoring the deployment for any issues and ensure a smooth transition.

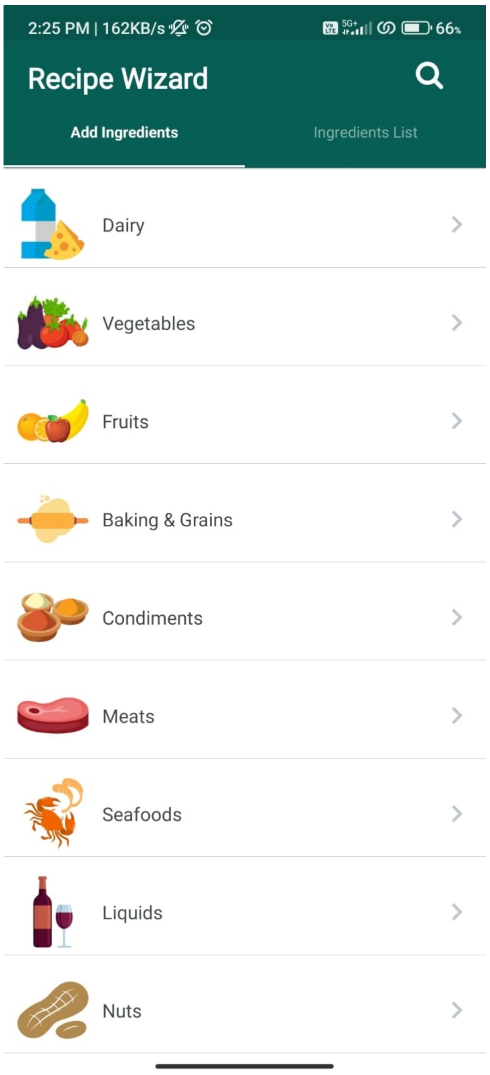
14. Continuous Improvement:
- Establishing a process for continuous improvement based on user feedback, analytics, and emerging technologies.

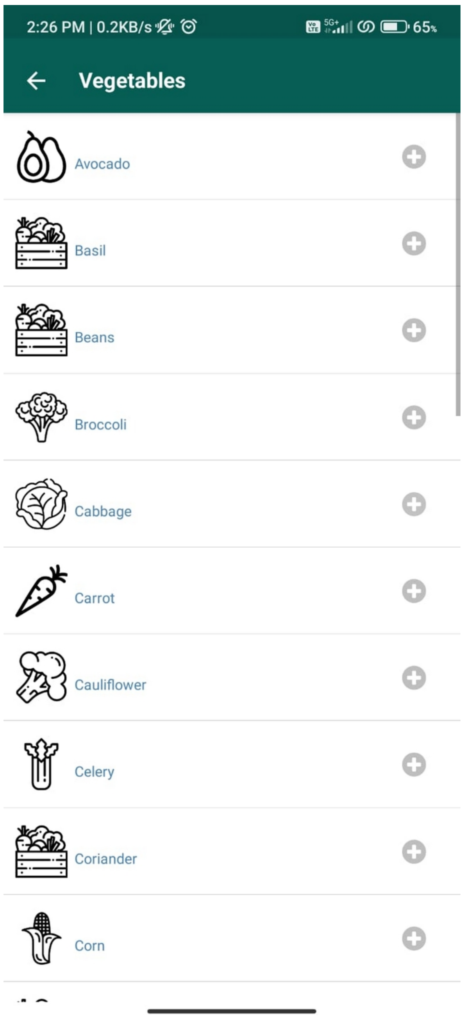
Project Legacy

a) Current Status of Project -

As of the latest update, the RecipeWizard project is progressing steadily through its development lifecycle. The development team successfully completed the backend infrastructure, implementing a robust recipe database and user authentication system. Frontend development, utilizing React Native and Expo, is in full swing, with core features such as recipe search, display, and user profiles taking shape. The integration of machine learning models for personalized recipe recommendations is underway, promising an enhanced user experience. The testing phase, encompassing functional, performance, and user acceptance testing, has identified and addressed several minor issues, ensuring the application's stability. The design implementation aligns seamlessly with approved wireframes, delivering a visually appealing and consistent user interface. The project is on track for a scheduled deployment, with a comprehensive marketing and promotion plan in development to increase RecipeWizard's visibility upon release. Continuous improvement processes are being established based on ongoing user feedback and emerging technologies, underscoring the commitment to delivering a high-quality and innovative culinary experience.

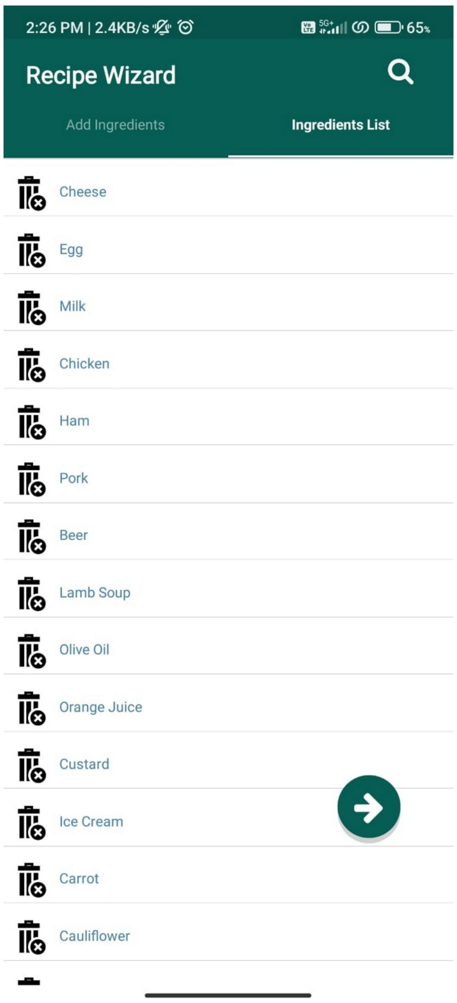





2:26 PM | 0.2KB/s 65%

Vegetables

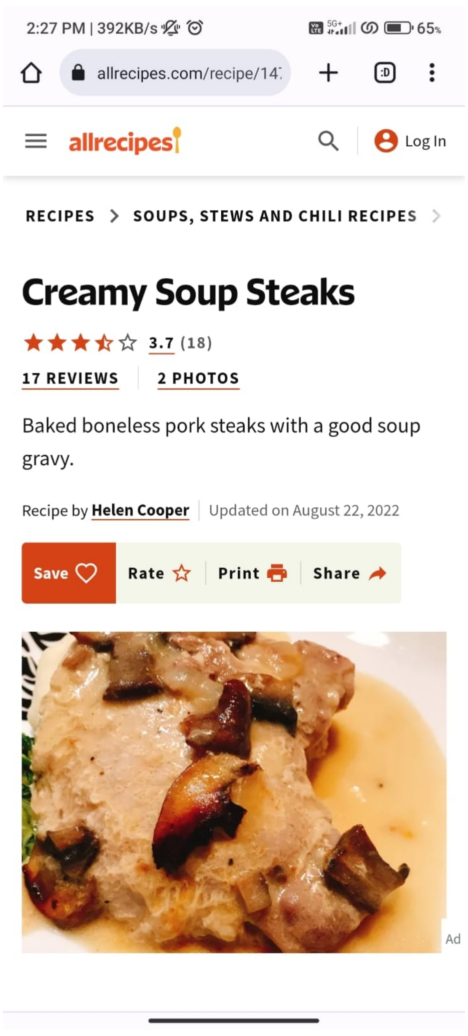
- Avocado
- Basil
- Beans
- Broccoli
- Cabbage
- Carrot
- Cauliflower
- Celery
- Coriander
- Corn


2:26 PM | 2.4KB/s 65%

Recipe Wizard

Add Ingredients | Ingredients List

- Cheese
- Egg
- Milk
- Chicken
- Ham
- Pork
- Beer
- Lamb Soup
- Olive Oil
- Orange Juice
- Custard
- Ice Cream
- Carrot
- Cauliflower


2:27 PM | 392KB/s 65%

allrecipes.com/recipe/14

allrecipes | Log In

RECIPES > SOUPS, STEWS AND CHILI RECIPES >

Creamy Soup Steaks

3.7 (18)

17 REVIEWS | 2 PHOTOS

Baked boneless pork steaks with a good soup gravy.

Recipe by Helen Cooper | Updated on August 22, 2022

Save | Rate | Print | Share

Ad

b) Remaining areas of concern -

Identifying remaining areas of concern in developing RecipeWizard would require a detailed analysis of the project's current status, potential challenges, and the scope of work. However, here are some common areas that often require attention in app development projects like RecipeWizard:

1. Performance Optimization:
   - Ensuring that the app's performance meets the desired standards, especially when handling a large database of recipes and user interactions.

2. User Experience (UX) Refinement:
   - Continuously evaluating and refine the user interface to enhance the overall user experience. Addressing any usability issues or potential points of confusion.

3. Security Measures:
   - Implementing robust security measures to protect user data, especially considering the sensitive nature of personal information and login credentials.

4. Cross-Platform Consistency:
   - Verifying that the app functions consistently across different devices and platforms, maintaining a uniform user experience.

5. Accessibility Features:
   - Ensuring the app is accessible to users with disabilities by incorporating features like screen reader compatibility and adherence to accessibility standards.

6. Testing Across Devices and Browsers:
   - Conducting thorough testing on various devices and browsers to identify and resolve any compatibility issues that may arise.

7. Scalability Planning:
   - Considering the potential growth of user data and the recipe database. Implement scalable solutions to accommodate increased traffic and data volume.

8. Data Integrity and Validation:
   - Implementing data validation measures to ensure the integrity of user inputs and maintain a clean and accurate recipe database.

9. Integration with External Platforms:
   - If RecipeWizard integrates with external platforms or APIs for recipe information, verify the robustness of these integrations and address any issues that may arise.

10. Offline Functionality:
    - Confirming that the offline functionality, such as accessing saved recipes without an internet connection, works seamlessly and provides a good user experience.

11. Backup and Recovery Procedures:
    - Implementing robust backup and recovery procedures to safeguard user data in case of unforeseen events or technical issues.

c) Technical and Managerial lessons learnt -

During the development of the RecipeWizard app, various technical and managerial lessons were learned that contributed to the project's success. Here are some key insights:

Technical Lessons Learned:

1. Cross-Platform Development:
   - Embracing cross-platform development with React Native proved beneficial, allowing the team to efficiently develop and maintain a single codebase for both iOS and Android platforms.

2. Expo for Rapid Prototyping:
   - Utilizing Expo for development streamlined the prototyping phase. Its tools and services accelerated the development process, enabling quick testing on physical devices during the early stages.

3. JSON for Data Interchange:
   - Employing JSON as a data interchange format facilitated seamless communication between the app's frontend, backend, and external APIs. It provided a human-readable and standardized format for data representation.

4. Integration with External Platforms:
   - Integrating external recipe websites using BeautifulSoup for web scraping enhanced the app's content significantly. However, it required careful consideration of ethical scraping practices and adherence to website policies.

5. Machine Learning for Personalization:
   - Implementing machine learning algorithms for personalized recipe recommendations enhanced user engagement. However, the complexity of these algorithms required meticulous testing and optimization.

6. Security Measures:
   - Robust security measures were crucial, especially when handling user data. Implementing encryption protocols and following best practices ensured the protection of sensitive information.

7. Scalability Planning:
   - Considering scalability from the early stages allowed the app to handle potential growth in user data and recipe databases. Scalable solutions were implemented to accommodate increased traffic.

Managerial Lessons Learned:

1. Agile Development Methodology:
   - Adopting an agile development methodology facilitated flexibility and adaptability to changing requirements. Regular sprints and feedback loops ensured continuous improvement.

2. Clear Communication:
   - Clear and open communication within the development team and with stakeholders was essential. Regular meetings and updates kept everyone aligned with project goals and progress.

3. User-Centric Design:
- Prioritizing user experience and continuously gathering user feedback shaped design decisions. User-centric design principles contributed to the app's intuitiveness and appeal.

4. Documentation Importance:
- Comprehensive documentation, including user guides and technical documentation, proved invaluable for future reference and knowledge transfer within the team.

5. Legal and Compliance Awareness:
- Staying informed about legal requirements and compliance, especially regarding data privacy and terms of service agreements, ensured a smooth and legally sound operation.

6. Post-Launch Support Planning:
- Planning for post-launch support, including addressing user issues and maintaining the app's performance, was crucial for ensuring a positive user experience.

7. Team Collaboration:
- Encouraging collaboration and fostering a positive team culture contributed to a productive work environment. Team members were motivated and engaged throughout the project.

8. Continuous Improvement Mindset:
- Adopting a mindset of continuous improvement allowed the team to iterate on features, address issues promptly, and stay responsive to user needs.

These technical and managerial lessons learned provide valuable insights for future app development projects, emphasizing the importance of flexibility, user focus, and effective collaboration to all the team.

d) Future Recommendation -

For future enhancements and recommendations for RecipeWizard, consider the following suggestions to further improve the app:

1. Advanced Search Filters:
   - Implement more advanced search filters, allowing users to refine their recipe searches based on specific criteria such as cooking time, difficulty level, or dietary preferences.

2. Enhanced User Profiles:
   - Expand user profiles to include features like personal recipe collections, a cooking history log, and the ability to share achievements or favorite recipes with other users.

3. Interactive Cooking Guides:
   - Integrate interactive cooking guides or tutorials within recipes, providing step-by-step instructions with visual cues, timers, and multimedia content for a more engaging cooking experience.

4. Community Features:
   - Foster a sense of community by incorporating social features such as user forums, recipe sharing, and the ability to connect with other cooking enthusiasts. User-generated content could add value to the platform.

5. Integration with Smart Appliances:
   - Explore partnerships or integrations with smart kitchen appliances. This could include features like sending recipes directly to smart ovens, adjusting cooking temperatures, or setting timers through the RecipeWizard app.

6. AI-Powered Personalization:
   - Further enhance personalization using advanced artificial intelligence (AI) algorithms. Analyze user behavior to provide even more accurate and tailored recipe recommendations over time.

7. Multi-Language Support:
   - Expand RecipeWizard user base by introducing support for multiple languages. This would make the app more accessible to a global audience, catering to users with different language preferences.

8. Collaborations with Chefs and Food Bloggers:
   - Collaborate with renowned chefs or food bloggers to feature exclusive recipes or cooking tips within the app. This can enhance the app's credibility and provide users with unique and high-quality content.

9. Augmented Reality (AR) Features:
   - Explore the integration of augmented reality features for a more immersive cooking experience. Users could use AR to visualize recipes in their own kitchens or get real-time guidance on cooking techniques.

10. Offline Community Challenges:

- Introduce offline community challenges or cooking contests to encourage user engagement. Users could submit their creations, vote on favorite recipes, and participate in themed cooking events.

11. Educational Content:
   - Include educational content on various cooking techniques, ingredients, or cultural aspects of different cuisines. This not only adds value to users but also positions RecipeWizard as an educational resource.

12. Integration with Grocery Shopping Apps:
   - Allow users to seamlessly add recipe ingredients to their grocery shopping lists or integrate with grocery delivery apps to streamline the cooking preparation process.

13. Adaptive Learning Algorithms:
   - Implement adaptive learning algorithms that evolve based on user feedback and preferences, continuously refining the recipe recommendations to better suit individual tastes.

14. Voice-Activated Commands:
   - Incorporate voice-activated commands for hands-free recipe navigation in the kitchen. Users could ask for the next step, set timers, or adjust quantities using voice commands.

15. Sustainability and Nutrition Information:
   - Provide information on the sustainability of ingredients and nutritional details for each recipe, catering to users who are conscious of their environmental impact and dietary choices.

By implementing these future recommendations, RecipeWizard can evolve into a more comprehensive and engaging platform, catering to a wider audience and providing users with an enriched culinary experience.

Source Code

Main File files → (files)

App.js

```
import {
  createStackNavigator,
} from 'react-navigation';

import React  from 'react';
import {
} from 'react-native';

import MainMenu from './src/components/MainMenu/MainMenu';
import Search from './src/components/Search/Search';
import LoadScreen from './src/components/LoadScreen/LoadScreen';
import Dairy from './src/components/Dairy/Dairy';
import Fruits from './src/components/Fruits/Fruits';
import Baking from './src/components/Baking/Baking';
import Meats from './src/components/Meats/Meats';
import Sweeteners from './src/components/Sweeteners/Sweeteners';
import Vegetables from './src/components/Vegetables/Vegetables';
import TabBar from './src/components/TabBar/TabBar';
import CheckOut from './src/components/CheckOut/CheckOut';
import Results from './src/components/Results/Results';
import Seafoods from './src/components/Seafoods/Seafoods';
import Liquids from './src/components/Liquids/Liquids';
import Nuts from './src/components/Nuts/Nuts';

const App = createStackNavigator({

  MainMenu: { screen: MainMenu },
  Search: { screen: Search },
  Meats: { screen: Meats },
  Liquids: { screen: Liquids },
  Seafoods: { screen: Seafoods },
  LoadScreen: { screen: LoadScreen },
  Sweeteners: { screen: Sweeteners },
  Dairy: { screen: Dairy },
  Baking: { screen: Baking },
  Fruits: { screen: Fruits },
  Vegetables: { screen: Vegetables },
  CheckOut: { screen: CheckOut },
  Nuts: { screen: Nuts },
  Results: { screen: Results }
},
  {
    initialRouteName: 'MainMenu', //TODO
    cardStyle: { backgroundColor: '#FFFFFF' },
  }
);

export default App;
```

1. Import necessary modules from react-navigation and react-native.
2. Import various components of your application, such as MainMenu, Search, LoadScreen, Dairy, Fruits, Baking, Meats, Sweeteners, Vegetables, TabBar, CheckOut, Results, Seafoods, Liquids, and Nuts.
3. Create a stack navigator (App) with screens corresponding to the imported components.
4. Set the initial route to be 'MainMenu'.
5. Set the card style to have a background color of #FFFFFF.

App.json

```
{
  "expo": {
    "name": "Recipe Wizard",
    "description": "Recipe generating application :)",
    "slug": "RW",
    "privacy": "public",
    "sdkVersion": "31.0.0",
    "platforms": ["ios", "android"],
    "version": "1.3.0",
    "orientation": "portrait",
     "androidStatusBar": {
      "backgroundColor": "#000000"
    },
    "icon": "logo1.png",
    "splash": {
      "image": "./assets/splash.png",
      "resizeMode": "contain",
      "backgroundColor": "#ffffff"
    },
    "updates": {
      "fallbackToCacheTimeout": 0
    },
    "assetBundlePatterns": [
      "**/*"
    ],
    "ios": {
      "supportsTablet": true
    },
     "android": {
       "versionCode": 3,
      "package": "com.recipewizard",
       "permissions": []
    }
  }
}
```

Scraper.py (Main search engine)

```
from recipe_scrapers import scrape_me
lines = [line.rstrip('\n') for line in open('output.txt')]

wordlist = ['baking soda', 'baking powder', 'batter', 'blueberries', 'bread', 'chocolate', 'coriander',
'cocoa', 'dough', 'flour', 'noodles', 'pasta', 'rice', 'vanilla', 'yeast', 'avocado', 'basil', 'beans',
'broccoli', 'cabbage', 'carrot', 'cauliflower', 'celery', 'corn', 'cucumber', 'eggplant', 'garlic', 'ginger',
'lettuce', 'mint', 'mushroom', 'olive', 'onion', 'oregano', 'pickle', 'potato', 'pumpkin', 'seed', 'shallot',
'spinach', 'sweet potato', 'thyme', 'tomato', 'zucchini','butter', 'butter milk', 'cheese', 'custard', 'egg',
'ice cream', 'milk', 'sour cream', 'yoghurt', 'apple', 'banana', 'blackberries', 'cherries', 'coconut',
'cranberries', 'grapes', 'kiwi', 'lemon', 'lime', 'mango', 'orange', 'peach', 'pear', 'pineapple',
'raspberries', 'strawberries', 'beef soup', 'beer', 'chicken soup', 'lamb soup', 'olive oil', 'red wine',
'vegetable oil', 'white wine', 'barbeque sauce', 'syrup', 'fish sauce', 'honey', 'mayonnaise',
'mustard', 'soy sauce', 'vinegar', 'carp', 'catfish', 'crab', 'eel', 'lobster', 'mackerel', 'mussel', 'oyster',
'prawn', 'salmon', 'sardine', 'scallop', 'shrimp', 'squid', 'trout', 'tuna', 'almond', 'cashew',
'macadamia', 'peanut', 'peanut butter', 'walnut', 'bacon', 'beef', 'chicken', 'ham', 'lamb', 'pork',
'salami', 'sausage', 'turkey', 'paprika', 'apple juice', 'orange juice', 'pecan', 'allspice', 'nutmeg',
'tomato sauce', 'watermelon']
exceptions = ['peanut butter', 'butter milk', 'chicken soup', 'lamb soup', 'beef soup', 'eggplant',
'pineapple', 'watermelon', 'vinegar', 'apple juice', 'orange juice', 'tomato sauce']
defaults = [' ice', ' water', ' oil', ' salt', ' pepper', ':', 'optional', 'sugar', 'butter', 'seasoning']

f = open('Recipes.json','w')
f.write('{\n')
f.write('  "recipes" : {\n')

recipesValid = 0
recipesTotal = 0

for each in lines:

        if 'pudding' in each:
                continue

        recipesTotal += 1
        try:
                scraper = scrape_me(each)
        except:
                continue
        ingredients = scraper.ingredients()
        title = scraper.title().replace('"',"'")
        matches = []
        matchedLength = 0

        for ingredient in ingredients:

                ingredient = ingredient.lower()

                if 'cherry' in ingredient:
                        matches.append('cherries')
                        matchedLength += 1
                        break
                if 'strawberry' in ingredient:
                        matches.append('strawberries')
                        matchedLength += 1
                        break

                if 'raspberry' in ingredient:
                        matches.append('raspberries')
                        matchedLength += 1
                        break

                if 'ketchup' in ingredient:
                        matches.append('tomato sauce')
                        matchedLength += 1
                        break

                if any(exception in ingredient for exception in exceptions):
                        for exception in exceptions:
                            if exception in ingredient:
                                matches.append(exception)
                                matchedLength += 1
                                break

                elif any(substring in ingredient for substring in wordlist):
                        for substring in wordlist:
                            if substring in ingredient:
                                matches.append(substring)
                                matchedLength += 1
                                break

                elif any(default in ingredient for default in defaults):
                        for default in defaults:
                            if default in ingredient:
                                matchedLength += 1
                                break

        if matchedLength == len(ingredients) and len(matches) > 0:

                f.write('    "' + title + '"' + ': {' + '\n')
                matches = list(dict.fromkeys(matches))

                recipesValid += 1
                print(str(recipesValid) + "/" + str(recipesTotal))

                for match in matches:
                        f.write('        "' + match.replace(" ", "") + '"' + ': 1,' + '\n')
                f.write('        "size": ' + str(len(matches)) + ',' + '\n')

                f.write('        "base":' + '"' + title + '"' + ',' + '\n')
                f.write('        "link":' + '"' + each + '"' + '\n')
                f.write('    },' + '\n')

f.write('}}')
f.close()
```

TestURL.py

```
from bs4 import BeautifulSoup
import requests

f = open('output.txt','w')
for i in range(2501, 2827):
    url = "http://allrecipes.com/recipes/?sort=Title&page=" + str(i)
    print(i)
    r = requests.get(url)
    soup = BeautifulSoup(r.content, "lxml")
    links = soup.find_all('article', {'class': "fixed-recipe-card"})
    for ele in links:
        f.write(ele.a["href"] + '\n')

f.close()
```

MainFile → Components folder → (files)

IngredientsList →
All.js

```
import React from 'react'
import { View, Text, ActivityIndicator, StyleSheet } from 'react-native'

export default class Loading extends React.Component {
  render() {
    return (
      <View style={styles.container}>
        <Text>Loading</Text>
        <ActivityIndicator size="large" />
      </View>
    )
  }
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
  }
})
```

IngredientsList (ingredientslist.js)

```
import React from 'react';
import styles from'./Styles';

import {
    Text,
    View,
    ScrollView,
    StyleSheet,
    StatusBar,
    Image,
    ImageBackground,
    BackHandler,
    TouchableOpacity,
    Dimensions,
} from 'react-native';

import {
  List,
  ListItem,
  Icon,
  Header,
} from 'react-native-elements';

import { createStackNavigator } from 'react-navigation';
import Dairy from '../Dairy/Dairy';
import Meats from '../Meats/Meats';
import Baking from '../Baking/Baking';
import Sweeteners from '../Sweeteners/Sweeteners';
import Vegetables from '../Vegetables/Vegetables';
import Fruits from '../Fruits/Fruits';
import HeaderBar from '../HeaderBar/HeaderBar';
import All from './All';

const list = [
  {
    name: 'Dairy',
    avatar_url: 'https://s3.amazonaws.com/recipe-icons/Dairy.png',
    selectedIndex: 1,
    key:1,
  },
  {
    name: 'Vegetables',
    avatar_url: 'https://s3.amazonaws.com/recipe-icons/Vegetables.png',
    selectedIndex: 2,
    key:2,

  },
  {
    name: 'Fruits',
    avatar_url: 'https://s3.amazonaws.com/recipe-icons/Fruits.png',
    selectedIndex: 3,
    key:3,

  },
  {
    name: 'Baking & Grains',
    avatar_url: 'https://s3.amazonaws.com/recipe-icons/Baking%20&%20Grains.png',
    selectedIndex: 4,
    key:4,
  },
  {
    name: 'Condiments',
    avatar_url: 'https://s3.amazonaws.com/recipe-icons/Spices.png',
    selectedIndex: 5,
    key:5,

  },
  {
    name: 'Meats',
    avatar_url: 'https://s3.amazonaws.com/recipe-icons/Meats.png',
    selectedIndex: 6,
    key:6,
  },
  {
    name: 'Seafoods',
    avatar_url: 'https://s3.amazonaws.com/recipe-icons/Seafood.png',
    selectedIndex: 7,
    key:7,
  },
  {
    name: 'Liquids',
    avatar_url: 'https://s3.amazonaws.com/recipe-icons/Alcohol.png',
    selectedIndex: 8,
    key:8,

  },
  {
    name: 'Nuts',
    avatar_url: 'https://s3.amazonaws.com/recipe-icons/Nuts.png',
    selectedIndex: 9,
    key:9,

  }
]

export default class IngredientList extends React.Component {

  add (value) {
        this.props.change(value);
  }
  del (value) {
        this.props.remove(value);
  }

  componentWillUnmount() {
    this.backHandler.remove();
  }
  handleUpdateIndex (selected, names) {
    this.setState({selectedIndex : selected})
    this.setState({name : names})
  }

  revertUpdateIndex (selectedIndex) {
    this.setState({selectedIndex : 0})
    this.setState({name : 'Add Ingredients'})
  }

  stopUpdateIndex (category) {
    if (category === 'Baking & Grains'){
      category = 'Baking'
    }
    if (category === 'Condiments'){
      category = 'Sweeteners'
    }

    this.props.navigation.navigate(category,
      {
        adds: this.add.bind(this),
        dels: this.del.bind(this)
      })
  }

  constructor() {
    super();
    this.state = {
      selectedIndex: 0,
      name: 'Add Ingredients'
    }
    this.handleUpdateIndex = this.handleUpdateIndex.bind(this)
    this.revertUpdateIndex = this.revertUpdateIndex.bind(this)
  }

  render () {

    const { selectedIndex } = this.state
    return(
      <View containerStyle = {{backgroundColor: '#FFF'}}>
      <ScrollView containerStyle = {{backgroundColor: '#FFF'}}>
        <View>

        <ScrollView>
        <List containerStyle = {{
          flex: 1,
          marginTop: 0,
          borderTopWidth: 0,
          }}>
          {list.map((l) => (
            <TouchableOpacity>
                  <ListItem containerStyle = {{ height: 80, paddingTop: 15, borderTopWidth:0,
borderBottomWidth: 0.25}}
              key = {l.key}
              avatar={
                <View>
                  <ImageBackground
                    style={{width: 60, height: 60}}
                    source={{uri: l.avatar_url}}
                  />
                </View>
              }
              title={l.name}
              onPress={() => this.stopUpdateIndex(l.name)}
            />
            </TouchableOpacity>
          ))
        }
        </List>
        </ScrollView>
        </View>
      </ScrollView>
      </View>
    )
  }
}
```

<u>LoadScreen →</u>

<u>Loadscreen.js</u>

```
import React from 'react'
import { View, Text, ActivityIndicator, StyleSheet, Dimensions } from 'react-native'

export default class Loading extends React.Component {
  render() {
    return (
      <View style={styles.container}>
        <ActivityIndicator size="large" />
      </View>
    )
  }
}

const styles = StyleSheet.create({
  container: {
    paddingTop: Dimensions.get('window').height * 0.2,
    justifyContent: 'center',
    alignItems: 'center',
  }
})
```

<u>Search →</u>

<u>Search.js</u>

```
import React from 'react'
import Dimensions from 'Dimensions';
import { Searchbar } from 'react-native-paper';
import {
    Text,
    View,
    ScrollView,
    StyleSheet,
    StatusBar,
    Image,
    ImageBackground,
    BackHandler,
    AsyncStorage,
    TouchableOpacity,
    ActivityIndicator
} from 'react-native';
```

```
import {
  List,
  ListItem,
  Icon,
  CheckBox
} from 'react-native-elements';

const DEVICE_WIDTH = Dimensions.get('window').width;
const DEVICE_HEIGHT = Dimensions.get('window').height;
const screen = Dimensions.get("window");

let _this = null;

export default class Search extends React.Component {

  constructor(props){
    super(props);
    this.state = {
      completelist: [
          'Baking Soda', 'Baking Powder', 'Batter', 'Blueberries', 'Bread', 'Chocolate', 'Cocoa',
'Dough', 'Flour', 'Noodles', 'Pasta', 'Rice', 'Vanilla', 'Yeast', 'Avocado', 'Basil', 'Beans', 'Broccoli',
'Cabbage', 'Carrot', 'Cauliflower', 'Celery', 'Corn', 'Cucumber', 'Eggplant', 'Garlic', 'Ginger',
'Lettuce', 'Mint', 'Mushroom', 'Olive', 'Onion', 'Oregano', 'Pickle', 'Potato', 'Pumpkin', 'Seed',
'Shallot', 'Spinach', 'Sweet Potato', 'Thyme', 'Tomato', 'Zucchini','Butter', 'Butter Milk', 'Cheese',
'Custard', 'Egg', 'Ice Cream', 'Milk', 'Sour Cream', 'Yoghurt', 'Apple', 'Banana', 'Blackberries',
'Cherries', 'Coconut', 'Cranberries', 'Grapes', 'Kiwi', 'Lemon', 'Lime', 'Mango', 'Orange', 'Peach',
'Pear', 'Pineapple', 'Raspberries', 'Strawberries', 'Beef Soup', 'Beer', 'Chicken Soup', 'Lamb Soup',
'Olive Oil', 'Red Wine', 'Vegetable Oil', 'White Wine', 'Barbeque Sauce', 'Syrup', 'Fish Sauce',
'Honey', 'Mayonnaise', 'Mustard', 'Soy Sauce', 'Vinegar', 'Carp', 'Catfish', 'Crab', 'Eel', 'Lobster',
'Mackerel', 'Mussel', 'Oyster', 'Prawn', 'Salmon', 'Sardine', 'Scallop', 'Shrimp', 'Squid', 'Trout',
'Tuna', 'Almond', 'Cashew', 'Macadamia', 'Peanut', 'Peanut Butter', 'Walnut', 'Bacon', 'Beef',
'Chicken', 'Ham', 'Lamb', 'Pork', 'Salami', 'Sausage', 'Turkey', 'Watermelon', 'Paprika', 'Apple
Juice', 'Orange Juice', 'Coriander', 'Pecan', 'Allspice', 'Nutmeg', 'Tomato Sauce'
      ],
      list: [],
      checked:[],
      included:false,
    }
    this.handleFilterUpdate = this.handleFilterUpdate.bind(this)
  }

  componentDidMount() {
      _this = this;
   }

  checkItem = checkbox => {
     const { checked } = this.state;
     if (!checked.includes(checkbox)) {
      this.setState({ checked: checked.concat([checkbox]) });
      var number = checkbox.toString();
      this.props.navigation.state.params.adds(number);
    } else {
      this.setState({ checked: checked.filter(a => a !== checkbox) });
      var number = checkbox.toString();
      this.props.navigation.state.params.dels(number);
    }
```

```
};

findstr(string){

  var sample = [];
  if (string != "" && string != " "){
    for(var i = 0; i < this.state.completelist.length; i++) {
      if (this.state.completelist[i].toLowerCase().includes(string.toLowerCase())){
        var word = this.state.completelist[i]
        sample.push(word)
      }
    }
  }
  this.setState({
      list: sample
  })
}

static navigationOptions = {
  headerTitle:
    <View style={{width: DEVICE_WIDTH - 80}}>
    <Searchbar
      placeholder="Search"
      onChangeText={(query) => _this.findstr(query)}
    />
    </View>,
  headerStyle: {height: 55, backgroundColor: '#075e54', marginTop: -20},
  headerTintColor: 'white',
}

handleFilterUpdate() {
}

render() {
  return (
    <View containerStyle = {{backgroundColor: '#FFF'}}>

     {this.state.list[0] === undefined ?
      <View style={styles.container}>
      <Text style = {{color: '#4c4c4c'}}>No Results Found</Text>
      </View>: null
     }

     {this.state.list[0] !== undefined ?
     <ScrollView containerStyle = {{backgroundColor: '#FFF'}}>
     <View>
     <List containerStyle = {{
       flex: 1,
       marginTop: 0,
       borderTopWidth: 0,
       }}>
      {this.state.list.map((l) => (
        <ListItem containerStyle = {{borderTopWidth:0, borderBottomWidth: 0.25}}

          avatar={
            <View style = {{flexDirection: 'row'}}>
              <Image
```

```
                style={{width: 45, height: 45}}
                source={{uri: 'https://i.imgur.com/0zfrvlw.png'}}/>

              <Text style = {{paddingTop:20, color: 'steelblue'}}>

              {'  '+l }
              </Text>
            </View>
          }
          title={
            <CheckBox
            containerStyle={{
              backgroundColor: 'transparent',
              borderWidth: 0,
            }}
            right
            checkedIcon='minus-circle'
            uncheckedIcon='plus-circle'
            iconRight
            onPress={
              () => {
                this.checkItem(l);
              }

            }
            checked={this.state.checked.includes(l)}

            />
          }
          hideChevron
        />
      ))
    }
    </List>
    </View>

    </ScrollView>:null
   }
    </View>
   )
  }
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    paddingTop: DEVICE_HEIGHT * 0.4,
    justifyContent: 'center',
    alignItems: 'center',
  }
})
```

Results →

Results.js

```
import React from 'react'
import Dimensions from 'Dimensions';
import { View, Text, StyleSheet, ScrollView, TouchableOpacity, Image, Linking, Switch } from
'react-native'
import {
  CheckBox,
  List,
  ListItem,
  Button,
  Avatar,
} from 'react-native-elements';
import data from './Recipes';

const DEVICE_WIDTH = Dimensions.get('window').width;
const DEVICE_HEIGHT = Dimensions.get('window').height;

export default class Results extends React.Component {

  constructor(props) {
    super(props)
    this.state = {
      recipes:[],
      newList: this.props.navigation.state.params.lists,
      bonusrecipes:[],
      toggle: false,
    }
    this.switchMore = this.switchMore.bind(this)
  }

  componentDidMount(props){

    this.props.navigation.setParams({
      toggle: this.state.toggle,
      switchMore: this.switchMore,
     });

    var obj = Object.values(data.recipes)
    var samples = []
    var moresamples = []
    for (var i = 0; i < obj.length; i++) {
        var matches = 0;
        for (var j = 0; j < this.state.newList.length; j++){
            if (this.state.newList[j].toLowerCase().replace(/ /g,'') in obj[i]){
                matches++;
            }
        }

        if ("vegetableoil" in obj[i] && !this.state.newList.includes("Vegetable Oil")){
          matches++;
        }

        if (matches === obj[i].s){
          samples.push(obj[i])
        }

        else if (matches === obj[i].s - 1 && obj[i].s !== 1 && (obj[i].s !== 2 && "vegetableoil" in
obj[i])){
          moresamples.push(obj[i])
        }
    }

    samples.sort((a, b) => (a.s) - (b.s)).reverse();
    moresamples.sort((a, b) => (a.s) - (b.s)).reverse();

    this.setState({
          recipes: samples,
          bonusrecipes: moresamples
    })
  }

  switchMore = () => {
    var current = this.state.toggle
    this.setState({
      toggle: !current
    })
    this.props.navigation.setParams({
      toggle: !this.state.toggle,
    })
  }

  static navigationOptions = ({ navigation }) => {
        const { params = {} } = navigation.state;
        return {
          headerTitle:
          <View style = {{flexDirection: 'row'}}>
            <View style = {{paddingTop: 4, paddingLeft: -3}}>
            </View>
            <View style = {{flexDirection: 'column'}}>
                <Text style={{paddingBottom:2, color: 'white', fontSize: 21, fontWeight: 'bold'}}>{"
Recipes"}</Text>
            </View>
          </View>,
          headerTintColor: 'white',
          headerStyle: {height: 55,backgroundColor: '#075e54', marginTop: -20},
          headerRight:
          <View style = {{flexDirection: 'row', paddingRight: DEVICE_WIDTH * 0.02 }}>
            <Switch value = {params.toggle} onValueChange = {() => params.switchMore()}/>
          </View>,
        };
    };

  render() {
    return (
      <View>

       {this.state.recipes[0] === undefined ?
         <View style={styles.othercontainer}>
         <Text style = {{color: '#4c4c4c'}}>No Results Found</Text>
         </View> : null
       }
    {this.state.recipes[0] !== undefined ?
    <ScrollView>


      {this.state.toggle === false ?
      <List containerStyle = {{
        flex: 1,
        marginTop: 0,
        borderTopWidth: 0,
        }}>
        {this.state.recipes.map((k) => (
         <TouchableOpacity>
              <ListItem containerStyle = {{ height: 80, paddingTop: 15, borderTopWidth:0,
borderBottomWidth: 0.25}}
          avatar={
            <Avatar
            medium
            rounded
            source={{uri: 'cloche.png'}}
            />
          }
          title = {
            k.b
          }
          subtitle = 'allrecipes.com'
          onPress={() => Linking.openURL("https://www.allrecipes.com/recipe/" + k.l)}
         />
         </TouchableOpacity>
        ))
      }
      </List> :
      <List containerStyle = {{
        flex: 1,
        marginTop: 0,
        borderTopWidth: 0,
        }}>
        {this.state.bonusrecipes.map((k) => (
         <TouchableOpacity>
              <ListItem containerStyle = {{ height: 80, paddingTop: 15, borderTopWidth:0,
borderBottomWidth: 0.25}}
          avatar={
            <Avatar
            medium
            rounded
            source={{uri: 'cloche.png'}}
            />
          }
          title = {
            k.b
          }
          subtitle = 'allrecipes.com'
          badge={{ value: 1 }}
          onPress={() => Linking.openURL("https://www.allrecipes.com/recipe/" + k.l)}
          hideChevron
         />
         </TouchableOpacity>
```

```
          ))
        }
        </List>
       }
      </ScrollView>:null
    }
      </View>
    )
  }
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
  },
  othercontainer: {
    flex: 1,
    paddingTop: DEVICE_HEIGHT * 0.4,
    justifyContent: 'center',
    alignItems: 'center',
  }
})
```

TabBar →

TabBar.js

```
import React from 'react';
import {
    Text,
    View
} from 'react-native';
import Icon from '@expo/vector-icons/FontAwesome';

import { createStackNavigator, createBottomTabNavigator } from 'react-navigation';
import MainMenu from '../MainMenu/MainMenu';
import LoadScreen from '../LoadScreen/LoadScreen';
import Dairy from '../Dairy/Dairy';
import IngredientList from '../IngredientList/IngredientList';
import Vegetables from '../Vegetables/Vegetables';
import CheckOut from '../CheckOut/CheckOut';
import Search from '../Search/Search';
import App from '../../../App';

const TabBar = createBottomTabNavigator({
  IngredientList: {
     screen: IngredientList,
     navigationOptions: () => ({
        tabBarLabel: 'Add',
        tabBarIcon: ({tintColor}) => (
           <Icon
              name="cart-plus"
              color={tintColor}
              size={24}
```

```
                />
            )
        })
    },
    CheckOut: {
        screen: CheckOut,
        navigationOptions: () => ({
            tabBarLabel: 'Ingredients',
            tabBarIcon: ({tintColor}) => (
                <Icon
                    name="shopping-basket"
                    color={tintColor}
                    size={24}
                />
            )
        })
    },
    Search: {
        screen: Search,
        navigationOptions: () => ({
            tabBarLabel: 'Search',
            tabBarIcon: ({tintColor}) => (
                <Icon
                    name="search"
                    color={tintColor}
                    size={24}
                />
            )
        })
    },
    LoadScreen: {
        screen: LoadScreen,
        navigationOptions: () => ({
            tabBarLabel: 'Bookmarks',
            tabBarIcon: ({tintColor}) => (
                <Icon
                    name="bookmark"
                    color={tintColor}
                    size={24}
                />
            )
        })
    },
    Vegetables: {
        screen: Vegetables,
        navigationOptions: () => ({
            tabBarLabel: 'Support',
            tabBarIcon: ({tintColor}) => (
                <Icon
                    name="heart"
                    color={tintColor}
                    size={24}
                />
            )
        })
    }
}, {
 tabBarOptions: {
     showLabel: true,
     activeTintColor: '#F8F8F8',
     inactiveTintColor: '#7181ae',
     style: {
        backgroundColor: '#171F33'
     },
     tabStyle: {
       //onPress={() => this.props.navigation.navigate('Dairy')}
     }
 }
});

export default TabBar
```

MainMenue →

MainMenu.js

```
import React from 'react';
import {
   View,
   StyleSheet,
   StatusBar
} from 'react-native';

import TabBar from '../TabBar/TabBar';
import HeaderBar from '../HeaderBar/HeaderBar';
import IngredientList from '../IngredientList/IngredientList';
import CheckOut from '../CheckOut/CheckOut';
import { Container, Header, Content, Footer, FooterTab, Button, Icon, Text } from 'native-base';
import { createStackNavigator, createBottomTabNavigator } from 'react-navigation';
import ScrollableTabView from 'react-native-scrollable-tab-view'


class MainMenu extends React.Component {

  static navigationOptions = {
       header: null,
    }

  constructor() {
    super()
    this.state = {
      isReady: false,
      list: [],

    }
    console.disableYellowBox = true;
    this.handleFilterUpdate = this.handleFilterUpdate.bind(this);
    this.removeFilterUpdate = this.removeFilterUpdate.bind(this);

  }

  handleFilterUpdate(filterValue) {
    var t = 0;
    for(var i = 0; i < this.state.list.length; i++) {
      if (this.state.list[i] === (filterValue)) {
        t = 1;
        break;
      }
    }

    if (t === 0){
      this.setState({
        list: this.state.list.concat(
          filterValue
        )
      })
    }
  }

removeFilterUpdate(filterValue) {
  for(var i = 0; i < this.state.list.length; i++) {
    if(this.state.list[i] === (filterValue)) {
      var array = this.state.list;
      array.splice(i, 1);
      this.setState({
        list: array
      })
    }
  }
}

componentWillMount() {

  Expo.Font.loadAsync({
  Roboto: require("native-base/Fonts/Roboto.ttf"),
  Roboto_medium: require("native-base/Fonts/Roboto_medium.ttf"),
  Ionicons: require("@expo/vector-icons/fonts/Ionicons.ttf")
  });
  this.setState({ isReady: true });

}

render () {
  if (!this.state.isReady) {
    return <Expo.AppLoading />;
    }
  const { navigation } = this.props;
  return (



    <View style={{
              flex: 1,
              flexDirection: 'column',
              justifyContent: 'space-between',
              backgroundColor: '#FFF'
             }}>
      <View style={{
              flex: 1,
          }}>
        <StatusBar
         backgroundColor="#06544b"
         barStyle="light-content"
         />
         <HeaderBar
           navigation = {navigation}
           change={this.handleFilterUpdate}
           remove={this.removeFilterUpdate}
         />

         <ScrollableTabView
              style={{
                flex: 1,
                flexDirection: 'column',
                justifyContent: 'space-between',
              }}
              initialPage={0}
              tabBarUnderlineStyle ={{
                backgroundColor: '#fff',
                height: 2,
              }}
              tabBarUnderlineColor="#fff"
              tabBarBackgroundColor ='#075e54'
              tabBarActiveTextColor="#fff"
              tabBarInactiveTextColor="#88b0ac"
              >
                 <IngredientList  navigation={navigation}  change={this.handleFilterUpdate}
remove={this.removeFilterUpdate} tabLabel="Add Ingredients" />
                           <CheckOut   navigation={navigation}   item={this.state.list}
remove={this.removeFilterUpdate} tabLabel="Ingredients List" />

         </ScrollableTabView>

         <View style={{}} />
        </View>
       </View>

     )
    }
   }

  export default MainMenu

const styles = StyleSheet.create({
  container: {
    flex: 1,
    //backgroundColor: '#fff',
  },
});
```

<u>HeaderBar →</u>

<u>HeaderBar.js</u>

```
import React from 'react';
import {
    Text,
    View,
    Dimensions,
    StyleSheet,
    AppRegistry,
    Image,
    TouchableOpacity,
} from 'react-native';
import { Font } from 'expo';
import Search from '../Search/Search';
import Icon from '@expo/vector-icons/FontAwesome';

import {
  Header
} from 'react-native-elements';

class HeaderBar extends React.Component {

    add (value) {
        this.props.change(value);
    }
    del (value) {
        this.props.remove(value);
    }

    state = {
      fontLoaded: false,
    };

    async componentDidMount() {
      await Font.loadAsync({
        'KlavikaBold': require('./KlavikaBold.ttf'),
      });
      this.setState({ fontLoaded: true });
    }

    render () {
      return (
        <View>
        {
          this.state.fontLoaded ? (
        <Header
          outerContainerStyles={{
            backgroundColor: '#075e54',
            borderBottomWidth: 0,
            height: 55
          }}

          placement = "left"

          leftComponent={
```

```
          <View style = {{flexDirection: 'row', paddingTop: 2}}>
          <Text style={{
            color: '#fff',
            fontSize: 24,
            fontFamily: 'KlavikaBold',
          }}> Super Cook </Text>
          </View>
        }
        rightComponent={
          <View style = {{flexDirection: 'row', paddingTop: 2, paddingRight: 20}}>
          <TouchableOpacity
            onPress={() => this.props.navigation.navigate('Search',
            {
              adds: this.add.bind(this),
              dels: this.del.bind(this)
            })
          }
          >
          <Icon
            name={"search"}
            size={24}
            type='entypo'
            color="#FFF"
             />
           </TouchableOpacity>
           </View>

        }
      />
    ) : null}
    </View>
    )
    }
  }

export default HeaderBar
```

<u>Light.js</u>

```
import Color from 'color';

import {Platform} from 'react-native';

export default {

  // Badge
  badgeBg: '#ED1727',
  badgeColor: '#fff',


  // Button
  btnFontFamily: (Platform.OS === 'ios' ) ? 'HelveticaNeue' : 'Roboto_medium',
  btnDisabledBg: '#b5b5b5',
  btnDisabledClr: '#f1f1f1',

  get btnPrimaryBg () {
     return this.brandPrimary;
  },
  get btnPrimaryColor () {
     return this.inverseTextColor;
  },
  get btnInfoBg () {
     return this.brandInfo;
  },
  get btnInfoColor () {
     return this.inverseTextColor;
  },
  get btnSuccessBg () {
     return this.brandSuccess;
  },
  get btnSuccessColor () {
     return this.inverseTextColor;
  },
  get btnDangerBg () {
     return this.brandDanger;
  },
  get btnDangerColor () {
     return this.inverseTextColor;
  },
  get btnWarningBg () {
     return this.brandWarning;
  },
  get btnWarningColor () {
     return this.inverseTextColor;
  },
  get btnTextSize () {
     return (Platform.OS==='ios') ? this.fontSizeBase* 1.1 :
     this.fontSizeBase-1;
  },
  get btnTextSizeLarge () {
     return this.fontSizeBase* 1.5;
  },
  get btnTextSizeSmall () {
     return this.fontSizeBase* .8;
  },
  get borderRadiusLarge () {
     return this.fontSizeBase* 3.8;
  },

  buttonPadding: 6,

  get iconSizeLarge () {
     return this.iconFontSize* 1.5;
  },
  get iconSizeSmall () {
     return this.iconFontSize* .6;
  },


  // card
  cardDefaultBg: '#fff',
  // checkbox
  checkboxBgColor: '#039BE5',
  checkboxSize: 23,
  checkboxTickColor: '#fff',


  // colors
  brandPrimary : '#5067FF',
  brandInfo: '#5bc0de',
  brandSuccess: '#5cb85c',
  brandDanger: '#d9534f',
  brandWarning: '#f0ad4e',
  brandSidebar: '#252932',


  // font
  fontFamily: (Platform.OS === 'ios' ) ? 'HelveticaNeue' : 'Roboto',
  fontSizeBase: 15,

  get fontSizeH1 () {
    return this.fontSizeBase*1.8;
  },
  get fontSizeH2 () {
    return this.fontSizeBase* 1.6;
  },
  get fontSizeH3 () {
    return this.fontSizeBase* 1.4;
  },


  // footer
  footerHeight: 55,
  footerDefaultBg: (Platform.OS === 'ios' ) ? '#F8F8F8' : '#4179F7',
  footerBtnMargin: 1,


  //footertab
  tabBarTextColor: (Platform.OS === 'ios' ) ? '#6b6b6b' : '#b3c7f9',
  tabBarActiveTextColor: (Platform.OS === 'ios' ) ? '#007aff' : '#fff',
  tabActiveBgColor: (Platform.OS=='ios') ? '#cde1f9' : undefined,
  tabActiveBorderRadius: (Platform.OS=='ios') ? 7 : undefined,
  tabBarTextSize:  (Platform.OS=='ios') ? 12.5 : 10,
  tabBarActiveTextSize:  (Platform.OS=='ios') ? 12.5 : 11,

  //tab
  tabDefaultBg: (Platform.OS === 'ios' ) ? '#F8F8F8' : '#4179F7',
  topTabBarTextColor: (Platform.OS === 'ios' ) ? '#6b6b6b' : '#b3c7f9',
  topTabBarActiveTextColor: (Platform.OS === 'ios' ) ? '#007aff' : '#fff',
  topTabActiveBgColor: (Platform.OS=='ios') ? '#cde1f9' : undefined,
  topTabBarBorderColor: (Platform.OS === 'ios' ) ? '#007aff' : '#fff',


  // header
  iosToolbarBtnColor: '#007aff',
  toolbarDefaultBg: (Platform.OS === 'ios' ) ? '#F8F8F8' : '#4179F7',
  toolbarHeight: (Platform.OS === 'ios' ) ? 64 : 56,
  toolbarIconSize: (Platform.OS === 'ios' ) ? 20 : 22,
  toolbarInputColor: '#CECDD2',
  toolbarInverseBg: '#222',
  toolbarTextColor: (Platform.OS==='ios') ? '#000' : '#fff',
  get statusBarColor() {
    return Color(this.toolbarDefaultBg).darken(0.2).hexString();
  },


  // icon
  iconFamily: 'Ionicons',
  iconFontSize: (Platform.OS === 'ios' ) ? 30 : 28,
  iconMargin: 7,


  // inputgroup
  inputFontSize: 15,
  inputBorderColor: '#D9D5DC',
  inputSuccessBorderColor: '#2b8339',
  inputErrorBorderColor: '#ed2f2f',

  get inputColor () {
    return this.textColor;
  },
  get inputColorPlaceholder () {
    return '#575757';
  },

  inputGroupMarginBottom: 10,
  inputHeightBase: 40,
  inputPaddingLeft: 5,

  get inputPaddingLeftIcon () {
    return this.inputPaddingLeft* 8;
  },


  // lineheight
  btnLineHeight: 19,
  lineHeightH1: 32,
  lineHeightH2: 27,
  lineHeightH3: 22,
  iconLineHeight: (Platform.OS === 'ios' ) ? 37 : 30,
  lineHeight: (Platform.OS === 'ios' ) ? 20 : 24,


  // list
  listBorderColor: '#ddd',
  listDividerBg: '#ddd',
  listItemHeight: 45,
  listItemPadding: (Platform.OS === 'ios' ) ? 12 : 16,
  listNoteColor: '#808080',
  listNoteSize: 13,
  listMarginLeft: 15,


  // progress
  defaultProgressColor: '#E4202D',
    inverseProgressColor: '#1A191B',


    // radiobtn
    radioBtnSize: (Platform.OS === 'ios') ? 25 : 23,
    radioColor: '#7e7e7e',

    get radioSelectedColor() {
      return Color(this.radioColor).darken(0.2).hexString();
    },


    // spinner
    defaultSpinnerColor: '#45D56E',
    inverseSpinnerColor: '#1A191B',


    // tabs
    tabBgColor: '#F8F8F8',
    tabFontSize: 15,
    tabTextColor: '#222222',


    // text
    textColor: '#000',
    inverseTextColor: '#fff',
    textBgColor: 'transparent',


    // title
    titleFontSize: (Platform.OS === 'ios' ) ? 17 : 19,
    subTitleFontSize: (Platform.OS === 'ios' ) ? 12 : 14,
    subtitleColor: '#8e8e93',


    // other
    borderRadiusBase: (Platform.OS === 'ios' ) ? 5 : 2,
    borderWidth: 1,
    contentPadding: 10,

    get darkenHeader() {
      return Color(this.tabBgColor).darken(0.03).hexString();
    },

    dropdownBg: '#000',
    dropdownLinkColor: '#414142',
    inputLineHeight: 24,
    jumbotronBg: '#C9C9CE',
    jumbotronPadding: 30
  }
```

CheckOut →

Checkout.js

```
import React from 'react'
import { View, Linking, TouchableOpacity, AsyncStorage, ScrollView, Image, ImageBackground, Text, BackHandler, ActivityIndicator, StyleSheet, Dimensions } from 'react-native'
import {
  Header,
  CheckBox,
  List,
  ListItem,
  Button,
  Avatar,
} from 'react-native-elements';
import Icon from '@expo/vector-icons/FontAwesome';
import Results from '../Results/Results';

export default class CheckOut extends React.Component {
  constructor() {
    super()
    this.state = {
      unavailable: [],
      list: [],
      selectedIndex: 0,
      recipelist: []
    }
    this.handleUpdateIndex = this.handleUpdateIndex.bind(this)
  }

  static navigationOptions = {
    header: null,
    };

  handleUpdateIndex (selected) {
    this.componentDidMount();
    this.setState({selectedIndex : selected})
  }

  componentDidMount() {
  }

  _onRefresh = () => {
    this.componentDidMount();
  }

  render() {

    const { selectedIndex } = this.state
    return(
      <View style={styles.container}>
     {this.state.selectedIndex === 1 ?
       <View>
       <ScrollView>
       <List containerStyle = {{
         flex: 1,
         marginTop: 0,
         borderTopWidth: 0,
         paddingBottom: 70
         }}>
         {this.state.recipelist.map((k) => (
          <TouchableOpacity>
                 <ListItem containerStyle = {{ height: 100, paddingTop: 15, borderTopWidth:0,
borderBottomWidth: 0.25}}
          avatar={

            <Avatar
            large
            rounded
            source={{uri: k.url}}
            />

          }
          title = {
            <Text style={{
              fontSize:18
            }}>
            {"  " + k.base}
            </Text>
          }
          subtitle = 'food.com'
          onPress={() => Linking.openURL('https://google.com')}

          />
          </TouchableOpacity>
         ))
       }
       </List>
       </ScrollView>
       </View>: null
      }

      {this.state.selectedIndex === 0 ?
      <View style = {{Color: '#FFF'}}>
      <ScrollView Style = {{flex: 1, backgroundColor: '#FFF'}}>

       <View Style = {{backgroundColor: '#FFF'}}>
       <ScrollView Style = {{backgroundColor: '#FFF'}}>
       <List containerStyle = {{
         flex: 1,
         marginTop: 0,
         borderTopWidth: 0,
         paddingBottom: 10,
         backgroundColor: '#FFF'
         }}>
         {this.props.item.map((l) => (
                 <ListItem containerStyle = {{ height: 50, paddingTop: 15, borderTopWidth:0,
borderBottomWidth: 0.25}}
          avatar={
            <View style = {{flexDirection: 'row', backgroundColor: '#FFF'}}>
            <TouchableOpacity
              onPress={() =>
                {
                  this.props.remove(l)
                }
              }
            >
            <Image
              style={{width: 30, height: 30}}
              source={{uri: 'https://i.imgur.com/9ukbPRu.png'}}/>
            </TouchableOpacity>
            <Text style = {{color: 'steelblue', paddingTop: 5}}>

            {'   '+l }
            </Text>

            </View>
          }
          hideChevron
          />
         ))
       }
       </List>
       </ScrollView>
       </View>
      </ScrollView>



      <TouchableOpacity
       style={{
         position: 'absolute',
         alignItems:'center',
         justifyContent:'center',
         width:55,
         height:55,
         backgroundColor:'rgba(30, 30, 30, 0.2)',
         borderRadius:100,
         alignSelf: 'flex-end',
         margin: 50,
         top: Dimensions.get('window').height * 0.595,
         left: Dimensions.get('window').width * 0.615,
         //position: 'absolute',
         //top: Dimensions.get('window').height*0.68,
         //left: Dimensions.get('window').width*0.66
       }}
       onPress={() => this.props.navigation.navigate('Results',
         {
          lists : this.props.item
         })}
      >
      <Icon
       size={30}
       type='entypo'
       color="#FFF"
        />
      </TouchableOpacity>
      <TouchableOpacity
       style={{
          position: 'absolute',
          alignItems:'center',
          justifyContent:'center',
          top: Dimensions.get('window').height * 0.59,
          left: Dimensions.get('window').width * 0.615,
          width:55,
          height:55,
          backgroundColor:"#075e54",
          borderRadius:100,
          alignSelf: 'flex-end',
          margin: 50,
          shadowColor:'#000000',
          shadowOpacity:0.7,
        }}
        onPress={() => this.props.navigation.navigate('Results',
          {
            lists : this.props.item
          })}

      >
        <Icon
          name={"arrow-right"}
          size={30}
          type='entypo'
          color="#FFF"
           />
      </TouchableOpacity>
      </View>:null
    }
      </View>

    )
  }
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#FFF',
    //justifyContent: 'center',
    //alignItems: 'center',
  }
})
```

<u>Vegetables →</u>

<u>Vegetables.js</u>

```
import React from 'react'
import { View, ScrollView, Image, ImageBackground, Text, ActivityIndicator, StyleSheet,
Dimensions } from 'react-native'
import {
  Header,
  CheckBox
} from 'react-native-elements';
import VegetablesList from './VegetablesList';
```

```
const CustomHeader = () => (
  <View style = {{flexDirection: 'row'}}>
    <View style = {{paddingTop: 4, paddingLeft: -3}}>
    </View>
    <View style = {{flexDirection: 'column'}}>
        <Text  style={{paddingBottom:2,  color:  'white',  fontSize:  21,  fontWeight:  'bold'}}>{"
Vegetables"}</Text>
    </View>
  </View>
);

export default class Vegetables extends React.Component {
  state = {
    checked: false,
  };

  change (value) {
    this.props.navigation.state.params.adds(value);
  }

  rem (value) {
    this.props.navigation.state.params.dels(value);
  }

  static navigationOptions = ({ navigation }) => ({
      headerTitle: <CustomHeader/> ,
      headerTintColor: 'white',
      headerStyle: {height: 55,backgroundColor: '#075e54', marginTop: -20},
  });

  render() {
    return (
      <View style={styles.container}>

      <VegetablesList changed = {this.change.bind(this)} deleted = {this.rem.bind(this)}/>
      </View>
    )
  }
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#FFF',
    //justifyContent: 'center',
    //alignItems: 'center',
  }
})
```

VegetablesList.js

```
import React from 'react';
import {
    Text,
    View,
    ScrollView,
    StyleSheet,
    StatusBar,
    Image,
    ImageBackground,
    BackHandler,
    AsyncStorage,
    TouchableOpacity,
} from 'react-native';

import {
  List,
  ListItem,
  Icon,
  CheckBox
} from 'react-native-elements';

import { createStackNavigator } from 'react-navigation';
import Dairy from '../Dairy/Dairy';
import Vegetables from '../Vegetables/Vegetables';
import HeaderBar from '../HeaderBar/HeaderBar';

const list = [
  {
    name: 'Avocado',
    avatar_url: 'https://i.imgur.com/ubwLE1E.png',
  },
  {
    name: 'Basil',
    avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
    name: 'Beans',
    avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
    name: 'Broccoli',
    avatar_url: 'https://i.imgur.com/sIxzkAH.png',
  },
  {
    name: 'Cabbage',
    avatar_url: 'https://i.imgur.com/PIgw93G.png',
  },
  {
    name: 'Carrot',
    avatar_url: 'https://i.imgur.com/iX6q5cx.png',
  },
  {
    name: 'Cauliflower',
    avatar_url: 'https://i.imgur.com/jepRRBh.png',
  },
  {
    name: 'Celery',
    avatar_url: 'https://i.imgur.com/cIc4BpR.png',
  },
  {
    name: 'Coriander',
    avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
    name: 'Corn',
    avatar_url: 'https://i.imgur.com/T1zgMRS.png',
  },
  {
    name: 'Cucumber',
    avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
    name: 'Eggplant',
    avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
    name: 'Garlic',
    avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
    name: 'Ginger',
    avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
    name: 'Lettuce',
    avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
    name: 'Mint',
    avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
    name: 'Mushroom',
    avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
    name: 'Olive',
    avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
    name: 'Onion',
    avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
    name: 'Oregano',
    avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
    name: 'Pickle',
    avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
    name: 'Potato',
    avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
    name: 'Pumpkin',
    avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
   name: 'Seeds',
   avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
   name: 'Shallot',
   avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
   name: 'Spinach',
   avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
   name: 'Sweet Potato',
   avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
   name: 'Thyme',
   avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
   name: 'Tomato',
   avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
   name: 'Zucchini',
   avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  }
]

export default class VegetablesList extends React.Component {

  constructor(props) {
   super(props);
   this.state = {
     checked : [],
     included: false,
   }
  }

  checkItem = checkbox => {

    const { checked } = this.state;
    if (!checked.includes(checkbox)) {
     this.setState({ checked: checked.concat([checkbox]) });
     var number = checkbox.toString();
     this.props.changed(number);
   } else {
     this.setState({ checked: checked.filter(a => a !== checkbox) });
     var number = checkbox.toString();
     this.props.deleted(number);
   }
  };

  render () {
   return(
      <View containerStyle = {{backgroundColor: '#FFF'}}>
      <ScrollView containerStyle = {{backgroundColor: '#FFF'}}>
      <View>
      <List containerStyle = {{
        flex: 1,
        marginTop: 0,
        borderTopWidth: 0,
        }}>
        {list.map((l) => (
          <ListItem containerStyle = {{borderTopWidth:0, borderBottomWidth: 0.25}}

            avatar={
              <View style = {{flexDirection: 'row'}}>
                <Image
                  style={{width: 45, height: 45}}
                  source={{uri: l.avatar_url}}/>

                <Text style = {{paddingTop:20, color: 'steelblue'}}>

                {'  '+l.name }
                </Text>
              </View>
            }
            title={
              <CheckBox
              containerStyle={{
                backgroundColor: 'transparent',
                borderWidth: 0,
              }}
              right
              checkedIcon='minus-circle'
              uncheckedIcon='plus-circle'
              iconRight
              onPress={
                () => {
                  this.checkItem(l.name);
                }
                //() => this.checkItem(l.checkbox)
              }
              checked={this.state.checked.includes(l.name)}

              />
            }
            hideChevron
          />
        ))
      }
      </List>
      </View>

      </ScrollView>
      </View>
    )
  }
}
```

Fruits →

Fruits.js

```
import React from 'react'
import { View, ScrollView, Image, ImageBackground, Text, ActivityIndicator, StyleSheet,
Dimensions } from 'react-native'
import {
  Header,
  CheckBox
} from 'react-native-elements';
import FruitsList from './FruitsList';

const CustomHeader = () => (
  <View style = {{flexDirection: 'row'}}>
    <View style = {{paddingTop: 4, paddingLeft: -3}}>
    </View>
    <View style = {{flexDirection: 'column'}}>
        <Text style={{paddingBottom:2, color: 'white', fontSize: 21, fontWeight: 'bold'}}>{"
Fruits"}</Text>
    </View>
  </View>
);

export default class Fruits extends React.Component {
  state = {
    checked: false,
  };

  change (value) {
    this.props.navigation.state.params.adds(value);
  }

  rem (value) {
    this.props.navigation.state.params.dels(value);
  }

  static navigationOptions = ({ navigation }) => ({
      headerTitle: <CustomHeader/> ,
      headerTintColor: 'white',
      headerStyle: {height: 55,backgroundColor: '#075e54', marginTop: -20},
  });

  render() {
    return (
      <View style={styles.container}>

      <FruitsList changed = {this.change.bind(this)} deleted = {this.rem.bind(this)}/>
      </View>
    )
  }
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#FFF',
    //justifyContent: 'center',
    //alignItems: 'center',
  }
})
```

Fruitslists.js

```
import React from 'react';
import {
    Text,
    View,
    ScrollView,
    StyleSheet,
    StatusBar,
    Image,
    ImageBackground,
    BackHandler,
    AsyncStorage,
    TouchableOpacity,
} from 'react-native';

import {
  List,
  ListItem,
  Icon,
  CheckBox
} from 'react-native-elements';

import { createStackNavigator } from 'react-navigation';
import Dairy from '../Dairy/Dairy';
import Fruits from '../Fruits/Fruits';
import Vegetables from '../Vegetables/Vegetables';
import HeaderBar from '../HeaderBar/HeaderBar';

const list = [
  {
    name: 'Apple',
    avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
    name: 'Banana',
    avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
    name: 'Blackberries',
    avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
    name: 'Blueberries',
    avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
    name: 'Cherries',
    avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
   name: 'Coconut',
   avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
   name: 'Cranberries',
   avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
   name: 'Grapes',
   avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
   name: 'Kiwi',
   avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
   name: 'Lemon',
   avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
   name: 'Lime',
   avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
   name: 'Mango',
   avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
   name: 'Orange',
   avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
   name: 'Peach',
   avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
   name: 'Pear',
   avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
   name: 'Pineapple',
   avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
   name: 'Raspberries',
   avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
   name: 'Strawberries',
   avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  },
  {
   name: 'Watermelon',
   avatar_url: 'https://i.imgur.com/0zfrvlw.png',
  }
]
```

```
export default class FruitsList extends React.Component {

 constructor(props) {
  super(props);
  this.state = {
    checked : [],
    included: false,
  }
 }

 checkItem = checkbox => {

   const { checked } = this.state;
   if (!checked.includes(checkbox)) {
    this.setState({ checked: checked.concat([checkbox]) });
    var number = checkbox.toString();
    this.props.changed(number);
  } else {
    this.setState({ checked: checked.filter(a => a !== checkbox) });
    var number = checkbox.toString();
    this.props.deleted(number);
  }
 };

 render () {
  return(
   <View containerStyle = {{backgroundColor: '#FFF'}}>
   <ScrollView containerStyle = {{backgroundColor: '#FFF'}}>
   <View>
   <List containerStyle = {{
    flex: 1,
    marginTop: 0,
    borderTopWidth: 0,
    }}>
    {list.map((l) => (
     <ListItem containerStyle = {{borderTopWidth:0, borderBottomWidth: 0.25}}

      avatar={
       <View style = {{flexDirection: 'row'}}>
        <Image
         style={{width: 45, height: 45}}
         source={{uri: l.avatar_url}}/>

        <Text style = {{paddingTop:20, color: 'steelblue'}}>

        {'  '+l.name }
        </Text>
       </View>
      }
      title={
       <CheckBox
       containerStyle={{
        backgroundColor: 'transparent',
        borderWidth: 0,
       }}
       right
```

```
          checkedIcon='minus-circle'
          uncheckedIcon='plus-circle'
          iconRight
          onPress={
            () => {
              this.checkItem(l.name);
            }
            //() => this.checkItem(l.checkbox)
          }
          checked={this.state.checked.includes(l.name)}

          />
        }
        hideChevron
      />
    ))
  }
  </List>
  </View>

  </ScrollView>
  </View>
 )
 }
}
```