

Project Report Format

1. INTRODUCTION

1.1 Project Overview

- A food tracking project would involve creating a system or tool for individuals to track their daily food and beverage intake. This could involve developing a mobile app or website that allows users to input their meals and snacks, as well as track specific nutrients or dietary restrictions.
- The project would begin with research into existing food tracking tools and methods, as well as user needs and preferences. This could involve conducting surveys or focus groups to gather feedback from potential users.
- Once the research is complete, the project team would begin developing the food tracking tool, which may involve working with developers, designers, and nutrition experts. The tool would need to be user-friendly and accessible, with features such as barcode scanning, meal planning, and data visualization.
- The final stage of the project would involve testing and refining the tool based on user feedback. This may involve conducting usability tests or surveys to gather feedback and make improvements.
- Overall, a food tracking project would aim to create a valuable tool for individuals looking to improve their eating habits and achieve their health

1.2 Purpose

The purpose of food tracking is to help individuals monitor and manage their food and beverage intake, with the goal of improving their overall health and well-being. By tracking what they eat and drink, individuals can identify patterns, make informed choices, and stay accountable to their goals. Food tracking can also be helpful for individuals with specific dietary needs or restrictions, such as those with allergies or intolerances. Overall, food tracking can be a powerful tool for promoting healthy eating habits and achieving optimal nutrition.

2. LITERATURE SURVEY

2.1 Existing problem

One existing problem with food tracking is that it can be time-consuming and tedious. Many individuals find it difficult to consistently track their meals and snacks throughout the day, especially if they have a busy schedule or are eating on the go. Additionally, some people may feel overwhelmed by the amount of information they need to input into their

tracking app or program, which can lead to frustration and discouragement. Another issue is that food tracking apps may not always provide accurate information about the nutritional content of certain foods, which can lead to incorrect tracking and potentially negative health outcomes. Finally, some individuals may become overly focused on tracking their food intake, which can lead to obsessive or disordered eating behaviors.

2.2 References

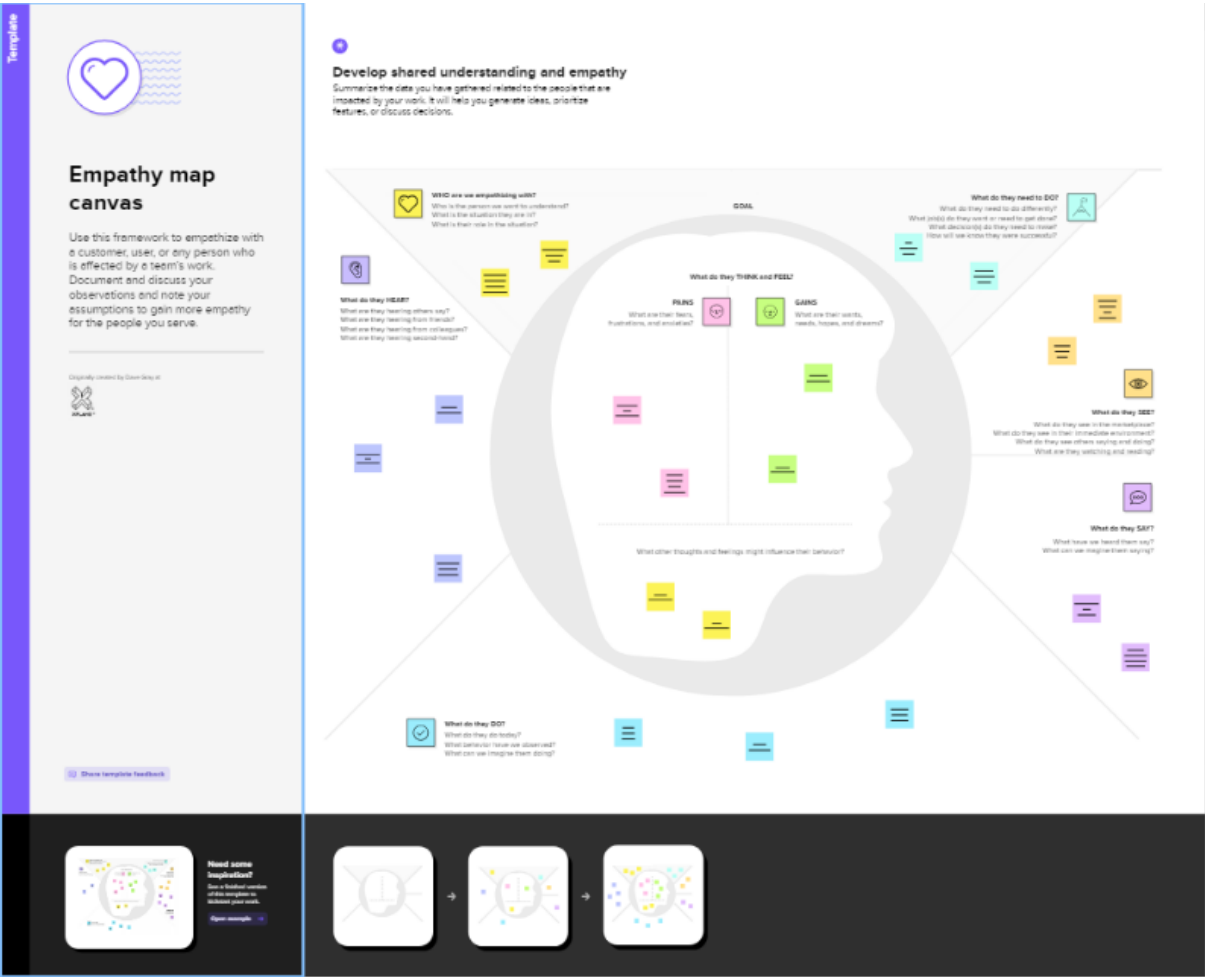
The article provides an overview of food tracking, including how it works and why it is worth doing. It discusses the benefits of keeping track of what you eat, such as increased awareness of your eating habits and the ability to make healthier choices. The article also offers tips for effective food tracking, such as using a food diary or mobile app, setting realistic goals, and seeking support from others. Overall, the article emphasizes the importance of food tracking as a tool for weight loss and improved health.

2.3 Problem Statement Definition

3. IDEATION & PROPOSED SOLUTION

- **Integration of food tracking with wearable technology:** With the increasing popularity of wearable devices such as smartwatches and fitness trackers, it would be beneficial to integrate food tracking with these devices. This would allow users to track their food intake seamlessly and effortlessly.
- **Gamification of food tracking:** Gamification can be used to make food tracking more engaging and motivating. For example, users could earn points for logging their meals, and these points could be redeemed for rewards such as discounts on healthy food or fitness classes.
- **Personalized meal plans:** Food tracking data can be used to create personalized meal plans for users. By analyzing their food intake, dietary preferences, and health goals, an AI-powered system could generate meal plans that are tailored to their specific needs.

3.1 Empathy Map Canvas



3.2 Ideation & Brainstorming

Brainstorm & idea prioritization

Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.

- 🕒 10 minutes to prepare
- 🕒 1 hour to collaborate
- 👥 2-8 people recommended

🕒 10 minutes

- A Team gathering**
Define who should participate in the session and send an invite. Share relevant information or pre-work ahead.
- B Set the goal**
Think about the problem you'll be focusing on solving in the brainstorming session.
- C Learn how to use the facilitation tools**
Use the Facilitation Superpowers to run a happy and productive session.

[Open article](#) →

🕒 5 minutes



Key rules of brainstorming

To run a smooth and productive session

- 🗣️ Stay in topic. 💡 Encourage wild ideas.
- 🕒 Defer judgment. 👂 Listen to others.
- 🗣️ Go for volume. 👁️ If possible, be visual.



Need some inspiration?
See a finished version of this template to kickstart your work.

[Open example](#) →

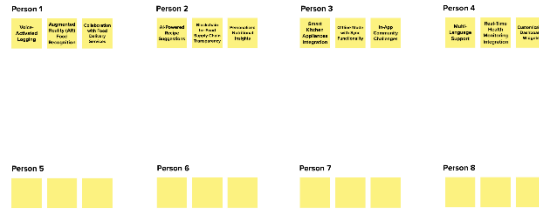
2

Brainstorm

Write down any ideas that come to mind that address your problem statement.

🕒 10 minutes

TIP
You can select a sticky note and, with the pencil (switch to select) icon to start drawing.



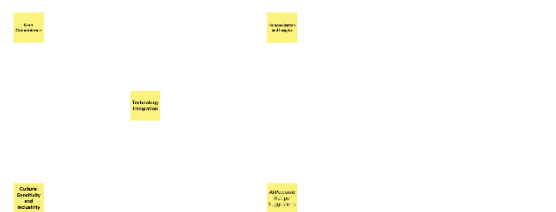
3

Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you can break it up into smaller sub-groups.

🕒 20 minutes

TIP
Just concentrate on sticky notes to make a group to find a theme, category, and category cluster ideas as themes or in your mind.



4

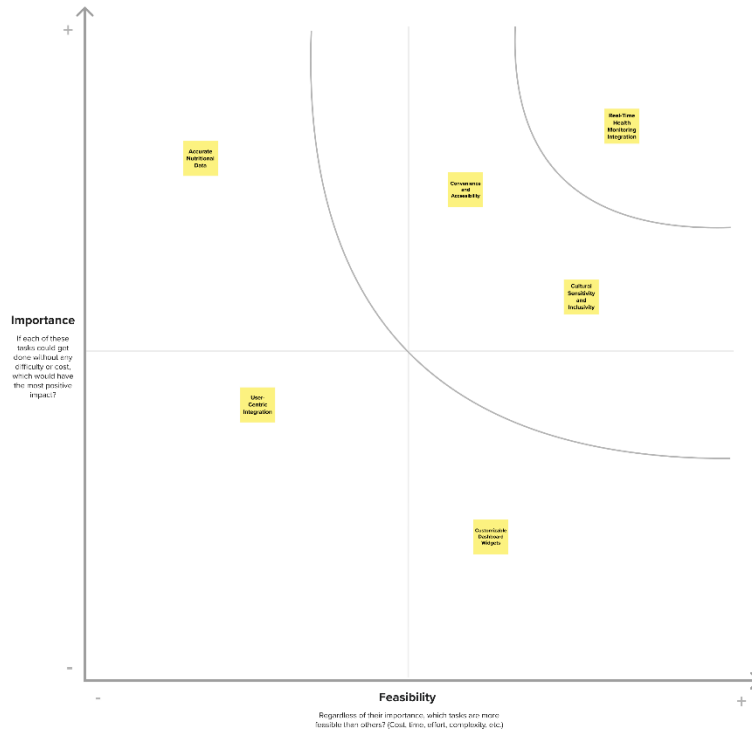
Prioritize

Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

⌚ 20 minutes

TIP

Participants can use their cursor to point at where sticky notes should go on the grid. The facilitator can confirm the spot by using the laser pointer holding the H key on the keyboard.



→

After you collaborate

You can export the mural as an image or pdf to share with members of your company who might find it helpful.

Quick add-ons

- A Share the mural**
Share a view link to the mural with stakeholders to keep them in the loop about the outcomes of the session.
- B Export the mural**
Export a copy of the mural as a PNG or PDF to attach to emails, include in slides, or save in your drive.

Keep moving forward

- Strategy blueprint**
Define the components of a new idea or strategy.
[Open the template →](#)
- Customer experience journey map**
Understand customer needs, motivations, and obstacles for an experience.
[Open the template →](#)
- Strengths, weaknesses, opportunities & threats**
Identify strengths, weaknesses, opportunities, and threats (SWOT) to develop a plan.
[Open the template →](#)

[Share template feedback](#)

4. REQUIREMENT ANALYSIS

4.1 Functional requirement

1. User Registration and Profile Management:

- Allow users to create accounts and profiles.
- Store user information securely.
- Enable users to update their profiles and preferences.

2. Food Database:

- Maintain a comprehensive database of food items, including nutritional information.
- Allow users to search for specific foods or scan barcodes for quick input.

3. Goal setting:

- Allow users to set personal health goals, such as calorie intake, weight loss, or nutrient targets.
- Provide recommendations based on the user's goals and

progress.

4. Data Security and Privacy:

- Ensure data encryption and user privacy in compliance with relevant regulations (such as GDPR).
- Allow users to control their data sharing preferences.

5. Support and Help:

- Provide a help section with FAQs.
- Offer customer support channels (email, chat) for user assistance.

4.2 Non-Functional requirements

1. Performance:

- The application should respond to user interactions within a maximum of 2 seconds.
- The database should handle a minimum of 1000 concurrent users without significant performance degradation.

2. Scalability:

- The system should be scalable to accommodate a growing user base without compromising performance.
- It should handle at least a 20% increase in users .

3. Reliability:

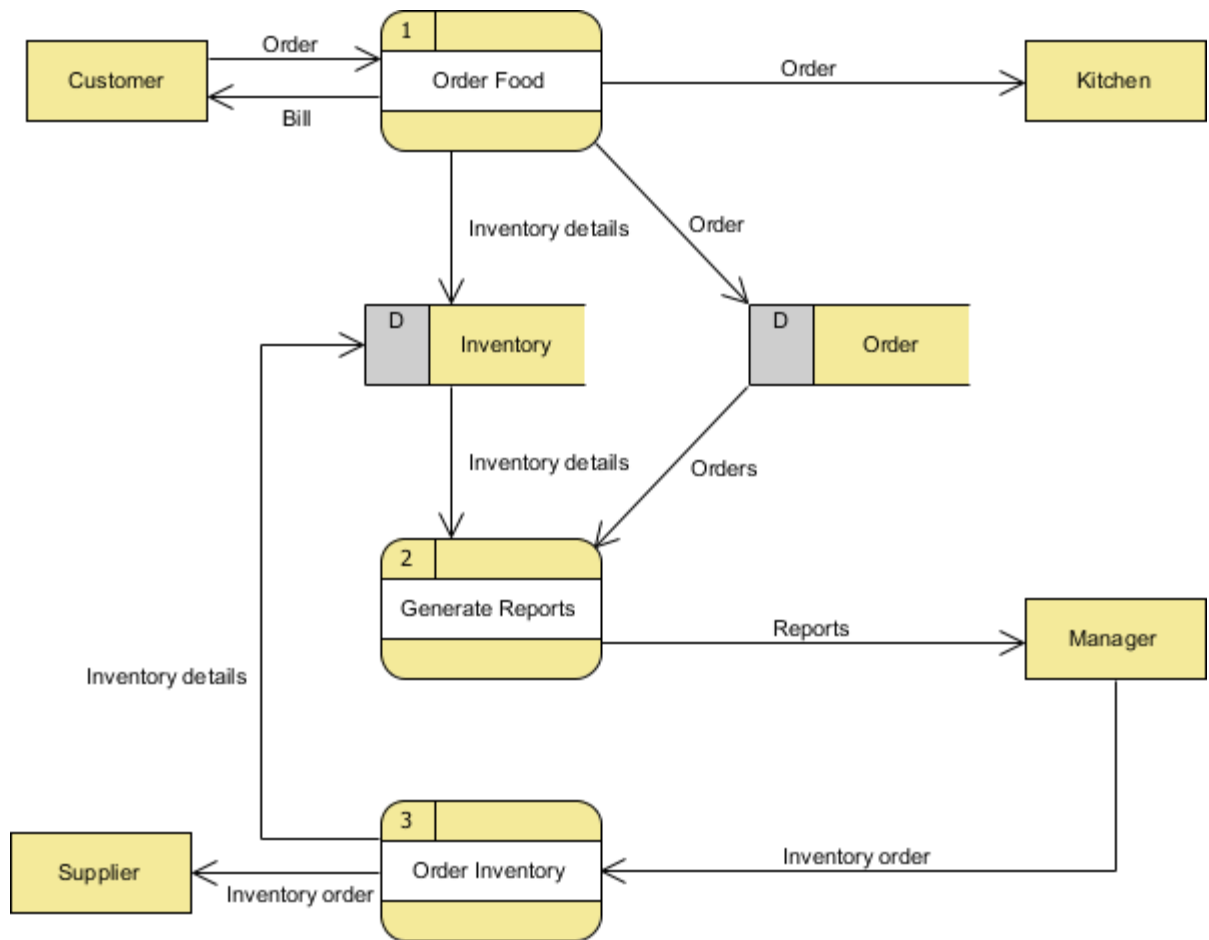
- The application should have a minimum uptime of 99.9%.
- Implement regular backups to prevent data loss in case of system failures.

4. Comprehensive Error Handling:

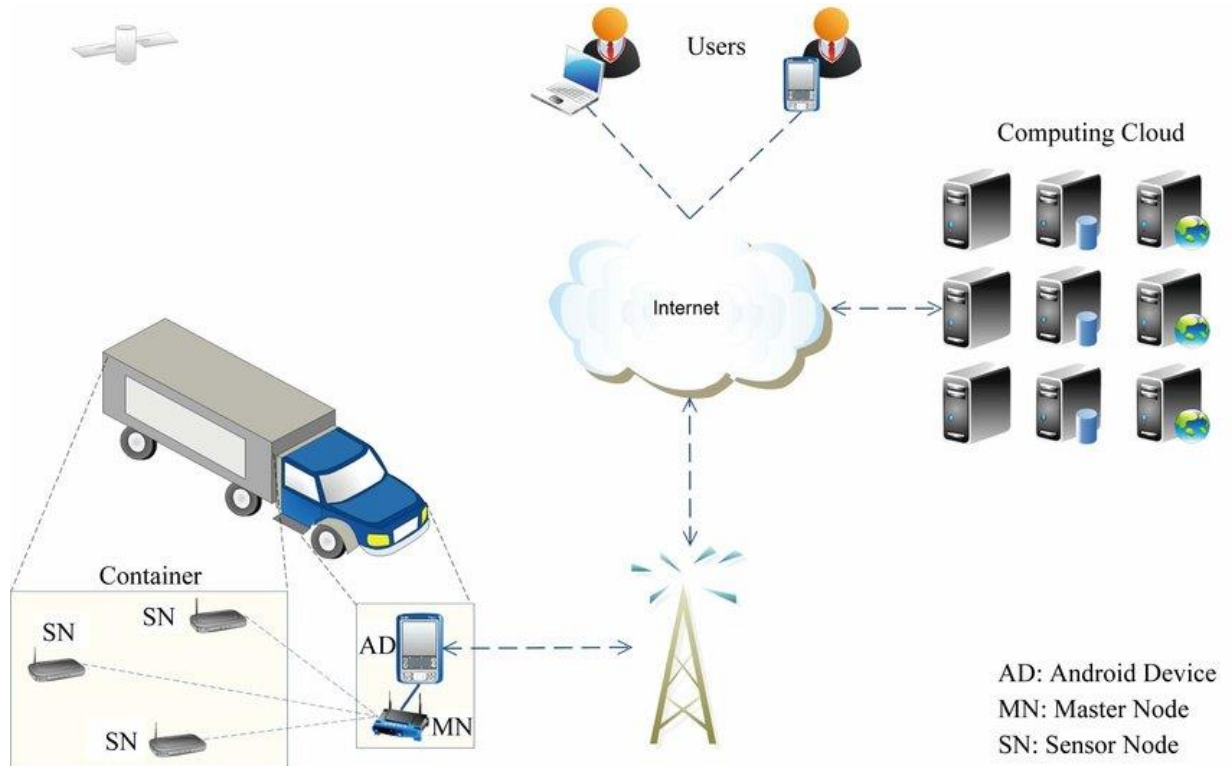
- Implement informative error messages to guide users in case of input errors or system failures.

5. PROJECT DESIGN

5.1 Data Flow Diagrams & User Stories

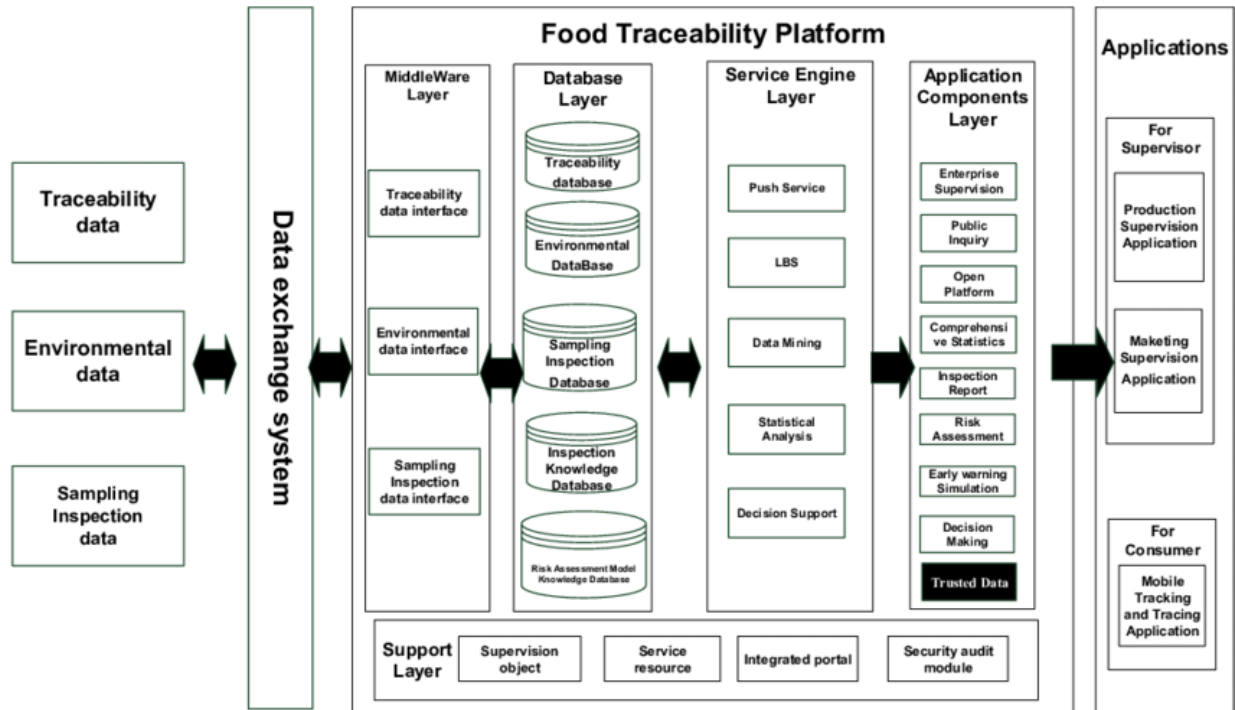


5.2 Solution Architecture:



6. PROJECT PLANNING & SCHEDULING

6.1 Technical Architecture



6.2 Sprint Planning & Estimation

- For the first sprint, we will focus on developing the basic features of the food tracking app, including the ability to log meals, track calories and macronutrients, and view nutritional information. We will also work on integrating a barcode scanner and incorporating visual aids to help users make healthier food choices.
- For the second sprint, we will focus on developing more advanced features, such as meal planning tools, food journaling, and integration with grocery delivery services. We will also work on incorporating social proof and incentives to motivate users to achieve their health goals.
- The first sprint is estimated to take approximately 2-3 weeks, with a team of 4 developers working full-time. The second sprint is estimated to take approximately 4-6 weeks, with additional resources allocated for design and testing. Overall, we estimate the entire project to take approximately 2-3 months to complete.

7. CODING & SOLUTIONING (Explain the features added in the project along with code)

1. Set up a virtual environment:

Create a virtual environment

```
python -m venv venv
```

```
# Activate the virtual environment
```

```
# On Windows
```

```
venv\Scripts\activate
```

```
# On macOS/Linux
```

```
source venv/bin/activate
```

2. Install Flask:

```
pip install flask
```

3. Create the project structure:

```
food_tracking_system/
```

```
|-- app/
```

```
|   |-- static/
```

```
|   |   |-- style.css
```

```
|   |-- templates/
```

```
|   |   |-- index.html
```

```
|   |-- __init__.py
```

```
|   |-- routes.py
```

```
|-- venv/
```

```
    |-- run.py
```

4. Create __init__.py in the app folder:

```
from flask import Flask app = Flask(__name__) from app import routes
```

5. Create routes.py in the app folder:

```
from app import app from flask import render_template @app.route('/')  
def index(): return render_template('index.html')
```

6. Create index.html in the templates folder:

```
<!DOCTYPE html> <html lang="en"> <head> <meta charset="UTF-  
8"> <meta name="viewport" content="width=device-width, initial-  
scale=1.0"> <link rel="stylesheet" href="{ { url_for('static',  
filename='style.css') } }"> <title>Food Tracking System</title>  
</head> <body> <h1>Food Tracking System</h1> <!-- Add your
```

HTML content for food tracking here --> </body> </html>

7. Create style.css in the static folder:

/* Add your CSS styles here */

8. Create run.py in the project root folder:

```
from app import app if __name__ == '__main__': app.run(debug=True)
```

9. Run the application:

```
python run.py
```

8. PERFORMANCE TESTING



project Development Phase



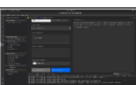
Model Performance Test

| | |
|----------------------|----------------------|
| Date | 30 October |
| Team ID | NX/2023TMD01707 |
| Project Name Project | FOOD TRACKING SYSTEM |
| Maximum Marks | 10 Marks |

Model Performance Testing:

Project team shall fill the following information when working for blockchain.

| S.No. | Parameter | Values | Screenshot |
|-------|-----------------------|-----------------------------|---|
| 1. | Information gathering | Setup all the Prerequisite: |  |
| 2. | Extract the zip files | Open to vs code |  |

| | | | |
|----|------------------------|--|---|
| 3. | Remix ide platform | <p>exploring</p> <p>Deploy the smart contract code</p> <p>Deploy and run the transaction. By selecting the environment - inject the MetaMask.</p> |  |
| 4. | Open file explorer | <p>Open the extracted file and click on the folder.</p> <p>Open src, and search for utils.</p> <p>Open cmd enter commands</p> <ol style="list-style-type: none"> 1. npm install 2. npm bootstrap 3. npm start |  |
| 5. | {LOCALHOST ADDRESS} IP | <p>copy the address and open it to chrome so you can see the front end of your project.</p> |  |

9. RESULTS

9.1 Output Screenshots

```
File Edit Selection View Go Run Terminal Help
Search

Welcome FoodTracking.sol

C:\Users\myrc> mypc > AppData > Local > Temp > Rar$Dia7272.24688 > FoodTracking.sol

1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.0;
3
4 contract FoodTracking {
5     address public owner;
6
7     enum FoodStatus {
8         Unverified,
9         Verified,
10        Consumed
11    }
12
13    struct FoodItem {
14        string itemId;
15        string productName;
16        string origin;
17        uint256 sentTimestamp;
18        FoodStatus status;
19    }
20
21    mapping(string => FoodItem) public foodItems;
22
23    event FoodItemSent(
24        string indexed itemId,
25        string productName,
26        string origin,
27        uint256 sentTimestamp
28    );
29    event FoodItemVerified(string indexed itemId);
30    event FoodItemConsumed(string indexed itemId);
31
32    constructor() {
33        owner = msg.sender;
34    }
35
36    modifier onlyOwner() {
37        require(msg.sender == owner, "Only contract owner can call this");
38    }
39}
```

3) WhatsApp My Drive - Google Drive Remix - Ethereum IDE

remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.18+commit.87f61d96.js

DEPLOY & RUN TRANSACTIONS

ENVIRONMENT
Remix VM (Shanghai)

ACCOUNT
0x5B3...eddC4 [99.999999%]

GAS LIMIT
3000000

VALUE
0 Wei

CONTRACT
FoodTracking - food tracking.sol

evm version paris

Deploy

Publish to IPFS

At Address Load contract from Address

Transactions recorded 1

Deployed Contracts

Balance: 0 ETH

consumeFoodItem string itemId

sendFoodItem string itemId, string product

```
83 }
84
85 function consumeFoodItem( infinite gas
86     string memory itemId
87 ) external onlyUnconsumed(itemId) {
88     foodItems[itemId].status = FoodStatus.Consumed;
89
90     emit FoodItemConsumed(itemId);
91
92
93 function getFoodItemDetails( infinite gas
94     string memory itemId
95 )
96     external
97     view
98     returns (string memory, string memory, uint256, FoodStatus)
99 {
100     FoodItem memory item = foodItems[itemId];
101     return (item.productName, item.origin, item.sentTimestamp, item.status);
102 }
103
104
```

listen on all transactions Search with transaction hash or address

[vm] from: 0x5B3...eddC4 to: FoodTracking.(constructor) value: 0 wei data: 0x698...28033 logs: 0 hash: 0x93e...80f4

status true Transaction mined and execution succeed

transaction hash 0x93e1579513ee057a8cfac228921659babb4189124c417c5d51bba8b6f4

block hash 0x96446b7e18063bc137742b3b51a7979fbdcc381fc805e526ef66aeb67ef95

block number 1

contract address 0xd9145CC520386f254917e4818B44e9943f39118

from 0x5B380da7781C568545dcfC803F8875F50ad8C4

DEPLOY & RUN TRANSACTIONS

ENVIRONMENT: Remix VM (Shanghai)

ACCOUNT: 0x5B3...eddC4 (99.999999%)

GAS LIMIT: 3000000

VALUE: 0 Wei

CONTRACT: FoodTracking - food_tracking.sol

Deploy

Published to IPFS

Transactions recorded: 1

Deployed Contracts: FOODTRACKING AT 0xD91...391

Balance: 0 ETH

consumeFood... string itemId

sendFoodItem... string itemId, string product

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.0;
3
4 contract FoodTracking {
5     address public owner;
6
7     enum FoodStatus {
8         Unverified,
9         Verified,
10        Consumed
11    }
12
13    struct FoodItem {
14        string itemId;
15        string productName;
16        string origin;
17        uint256 sentTimestamp;
18        FoodStatus status;
19    }
20
21    mapping(string => FoodItem) public foodItems;
22
23    event FoodItemSent(
24        string indexed itemId,
25        string productName,
26        string origin,
27        uint256 sentTimestamp
28    );
29    event FoodItemVerified(string indexed itemId);
30    event FoodItemConsumed(string indexed itemId);
31
32    constructor() {
33        owner = msg.sender;
34    }
35
36    modifier onlyOwner() {
37        require(msg.sender == owner, "Only contract owner can call this");
38    }
39}
```

Transaction View - cc_5 - Visual Studio Code

Manual input Transaction data directory

Transaction name: TransferToDeliverer

Transaction arguments: { "arg0": "Onion1" }

Transient data (optional):

Target specific peer (optional): 1 x Select peers

Evaluate transaction Submit transaction

Returned value from ReadAsset: {"Amount": "5kg", "AssetStatus": "Delivering", "ID": "Onion1", "Owner": "Deliverer1", "SalePrice": "20TL", "TimeStamp": "2021-08-13 21:17:53", "docType": "asset"}

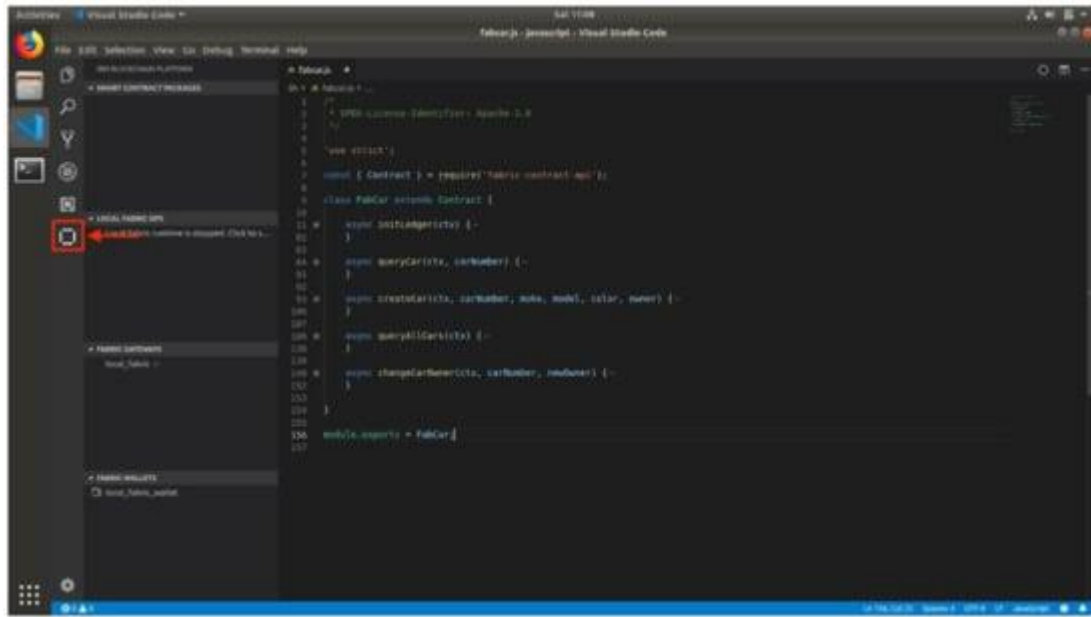
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

[8/14/2021 12:17:54 PM] [SUCCESS] Returned value from TransferToDeliverer: {"type": "Buffer", "data": []}

[8/14/2021 12:18:00 AM] [INFO] submitTransaction

[8/14/2021 12:18:00 AM] [INFO] submitting transaction ReadAsset with args Onion1 on channel mychannel

[8/14/2021 12:18:01 AM] [SUCCESS] Returned value from ReadAsset: {"Amount": "5kg", "AssetStatus": "Delivering", "ID": "Onion1", "Owner": "Deliverer1", "SalePrice": "20TL", "TimeStamp": "2021-08-13 21:17:53", "docType": "asset"} Successfully submitted transaction



10.ADVANTAGES

11.1 ADVANTAGES:

- **Awareness:** Tracking your food intake can help you become more aware of what you're eating and how much you're consuming. This can help you make healthier choices and avoid overeating.
- **Accountability:** By tracking your food intake, you hold yourself accountable for what you eat. This can help you stay on track with your dietary goals and make adjustments as needed.
- **Personalization:** Tracking your food intake allows you to personalize your diet to meet your individual needs and goals. You can track your macros, calories, and other nutrients to ensure you're getting the right balance of nutrients for your body.
- **Motivation:** Seeing progress over time can be motivating and help you stay committed to your dietary goals. Tracking your food intake allows you to see how far you've come and how much progress you've made.

12.CONCLUSION

- While food tracking can have its drawbacks, the use of blockchain technology can potentially address some of these issues. By securely storing and tracking food data on a decentralized ledger, individuals can have a more streamlined and accurate way of monitoring their food intake. This can also help to reduce the risk of fraud or tampering in the food industry, promoting greater transparency and accountability. However, it is important to remember that technology alone cannot solve all issues related to food tracking and healthy eating. It is still up to individuals to approach their diet in a balanced

and sustainable way, with the support of healthcare professionals and trusted sources of information.

- In conclusion, blockchain technology has the potential to revolutionize food tracking and promote greater transparency and accountability in the food industry. However, it is important to remember that technology alone cannot solve all issues related to food tracking and healthy eating. It is still up to individuals to approach their diet in a balanced and sustainable way, with the support of healthcare professionals and trusted sources of information.

13.FUTURE SCOPE

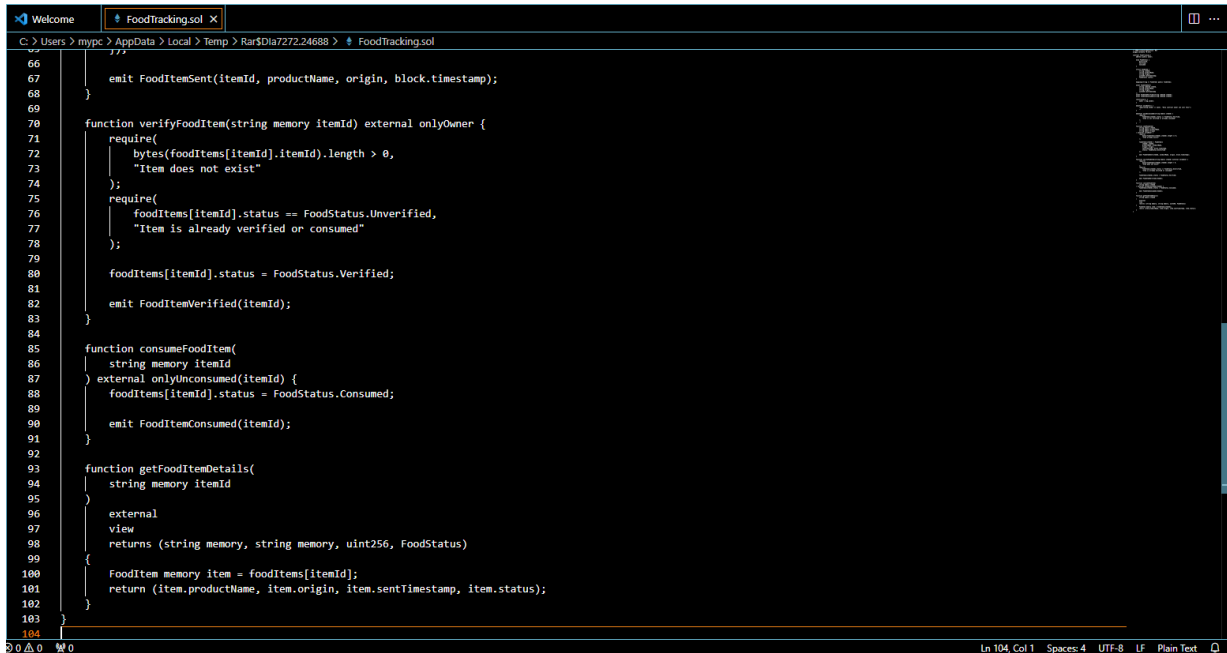
- **Improved supply chain management:** Blockchain technology can help track the entire journey of food from farm to fork, ensuring that it is safe, fresh, and of high quality.
- **Better food safety:** By using blockchain to track food products, it is possible to quickly identify and respond to outbreaks of foodborne illnesses, reducing the risk of widespread contamination.
- **Increased transparency:** Consumers can have access to information about the origin, production, and distribution of their food, allowing them to make informed decisions about what they eat.
- **Enhanced sustainability:** Blockchain can be used to track the environmental impact of food production and distribution, enabling consumers to choose products that are produced in a sustainable and ethical manner.
- **Personalized nutrition:** By using blockchain to track individual food intake, it is possible to create personalized nutrition plans that are tailored to an individual's specific needs and goals.

14.APPENDIX

Source Code

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.0;
3
4 contract FoodTracking {
5     address public owner;
6
7     enum FoodStatus {
8         Unverified,
9         Verified,
10        Consumed
11    }
12
13    struct FoodItem {
14        string itemId;
15        string productName;
16        string origin;
17        uint256 sentTimestamp;
18        FoodStatus status;
19    }
20
21    mapping(string => FoodItem) public foodItems;
22
23    event FoodItemSent(
24        string indexed itemId,
25        string productName,
26        string origin,
27        uint256 sentTimestamp
28    );
29    event FoodItemVerified(string indexed itemId);
30    event FoodItemConsumed(string indexed itemId);
31
32    constructor() {
33        owner = msg.sender;
34    }
35
36    modifier onlyOwner() {
37        require(msg.sender == owner, "Only contract owner can call this");
38        _;
39    }
```

```
28    };
29    event FoodItemVerified(string indexed itemId);
30    event FoodItemConsumed(string indexed itemId);
31
32    constructor() {
33        owner = msg.sender;
34    }
35
36    modifier onlyOwner() {
37        require(msg.sender == owner, "Only contract owner can call this");
38        _;
39    }
40
41    modifier onlyUnconsumed(string memory itemId) {
42        require(
43            foodItems[itemId].status == FoodStatus.Verified,
44            "Item is not verified or already consumed"
45        );
46        _;
47    }
48
49    function sendFoodItem(
50        string memory itemId,
51        string memory productName,
52        string memory origin
53    ) external onlyOwner {
54        require(
55            bytes(foodItems[itemId].itemId).length == 0,
56            "Item already exists"
57        );
58
59        foodItems[itemId] = FoodItem({
60            itemId: itemId,
61            productName: productName,
62            origin: origin,
63            sentTimestamp: block.timestamp,
64            status: FoodStatus.Unverified
65        });
66    }
```

```
66     emit FoodItemSent(itemId, productName, origin, block.timestamp);
67 }
68
69
70 function verifyFoodItem(string memory itemId) external onlyOwner {
71     require(
72         bytes(foodItems[itemId].itemId).length > 0,
73         "Item does not exist"
74     );
75     require(
76         foodItems[itemId].status == FoodStatus.Unverified,
77         "Item is already verified or consumed"
78     );
79     foodItems[itemId].status = FoodStatus.Verified;
80
81     emit FoodItemVerified(itemId);
82 }
83
84
85 function consumeFoodItem(
86     string memory itemId
87 ) external onlyUnconsumed(itemId) {
88     foodItems[itemId].status = FoodStatus.Consumed;
89
90     emit FoodItemConsumed(itemId);
91 }
92
93 function getFoodItemDetails(
94     string memory itemId
95 )
96     external
97     view
98     returns (string memory, string memory, uint256, FoodStatus)
99 {
100     FoodItem memory item = foodItems[itemId];
101     return (item.productName, item.origin, item.sentTimestamp, item.status);
102 }
103
104 }
```

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract FoodTracking {
 address public owner;

enum FoodStatus {
 Unverified,
 Verified,
 Consumed
}

struct FoodItem {
 string itemId;
 string productName;
 string origin;
 uint256 sentTimestamp;
 FoodStatus status;
}

```
mapping(string => FoodItem) public foodItems;
```

```
event FoodItemSent(  
    string indexed itemId,  
    string productName,  
    string origin,  
    uint256 sentTimestamp  
);
```

```
event FoodItemVerified(string indexed itemId);
```

```
event FoodItemConsumed(string indexed itemId);
```

```
constructor() {  
    owner = msg.sender;  
}
```

```
modifier onlyOwner() {  
    require(msg.sender == owner, "Only contract owner can call this");  
    _;  
}
```

```
modifier onlyUnconsumed(string memory itemId) {  
    require(  
        foodItems[itemId].status == FoodStatus.Verified,  
        "Item is not verified or already consumed"  
    );  
    _;  
}
```

```
function sendFoodItem(  
    string memory itemId,  
    string memory productName,
```

```

    string memory origin
) external onlyOwner {
    require(
        bytes(foodItems[itemId].itemId).length == 0,
        "Item already exists"
    );

    foodItems[itemId] = FoodItem({
        itemId: itemId,
        productName: productName,
        origin: origin,
        sentTimestamp: block.timestamp,
        status: FoodStatus.Unverified
    });

    emit FoodItemSent(itemId, productName, origin, block.timestamp);
}

function verifyFoodItem(string memory itemId) external onlyOwner {
    require(
        bytes(foodItems[itemId].itemId).length > 0,
        "Item does not exist"
    );
    require(
        foodItems[itemId].status == FoodStatus.Unverified,
        "Item is already verified or consumed"
    );

    foodItems[itemId].status = FoodStatus.Verified;

    emit FoodItemVerified(itemId);
}

```

```
}
```

```
function consumeFoodItem(  
    string memory itemId  
) external onlyUnconsumed(itemId) {  
    foodItems[itemId].status = FoodStatus.Consumed;  
  
    emit FoodItemConsumed(itemId);  
}
```

```
function getFoodItemDetails(  
    string memory itemId  
)  
    external  
    view  
    returns (string memory, string memory, uint256, FoodStatus)  
{  
    FoodItem memory item = foodItems[itemId];  
    return (item.productName, item.origin, item.sentTimestamp, item.status);  
}  
}
```

GitHub & Project Demo Link