—---

Phase 3 Backend Framework: Go to file Project Structure.doc

- Use a backend framework like Flask (Python), Express (Node.js), or Ruby on Rails (Ruby) to create API endpoints for your application.
- ● Javascript (NodeJS)
- ○ NodeJS is an open source server
- environment that will allow the use of

—---

Below are the main components and considerations for building this platform:

1. Project Structure:

   - Start by setting up the project folder structure. For a Node.js application, you can use a structure like this:

Markdown code:

```
- /project-root
  - /controllers
  - /models
  - /routes
  - /config
  - /middlewares
  - /public (for static files)
  - /views (if you're using server-side rendering)
  - app.js (main application file)
  - package.json (for npm packages)
```

2. Backend Framework (Node.js):

   - Use Node.js to create the backend of your application. Set up routes and controllers to handle various API endpoints.

3. Express.js (Web Framework):

   - If you're using Node.js, consider using Express.js, a popular web framework that simplifies routing, middleware handling, and API development.

Basic CRUD Operations(CREATE, READ, UPDATE and DELETE**)**
4. Authentication and Authorization:

- Implement user authentication and authorization mechanisms, especially if users can create accounts or interact with sensitive data.

Basic CRUD Operations

5. API Endpoints:

   - Define API endpoints for various functionalities:
     - User registration and login (if required)
     - Retrieving food sources and supplier information
     - Marking food sources as favorites for registered users
     - Managing user sessions
     - Implementing policies acceptance functionality

6. Database Interaction:

   - Create database models (using an ORM like Sequelize or writing raw SQL queries) to interact with the MySQL database as defined in your schema. **Basic CRUD Operations**

7. Middleware:

   - Use middleware for tasks like handling CORS (Cross-Origin Resource Sharing), parsing request bodies, and adding authentication checks.

8. Error Handling:

   - Implement error handling middleware to gracefully handle errors and provide meaningful responses to clients.

9. Security:

   - Ensure that your application follows best practices for security, including input validation, password hashing, and protection against common vulnerabilities like SQL injection and XSS (Cross-Site Scripting).

10. Frontend Development (Optional):

    - Create a user-friendly frontend using HTML, CSS, and JavaScript. Consider using a frontend framework like React or Vue.js if you want a dynamic and responsive interface.

11. Testing:

- Write unit tests and integration tests for your application to ensure its reliability and functionality.

12. Deployment:

    - Deploy your application to a web server or cloud platform of your choice (e.g., AWS, Heroku, or DigitalOcean).

13. Documentation:

    - Provide clear and comprehensive documentation for your API endpoints, database schema, and any other relevant aspects of your project.

14. Continuous Development:

    - Plan for continuous development and maintenance of your platform, including updates, bug fixes, and feature enhancements.

Remember to follow best practices, modularize your code, and maintain a structured and organized development approach throughout the project. Additionally, make use of npm to manage packages and dependencies effectively.