Project Report: Adopt-A-Tree Backend

1. Introduction:
The Adopt-A-Tree backend is a modular and distributed system designed to manage users, trees, and produce production. The system is comprised of three main services: UserService, TreeService, and ProduceService, each serving specific functionalities. Additionally, there is a utility module, Utils, providing common functions, and an API Gateway (ApiGateWay.js) serving as a centralized entry point for communication between services.

2. Services:

2.1 UserService:

- Responsible for user registration, login, and management.
- Utilizes bcrypt for password hashing and jwt for authentication.
- Supports user details retrieval and listing all users.
- Listens on port 3001.

2.2 TreeService:

- Manages tree-related operations, including tree addition, deletion, and retrieval.
- Implements tree adoption and information calculation functionalities.
- Requires authentication using a middleware.
- Listens on port 3002.

2.3 ProduceService:

- Handles produce production, sharing, and information calculation.
- Integrates with TreeService and UserService for data retrieval.
- Supports product creation and sharing calculation.
- Listens on port 3003.

2.4 Utils:

- Contains utility functions shared between ProduceService and TreeService.
- Functions include fetching user trees and calculating the time since adoption.

2.5 API Gateway (ApiGateWay.js):

- Acts as a centralized entry point for external communication.
- Implements authentication using a middleware.
- Forwards requests to the appropriate service based on the path.

3. Communication:

- Services communicate with each other through HTTP requests.
- The API Gateway handles authentication and routing of requests to the respective services.
- Axios is used for making HTTP requests between services.

4. Technologies Used:

- Node.js: For server-side JavaScript runtime.
- Express.js: For building web applications and APIs.

- Sequelize: As an ORM for interacting with the SQLite database.
- Axios: For making HTTP requests between services.
- bcrypt: For password hashing.
- jwt: For authentication and token generation.

5. Deployment:

- Each service is deployed independently, and their instances can be scaled horizontally.
- Services are deployed on different ports to avoid conflicts.
- Docker or a similar containerization tool can be used for ease of deployment and management.

6. Future Enhancements:

- Implement containerization for better scalability and maintainability.
- Enhance security measures, such as using HTTPS for communication.
- Introduce logging and monitoring tools for better insights into system behavior.
- Consider implementing a message queue for asynchronous communication between services.

7. Product Share Calculation Algorithm:

The algorithm for calculating product shares in the ProduceService involves a multi-step process to fairly assess a user's contribution based on the number of trees owned and the time since adoption. This algorithm is implemented in the `/production/calculate-share` endpoint. The key steps are summarized as follows:

1. **Fetch User Data:**
    - Retrieve all user information from the `/user/find/all` endpoint.
2. **Retrieve Share Information:**
    - Fetch all share records for each user from the database.
3. **Calculate Product Shares:**
    - Iterate through each user's share records.
    - For each product shared, calculate:
        - The maximum and minimum number of trees owned by any user (`maxNumberOfTrees` and `minNumberOfTrees`).
        - The maximum and minimum relationship time (time since adoption) for the product (`maxRelationshipTime` and `minRelationshipTime`).
4. **Calculate Scores:**
    - For each user's share of a specific product, compute two scores:
        - `numberOfTreesScore`: The ratio of the user's number of trees to the maximum number of trees owned by any user.
        - `timeSinceAdoptionScore`: The ratio of the user's relationship time to the maximum relationship time for that product.
5. **Composite Score:**
    - Combine the two scores using a weighted average:

        - `compositeScore = 0.6 * numberOfTreesScore + 0.4 * timeSinceAdoptionScore`.

6. **Aggregation:**
    - Aggregate the calculated scores for each user based on two factors (i.e total no of trees by the user for the produce (weight = 60%), time since adoption (weight = 40%)) into a composite score which could be used to compute the percentage for fair-share distribution.

This algorithm provides a balanced assessment of a user's share, taking into account both the quantity of trees owned and the duration of the relationship with those trees. The weighted average allows for flexibility in adjusting the importance of each factor, ensuring a fair representation of user contributions to the shared product.

8. Conclusion:
The Adopt-A-Tree backend is a well-structured and modular system that efficiently manages users, trees, and produce-related operations. The use of microservices allows for independent development and deployment of each service, promoting scalability and maintainability. The system's modular nature makes it adaptable to future enhancements and improvements.

This project report provides an overview of the system's architecture, services, technologies used, and recommendations for future development. It serves as a comprehensive guide for understanding the design and functionality of the Adopt-A-Tree backend.