Food Delivery App Backend Documentation

1. Introduction:

This backend server is designed to handle dynamic pricing for a food delivery app. It provides an API to calculate delivery costs for different types of food items across various zones based on the distance and item type.

2. Technologies Used:

- Node.js
- Express.js
- PostgreSQL
- Sequelize ORM
- Swagger

3**. Code Structure:**

- models/: Contains Sequelize models for Organization, Item, and Pricing.
- services/: Contains PriceCalculator service object to calculate delivery prices.
- routes/: Contains API routes for handling price calculations.
- db.js: Configures the connection to the PostgreSQL database.
- server.js: Express server setup and initialization.

4. **Models:**

- Organization:
  - Fields: id (auto-generated), name
- Item:
  - Fields: id (auto-generated), type, description
- Pricing:
  - Fields: organization_id, item_id, zone, base_distance_in_km, km_price, fix_price

5. **API Endpoints:**

- POST /api/price/calculate-price
  - Description: Calculate the total price for delivery of specified food items in a given zone for a particular organization.
  - Request Payload:

```
{
  "zone": "central",
  "organization_id": "005",
  "total_distance": 12,
```

"item_type": "perishable"
}
- Response:
{
"total_price": 20.5
}
- And other CRUD endpoint with dummy values

6**. Service Object:**
- PriceCalculator:
- Provides a method to calculate the total price for delivery based on pricing details and input parameters.

7. **Setup Instructions:**
- Clone the repository.
- Install dependencies using npm install.
- Set up PostgreSQL database and configure connection in db.js.
- Run the server using npm start.

8. **Error Handling:**
- Proper error handling is implemented for database queries and API requests.
- Errors are returned with appropriate HTTP status codes and error messages.

9**. Validation:**
- Input payload validation is performed to ensure required fields are provided.
- Sequelize model validations are implemented for data integrity.

10. **Security:**
- No sensitive data is exposed through API endpoints.
- SQL injection is prevented by using Sequelize ORM and parameterized queries.

11. **Future Improvements:**
- Implement authentication and authorization for accessing API endpoints.
- Add logging and monitoring for better debugging and performance tracking.
- Improve input validation and error handling for robustness.
- Implement pagination for handling large datasets.