Expanding this application into a full-stack solution involves several steps:

1. Front-End Development
- **Framework**: Continue using Dash for the front-end. You can also consider integrating React for more complex UI components.
- **Styling**: Use CSS frameworks like Bootstrap or Tailwind CSS to enhance the visual appeal.

2. Back-End Development
- **Framework**: Use Flask or Django (Python) for the back-end. These frameworks will help you manage routes, handle requests, and serve data to the front-end.
- **API Development**: Create RESTful APIs to handle CRUD operations. This will allow your front-end to communicate with the back-end.

3. Database Integration
- **Database**: Choose a database like PostgreSQL, MySQL, or MongoDB to store your data.
- **ORM**: Use an Object-Relational Mapping (ORM) tool like SQLAlchemy (for SQL databases) or PyMongo (for MongoDB) to interact with your database.

4. User Authentication
- **Authentication**: Implement user authentication using libraries like Flask-Login or Django's built-in authentication system.
- **Authorization**: Manage user roles and permissions to control access to different parts of the application.

5. Deployment
- **Server**: Deploy your application on a cloud platform like AWS, Heroku, or Azure.
- **CI/CD**: Set up Continuous Integration and Continuous Deployment (CI/CD) pipelines using tools like GitHub Actions or Jenkins to automate testing and deployment.

6. Testing
- **Unit Testing**: Write unit tests for your back-end and front-end components.
- **Integration Testing**: Ensure that different parts of your application work together as expected.
- **End-to-End Testing**: Use tools like Selenium or Cypress to test the entire application flow.

Example Implementation

Integrating a back-end using Flask and a PostgreSQL database:

Back-End (Flask)
1. **Install Flask and SQLAlchemy**:
2. pip install Flask SQLAlchemy psycopg2-binary
3. **Create a Flask App**:

```
from flask import Flask, jsonify, request
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] =
'postgresql://username:password@localhost/dbname'
db = SQLAlchemy(app)
```

```
class Automobile(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    year = db.Column(db.Integer)
    sales = db.Column(db.Integer)

@app.route('/automobiles', methods=['GET'])
def get_automobiles():
    automobiles = Automobile.query.all()
    return jsonify([auto.to_dict() for auto in automobiles])

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

AI-generated code. Review and use carefully. More info on FAQ.

4. **Create and Migrate the Database**:
5. flask db init
6. flask db migrate -m "Initial migration."

flask db upgrade

Front-End (Dash)
Modify Dash to Fetch Data from Flask API:

```
import dash
from dash import dcc, html
import requests

app = dash.Dash(__name__)

def fetch_data():
    response = requests.get('http://localhost:5000/automobiles')
    return response.json()

app.layout = html.Div([
    html.H1("Automobile Sales Dashboard"),
    dcc.Graph(
        id='sales-graph',
        figure={
            'data': [{'x': [d['year'] for d in fetch_data()], 'y': [d['sales'] for d in fetch_data()], 'type':
'line'}],
            'layout': {'title': 'Automobile Sales'}
        }
    )
])

if __name__ == '__main__':
    app.run_server(debug=True)
```