Here's a **phased approach** for developing your web application that allows users to select a date range and generate an Excel report of orders without guest counts.

Phased Approach for Commerce7 Guest Count Check Web App

Phase 1: Project Setup

✅ Goal: Establish a structured development environment.

- Create a **GitHub repository** for the project.
- Set up a **Node.js + Express** backend.
- Set up a **React.js frontend** (using Vite or Create React App).
- Configure a **`.env` file** to securely store Commerce7 API credentials.
- Install required packages:
- `npm install express axios dotenv cors body-parser xlsx`
- `npm install react react-dom react-router-dom @mui/material @mui/icons-material`

Phase 2: Backend Development

✅ Goal: Develop an API that fetches orders from Commerce7 and filters out those missing guest counts.

🔹 2.1 - API Authentication

- Implement **secure authentication** for Commerce7 API using `.env` variables.

```
require("dotenv").config();
const authConfig = {
  auth: {
    username: process.env.C7_APP_ID,
    password: process.env.C7_API_KEY,
  },
  headers: {
    Tenant: process.env.C7_TENANT_ID,
    "Content-Type": "application/json",
  },
};
```

🔹 2.2 - Fetch Orders from Commerce7

- Develop an **API route** (`/orders`) that retrieves orders within a given date range.

```
app.get("/orders", async (req, res) => {
  const { startDate, endDate } = req.query;
  try {
    const response = await axios.get(
```

```
      `https://api.commerce7.com/v1/order?startDate=${startDate}&endDate=${endDate}`,
      authConfig
    );

    const orders = response.data.orders;
    const filteredOrders = orders.filter(order => !order.guestCount);

    res.json(filteredOrders);
  } catch (error) {
    res.status(500).json({ message: "Failed to fetch orders", error: error.message });
  }
});
```

🔹 2.3 - Generate Excel File

- Implement an **endpoint (`/export`)** that generates an **Excel file** with the orders missing guest counts.

```
const XLSX = require("xlsx");

app.get("/export", async (req, res) => {
  const { startDate, endDate } = req.query;
  try {
    const response = await axios.get(
      `https://api.commerce7.com/v1/order?startDate=${startDate}&endDate=${endDate}`,
      authConfig
    );

    const orders = response.data.orders.filter(order => !order.guestCount);
    const worksheet = XLSX.utils.json_to_sheet(orders);
    const workbook = XLSX.utils.book_new();
    XLSX.utils.book_append_sheet(workbook, worksheet, "Missing Guest Counts");

    const buffer = XLSX.write(workbook, { type: "buffer", bookType: "xlsx" });

    res.setHeader("Content-Disposition", "attachment;
filename=missing_guest_counts.xlsx");
    res.setHeader("Content-Type", "application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet");
    res.send(buffer);
  } catch (error) {
    res.status(500).json({ message: "Failed to export orders", error: error.message
});
  }
});
```

Phase 3: Frontend Development

✅ Goal: Build a simple React frontend for selecting a date range and running the report.

🔹 3.1 - UI Design

- Use **Material UI** for a clean interface.
- Implement:
  - **Date pickers** for selecting `startDate` and `endDate`
  - A **"Run Report" button**
  - A **"Download Excel" button**

🔹 3.2 - Fetch Orders from Backend

- Create a **React component** that fetches orders from the backend.

```
import { useState } from "react";
import axios from "axios";
import { Button, TextField } from "@mui/material";

function OrderReport() {
  const [startDate, setStartDate] = useState("");
  const [endDate, setEndDate] = useState("");
  const [orders, setOrders] = useState([]);

  const fetchOrders = async () => {
    const response = await
axios.get(`/orders?startDate=${startDate}&endDate=${endDate}`);
    setOrders(response.data);
  };

  const downloadReport = async () => {
    window.location.href = `/export?startDate=${startDate}&endDate=${endDate}`;
  };

  return (
    <div>
      <h2>Guest Count Missing Report</h2>
      <TextField type="date" label="Start Date" onChange={(e) =>
setStartDate(e.target.value)} />
      <TextField type="date" label="End Date" onChange={(e) =>
setEndDate(e.target.value)} />
      <Button variant="contained" onClick={fetchOrders}>Run Report</Button>
      <Button variant="contained" color="primary" onClick={downloadReport}>Download
Excel</Button>

      <ul>
        {orders.map(order => (
          <li key={order.id}>{order.salesAssociate} - Order #{order.id}</li>
        ))}
      </ul>
    </div>
  );
}

export default OrderReport;
```

Phase 4: Deployment

✅ Goal: Deploy the app to Kinsta and make it accessible.

🔹 4.1 - Backend Deployment

- **Use GitHub Actions** to deploy the backend to **Kinsta**.
- Set up **environment variables** in Kinsta.
- Use **PM2 or Docker** to keep the backend running.

🔹 4.2 - Frontend Deployment

- Build the frontend using:
- `npm run build`
- Deploy the frontend to Kinsta using **Nginx** or **a static hosting service**.

Phase 5: Enhancements & Security

✅ Goal: Improve security and performance.

🔹 5.1 - Secure API Requests

- Use **JWT authentication** or an **API key** to restrict backend access.
- Move **Commerce7 API calls** to a **server-side proxy** (never expose API keys in frontend).

🔹 5.2 - Improve UX

- Add a **loading indicator** while fetching orders.
- Use **pagination** for large datasets.

🔹 5.3 - Logging & Monitoring

- Use **Winston** or **Datadog** for **logging errors**.
- Monitor **API usage** to avoid hitting Commerce7 rate limits.

Next Steps

Once the basic app is functional, we can:

- Integrate **email notifications** when reports are generated.
- Store **past reports** in a **database** for historical tracking.
- Enhance **error handling** with better user feedback.
