Next.js Chalet Booking Application Requirements

GOAL: Create a single-file, highly functional, and responsive Next.js/React application for collaborative chalet night booking, focusing on custom non-linear pricing and availability tracking.

TECHNOLOGY: Next.js (Functional Components, Hooks), Tailwind CSS (for styling).

1. Data Structures (MANDATORY CONSTANTS)

The application MUST embed the following fixed constants for the dates and the non-linear pricing model.

1.1 Booking Dates

The calendar view must only display the following 20 specific nights:

```
const CHALET_DATES = [
    '2025-12-19', '2025-12-20', '2025-12-21', '2025-12-22', '2025-12-23',
    '2025-12-24', '2025-12-25', '2025-12-26', '2025-12-27', '2025-12-28',
    '2025-12-29', '2025-12-30', '2025-12-31',
    '2026-01-01', '2026-01-02', '2026-01-03',
    '2026-01-09', '2026-01-10', '2026-01-16', '2026-01-17',
];
```

1.2 Non-Linear Pricing Table

The pricing calculation MUST use this exact `Nights` to `Total Cost` mapping. The total cost is **non-linear and step-wise**. Nights booked beyond 20 use the 20-night price.

Nights (N)	Total Cost ($)	Nights (N)	Total Cost ($)
1	45	11	375
2	90	12	395
3	130	13	415
4	170	14	430
5	205	15	445
6	240	16	455
7	270	17	465
8	300	18	475
9	325	19	480
10	355	20	485

2. Core Features and Persistence

2.1 Authentication & Session Management

- **Method:** Simple email input for identification (User View).
- **Persistence:** Use `localStorage` to save the user's email/ID to maintain the session upon return. No password verification is required for the User View.

2.2 Calendar & Booking Logic

- **Calendar View:** Display all dates from `CHALET_DATES`.
- **Selection:** Users click on a date to book/unbook a seat for that night.
- **Availability:** Maximum capacity per night is **11 seats**.
- **Conflict Prevention:** If a night is full (11 bookings), prevent further bookings unless the current user is unbooking their own reservation.
- **Data Persistence:** All bookings and payment status must be saved to a persistent database (e.g., mock a simple persistence layer or specify Firebase Firestore, if available).

2.3 Pricing & Payment

- **Real-time Calculation:** The total cost due MUST update immediately based on the number of nights selected, using the specific `Total Cost` table.
- **Payment Status:** For each night booked by the user, provide a toggle/button to mark the booking as **Paid** or **Unpaid**.

2.4 Availability Display

- For every date on the calendar, show the remaining availability (e.g., "5 / 11 Seats").
- Clearly indicate if a night is fully booked or if the current user has booked a seat.

3. Required Application Views

3.1 Login Screen

- A simple input field for the user's email address.

3.2 Main Dashboard (Post-Login)

- **Header:** Show the currently logged-in email.
- **Pricing Card:** Display the mandatory non-linear pricing table clearly.
- **Booking Summary:** Show the user's total nights booked and the corresponding final **Total Cost Due** (calculated from the table).
- **Calendar:** The main interactive calendar view.

3.3 Admin Dashboard (Password Protected)

- **Access:** Requires a password (hardcoded for simplicity) to access.
- **Overview:** Display key financial metrics:
  - Total Revenue Due (Sum of all users' "Total Cost Due").
  - Total Revenue Collected (Sum of proportional payments based on paid nights).
  - Total Revenue Pending.
- **User Management Table:** A comprehensive table listing every participant with:
  - User Email
  - Total Nights Booked
  - Total Cost Due

- Paid Amount
- Balance Due

The resulting application should be a single, self-contained file using React and Tailwind CSS.