AbsoluteSport SSR Web Application – Project Overview

Project Purpose

AbsoluteSport is a Nuxt 4 SSR web application for managing bookings, payments, and content for sports camps, clubs, football academies, events, and parties. It provides a seamless booking experience for parents and administrators, integrating with Airtable for CMS and GoCardless for payments. The app is deployed to Netlify and designed for resilience, security, and scalability.

Tech Stack

- **Framework:** Nuxt 4 (SSR)
  - [Nuxt Documentation](https://nuxt.com/docs)
- **Language:** TypeScript
- **Styling:** Tailwind CSS (with custom color variables and breakpoints)
  - [Tailwind CSS Docs](https://tailwindcss.com/docs)
- **State & Data:**
  - Airtable (CMS & booking data)
    - [Airtable API Docs](https://airtable.com/developers/web/api/introduction)
  - GoCardless (Payments)
    - [GoCardless API Docs](https://developer.gocardless.com/api-reference/)
  - Mailchimp (Email marketing)
    - [Mailchimp API Docs](https://mailchimp.com/developer/)
- **Deployment:** Netlify (with serverless and edge functions)
  - [Netlify Docs](https://docs.netlify.com/)
- **Testing:** Vitest (unit and Nuxt runtime tests)
  - [Vitest Docs](https://vitest.dev/)
- **Other:**
  - Font Awesome (icons)
  - Cloudinary (image hosting)

Key Features

- **Multi-domain Booking Flows:**
  - Holiday camps, after-school clubs, football academies, events, parties
- **Airtable Resilience Layer:**
  - All Airtable operations go through centralized utilities with error handling and graceful degradation (see `/server/api/utils/airtable.ts` and `useBookingApi.ts`)
- **GoCardless Integration:**
  - Secure payment flows for bookings
- **Content Management:**
  - Dynamic content blocks, FAQs, reviews, testimonials, and policies managed via Airtable
- **Security:**
  - Global bot protection middleware, CSP reporting, legacy URL logging, and edge function probes
- **Responsive UI:**

  - Custom breakpoints, semantic color classes, and modern component design
- **Testing:**
  - Unit and Nuxt runtime tests with composable and component mocks

Architecture Overview

Framework

- Nuxt 4.2.x (Server-Side Rendering)
- Nitro preset: `netlify`
- Deployed as Netlify Functions

Runtime Services

- **Airtable** – CMS, booking data, FAQs, policies, parties, clubs, camps
- **GoCardless** – Payments
- **Mailchimp** – Email marketing
- **Cloudinary** – Image delivery
- **Netlify** – Hosting, routing, deploy previews

Key System Domains

- Booking flows (camps, clubs, football, events)
- Contact / enquiries (party enquiries)
- CMS-driven content pages
- Payment processing
- Bot-safe SSR rendering
- Legacy URL telemetry and CSP reporting

Booking to Payment Flow: Camps

Overview

The camps booking flow allows parents to book holiday activity camps for their children. The process involves collecting parent details, child information, camp selections, and then processing payment through GoCardless.

Step-by-Step Flow

1. **Initial Page Load** (`/camps/booking`)

- User navigates to the camps booking page
- Page fetches available camps from `/api/camps/campsList` (filtered to show only camps with status "current")
- Page fetches camp locations from `/api/camps/campLocList`
- Uses `useBookingApi().guardedFetch()` which handles Airtable errors gracefully (redirects to `/booking-paused` on 429/503 errors)

2. **Parent/Guardian Details Form**

- User fills in:
  - Parent name
  - Mobile number
  - Email address
  - Terms & Conditions acceptance
- On submit, `handleSaveParent()` is called:
  - Creates a unique `paymentRef` (timestamp-based, base36)
  - Stores parent details in `savedParent` ref
  - Sets `parentAdded` flag to enable child booking form

3. **Child & Camp Selection Form**

- For each child, user selects:
  - Child first name and surname
  - Age (5–14)
  - HAF (Holiday Activities and Food) booking option (if applicable)
  - HAF ID (if HAF booking)
  - Photo permission
  - Camp location
  - Camp week (filtered by location)
  - Specific days within the selected camp week
- On "Save to booking":
  - Creates a unique `bookingRef` (timestamp-based, base24)
  - Calculates price: `totalDays × pricePerDay` (free for pupil premium)
  - Creates a `CampBookingItem` with status "reserved" or "pupil premium"
  - Adds item to `campBooking` array
  - Form resets for next child/camp selection

4. **Review Booking Details**

- User can toggle between "CampForm" and "CampBookingDetails" tabs
- Review page shows:
  - Parent details
  - All camp booking items
  - Total cost calculation
  - Payment reference
- User can remove individual booking items
- User can cancel entire booking

5. **Confirm Booking** (`confirmBooking()`)

When user clicks "Confirm Booking":

a. Create Payment Record:

```typescript
POST /api/camps/campPayment
Body: {
  parentName, mainPhone, email, paymentRef,
  numChildren, childrenNames, amountDue,
  status: "awaiting payment"
```

```
}
```

- Creates record in Airtable `camp-payments` table
- Returns payment record with `id`

b. Create Booking Records:

- Loops through each `campBooking` item
- For each item:

```typescript
POST /api/camps/campBooking
Body: {
  childName, childSurname, parentName, childAge,
  photos, location, campName, daysBooked, totalDays,
  bookingRef, pupilPrem, hafID, price, paymentRef,
  status: "reserved" | "pupil premium"
}
```

- Creates record in Airtable `camp-bookings` table
- Returns booking record with `id`

c. Redirect to Success Page:

- If both payment and booking records created successfully:

  - Redirects to `/camps/success` with query parameters:
    - `name`, `phone`, `email`
    - `children` (array)
    - `paymentRef`
    - `amountDue`
    - `bookingDate`
    - `status` ("awaiting payment" or "pupil premium")

6. **Success Page** (`/camps/success`)

- Displays booking summary (printable)
- Shows payment reference and bank transfer details
- **Payment Options:**

  Option 1: Instant Payment via GoCardless

  - User clicks "Request Pay Now"
  - `payNow()` function executes:

    Step 1: Create Billing Request

    ```typescript
    POST /api/gocardless/billing-request
    Body: {
      paymentRef,
      amount: amountDue * 100, // in pence
```

````
      description: "AbsoluteSport Holiday Camps : bookings payment:
{paymentRef}"
    }
    ```

    - Creates GoCardless billing request
    - Returns billing request `id`

    **Step 2:** Create Billing Request Flow

    ```typescript
    POST /api/gocardless/billing-request-flow
    Body: {
      id: billingRequestId,
      paymentRef,
      amount: amountDue
    }
    ```

    - Creates GoCardless billing request flow
    - Returns `authorisation_url`

    **Step 3:** Redirect to Payment

    - Button updates to "Pay Now"
    - User clicks and is redirected to GoCardless
`authorisation_url`
    - User authorizes payment with their bank
    - On success: Redirects to `/payment/success`
    - On exit: Redirects to `/payment/exit?amount={amount}
&paymentRef={paymentRef}`

  **Option 2: Bank Transfer**

  - User can pay via bank transfer using:
    - Account Name: ABSOLUTESPORT
    - Account Number: 36771585
    - Sort Code: 09-01-29
    - Reference: `{paymentRef}`
  - Booking status updated manually when payment received

### Error Handling

- All API calls use `guardedFetch()` which:
  - Catches 429 (rate limit) and 503 (service unavailable) errors
  - Redirects to `/booking-paused?context=booking`
  - Handles network errors gracefully
- Booking confirmation validates that both payment and booking
records were created
- If any step fails, user sees error message and can retry

---

## Booking to Payment Flow: Clubs
````
````

Overview

The clubs booking flow allows parents to book after-school clubs for their children. Unlike camps, clubs are term-based with fixed pricing per term. A single child can book multiple clubs in one transaction.

Step-by-Step Flow

1. **Initial Page Load** (`/clubs/booking`)

- User navigates to the clubs booking page
- Page fetches:
  - Available clubs from `/api/clubs/clubsList` (filtered to show only clubs with status "upcoming")
  - Schools from `/api/clubs/schoolList`
- Uses `useBookingApi().guardedFetch()` for error handling

2. **Form Completion**

User fills in a single form with:

Parent/Guardian Details:

- Parent name
- Mobile number
- Email address
- Alternate parent name (optional)
- Alternate contact number (optional)

Child Details:

- Child first name
- Surname
- Year group (dynamically populated from club data)
- School (selected from dropdown)
- Medical conditions

Club Selection:

- Year group and school selection filters available clubs
- User can select multiple clubs via checkboxes
- Each club shows: name, reference, and year ranges
- Cost calculation uses `useSelectionCost()` composable:
  - Sums `termCost` for all selected clubs
  - Updates dynamically as clubs are selected/deselected

Terms & Conditions:

- User must accept terms before submission

3. **Form Validation**

- Validates all required fields
- Ensures at least one club is selected
- Validates email format
- Validates year group and school selection

4. **Submit Booking** (`handleSubmitClubBooking()`)

When user clicks "Submit Booking":

a. Create Payment Record:

```typescript
POST /api/clubs/clubPayment
Body: {
  paymentRef,
  status: "awaiting payment",
  surname, childName, parentName,
  contactNumber, email,
  clubsBooked: JSON.stringify(checkedClubRefs),
  clubsQty: number of clubs,
  amountDue: totalCost,
  school, yearGroup, medicalConds
}
```

- Creates record in Airtable `club-payments` table
- Returns payment record with `id`
- If creation fails, shows error and allows retry

b. Create Booking Records:

- Loops through each selected club reference
- For each club:

  - Creates unique `bookingRef`: `{paymentRef}-{clubRef}`
  - Gets club details from filtered list

  ```typescript
  POST /api/clubs/clubBooking
  Body: {
    club: clubName,
    paymentRef, bookingRef,
    surname, childName, parentName,
    contactNumber, email,
    altParentName, altParentContact,
    medicalConds, yearGroup, school,
    startDate, endDate,
    sessionCost, sessionsPerTerm, termCost,
    status: "reserved awaiting payment",
    sessions,
  }
  ```

  - Creates record in Airtable `club-bookings` table

  - Adds to `bookingSummary` array

c. Redirect to Success Page:

- Redirects to `/clubs/success` with query parameters:
  - `name`, `childName`, `surname`
  - `phone`, `email`
  - `paymentRef`
  - `school`, `yearGroup`, `medicalConds`
  - `clubsBooked` (JSON string of club refs)
  - `clubsQty`
  - `amountDue`
  - `bookingDate`

5. **Success Page** (`/clubs/success`)

- Displays booking summary (printable)
- Shows payment reference and bank transfer details
- **Payment Options:**

  Option 1: Instant Payment via GoCardless

  - Same flow as camps:
    - "Request Pay Now" → Create billing request → Create billing request flow → Redirect to GoCardless
    - Description: `"AbsoluteSport School Clubs : bookings payment: {paymentRef}"`

  Option 2: Bank Transfer

  - Same bank details as camps
  - Uses same payment reference format

Key Differences from Camps Flow

1. **Single Form vs. Multi-Step:**

   - Clubs: One comprehensive form
   - Camps: Multi-step (parent → child/camp → review)

2. **Multiple Bookings per Transaction:**

   - Clubs: One child can book multiple clubs in one transaction
   - Camps: Multiple children/camps, but each is a separate booking item

3. **Pricing Model:**

   - Clubs: Fixed term cost per club (summed for multiple selections)
   - Camps: Per-day pricing (days × price per day)

4. **Filtering:**

   - Clubs: Filtered by school + year group (both required)
   - Camps: Filtered by location, then by HAF eligibility

5. **Booking References:**
   - Clubs: `{paymentRef}-{clubRef}` (unique per club)
   - Camps: Single `bookingRef` per child/camp combination

Error Handling

- Same error handling as camps via `guardedFetch()`
- Validates payment record creation before proceeding to bookings
- If any booking fails, previous bookings remain (partial success scenario)
- User can retry if submission fails

Common Components & Utilities

`useBookingApi()` Composable

- Provides `guardedFetch()` wrapper around `$fetch`
- Handles Airtable rate limiting (429/503) by redirecting to `/booking-paused`
- Catches network errors gracefully
- Used throughout booking flows for all API calls

`useSelectionCost()` Composable

- Calculates total cost for selected items (used in clubs)
- Dynamically updates as selections change
- Generic utility that works with any option type

Airtable Utilities (`server/api/utils/airtable.ts`)

- Centralized Airtable access
- Respects `AIRTABLE_DISABLED` flag
- Unified logging and error handling
- Provides: `getAirtableBase()`, `getAirtableTable()`, `airtableSelect()`, `airtableCreate()`

GoCardless Integration

- **Billing Request:** Creates payment request in GoCardless
- **Billing Request Flow:** Creates authorization flow for user to complete payment
- Redirects configured:
  - Success: `/payment/success`
  - Exit: `/payment/exit?amount={amount}&paymentRef={paymentRef}`

Environment Variables

```
The following environment variables are required for proper
operation:

- **Airtable-CMS**

  - `AT_API_KEY`
  - `AT_BASE_ID`

- **Mailchimp**

  - `MC_API_KEY`
  - `MC_AUDIENCE_ID`
  - `MC_SERVER_PREFIX`

- **GoCardless**

  - `GC_ACCESS_TOKEN`

- **CSP Logging**

  - `NUXT_CSP_AIRTABLE_TOKEN`
  - `NUXT_CSP_AIRTABLE_BASE_ID`

- **Cloudinary**

  - `CLOUD_NAME`

- **Operational Flags**
  - `AIRTABLE_DISABLED` (set to "true" to disable Airtable calls)
  - `NUXT_PUBLIC_BOOKING_PAUSED` (set to "true" to pause bookings)

---

## Development Workflow

- `npm run dev` — Start dev server
- `npm run typecheck` — TypeScript checks
- `npm run test` — Run all tests
- `npm run test:nuxt` — Nuxt runtime tests
- `npm run test:unit` — Unit tests
- `npm run build` — Production build

---

## Official Documentation & Resources

- [Nuxt 4 Docs](https://nuxt.com/docs)
- [Airtable API](https://airtable.com/developers/web/api/
introduction)
- [GoCardless API](https://developer.gocardless.com/api-reference/)
- [Mailchimp API](https://mailchimp.com/developer/)
- [Tailwind CSS](https://tailwindcss.com/docs)
- [Netlify Docs](https://docs.netlify.com/)
- [Vitest](https://vitest.dev/)
```

Suggested Annexures

- `DATABASE_SCHEMA.md` — Full Airtable schema and modelling instructions
- `README.md` — Technical setup and usage
- `README-CLIENT.md` — Client-facing overview
- `server/api/database/schema.ts` — TypeScript interfaces for all tables

For further details, see the annexure files and in-code documentation. This overview should be kept up to date as the project evolves.