1. High-level overview

The project is a small multi-page storefront driven by a shared JavaScript cart and a single central product catalog.

Main pages:

index.html

- Start page with:

    Event basics form (name, date, players, field length).

    Accordion sections for Discs, Field Supplies, Extras.

    A mini “starter pack” builder.

    A mini per-player / cart summary widget.

    A “Based on the items you have selected we recommend also” block.

TD_Guided_Discs.html

- Guided flow page that shows only disc and marker products.

TD_Guided_Supplies.html

- Guided flow page that shows field supplies, including suggested quantities based on saved field length.

TD_Guided_Extras.html

- Guided flow page that shows bottles and apparel.
-

TD_Cart.html

- Cart and checkout summary page, with quantity editing, tax and shipping, and a form that posts to checkout.php.

checkout.php

- Server side handler that reads the submitted cart JSON payload, recalculates totals, and emails a plain text receipt.

Shared assets:

- Styles: assets/tdpro.css
- Images: assets/images/*.png

Image filenames are tied directly to product ids, for example disc-driver-custom.png ⇐ id: "disc-driver-custom".

Core JavaScript modules:

assets/tdpro-cart-core.js

- Shared cart logic, localStorage storage, “Add to cart” handling, currency formatting.

assets/tdpro-guided.js

- Product catalog (window.TD_PRODUCTS), catalog renderer (window.renderCatalog), suggested quantities for supplies, and recommendation rules.

assets/tdpro-cartcount.js

- Keeps the header cart badge in sync on all pages.

assets/tdpro-widget.js

- Renders the right-hand “per player” mini widget that shows totals per player and per tournament.

assets/tdpro-progress.js

- Renders the guided flow progress bar on guided pages.

* All pages share the same cart via localStorage under the key "cart", wrapped by the window.__td helper in tdpro-cart-core.js.

2. How the cart works

2.1 LocalStorage format and __td

tdpro-cart-core.js is responsible for storing and retrieving the cart:

- Storage key: cart
- Shape of each cart entry:

```
{
  id: "disc-driver-custom",
  name: "Custom Driver Disc",
  category: "Discs",
  price: 14.0,          // base USD
  qty: 6,
  imageUrl: "assets/images/disc-driver-custom.png",
  customizable: true,   // used to show custom fields
  customNotes: "",
  customImages: []      // reserved for uploads later
}
```

Helpers exported on window.__td:

- __td.loadCart() → array of cart line objects.
- __td.saveCart(cartArray) → writes to localStorage.
- __td.clearCart() → empties the cart.
- __td.fmt(amount) → formats a base USD number into the selected currency.

2.2 Add-to-cart buttons

Product cards generated by renderCatalog contain a button like:

```
<button class="btn"
        data-add="1"
        data-id="disc-driver-custom"
        data-customizable="1">
    Add
```

</button>

tdpro-cart-core.js attaches a delegated click handler:

- Finds the nearest .card.
- Reads:
  - Title from .title
  - Unit price from .price[data-price]
  - Quantity from .qty-inline input or an input[type=number].
- Constructs or updates a cart entry with that id.
- Saves via __td.saveCart.
- Fires a cartchange CustomEvent.
- Shows a small temporary “Added to cart” message near the button.

Any button with data-add="1" automatically participates in this logic as long as it lives inside a .card with a properly structured body and price element.

2.3 Cart badge and mini widget

- tdpro-cartcount.js listens to:
  - DOMContentLoaded
  - cartchange
- It reads __td.loadCart() and updates #cartCount in the header.
- tdpro-widget.js uses the same cart plus the saved players count to:
  - Show per-player cost.
  - Show tournament subtotal and estimated totals.
  - It renders inside the element that has the widget markup (on the home page).

3. Central product catalog

3.1 window.TD_PRODUCTS

In assets/tdpro-guided.js you will find:

```
window.TD_PRODUCTS = [
  // Discs
  { id: "disc-driver-plain",   name: "Plain Driver Disc",  price: 10.00, category: "Discs",
customizable: false, details: "High-speed driver • stable flight" },
  { id: "disc-driver-custom",  name: "Custom Driver Disc", price: 14.00, category: "Discs",
```

```
customizable: true,  details: "Customizable • High-speed driver • stable flight" },
  // … midrange, putter, markers …
  // Bottles
  // Apparel
  // Supplies (spray paint, flags, bug spray, sunscreen, etc.)
];
```

Key fields:

- id

Unique identifier and primary key used everywhere. Must match the base filename of the corresponding product image in assets/images/, for example:

    - id: "spray-paint" ⇒ assets/images/spray-paint.png
    - id: "flags-pack" ⇒ assets/images/flags-pack.png
- name

Human readable name shown in tiles and cart.

- price

Base USD price used for calculations. Currency conversion is applied when rendering.

- category

Used to group products. Existing values include "Discs", "Markers", "Bottles", "Apparel", "Supplies".

- customizable

Boolean that controls whether the item has customization fields in the cart (notes, uploads) and whether it appears in the “custom disc” tab versus “plain disc” tab on the home page.

- details

Short descriptive text shown in the product tile.

From this catalog, tdpro-guided.js also builds a fast lookup map (window.TD_PRODUCT_MAP) and a set of supply ids for some helper functions.

4. How pages use the catalog

4.1 Home page (index.html)

Key locations:

- Tournament basics form

Saves to localStorage keys like:

  - td_event_name
  - td_event_date
  - td_players
  - td_distance_feet

That last one is used for supplies “Suggested: N” logic.

- Disc accordion

```
<div class="disc-tabs">
  <button class="disc-tab-btn active" type="button" data-target="discCustomGrid">Custom discs</button>
  <button class="disc-tab-btn" type="button" data-target="discPlainGrid">Plain discs</button>
</div>
<div id="discCustomGrid" class="disc-tab-panel active"></div>
<div id="discPlainGrid" class="disc-tab-panel"></div>
```

At the bottom of the file:

```
// Render guided catalogs into our single page
if (window.renderCatalog && Array.isArray(window.TD_PRODUCTS)) {
  // Discs and markers into both grids
  renderCatalog('discCustomGrid', { categories: ['Discs', 'Markers'] });
  renderCatalog('discPlainGrid',  { categories: ['Discs', 'Markers'] });

  function filterDiscGrid(id, wantCustom) {
    var root = document.getElementById(id);
    if (!root) return;
    var cards = root.querySelectorAll('.card[data-customizable]');
    cards.forEach(function (card) {
```

```
            var isCustom = card.getAttribute('data-customizable') === '1';
            card.style.display = (isCustom === wantCustom) ? '' : 'none';
        });
    }
    filterDiscGrid('discCustomGrid', true);
    filterDiscGrid('discPlainGrid',  false);

    // Supplies and extras
    renderCatalog('catalogSupplies', { category: 'Supplies' });
    renderCatalog('catalogExtras',   { categories: ['Bottles', 'Apparel'] });
}
```

All disc and marker products in TD_PRODUCTS appear in these grids automatically. The actual filtering between “custom” and “plain” is controlled by each product’s customizable flag.

- Field Supplies accordion

```
<div id="catalogSupplies"></div>
```

Filled via renderCatalog('catalogSupplies', { category: 'Supplies' }). Supply cards for flags and spray paint get an additional “Suggested: N packs / cans” label based on td_distance_feet.

- Extras accordion

```
<div id="catalogExtras"></div>
```

Filled via renderCatalog('catalogExtras', { categories: ['Bottles', 'Apparel'] }).

- Starter pack cards

There is a script that binds each “disc pack” card using a mapping from logical keys to product ids:

```
var ITEM_MAP = {
    driver:  { plain: "disc-driver-plain",   custom: "disc-driver-custom" },
    mid:     { plain: "disc-midrange-plain", custom: "disc-midrange-custom" },
    putter:  { plain: "disc-putter-plain",   custom: "disc-putter-custom" },
    marker:  { plain: "disc-marker-plain",   custom: "disc-marker-custom" },
    hoodie:  { plain: "hoodie-podium",       custom: "hoodie-podium" },
```

```
  steel:   { plain: "bottle-steel-34-plain",   custom: "bottle-steel-34-custom" },
  plastic: { plain: "bottle-squeeze-26-plain", custom: "bottle-squeeze-26-custom" }
};
```

When the user chooses plain/custom in the select menus and clicks “Add pack”, the script:

  - Pulls players count from the .pack-players input.
  - Translates each selection into a product id using ITEM_MAP.
  - Looks up the product in TD_PRODUCTS for price and name.
  - Adds appropriate quantities to the cart using __td helpers.
- “Based on the items you have selected we recommend also” block

This uses TD_renderSidebarRecommendations and TD_buildConfigsForAdd in tdpro-guided.js. That module looks at the cart contents and suggests complementary items such as:

  - Bugspray when you have sunscreen but no bug spray.
  - Matching molds of discs when your counts are unbalanced.

4.2 Guided pages

Each guided page has a similar pattern:

- A main content <main> element.
- A catalog container, for example on TD_Guided_Discs.html:

```
<div id="catalogDiscs"></div>
```

- At the bottom:

```
document.addEventListener('DOMContentLoaded', function () {
  // Mark page as guided and update progress
  markFlowGuided();
  progress.render('guided', { page: 'discs' });

  // Render only discs and markers
  renderCatalog('catalogDiscs', { categories: ['Discs', 'Markers'] });
});
```

Supplies and extras guided pages call renderCatalog with appropriate categories. Supplies guided pages also benefit from the same “Suggested: N” logic as the home page since it is built into renderCatalog itself (TD_applySuppliesSuggestions).

4.3 Cart page (TD_Cart.html)

Key elements:

- Cart summary container with line items and totals.
- Header badge uses the same #cartCount and tdpro-cartcount.js.
- Hidden field #checkoutCartPayload inside the checkout <form> is populated with a JSON string of the cart prior to form submission.

Inline script on the cart page:

- Calls __td.loadCart() to get items.
- Renders each line as a card row with:
    - Product image.
    - Name and SKU (id).
    - Unit price.
    - Quantity control (plus / minus and direct entry).
    - Line total.
    - Customization fields if customizable is true (notes textarea and a Save button).
- Computes:
    - Subtotal.
    - Tax at 7%.
    - Shipping: flat 9.99 if there is at least one item, zero otherwise.
    - Grand total.

When the user clicks “Checkout”, before the form is submitted, the current cart is serialized into JSON and placed in the cartPayload hidden input, which is then posted to checkout.php together with contact information.

4.4 Checkout backend (checkout.php)

- Accepts POST only.
- Reads:

```
$name    = trim($_POST["fullname"] ?? "");
$email   = trim($_POST["email"] ?? "");
$address = trim($_POST["address"] ?? "");
```

```
$city    = trim($_POST["city"] ?? "");
$state   = trim($_POST["state"] ?? "");
$zip     = trim($_POST["zip"] ?? "");
$cartJson = $_POST["cartPayload"] ?? "[]";
$cart     = json_decode($cartJson, true) ?: [];
```

- Iterates the cart entries, calculating:

```
$lineTotal = $qty * $unitPrice;
$subtotal += $lineTotal;
```

- Reapplies the same 7% tax and flat 9.99 shipping.
- Builds a plaintext receipt string listing all items and totals.
- Uses mail() to email that receipt to the buyer if the email address is valid.
- Echoes a very simple confirmation page and prints the receipt inside <pre>.

There is intentionally no payment processing logic. This is receipt and confirmation only.

5. How to add a new store item

5.1 Steps for a standard product

1. **Create image asset**
    a. Add a PNG into assets/images/ with a clear, unique base name, for example:

```
assets/images/hoodie-staff.png
```

2. **Add product definition to TD_PRODUCTS**
    a. Open assets/tdpro-guided.js.
    b. Find the window.TD_PRODUCTS = […]; block near the top.
    c. Add a new object in the appropriate category section:

```
{ id: "hoodie-staff",
  name: "Staff Hoodie",
  price: 48.00,
  category: "Apparel",
  customizable: true,
  details: "Heavy-weight fleece • S–3XL"
```

},

   d. Make sure:
      i. id matches the image file name (hoodie-staff.png).
      ii. category is one of the existing groupings so that renderCatalog picks it up.

3. **Verify where it should appear**
   a. Guided Extras page: any "Apparel" will show automatically.
   b. Home page Extras accordion: renderCatalog('catalogExtras', { categories: ['Bottles', 'Apparel'] }) will also include it automatically.
   c. It will also be visible in the cart and recommendation logic once added.
4. **Optional: include in packs**

If the new product should be part of one of the starter packs:

   a. Open index.html.
   b. Find the var ITEM_MAP = { … }; block.
   c. Add a mapping that points to your new id, and make sure there is a matching row in the HTML for the appropriate pack.

5. **Optional: extend recommendation rules**
   a. If this product should be recommended automatically as a companion to another product, update TD_buildConfigsForAdd in tdpro-guided.js and follow the existing patterns, for example the disc bundles and bug spray / sunscreen examples.
6. **Test**
   a. Reload index.html, the guided page, and TD_Cart.html.
   b. Confirm:
      i. New product tile appears.
      ii. Add button works and updates the header cart badge.
      iii. Product shows correctly on the cart page and in the emailed receipt.

5.2 Adding a new supply that uses “Suggested: N” labels

If you are adding or renaming the **flags** or **spray paint** products:

1. In TD_PRODUCTS use ids like:
   a. flags-pack
   b. spray-paint
2. In TD_applySuppliesSuggestions in tdpro-guided.js you will see:

```
cards.forEach(function (card) {
  var id = card.getAttribute('data-id') || '';
  if (id === 'flags-pack') flagCard = card;
  if (id === 'spray-paint') paintCard = card;
});
```

Update these comparisons if you change the ids.

3. The suggestion formula is:

```
var flags = Math.ceil(dist / 50);
var paint = Math.ceil(dist / 200);
```

You can adjust those divisors to change the recommendation density.

The function reads td_distance_feet from localStorage, which is set on the home page when the user saves their field distance.

6. How to remove or hide a store item

To remove a product everywhere:

1. **Remove or comment out its entry in TD_PRODUCTS**
    a. Open assets/tdpro-guided.js.
    b. Delete or comment the object for that id.
2. **Clean up references**
    a. Search across the repo for the id string, for example "disc-trophy-custom".
    b. Update or remove:
        i. Any entry in ITEM_MAP in index.html.
        ii. Any logic in TD_buildConfigsForAdd or TD_applySuppliesSuggestions that refers to that id.
3. **Optionally delete its image**
    a. Remove assets/images/<id>.png once you are sure it is not referenced anywhere.
4. **Test**
    a. Reload each page and confirm the tile is gone and no JavaScript errors appear in the browser console.

7. Where to tune business rules

- **Prices, names, and descriptions**

assets/tdpro-guided.js in the TD_PRODUCTS array.

- **Which items show up in each section or page**

Controlled by category and the renderCatalog calls:

    - Discs and markers: ['Discs', 'Markers']
    - Supplies: 'Supplies'
    - Extras: ['Bottles', 'Apparel']
- **Supplies quantity suggestions**

TD_applySuppliesSuggestions in assets/tdpro-guided.js.

- **Recommended companion items when cart changes**

TD_buildConfigsForAdd and TD_renderSidebarRecommendations in assets/tdpro-guided.js.

- **Starter pack composition**

ITEM_MAP object and data attributes on the pack card elements in index.html.

- **Tax and shipping**
    - Front end: inline script in TD_Cart.html.
    - Backend: checkout.php uses the same 7% tax and 9.99 flat shipping constants.

8. Open items / future work

The following work items are still pending and should be treated as the handover backlog:

1. Tournament save and repeat ordering

Create a mechanism to save a complete “tournament profile”, not just the basic details. The goal is that returning buyers can load a previous tournament and:

a. Automatically restore their previous cart contents.
b. Update logos and customization as needed.
c. Slightly tailor quantities based on changed player counts or field length.

My thought process to do this would be to set a unique ID to each new login/email address into a database. Then in that database once a purchase is made for a tournament or a tournament is saved aa unique ID is made for that tournament name along with all item quantities that were chosen. Then each of those tournament IDs can be linked to each individual's email ID for future population onto the site when someone chooses a previous tournament they've saved or purchased for.

2. Seasonal prompts for bug spray and sunscreen

Implement a script that reads the saved event **date** on the home/start page and, based on the season, checks whether any bug spray or sunscreen are present in the cart. If the date falls in a warm season and none of those items are in the cart, prompt the buyer to confirm they really want to proceed without them.

This should run on the cart page and the home page in the field supplies section, using the saved date and current cart to decide whether to show the gentle reminder.

3. Responsive design with media queries

Extend assets/tdpro.css with proper media queries so the layout adapts cleanly to different screen widths:

d. Collapse nav into a more compact form on small screens.
e. Stack cards in one column on mobile.
f. Ensure the mini widget and cart summary remain readable on tablets and phones.

4. Suggested item quantities by tournament size

Go beyond the current field-length based suggestions and add rules by tournament size tiers, for example:

g. Small tournament: up to N players.
h. Medium tournament: between N and M players.
i. Large tournament: above M players.

For each tier, define suggested quantities for key items (discs, flags, paint, apparel, bottles). Then use both field length and player count to compute and display "Suggested: N" quantities for those products.