Software Design Document for Salesbricks Order Builder

Revision History

Date	Author	Change Summary
2025-08-08	Daniel Nebreja	Initial draft of SDD

1. Introduction

1.1 Purpose

This Software Design Document describes the architecture, requirements, tools, data models, risks, possible improvements and implementation approach for Salesbricks Order Builder. This will help the Project Managers, Software Engineers and other people who will be involved in the implementation of the app

1.2 Scope

- A SaaS product that allows sellers to configure an order for their buyers through the order builder page
- The page will consist of 4 major stages: **Customer Info**, **Product & Plan**, **Contract Terms**, and **Review & Fine-Tune**.
- Hard-coded sample data, no real backend integration yet

1.3 Audience

- Project Managers who will facilitate the development of the application
- Engineers who will be implementing the SDD

---

2. System architecture

2.1 Folder Structure

```
|-- docs/                   # Project documentation
|    `-- SDD.md             # Software Design Document
|-- public/                 # Static assets served as-is
|    `-- data/              # Sample JSON data
|        |-- addons.json    # Add-ons master data
|        |-- customers.json # Customer master data
|        `-- products.json  # Products & plans master data
|-- src/                    # Application source code
|   |-- components/         # React UI components
|   |   |-- customer/       # Customer Info step components
|   |   |   |-- index.tsx       # Entry for customer step
|   |   |   `-- customer-info.tsx
|   |   |-- product/        # Product & Plan step components
|   |   |   |-- index.tsx       # Entry for product step
|   |   |   |-- plans.tsx
|   |   |   `-- edit-price.tsx
|   |   |-- review-order/     # Review & Fine-Tune step components
|   |   |   |-- index.tsx       # Entry for review step
|   |   |   |-- order-summary.tsx
```

```
|   |   |   `-- add-ons.tsx
|   |   |-- terms/            # Contract Terms step components
|   |   |   `-- index.tsx        # Entry for terms step
|   |   `-- workflow.tsx      # Tab/stepper container
|   |-- constants/          # Application constants
|   |   `-- constants.ts     # Lists (STAGES, US_STATES, CONTRACT_PERIODS)
|   |-- context/            # React Context providers
|   |   `-- order.context.ts # Order state & navigation
|   |-- hooks/              # Custom React hooks
|   |   `-- order.hooks.ts   # Data-loading & mutation hooks
|   |-- interfaces/         # TypeScript type definitions
|   |   `-- models.ts        # ICustomer, IProduct, IPlan, etc.
|   |-- services/           # Data service abstractions
|   |   `-- order.service.ts # Fetching from /public/data - For ease of scaling when
we implement actual backend
|   `-- utils/              # Utility functions
|       |-- date.ts          # Date manipulation helpers
|       |-- formatter.ts     # Display formatting helpers
|       `-- math.ts          # Price & total-calculation helpers
`-- App.tsx                # Root application component
```

2.2 Deployment

- **Hosting**: Deployed as a static build on Vercel
- **Build command**: `yarn build`
- **Env vars**:
    - `REACT_APP_API_URL` (when integrated later)
- **Preview URLs**:
    - Pull Request previews on Vercel
    - Production: `https://salesbricks-app.vercel.app`

---

3. Technology Stacks

Language

- Typescript
- Javascript

Frameworks & Libraries

- React JS
- Node JS

Styling

- Tailwind CSS
- ShadCN
- MUI

State management & Form

- API Context
- React Hook Form
- Zod

Data Fetching

- React Query

Linting
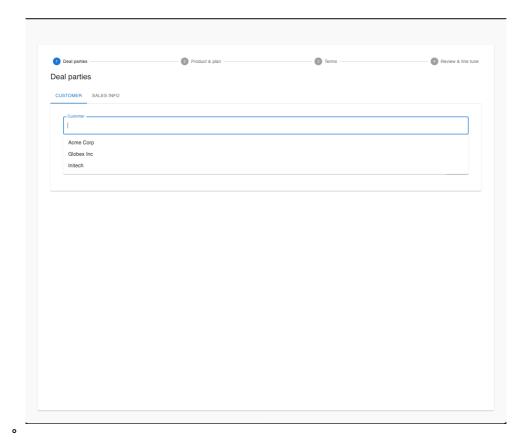
- Prettier
- ESLint

---

4. Data models

```typescript
interface Customer {
  id: string;
  customer: string;
  street?: string;
  address1?: string;
  address2?: string;
  city?: string;
  state?: string;
  zipCode?: string;
}

interface Plan {
  id: string;
  name: string;
  basePrice: number;
}

interface Product {
  id: string;
  name: string;
  plans: Plan[];
}

interface AddOn {
  id: string;
  name: string;
  unitPrice: number;
  quantity?: number;
}

interface ContractTerms {
  startDate: string;    // formatted date string
  endDate: string;      // formatted date string
}

interface Order {
  customer?: Customer;
  product?: Product;
  plan?: Plan;
  addOns: AddOn[];
  contract?: ContractTerms;
}
```

---

5. UI Flow & Wireframes

Figure 1: Customer Information Stage

- Select a customer account from the dropdown or create a new one if the customer doesn't exist on the data

  ○

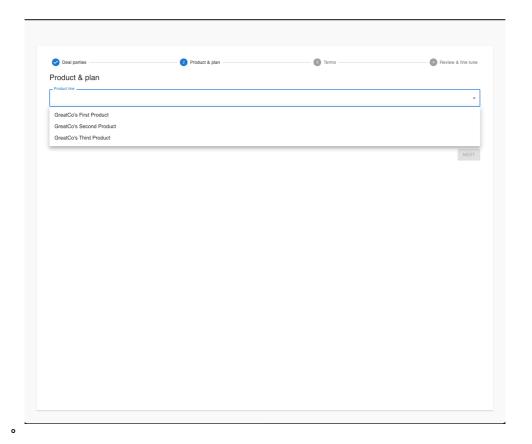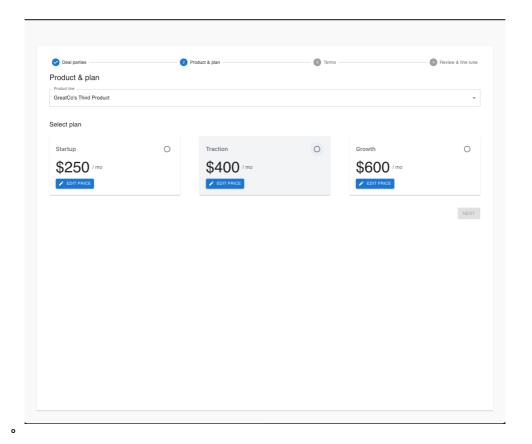- Check the `Pre-populate customer information` checkbox to input the customer's address
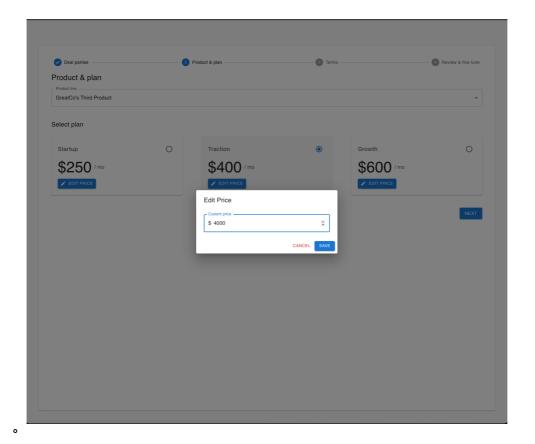
- Deal parties — Product & plan — Terms — Review & fine tune

Deal parties

CUSTOMER   SALES INFO

Customer
Acme Corp

☑ Pre-populate customer information

Address 1
123 Elm St

Address 2

City
Springfield

State
IL

Zip code
62701

NEXT

Figure 2: Product & Plan Stage

- Select a product line from the dropdown

- Select a plan, the price are already pre-populated but it should be editable

- ○
- Cicking the Edit button will open a modal and user can edit the price

-
  - Clicking `Save` will update the current selected plan

○

Figure 3: Contract Terms Stage
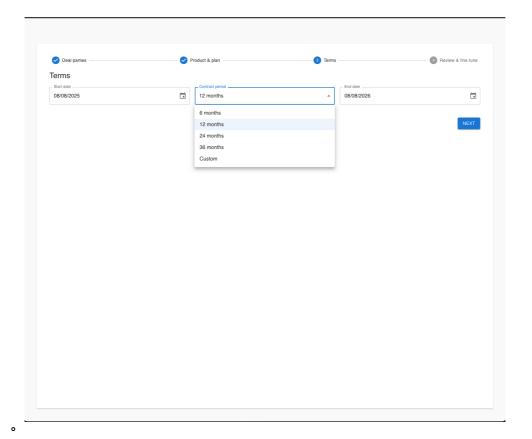- User can select a Start Date from a datepicker
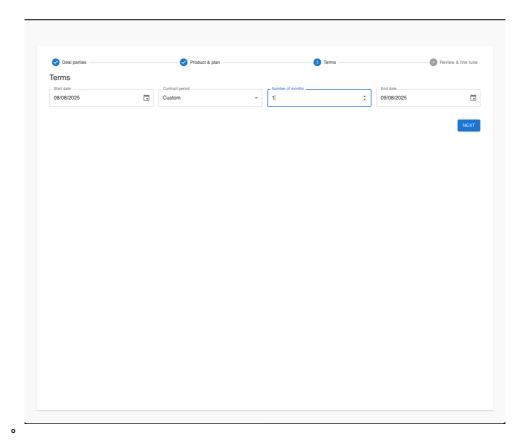
- ○

- User selects a duration:

    - 6 months
    - 12 months
    - 24 months
    - 36 months
    - Custom (reveals a *Number of months input*, minimum 1)

- End date is auto-calculated and read-only.

    - ○ > **Contract Period Options**
        >
        > - `6 months`
        > - `12 months`
        > - `24 months`
        > - `36 months`
        > - `Custom` *(user-entered integer ≥ 1)*

○

- If `Custom` is selected on the contract dropdown it should show a free text input and they can input a **number** of months with the minimum of 1

- End date should be automatically updated accordingly based on the selected contract duration or custom input month/s, the field should be read-only
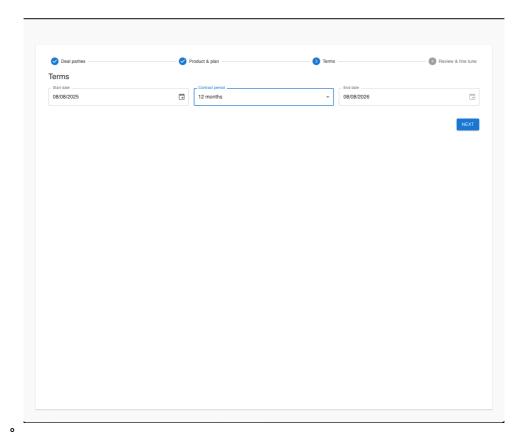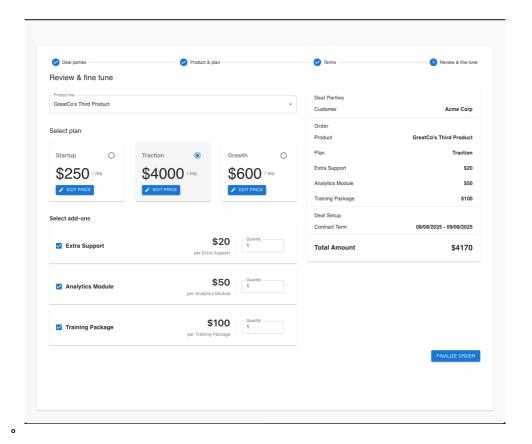
    o

Figure 4: Review & Fine-Tune Stage
FIGURE 4.1: EDIT ORDER SECTION (RIGHT SIDE OF THE PAGE)

- It should show all the information of their order along with the add-ons capability

- It should show all the available add-ons at the bottom and can edit the quantity with the minimum to **0**, unchecking the add-on will remove the specific add-on from the order
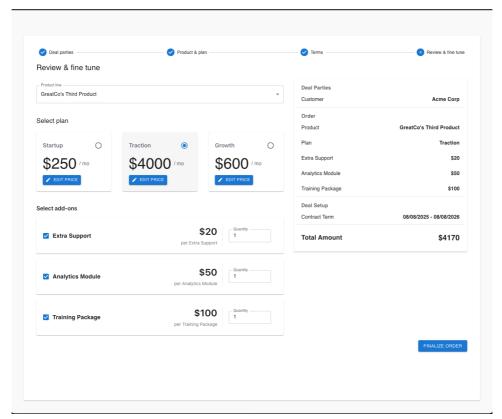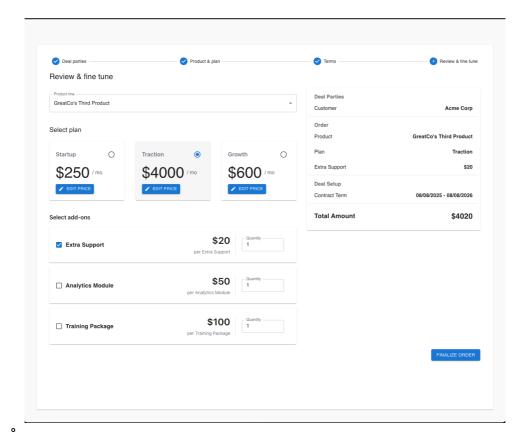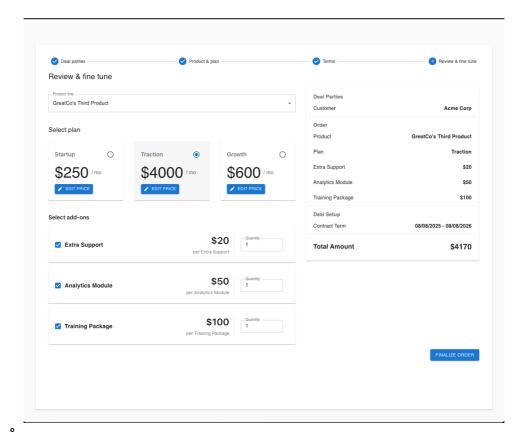
FIGURE 4.2: SUMMARY SECTION (LEFT SIDE OF THE PAGE)

- Unchecking/Removing an add-on on the Edit order section should also reflect on this page as well
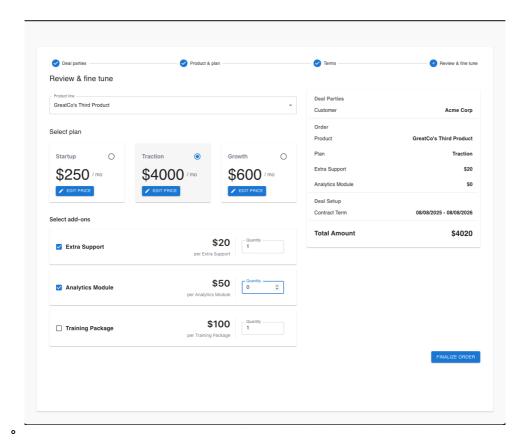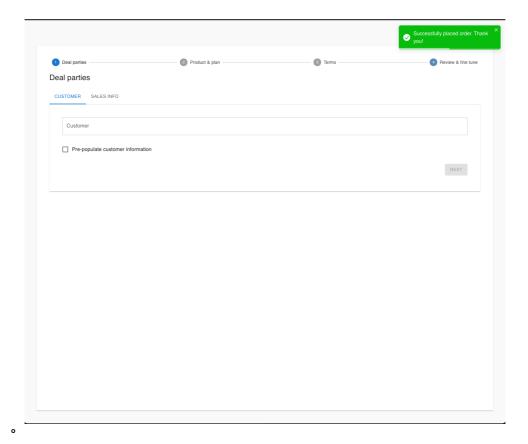
- Left column of the add-on list is the name and the right side is the _total amount (quantity _ price per unit)

○

- Below is the **total price (plan price + all add-ons' total)**

- Clicking the **Finalize Order** button will show a success message and will redirect step 1

---

6. Validation Rules

Customer information

- Name is `required`
- Billing address is `optional` **Product & Plan**
- Product line is `required`
- Plan is `required` **Contract Terms**
- Start date is `required`
- Contract Period is `required`
- End date is `required` (auto-populated) **Review**
- Add-ons are `optional`

---

7. Quality Attributes & Edge Cases

Possible risk/problem	Solution
User navigates backward	Persist the data to localStorage
User refreshes the page	Persist the data to localStorage
User input an invalid character	Sanitize or validate (handled by zod)
User forgot to input some required field/s	Disable the **Next** button or show an error notification

---

8. Implementation Timeline

Task	Estimate
Setup project, contexts, interfaces, custom hooks	**20 mins**
Steps 1-4 development (validations, setup utils, styling, etc.)	**1.5 hours**
Manual e2e testing, polish, and optimization	**25 mins**

9. Assumptions & Future Works

9.1 Assumptions
- No real backend, static JSON data is provided
- Currency is USD accross the app
- No authentication
- Single Page Application (SPA)

9.2 Future Works
- Implement user authentication
- Implement unit testing & e2e testing (*Jest, Playwright or Cypress*)
- Integrate the backend server using **Node JS**, **PostgreSQL**, **Prisma for ORM**
  - Rest API endpoints
    - CUSTOMERS
      - GET `/api/customers`
      - POST/GET `/api/customer`
      - PATCH/PUT `/api/customer/:id`
    - PRODUCTS (includes plan)
      - GET `/api/products`
      - GET/POST `/api/product`
      - PATCH/PUT `/api/product/:id`
    - PLANS
      - GET `/api/plans`
      - GET/POST `/api/plan`
      - PATCH/PUT `/api/plan/:id`
    - ADD-ONS - GET `/api/add-ons` - GET/POST `/api/add-on` - PATCH/PUT `/api/add-on/:id` *All endpoints return JSON*

10 Non-Functional requirements
- Performance optimization
  - Lazy loading
  - Memoize child components
  - Memoize expensive calculations and functions
  - Debounce input events
  - Map array should have key property
  - Add linter
- Accessibility
  - Use semantic HTML
  - Apply ARIA roles
- Security
  - Communicate through HTTPS only
  - Sanitize inputs

- Hash passwords
- Store sensitive information thru `.env`
- Scalability
  - Break large components into smaller ones
  - Store repetitive functions into utility
  - Create custom hooks for reusability
- Deployment
  - Host in Vercel or Netlify
  - Implement CI/CD pipeline for build, lint, test and deployment on each push or merge