# à la carte app

## Design

'A la carte' SaaS application:

Hey there! This is all about **'A la carte'**, a super cool **Software-as-a-Service (SaaS) platform** built just for restaurants. It's designed to help both the folks running the place and the awesome customers have a better time. And guess what? It's perfect whether you've got one cozy spot or a bunch of busy locations!

### What's This App All About? (Key Tech Stuff & What It Does):

This section dives into the main technical bits and what 'A la carte' can actually do for a restaurant.

#### 1. Getting Restaurants Onboard & Subscriptions:

Signing up and managing your plan should be a breeze!

* **Easy Self-Service Signup:** Restaurants can jump right in and sign up for 'A la carte' themselves, no fuss!
* **Tell Us About Your Business:** When signing up, restaurants will share all their important deets, like:
  + Your restaurant's name, where it is, and how to reach you.
  + When you're open, what kind of food you serve, and your cool logos/pictures.
  + A quick first menu setup (or just import what you've got!).
* **Got Multiple Spots? No Problem!:** The sign-up process lets you add and set up **all your physical locations** under one main restaurant account. Each one gets its own specific info too – address, contact, hours, you name it!
* **Payment Ready:** We'll securely grab credit card info for billing your subscription. This means we'll hook up with a **super safe payment system** (like Stripe or Braintree).
* **Pick Your Features!:** Restaurants can pick and choose what they want from us when they sign up or later, like:
  + **Core Menu Magic:** Just simple digital menu stuff.
  + **Customers Order Themselves:** Let folks order right from their phones using QR codes. How cool is that?
  + **Waiter Help:** Turn on the admin portal so your waiters can take orders on their devices.
  + **Seat-by-Seat Ordering:** This is a neat extra feature where you can tag each item to a specific seat.
  + **Built-in Payments:** If you want, customers can pay right through the app.
  + **Printer Hook-Up:** Set up all your kitchen and bar printers.
* **Subscription Smarts:** We've got a system to handle your subscription plans, upgrades, downgrades, and billing cycles. **Your subscription will totally factor in how many locations you have**, so pricing can be tiered or per-location.
* **Feature Control:** The platform will keep track of exactly which features and options each restaurant gets, based on their plan and what you've set up.

#### 2. How It Gets Out There & How You Get In (Deployment & Access):

 **Custom Web Address Options:** Each restaurant can use their very own custom website address (like yourrestaurant.com) to get to the app. This provides a strong brand presence. Alternatively, they can use a subdomain provided by 'A la carte' (e.g., [restaurant-name].alacarteapp.com). For multi-location restaurants, this gets even smarter:

* **Unified Domain with Location Picker:** yourrestaurant.com (or [restaurant-name].alacarteapp.com) can lead to a page where customers or staff select a specific location before proceeding.
* **Location-Specific Subdomains:** Each location can have its own distinct subdomain, like main-street.yourrestaurant.com or downtown.[restaurant-name].alacarteapp.com. This setup requires advanced technical capabilities for **custom web address mapping**, **DNS management**, and **security certificate provisioning** (including wildcard certificates for subdomains).

 **Access Portals and URLs:**

* **Restaurant Sign-up:** New restaurants can begin their journey at the main platform website: alacarteapp.com
* **Restaurant Admin Portal:** Restaurant owners and managers can administer their data and operations. Access can be:
  + For a single main account or primary location: yourrestaurant.com/admin or [restaurant-name].alacarteapp.com/admin
  + For direct access to a specific location's admin panel: yourrestaurant.com/[location-slug]/admin or [location-slug].[restaurant-name].alacarteapp.com/admin
* **Waiter Portal:** Your staff can access their order-taking and table management tools. Access can be:
  + For a specific location: yourrestaurant.com/[location-slug]/waiter or [location-slug].[restaurant-name].alacarteapp.com/waiter
  + A waiter typically operates within a single physical location.
* **Customer Portal (Your Menu!):** Customers can easily find your delicious menu and place orders. Access can be:
  + For a unified experience with a location picker: yourrestaurant.com/menu or [restaurant-name].alacarteapp.com/menu
  + For direct access to a specific location's menu: yourrestaurant.com/[location-slug]/menu or [location-slug].[restaurant-name].alacarteapp.com/menu
  + For direct ordering (e.g., via QR code): yourrestaurant.com/[location-slug]/order or [location-slug].[restaurant-name].alacarteapp.com/order
* **Kitchen Display System (KDS) Portal:** Your kitchen staff can see orders flow in, organized by prep area. Access is always location-specific:
  + yourrestaurant.com/[location-slug]/kds/[area] or [location-slug].[restaurant-name].alacarteapp.com/kds/[area]
  + The [area] part means specific kitchen sections you set up, like drinks, salad-and-appetizers, main-course, or desserts-and-coffee!

#### 3. All Your Data, All in One Place (Centralized Data Management):

This is the brain of the operation, keeping all your restaurant's info organized.

* **Multi-Location Data Smarts:** Our PostgreSQL database is built to handle all your data across multiple restaurant locations. It keeps everything separate but easy to find for each spot.
* **Menu Management Hub:** This is where restaurants can easily create, update, and organize all their menu items. Think prices, descriptions, dietary info, and what's available. You can set up menus for all locations at once or customize them for each individual spot. We're using a powerful PostgreSQL database for this, it's great for relationships and flexible data!
  + **Lots of Pictures for Your Food!:** Restaurants can upload **multiple pictures for every menu item**. We'll store these safely (like on Google Cloud Storage or AWS S3) and link them in the database.
  + **Make Your Own Menu Sections:** Restaurants can totally **create their own menu sections or categories** (like "Drinks," "Appetizers," "Desserts"). Then, each menu item gets **assigned to one or more of these custom sections** to keep things super organized for customers.
  + **Dietary Tags:** Restaurants can **tag each menu item with important dietary info** (like "Gluten-Free," "Vegetarian," "Contains Nuts," "Spicy"). This helps customers filter and get alerts!
  + **Global vs. Local Menus:** Restaurants with multiple locations can choose to have:
    - **Global Menu Items:** These are available everywhere, with the same price, description, pictures, etc. Change it once, and it updates everywhere!
    - **Location-Specific Menu Items:** These are unique to a particular spot, letting you offer local specials, different prices, or availability. You've got total control over each branch's menu!
* **Restaurant Info Central:** Admins can set up all the important restaurant details like hours, contact info, location data, and branding stuff, all managed for each individual location.

#### 4. Customer-Facing Mobile Ordering (The Fun Part for Diners!):

This is how customers interact with your menu and place orders.

* This is a **super responsive website or a Progressive Web App (PWA)** that customers use.
* **QR Code Magic:**
  + **Dine-in Ordering:** If customers are actually at the restaurant, they'll find QR codes (on tables, for example). Scanning these will tell the app **exactly which table they're at (and the location ID)**, so they can order themselves and get food delivered right to them!
  + **Takeout/Delivery:** For folks ordering from home, QR codes (on marketing, your website) will let them see the menu and **order for delivery or pickup**. These QR codes will just link to the general menu, then ask for delivery details.
* **Works on Any Device:** Customers can get to the menu and order system from their phones, laptops, or tablets. Super flexible!
* **Pick Your Location (for multi-location spots):** If a customer just goes straight to your main menu link (like restaurant.com/menu), they'll be asked to pick a specific restaurant location first to see its menu and order.
* **Slick Design & Easy to Use:** We're building a responsive web app or PWA that's really easy to navigate and makes ordering a breeze. We're using modern web tech to make it look great and feel smooth. The menu will automatically organize items into the sections you've defined.
* **Orders Go Through Instantly:** Ensures customer orders get to the restaurant's system right away.
* **Seat-Based Ordering (Optional Extra!):** The app will have an **optional feature for restaurants to turn on seat-based ordering**. If it's on, you can link individual items in an order to specific diners or seat numbers. This can happen in a couple of ways:
  + **Customers Do It Themselves:** When customers scan the QR code, they might be asked to pick their seat number or enter their name before adding food.
  + **Waiters Help Out:** The admin portal's ordering screen will let waiters tag items with seat numbers or guest names as they take orders.
  + **Choosing Between Names & Seats:**
    - **Using Seat Position (e.g., "Seat 1", "Seat 2"):**
      * **Pros:** It's super clear, especially in a restaurant where seat numbers are often set or easy to spot. Makes talking to the kitchen and food runners simpler. Less chance of typos than names!
      * **Cons:** A bit less personal. If people move seats, it can get confusing.
    - **Using the Person's Name:**
      * **Pros:** Way more personal and friendly for the customer! Easier for staff to remember who ordered what if they're chatting with guests.
      * **Cons:** Names can get misspelled, leading to mix-ups. Guests might move around. Requires the customer to type in a name, which adds a tiny extra step.
    - **Best Approach:** For maximum flexibility and user-friendliness, offering **both options** (name or seat number) and letting the restaurant decide which to enable (or even allowing the customer to choose at the start of their order) is ideal. For most self-service scenarios, seat numbers are often the most practical and efficient. But for a more personal touch, names can be great! The current design document already supports this flexibility, which is fantastic!
* **Customer Accounts & Order History (for Website Orders):**
  + For orders placed through the website (not just QR codes), customers can **sign up for an account** using their email/password or maybe even social logins.
  + When signing up, customers will give us important info like their **address, phone number, and any dietary restrictions**.
  + **Safe Payment Storage:** Customers can securely save their favorite payment methods (like credit card info, safely tokenized with our payment system).
  + Once signed up, customers will just use their **login to place orders**.
  + **Past Orders & Reordering:** Logged-in customers can see their **past order history**. They can **view details of old orders** and even **reorder a past meal**, with the option to **tweak items** (like quantity or modifications) before placing it again.
* **Customize Your Food! (Notes & Modifiers):**
  + For every item added to an order, customers will see an **optional "observation" box** (like a text field). This lets them add personal requests or changes, like "less salt," "no sugar," "extra spicy," or "sauce on the side," if that menu item is set up to allow it.
  + Plus, for items that have specific customization options (like steak or pizza), customers will get predefined **modifier choices**. For example:
    - **Steak Doneness:** ("Rare," "Medium-Rare," "Well Done")
    - **Pizza Toppings/Crust:** ("No Onion," "Swap Mozzarella for Provolone," "Thin Crust")
    - **Whole/Half Customization (e.g., Pizza):** For items like pizza, customers can pick one flavor for the whole thing or choose two different flavors/styles for a "half and half" pie! This will have a specific way to select options for each half.
  + **Combo Meal Building:** For combo meals or executive lunches, customers will be guided step-by-step to build their meal. This includes:
    - **Main Course Pick:** Choose from a list of main courses (like fish, chicken, or a veggie option).
    - **Side Dish Pick:** Select a certain number of side dishes (e.g., "pick 2 sides") from a list (like rice, beans, salad). The system will make sure they pick the right amount!
    - **Optional/Must-Have Add-ons:** Options for things that are included (like a specific drink or dessert) or optional extras (like an extra drink or upgraded dessert) that might cost a bit more.
    - Restaurants can fully set up these complex structures in the Admin Portal, defining all the parts, choices, and rules for each combo.
* **Awesome Picture Display:**
  + When customers check out the menu, menu items will show **multiple pictures in a cool rotating carousel or slideshow**.
  + When a customer taps on a menu item for more details, a special **item detail screen will show all the pictures in a gallery**, letting them swipe, zoom, or see them in high-res.
* **Pay Yourself! (Self-Service Payment - Optional):**
  + Customers (whether they're dining in or ordering out) will have the **option to pay right through the app**.
  + **Multiple Credit Card Support:** The payment interface will allow for **splitting the bill across multiple credit cards** (like by amount or by specific items).
  + **Split the Bill by Seat!:** For dine-in customers, the system will let each person **pay for their own items by seat**. The app will automatically **split taxes and tips fairly** among those paying, or based on how the restaurant sets it up.
  + **Flexible Partial Payments & Early Departure:** This is super handy! Customers will be able to **pay their portion of the bill and leave before others**, even if the rest of the party is still hanging out (think happy hour!). They can choose to pay for:
    - **One or more specific seats.**
    - **Their share divided by the number of people** currently on the bill.
    - A fixed amount they want to pay.  
      This can be done either self-service through the app or with the waiter's help through the Admin Portal. The system will keep track of who paid what and the remaining balance for the table.
  + **Super Secure Payments:** We'll hook up with a **PCI-compliant payment gateway** to handle all credit card transactions safely, using tokenization so we don't store sensitive card data on our servers.
* **Dietary Restriction Check (Optional):**
  + **Ask Before You Order:** Customers will see an **optional pop-up or section** where they can tell us their dietary restrictions or allergies before they even look at the menu or order. This can cover common allergens (nuts, dairy, gluten), dietary choices (vegetarian, vegan), or other needs.
  + **Menu Filters/Highlights:** Based on what the customer says, the menu will dynamically:
    - **Filter out** items that definitely don't work with their restrictions.
    - **Highlight** items that are safe or good to go.
    - **Show alerts or warnings** on items that *might* have an allergen or need a tweak, even if they're not totally filtered out (e.g., "Has Dairy," "Can be made Gluten-Free if you ask").
  + **Confirm Before You Buy:** If a customer adds something to their cart that clashes with their restrictions, an **alert or confirmation box** will pop up, asking them to acknowledge the potential issue before they continue.
* **Customer Loyalty Program (Optional!):**
  + The app will offer a **points-based loyalty system** where customers earn points every time they buy something.
  + **Tiered Rewards:** Restaurants can set up different loyalty levels (like Bronze, Silver, Gold) with better perks as customers earn more points.
  + **Personalized Deals:** The system will let restaurants send **special offers and promotions** to loyal customers based on what they've bought before, their preferences, and their loyalty level. This could be birthday treats, exclusive discounts, or early access to new menu items!
  + Customers can check their loyalty points and available rewards right in their account profile.
* **Customer Notifications:**
  + The system will send **real-time alerts to customers** for important order and promo stuff, including:
    - **Order Confirmed!:** You'll get an instant confirmation when your order goes through.
    - **Order Updates:** Get notifications for big changes in your order status (like "Order Accepted," "Cooking Started," "Ready for Pickup/Delivery," "Out for Delivery," "Delivered").
    - **Promo Messages:** Get special messages about offers, loyalty rewards, or new menu items (if you opt in, of course!).
  + These alerts can pop up in the app, as push notifications (for PWA users), or even as text messages (if the restaurant sets it up and you say yes!).
* **Customer Feedback & Review Management:**
  + **Tell Us What You Think!:** Customers can give direct feedback (like star ratings, comments) on specific dishes or their whole experience right after they order or pay. This can happen **at the moment of payment** or **afterward via email or other methods** (like an in-app notification prompt after the meal is complete).
  + **Feedback Per Item:** Customers will also have the option to provide **feedback on individual menu items** they ordered (e.g., a star rating or short comment specifically for the "Spicy Chicken Sandwich"). This gives restaurants super granular insights into what's hitting the mark and what might need a tweak!
  + **Review Site Integration:** We'll smoothly connect with popular review sites (like Google, Yelp) so customers can easily leave public reviews, and restaurants can see and reply to them right from their Admin Portal.
  + **AI for Feedback:** (Future AI cool stuff!) We'll automatically figure out the vibe of customer reviews to quickly spot common issues or things you're doing great!

#### 5. Admin Portal (The Restaurant's Command Center!):

This is the secure hub where restaurant staff manage everything.

* This is a **secure website** that gives restaurant staff super detailed control over their part of the platform.
* **Waiter Management & Ordering:**
  + **Waiter ID:** Waiters will log into the app using their own usernames/passwords or unique staff IDs.
  + **Take Orders by Table:** Waiters will start taking orders by first picking the right table number.
  + **Optional Customer-Specific Ordering:** If you want, waiters can even tag each item in an order to a specific customer at the table (like by seat number or guest name). This uses the same "Seat-Based Ordering" tech that self-service customers use.
* **Multi-Location Management:** Managers of restaurants with multiple locations will have one central dashboard to handle all their branches. This includes viewing and setting up menus, orders, staff, and settings for each individual location.
* **Login & Permissions:** Getting into the admin portal is secure, using **passwords** (maybe even with an extra security step!).
* **Who Can Do What (Role-Based Access Control):** Different staff roles (like manager, chef, cashier) will have different levels of access, and these can even be set for specific locations.
* **Order Management System:** This shows incoming orders in real-time, letting staff update their status (like pending, preparing, ready for pickup/delivery). This system will also show **seat-specific item details** if that feature is on, and clearly tell you which location the order came from. **Any notes from customers, chosen modifiers, combo meal setups, and declared dietary restrictions will be clearly visible** in the order details.
  + **Split Bill Helper:** The Order Management System will have tools for staff to handle and keep an eye on split bills, including seeing individual seat payments and any remaining balances.
* **Check Self-Service Payments:**
  + The Order Management System will clearly show the **payment status for each order and table** (like "Unpaid," "Partially Paid," "Fully Paid").
  + Restaurant staff (like hosts, managers, waiters) can see this status instantly on their **tables, PCs, or dedicated POS terminals**.
  + For partially paid orders, the system will show how much is still owed.
  + This helps staff quickly see if a party has paid up before they leave, avoiding mix-ups and making checkout smoother!
* **Menu Customization Setup:** The admin portal will have tools for restaurants to define and manage structured modifier options for their menu items. This includes setting prices for add-on modifiers, setting up complex options like "half and half" selections, and **defining how combo meals are built**. It'll also have a place to **upload and manage multiple pictures per menu item**, letting you reorder or delete them. **Restaurants can also create, edit, and organize their menu sections/categories** and put individual menu items into them. Plus, the system will have a way to **tag menu items with all sorts of dietary info** (allergens, dietary types).
* **Promotions & Discounts Manager:**
  + Restaurants will have a dedicated screen to **create and manage all sorts of promotions and discounts**.
  + This includes setting up:
    - **Percentage-off deals** (like "10% off your total bill!").
    - **Fixed amount discounts** (like "$5 off!").
    - **Buy-one-get-one (BOGO) offers** (like "Buy one pizza, get one free!").
    - **Happy Hour specials** (discounts on certain items during specific times).
    - **Combo deals** (different from regular combo meals, these are promo bundles!).
  + Promos can be applied based on:
    - **Specific items or categories.**
    - **Minimum order amount.**
    - **Time of day or day of the week.**
    - **Using a promo code.**
  + The system will let you set **start and end dates/times** for promos, and **how many times they can be used** (like per customer, or total uses). Discounts can kick in **automatically** based on rules (like happy hour) or **manually with promo codes** entered by customers or staff.
* **Loyalty Program Setup:**
  + The admin portal will have tools for restaurants to **set up and manage their loyalty program**.
  + This includes setting up **how points are earned** (like points per dollar spent), defining **loyalty levels** and their perks, and creating and managing **special offers** for different customer groups.
  + Restaurants can track customer loyalty status and how rewards are used.
* **Order Prioritization:**
  + Restaurant staff (like managers, expeditors) can **manually bump up the priority of individual orders or specific items** on the Order Management System or KDS.
  + When they prioritize, staff will be asked to **pick a reason or type in a quick note** (like "VIP customer," "Rush order," "Kitchen oops," "Allergy alert!").
  + Prioritized orders will really stand out on the KDS and Order Management System so kitchen staff know it's urgent!
* **Manager Notifications:**
  + The system will send **real-time alerts to managers** for important stuff, including:
    - **Payment Trouble:** Alerts for failed payments, chargebacks, or any weirdness with self-service payments.
    - **Big Problems:** Notifications for system glitches, integrations breaking (like with payment systems), or major operational hiccups.
    - **Overdue Orders:** Alerts for orders that are taking too long to make or deliver, so managers can jump in.
    - **Low Stock Alerts:** (Coming soon!) Notifications when you're running low on key ingredients.
  + These alerts can pop up on the Admin Portal, be sent via email, or even through other messaging apps you use.
* **Analytics & Reports:** Get awesome insights into sales data, what's selling hot, and how efficient things are running. You can filter and analyze data per location or for all of them! This includes generating **important reports and graphs** to help management make smart choices.
* **Price Display Control:** Restaurants can choose in the admin portal whether prices are only shown when someone scans a QR code (like for dine-in) or always visible across all customer access methods (phone, tablet, PC). This setting can be customized for each location.

#### 6. Distributed Kitchen Display System (KDS) & Printer Integration (The Back-of-House Heroes!):

This is all about getting orders from the customer to the kitchen and out to the printers, smoothly!

* **Knows Where to Go:** The system will smartly send orders to the right KDS and printers based on where the order came from.
* **Networked Printing:** Helps send specific parts of an order to the right thermal printers in different kitchen stations (like the bar for drinks, prep area for salads, main kitchen for entrees, pastry station for desserts). If seat-based ordering is on, printouts will include **individual item tags** (like seat number, guest name) for super accurate delivery! **Customer notes, chosen modifiers, combo meal parts, and any declared dietary restrictions/allergies will be clearly printed** with the item, with big alerts for critical info!
* **Print Queue Management:** Makes sure print jobs get delivered reliably and handles any internet hiccups.
* **Communication Smarts:** Uses solid communication methods like **TCP/IP sockets, HTTP POST requests, or special printer APIs** to talk to networked printers reliably and efficiently.
* **Order Splitting Brains:** Has clever ways to automatically sort and send order items to the right prep stations based on rules you set. Now, it's even smarter with **seat-based item allocation**, **location-specific routing**, and **combo meal component routing**!
* **Real-time Kitchen Notifications:** The KDS will give **real-time visual and/or sound alerts** to kitchen staff for:
  + **New orders!:** Lets them know new tickets are in right away.
  + **Order changes!:** Highlights any tweaks to existing orders (like added items, special requests).
  + **Item ready!:** Shows when items are marked "prepared" or "ready for delivery" by other stations.
  + **Orders taking too long!:** Alerts staff if prep times are going over what's expected.
  + **Priority orders!:** Clearly shows orders that need to be rushed, along with why!

#### 7. How We Build It (Tech Stack & Quality Assurance):

This section covers the technical foundation and how we ensure everything runs smoothly and securely.

* **Backend Power:** We're using Node.js with Express.js to build strong and scalable APIs.
* **Testing, Testing, 1-2-3!:**
  + **Unit Tests:** Lots of detailed tests for individual parts of the backend and frontend to make sure everything works perfectly.
  + **End-to-End (E2E) Tests:** Automated tests that act like a real user, checking the whole app from start to finish.
  + **Integration Tests:** Testing how different parts of the system talk to each other (like API calls to the PostgreSQL database).
  + **Performance Testing:** Simulating tons of users to make sure the app stays fast and stable even when it's super busy.
* **Security First!:**
  + **No SQL Injection!:** We're super careful with all database queries to prevent nasty SQL injection attacks.
  + **Clean Inputs (No XSS!):** We rigorously clean all user-entered info on both the customer and server sides to stop Cross-Site Scripting (XSS) attacks.
  + **Login & Permissions:**
    - **Password Hashing:** Using strong, one-way encryption for super secure password storage.
    - **Session Management:** Securely managing user sessions (like using JWTs with smart expiration or secure server-side sessions).
    - **Rate Limiting:** Protecting against folks trying to guess passwords too many times.
  + **API Security:**
    - **CORS Setup:** Properly setting up Cross-Origin Resource Sharing (CORS) to control who can access our APIs.
    - **HTTPS/TLS:** Everything is encrypted when it travels over the internet!
    - **Parameter Validation:** Rigorously checking all incoming requests to prevent bad data and potential exploits.
  + **Content Security Policy (CSP):** Using a strict CSP on the frontend to stop all sorts of content injection attacks.
  + **Environment Variables:** All sensitive stuff (like database passwords, API keys) is stored as environment variables, never hardcoded!
* **Easy to Maintain & Operate:**
  + **Centralized Error Handling:** A solid system for catching and gracefully handling any app errors.
  + **Detailed Logging:** Keeping detailed records of everything that happens in the app – events, errors, security stuff – for monitoring and debugging.
  + **Dependency Management:** Regularly updating all our tech (Node.js, Express, and other libraries) to fix security holes and get new features.
* **Getting Found on Google (SEO Strategy):**
  + **Platform SEO:** We'll focus on attracting restaurants with keywords like "Restaurant SaaS," "online ordering system," etc., through detailed feature pages, case studies, and blog content on the main alacarteapp.com site.
  + **Restaurant-Specific SEO:** Helping individual restaurants show up well in local searches and for their menu items by:
    - Encouraging unique, awesome descriptions for their business and menu items.
    - Guiding them on how to optimize their **Google My Business (GMB)** profiles with consistent info and direct menu links.
    - Adding **Schema Markup (Structured Data)** (like Restaurant, Menu) directly to public-facing pages so search engines understand them better.
    - Making it easy to get reviews and build local mentions.
    - Making sure all restaurant pages load super fast!
    - Using **Server-Side Rendering (SSR) or Pre-rendering** for important public pages to help search engines find them.
    - Providing dynamic **Sitemaps** for each restaurant's content.
    - Letting them customize **Meta Tags** (title, meta description) and \*\*Open Graph (OG) tags`).

### Workflow: On-Premises Customer Journey with 'A la carte'

This is how a customer typically experiences 'A la carte' when they're dining in, from scanning a QR code to leaving the restaurant.

#### 1. Arrival & Initial Engagement

* **Customer Action:** Customer arrives at the restaurant and is seated at a table.
* **System Interaction:** The customer locates a **QR code** (e.g., on the table, menu stand).
* **Customer Action:** Customer scans the QR code using their smartphone.
* **System Interaction:** The 'A la carte' system identifies the **specific table and location ID** embedded in the QR code. The customer's **responsive web application (PWA)** loads in their browser, displaying the menu relevant to that specific location.
* **Customer Action (Optional):** If the restaurant has enabled **seat-based ordering**, the customer may be prompted to enter their seat number or a name for their seat.

#### 2. Browsing & Ordering

* **Customer Action:** Customer browses the **dynamic menu**. They can navigate through **customizable menu sections/categories**.
* **System Interaction (Dietary Restriction Control - Optional):**
  + **Customer Action:** Customer is presented with an **optional prompt** (e.g., pop-up, dedicated section) to declare their **dietary restrictions or allergies**.
  + **System Interaction:** Based on the declared restrictions, the menu dynamically **filters out or highlights** items, and **displays alerts/warnings** on conflicting items.
* **Customer Action:** Customer selects desired menu items.
* **System Interaction:** For each selected item, the system presents **item-level customization options** (e.g., "notes" field for notes, predefined modifier options like meat doneness, pizza toppings, or structured combo meal configurations).
* **Customer Action:** Customer customizes items as needed and adds them to their cart.
* **System Interaction (Confirmation on Order):** If an item conflicts with declared dietary restrictions, an **alert or confirmation box** will pop up, asking them to acknowledge the potential issue before they continue.
* **Customer Action:** Customer reviews their order in the cart.
* **System Interaction:** The system calculates the subtotal.

#### 3. Order Placement & Kitchen Processing

* **Customer Action:** Customer confirms and places the order via the app.
* **System Interaction (Real-time Order Synchronization):** The order is immediately transmitted to the restaurant's system.
* **System Interaction (Admin Portal - Order Management System):** The order appears in real-time on the Admin Portal, clearly showing the table number, items, customizations, and any declared dietary restrictions.
* **System Interaction (KDS & Printer Integration):**
  + The system intelligently **routes order components** to the correct **Kitchen Display System (KDS)** stations (e.g., drinks to bar, entrees to main kitchen) and designated thermal printers.
  + KDS screens display the new order with **real-time visual/audible notifications** for kitchen staff.
  + Printouts include **seat-specific item tags**, customer observations, selected modifiers, combo meal components, and prominent alerts for dietary restrictions/allergies.
* **Staff Action (Kitchen/Expeditor):** Kitchen staff prepare the items.
* **Staff Action (Admin Portal/KDS - Order Prioritization):** Managers or expeditors can **manually bump up the priority of individual orders or specific items**, with a reason, which is highlighted on the KDS.

#### 4. Service & Delivery

* **Staff Action (Kitchen/Expeditor):** As items are prepared, kitchen staff mark them as "prepared" or "ready for delivery" on the KDS.
* **System Interaction (Waiter Notification):** Waiters receive an **automated notification** (e.g., on their mobile device) when all items for a specific table's delivery stage are ready.
* **Staff Action (Waiter):** Waiter collects the prepared items from the pass.
* **Staff Action (Waiter):** Waiter delivers the items to the correct table and seat (if seat-based ordering is active).
* How Waiters Know What to Deliver (App vs. Printouts):  
  This is a super important question for smooth service! Here's the lowdown on how waiters know who gets what, using both the app and printouts.
  + **Using the App (The Smart Way!):**
    - The "Waiter Notification" feature means waiters get an awesome **automated alert** (probably on their mobile device, running a special waiter version of the 'A la carte' app) when dishes for a specific table are all set.
    - If "Seat-Based Ordering" is turned on, the app won't just tell the waiter *what* food is ready for a table, but also *exactly which seat* (or person's name) each dish belongs to! This helps them deliver food with pinpoint accuracy.
    - The "Order Management System" in the Admin Portal also gives staff a live view of all orders, including all those neat seat-specific details.
  + **Why Restaurants Still Print Stuff (It's About Backup & Clarity!):**
    - The "KDS & Printer Integration" is there for a reason! While the app is key for real-time updates and digital management, physical printouts still do some really important jobs, especially in the kitchen and for food runners:
      * **Kitchen Flow:** Printouts often go straight to the kitchen (or show up on KDS screens) to guide the cooking process. A physical ticket can be super quick and easy for chefs to glance at while they're busy, especially for complicated orders or specific tweaks.
      * **Backup Plan:** Tech can sometimes glitch, right? A physical printout is a fantastic backup in case a device battery dies, the Wi-Fi acts up, or there's a software hiccup.
      * **Smooth Hand-offs:** For food runners or different stations (like the bar or main kitchen), a physical ticket can be a clear, tangible item to pass along with the prepared food.
      * **Seat-Specific Tags:** Like the Canvas says, printouts can include "seat-specific item tags." This means even if the waiter uses the app for general delivery, a small tag on the plate itself (from the printer) can confirm *exactly* who gets that dish.
      * **Customer Notes & Allergy Alerts:** Printouts can clearly show customer notes, modifications, and super important dietary restrictions right there with the order for kitchen staff. This helps make sure nothing is missed during cooking.
  + **So, It's a Team Effort!** The app gives waiters real-time, dynamic info and alerts, while printouts offer a reliable backup, detailed instructions for the kitchen, and clear physical tags for getting food to the right person!

#### 5. Payment Process

* **Customer Action (Self-Service Payment - Optional):** Customer chooses to pay via the app.
* **System Interaction:** The app displays the bill, including items, taxes, and tips.
* **Customer Action (Multiple Credit Card Support):** The payment interface will allow for **splitting the bill across multiple credit cards** (like by amount or by specific items).
* **Customer Action (Split the Bill by Seat!):** For dine-in customers, the system will let each person **pay for their own items by seat**. The app will automatically **split taxes and tips fairly** among those paying, or based on how the restaurant sets it up.
* **Customer Action (Flexible Partial Payments & Early Departure):** This is super handy! Customers will be able to **pay their portion of the bill and leave before others**, even if the rest of the party is still hanging out (think happy hour!). They can choose to pay for:
  + **One or more specific seats.**
  + **Their share divided by the number of people** currently on the bill.
  + A fixed amount they want to pay.  
    This can be done either self-service through the app or with the waiter's help through the Admin Portal. The system will keep track of who paid what and the remaining balance for the table.
* **System Interaction (Super Secure Payments):** We'll hook up with a **PCI-compliant payment gateway** to handle all credit card transactions safely, using tokenization so we don't store sensitive card data on our servers.
* **System Interaction (Admin Portal - Payment Verification):** The Order Management System updates the table's payment status (e.g., "Unpaid," "Partially Paid," "Fully Paid"). Staff can view this in real-time.
* **Customer Action:** Customer receives a digital receipt (optional).

#### 6. Departure & Feedback

* **Customer Action:** Customer leaves the restaurant.
* **System Interaction (Customer Feedback & Review Management):**
  + **Tell Us What You Think!:** Customers can give direct feedback (like star ratings, comments) on specific dishes or their whole experience right after they order or pay. This can happen **at the moment of payment** or **afterward via email or other methods** (like an in-app notification prompt after the meal is complete).
  + **Feedback Per Item:** Customers will also have the option to provide **feedback on individual menu items** they ordered (e.g., a star rating or short comment specifically for the "Spicy Chicken Sandwich"). This gives restaurants super granular insights into what's hitting the mark and what might need a tweak!
  + **Review Site Integration:** We'll smoothly connect with popular review sites (like Google, Yelp) so customers can easily leave public reviews, and restaurants can see and reply to them right from their Admin Portal.
  + **AI for Feedback:** (Future AI cool stuff!) We'll automatically figure out the vibe of customer reviews to quickly spot common issues or things you're doing great!

## Development

Based on the comprehensive design document for your 'A la carte' SaaS application, the best way to develop it would be primarily as a **Progressive Web App (PWA) for the customer-facing interface** and a **responsive web application for the Admin Portal**.

Here's a breakdown of why this approach is highly suitable for your needs:

### 1. Progressive Web App (PWA) for Customer-Facing Interface:

* **Seamless Access via QR Codes:** Customers can simply scan a QR code and immediately access the menu and ordering system in their web browser, without needing to download or install anything. This is a huge advantage for on-premises self-service.
* **Cross-Platform Compatibility:** A single codebase (HTML, CSS, JavaScript) works across all devices (Android phones, iPhones, tablets, PCs) that have a modern web browser. This significantly reduces development time and cost compared to building separate native apps for iOS and Android.
* **Near-Native Experience:** PWAs can offer many features traditionally associated with native apps, such as:
  + **Offline Capabilities:** Customers can still browse the menu or even place orders (which would sync once online) even with intermittent internet connectivity, using service workers for caching. This aligns well with your "Offline Mode / Resilience" future enhancement.
  + **Push Notifications:** For real-time order status updates and promotional messages.
  + **"Add to Home Screen":** Customers can "install" the PWA to their device's home screen, making it feel like a native app without going through an app store.
* **Instant Updates:** When you deploy a new version or update the menu, changes are immediately available to all users without requiring them to manually update an app.
* **SEO Benefits:** Being a web application, it benefits from standard web SEO practices, making it discoverable through search engines.

### 2. Responsive Web Application for the Admin Portal:

* **Accessibility:** Restaurant staff can access the management dashboard from any device with a web browser – a desktop PC in the back office, a tablet carried by a manager, or even a smartphone.
* **Unified Development:** Maintaining a single web technology stack for both the customer and admin interfaces streamlines development and reduces complexity.
* **Real-time Management:** WebSockets (as noted in your design document) are ideal for the real-time updates needed for order management, kitchen display systems, and various notifications.

**Why Native Mobile Apps (Android/iOS) might be less ideal for your initial launch:**

* **Higher Development Cost & Time:** Building and maintaining separate native applications for both Android (Kotlin/Java) and iOS (Swift/Objective-C) is significantly more expensive and time-consuming.
* **App Store Friction:** Users must download the app from an app store, which can be a barrier to adoption, especially for casual diners using QR codes. App store review processes can also delay updates.
* **Maintenance Overhead:** You'd have two distinct codebases to manage, debug, and update.

### In summary:

Starting with a **PWA for the customer-facing side** and a **responsive web application for the Admin Portal** provides the most efficient, flexible, and accessible solution for the 'A la carte' application, perfectly aligning with your current feature set and planned future enhancements. This approach allows you to quickly launch, iterate, and scale across multiple restaurant locations without the overhead of native app development. Native apps could be considered in a later phase if specific, highly specialized native features become absolutely essential.

## Project Folder Structure

**Project Folder Structure for 'A la carte'**

This structure promotes modularity, separation of concerns, and ease of maintenance for both your backend API and your frontend application.

alacarte-app/

├── backend/

│ ├── src/

│ │ ├── controllers/ # Handles incoming requests, orchestrates service calls, prepares responses.

│ │ │ ├── authController.js

│ │ │ ├── menuController.js

│ │ │ └── orderController.js

│ │ ├── services/ # Contains core business logic. Interacts with models.

│ │ │ ├── authService.js

│ │ │ ├── menuService.js

│ │ │ └── orderService.js

│ │ ├── models/ # Database interaction logic (e.g., PostgreSQL queries, ORM definitions).

│ │ │ ├── userModel.js

│ │ │ ├── menuItemModel.js

│ │ │ └── orderModel.js

│ │ ├── routes/ # Defines API endpoints and links them to controllers.

│ │ │ ├── authRoutes.js

│ │ │ ├── menuRoutes.js

│ │ │ └── orderRoutes.js

│ │ ├── middleware/ # Custom Express middleware (e.g., authentication, validation, logging).

│ │ │ ├── authMiddleware.js

│ │ │ ├── errorHandler.js

│ │ │ └── requestLogger.js

│ │ ├── utils/ # Helper functions, common utilities (e.g., data formatting, validators).

│ │ │ ├── jwtUtils.js

│ │ │ └── validationUtils.js

│ │ ├── config/ # Configuration files (e.g., database settings, environment-specific configs).

│ │ │ └── db.js # Database connection setup

│ │ └── db/ # Database migration scripts, schema definitions.

│ │ ├── migrations/

│ │ └── seeds/

│ ├── tests/ # Unit and integration tests for backend components.

│ │ ├── controllers/

│ │ ├── services/

│ │ └── models/

│ ├── .env # Environment variables (e.g., DB credentials, JWT secret, PORT).

│ ├── package.json # Backend project metadata and dependencies.

│ └── server.js # Main backend application entry point.

│

├── frontend/

│ ├── public/ # Static assets directly served by the web server (e.g., index.html, favicon).

│ │ └── index.html

│ │ └── favicon.ico

│ ├── src/

│ │ ├── components/ # Reusable UI components (e.g., Button, Card, Modal).

│ │ │ ├── common/

│ │ │ ├── menu/

│ │ │ └── cart/

│ │ ├── pages/ # Top-level components representing different views/routes (e.g., MenuPage, CartPage).

│ │ │ ├── MenuPage.jsx

│ │ │ ├── CartPage.jsx

│ │ │ └── OrderStatusPage.jsx

│ │ ├── assets/ # Images, fonts, icons (prefer SVGs/emojis where possible).

│ │ │ ├── images/

│ │ │ └── icons/

│ │ ├── hooks/ # Custom React hooks for reusable logic.

│ │ │ └── useAuth.js

│ │ ├── contexts/ # React Context API for global state management (e.g., AuthContext, CartContext).

│ │ │ ├── AuthContext.jsx

│ │ │ └── CartContext.jsx

│ │ ├── utils/ # Utility functions (e.g., date formatters, data transformers).

│ │ │ └── api.js # Centralized API call functions

│ │ ├── services/ # Logic for interacting with the backend API (e.g., fetchMenu, placeOrder).

│ │ │ ├── authService.js

│ │ │ └── menuService.js

│ │ ├── styles/ # Main CSS files, Tailwind config, any global styles.

│ │ │ └── index.css # Contains Tailwind directives and custom global CSS

│ │ └── main.jsx # Main entry point for the React application.

│ ├── tests/ # Component and end-to-end tests for the frontend.

│ │ ├── components/

│ │ └── pages/

│ ├── .env # Environment variables (e.g., VITE\_API\_BASE\_URL).

│ ├── package.json # Frontend project metadata and dependencies.

│ ├── vite.config.js # Vite build configuration.

│ └── tailwind.config.js # Tailwind CSS configuration.

│

├── .gitignore # Specifies intentionally untracked files to ignore.

└── README.md # Project overview, setup instructions, and deployment notes.

**Explanation of Key Directories:**

**Backend (**backend/**)**

* src/: Contains all the core application logic.
  + controllers/: These are your API endpoints' first stop. They receive requests, call the necessary services, and send back responses. Keep them light!
  + services/: This is where your main business rules live. If a task needs multiple steps or involves complex logic, it belongs here. Services talk to models.
  + models/: Handles all the database talk. This is where you'll define how your data looks and how to interact with your PostgreSQL database (e.g., raw SQL queries, or using an ORM like Sequelize/Knex).
  + routes/: Defines the different URLs (endpoints) your API will expose and maps them to the right controller functions.
  + middleware/: Functions that run *before* or *after* your main route handlers. Perfect for things like checking if a user is logged in (authMiddleware), logging requests, or catching errors.
  + utils/: General helper functions that don't fit neatly into other categories but are used across your app.
  + config/: Configuration files (e.g., database settings, environment-specific configs). Your database connection details would go here.
  + db/: For database-specific scripts like migrations (to manage changes to your database schema over time) and seeders (to populate initial data).
* tests/: Where all your backend tests live. Keep them organized by the type of code they're testing.
* .env: Crucial for storing sensitive information (like database passwords, API keys) and configuration that changes between environments. **Never commit this file to version control!**
* package.json: Backend project metadata and dependencies.
* server.js **(or** app.js**)**: The main file that starts your Express server, sets up middleware, and imports your routes.

**Frontend (**frontend/**)**

* public/: Files placed here are served directly by the web server without being processed by Vite. Good for index.html, favicon.ico, etc.
* src/: Contains all your React source code.
  + components/: This is the heart of your UI. Break down your interface into small, reusable pieces (e.g., a Button, a MenuItemCard, a CartSummary). You can further organize them into subfolders like common, menu, cart.
  + pages/: Components that represent entire views or "pages" of your application. They often compose multiple smaller components.
  + assets/: For static media files like images, custom fonts, or SVG icons.
  + hooks/: Custom React Hooks to extract and reuse stateful logic across components.
  + contexts/: If you're using React's Context API for global state (like user authentication status or cart items), the context providers and consumers live here.
  + utils/: General-purpose JavaScript functions that are useful across your frontend, but don't directly relate to UI or API calls.
  + services/: Contains functions specifically for making API calls to your backend. This keeps your API logic separate from your UI components.
  + styles/: Where your main CSS files and Tailwind CSS configuration (tailwind.config.js) reside. index.css typically imports Tailwind's base styles.
  + main.jsx **(or** .tsx**)**: The entry point for your React application, where you render your App component into the DOM.
* tests/: Where your frontend tests (unit tests for components, potentially end-to-end tests with tools like Cypress or Playwright) are stored.
* .env: Similar to the backend, but for frontend environment variables (e.g., the URL of your backend API).
* package.json: Frontend project metadata and dependencies.
* vite.config.js: Configuration file for Vite, your build tool.
* tailwind.config.js: Configuration for Tailwind CSS, where you can customize your design system.

This structure provides a clear separation of concerns, making it easier for new developers to understand the codebase and for existing developers to find and modify code without breaking other parts of the application.

## Functionalities

All app screens must be in Brazilian Portuguese.

You're looking for a more detailed breakdown of the functionalities required to develop the 'A la carte' app! I can certainly provide that, drawing directly from the "Design Document for 'A la carte' SaaS Application" Canvas.

Here's a comprehensive list of the functionalities needed, broken down by their respective areas

### 1. Restaurant Onboarding and Subscription Management

* **Self-Service Signup:**
  + User interface for new restaurants to create an account.
  + Secure collection of basic restaurant information (name, contact, address).
  + Initial setup of restaurant branding (logo, images).
  + Option for quick initial menu setup or import.
* **Multi-Location Registration:**
  + Ability to add and configure multiple physical locations under a single restaurant account.
  + Capture location-specific details (address, phone, operating hours).
* **Subscription Plan Selection:**
  + Presentation of different subscription tiers/features.
  + Mechanism for selecting a plan during or after signup.
* **Payment Integration:**
  + Secure collection of credit card information (PCI-compliant gateway integration like Stripe/Braintree).
  + Handling of recurring billing cycles.
* **Feature Entitlement Management:**
  + Backend logic to track and enforce which features are available to each restaurant based on their subscription.
  + UI in the admin portal for restaurants to enable/disable purchased features (e.g., Seat-Based Ordering).

### 2. Deployment and Access Architecture

* **Custom Domain Mapping:**
  + Backend system to manage DNS records and map custom domains (e.g., yourrestaurant.com) to the 'A la carte' application.
* **Subdomain Management:**
  + Support for [restaurant-name].alacarteapp.com and [location-slug].[restaurant-name].alacarteapp.com subdomains.
* **SSL Certificate Provisioning:**
  + Automated provisioning and renewal of SSL certificates (including wildcard certificates for subdomains) to ensure HTTPS.
* **URL Routing for Portals:**
  + Frontend and backend routing logic to direct users to the correct portal (/admin, /waiter, /menu, /kds/[area]) based on the URL path and subdomain.
* **Location-Aware Access:**
  + System to identify the specific restaurant and location from the URL or QR code scan.

### 3. Centralized Data Management System

* **PostgreSQL Database Design:**
  + Schema design to support multi-tenancy and multi-location data, ensuring data isolation and efficient querying.
  + Tables for restaurants, locations, users (staff, customers), menu categories, menu items, orders, order items, modifiers, images, dietary tags, promotions, loyalty data, feedback, etc.
* **Menu Item Management (CRUD):**
  + Admin portal UI for creating, reading, updating, and deleting menu items.
  + Fields for name, description, price, availability, images, dietary tags.
* **Menu Category Management (CRUD):**
  + Admin portal UI for creating, organizing, and assigning menu categories/sections.
* **Image Management:**
  + Functionality to upload multiple images per menu item (integration with cloud storage like Google Cloud Storage/AWS S3).
  + Image display and management within the admin portal.
* **Dietary Tagging:**
  + Admin portal UI to assign predefined or custom dietary tags to menu items.
* **Global vs. Local Menu Configuration:**
  + Logic to differentiate and manage menu items that are global across all locations versus those specific to certain locations.
* **Restaurant Information Management (CRUD):**
  + Admin portal UI for managing restaurant details like hours, contact info, and branding, on a per-location basis.

### 4. Customer-Facing Mobile Ordering Interface (PWA)

* **QR Code Scanning Integration:**
  + Frontend logic to interpret QR code data (table number, location ID) and load the appropriate menu.
* **Dynamic Menu Display:**
  + UI to display menu items organized by categories.
  + Filtering/highlighting options based on dietary restrictions.
  + Carousel/gallery for multiple item images.
* **Order Customization:**
  + Text input for "notes" or "observations" per item.
  + Predefined modifier selection (e.g., steak doneness, pizza toppings).
  + Structured combo meal building interface with required/optional selections.
* **Shopping Cart Management:**
  + Add/remove items, adjust quantities.
  + Display of item price and subtotal.
* **Seat-Based Ordering (Optional):**
  + Prompt for seat number or name upon QR code scan.
  + Ability to tag individual cart items with a seat identifier.
* **Customer Accounts & Order History (for website orders):**
  + User registration and login (email/password, potentially social logins).
  + Secure storage of customer addresses, phone numbers, dietary preferences.
  + Secure storage of tokenized payment methods.
  + Display of past order history with reorder functionality.
* **Self-Service Payment:**
  + UI for viewing the bill (items, taxes, tips).
  + Option to split the bill by specific items, fixed amounts, or by seat.
  + Integration with payment gateway for credit card processing.
  + Handling of flexible partial payments and early departure scenarios.
* **Dietary Restriction Declaration:**
  + Optional pre-order prompt for customers to declare allergies/restrictions.
  + Dynamic menu adjustments (filtering, highlighting, warnings) based on declarations.
  + Confirmation alerts for conflicting items in the cart.
* **Customer Loyalty Program (Optional):**
  + Display of loyalty points and reward tiers.
  + Redemption of rewards.
* **Customer Notifications:**
  + Real-time in-app, push, or SMS notifications for order confirmations, status updates, and promotions.
* **Customer Feedback & Review Management:**
  + UI for star ratings and comments for overall experience.
  + UI for **feedback per individual menu item**.
  + Integration with external review sites (e.g., Google, Yelp).

### 5. Admin Portal (Restaurant Management Dashboard)

* **Staff Authentication & RBAC:**
  + Secure login for different staff roles (manager, waiter, chef, cashier).
  + Role-based access control to specific features and locations.
* **Multi-Location Dashboard:**
  + Centralized view and management for all registered restaurant locations.
* **Order Management System:**
  + Real-time display of incoming orders (table number, items, customizations, dietary notes, location).
  + Ability to update order status (pending, preparing, ready).
  + Clear visibility of seat-specific item details.
  + Tools for managing and tracking split bills and remaining balances.
* **Waiter Management & Ordering:**
  + Waiter login and selection of table.
  + Interface for waiters to take orders and tag items with seat numbers/guest names.
* **Promotions & Discounts Manager:**
  + UI for creating and managing various discount types (percentage, fixed, BOGO, happy hour, combos).
  + Configuration of application rules (items, minimums, time, promo codes).
  + Tracking of promo usage.
* **Loyalty Program Setup:**
  + Admin UI for configuring point earning rules, reward tiers, and personalized offers.
  + Tracking customer loyalty status.
* **Order Prioritization:**
  + Ability for staff to manually prioritize orders or items with reasons.
  + Visual highlighting of prioritized orders on KDS/OMS.
* **Manager Notifications:**
  + Real-time alerts for payment failures, system errors, overdue orders, low stock (future).
* **Analytics & Reporting:**
  + Dashboards and reports for sales data, popular items, operational efficiency.
  + Filtering data by location.
* **Price Display Control:**
  + Admin setting to control whether prices are always visible or only via QR code access, configurable per location.

### 6. Distributed Kitchen Display System (KDS) & Printer Integration

* **Order Routing Logic:**
  + Backend logic to intelligently route order items to the correct KDS screens and thermal printers based on item category and location.
* **KDS Interface:**
  + Real-time display of orders for kitchen staff.
  + Visual and audible notifications for new orders, changes, and delays.
  + Clear display of customer notes, modifiers, combo components, and prominent dietary/allergy alerts.
  + Ability for staff to mark items/orders as prepared/ready.
* **Networked Printing:**
  + Integration with thermal printers using protocols like TCP/IP sockets or HTTP POST.
  + Print queue management for reliable delivery.
  + Printouts including seat-specific item tags for accurate delivery.
* **Order Splitting & Allocation:**
  + Logic to split orders by prep station, seat, and combo components.

This detailed breakdown covers the essential functionalities required for 'A la carte', encompassing everything from initial setup to day-to-day operations and customer experience.

## Database

That's a great name! "A la cart app" perfectly fits the restaurant and ordering theme.

Yes, there are indeed free PostgreSQL services available for your development phase, similar to MongoDB Atlas's free tier. These are excellent for building and testing your "A la cart app" without initial costs.

Here are some of the most popular and generous free PostgreSQL options:

1. **Supabase:**
   * **What it is:** Supabase positions itself as an open-source Firebase alternative, providing a full backend-as-a-service with a PostgreSQL database at its core. It includes features like authentication, real-time subscriptions, and storage.
   * **Free Tier Limits:**
     + **Database Size:** 500 MB
     + **File Storage:** 1 GB
     + **Bandwidth:** 5 GB per month
     + **Monthly Active Users:** 50,000 (for authentication)
     + **API Requests:** Unlimited
     + **Projects:** 2 active free projects
     + **Note:** Free projects are paused after 1 week of inactivity (you can easily unpause them).
   * **Why it's good for you:** Supabase is a very strong contender because it gives you a full PostgreSQL database, along with other backend services that you might need for your "A la cart app" (like authentication for the admin portal and file storage for menu images). Its real-time capabilities are also a big plus.
2. **Neon:**
   * **What it is:** Neon provides a serverless PostgreSQL database designed for modern applications. Its key features include "database branching" (like Git branches for your database) and "scale to zero" (meaning it can pause when inactive to save costs).
   * **Free Tier Limits:**
     + **Storage:** 0.5 GB (regular and archive combined)
     + **Compute Hours:** 191.9 compute hours per month (enough to run a primary 0.25 CU compute 24/7)
     + **Data Transfer (Egress):** 5 GB per month
     + **Projects:** Up to 10 projects, with 10 branches per project.
   * **Why it's good for you:** Neon's serverless architecture means you only pay for what you use, and its branching feature can be incredibly useful for development, allowing you to create isolated environments for different features without affecting your main database.
3. **Aiven for PostgreSQL:**
   * **What it is:** Aiven offers managed open-source data technologies, including PostgreSQL. Their free plan provides a dedicated virtual machine.
   * **Free Tier Limits:**
     + **Storage:** 1 GB
     + **CPU:** 1 CPU per VM
     + **RAM:** 1 GB per VM
     + **Dedicated VM:** You get an exclusive virtual machine, not shared resources.
   * **Why it's good for you:** Aiven's free tier provides a dedicated instance, which can offer more consistent performance compared to shared environments. It's built on a robust platform used by enterprise customers.
4. **ElephantSQL:**
   * **What it is:** ElephantSQL provides managed PostgreSQL instances. They have a "Tiny Turtle" free plan.
   * **Free Tier Limits:**
     + **Data:** 20 MB
     + **Concurrent Connections:** 5
     + **Shared Server:** Runs on a shared high-performance server.
   * **Why it's good for you:** It's very easy to set up and great for learning PostgreSQL or for very small, simple projects. However, the 20 MB data limit might be restrictive as your "A la cart app" grows, even during development. (Note: There have been discussions about ElephantSQL's future, so it's good to keep an eye on their announcements, but for now, the free tier is still listed.)

**Recommendation for "A la cart app" development:**

For your "A la cart app," **Supabase** is likely the most comprehensive and convenient free option. It provides a solid PostgreSQL database with generous limits, plus additional features like authentication and storage that you'll probably need anyway. Neon is also an excellent choice, especially if you value its serverless nature and branching capabilities for your development workflow.

Remember to always review the most up-to-date free tier limits on their respective websites, as these can change over time.

## **Future Enhancements (Version 2.0):**

These are some cool features we're planning for the next big update!

#### Controlled Item Delivery Process (Optional):

* + **Functionality:** The system will offer an optional feature to control the delivery sequence of items within an order. This ensures items are delivered in a logical progression (e.g., drinks first, then appetizers, salads, main courses, and finally desserts).
  + **Configuration:** Restaurants will configure the desired delivery order of their custom menu sections/categories (e.g., "Drinks" before "Appetizers"). Each menu item will be classifiable into a **delivery stage** (e.g., Drinks, Appetizers, Main Course, Dessert).
  + **Kitchen/Expeditor Control & Waiter Notification:** The Kitchen Display System (KDS) will be the primary interface for kitchen staff or a dedicated expediter to mark items or entire delivery stages as "prepared" or "ready for delivery" (e.g., via tap, click, or scan). The waiter will receive an automated notification (e.g., on their mobile device or a central display) when all items for a specific table's delivery stage are ready.
  + **Automated Stage Progression:** Completing a delivery stage (e.g., all drinks are ready) can optionally trigger the preparation of the next logical item group (e.g., "now prepare the snacks/appetizers") on the KDS, streamlining kitchen workflow.
  + **Order Status Tracking (for Self-Service Customers):** Self-service customers (those who ordered via QR code or directly through the website) will have an interface to **view the real-time status of their active order(s)**. This status will reflect the progression through the configured delivery stages, with visual cues or notifications to update customers on significant status changes.
  + **Estimated Delivery Time per Course/Stage:** The system will provide estimated delivery times for each course or delivery stage. This will be based on:
    - **Backend Data & Configuration:** Default estimated preparation times set by the restaurant for each delivery stage/category.
    - **Dynamic Adjustment Factors:** Optional dynamic adjustments based on current kitchen load and historical performance.
    - **KDS Integration:** KDS will capture timestamps for stage progression.
    - **Real-time Customer Updates:** Enhanced order status view with visual progress bars/timelines and notifications.

#### Table Management (beyond just ordering):

* + **Visual Table Layout:** The Admin Portal will provide a visual representation of the restaurant's table layout, showing occupied/available tables in real-time.
  + **Waiter Assignment:** Ability for managers to assign waiters to specific tables or sections.
  + **Table Status Management:** Tools to manage table status (e.g., "Occupied," "Available," "Needs Cleaning," "Reserved"), with real-time updates for hosts and waiters.
  + **Basic Reservation System (Optional):** A simple reservation system allowing customers to book tables and staff to manage reservations.

#### Broader Third-Party Integrations:

* + **Point of Sale (POS) Systems:** Seamless integration with existing restaurant POS systems for unified order processing, inventory management, and sales reporting.
  + **Delivery Aggregators:** Integration with popular food delivery platforms (e.g., Uber Eats, DoorDash) to consolidate orders from various channels into the 'A la carte' system.
  + **Accounting Software:** Integration with accounting platforms (e.g., QuickBooks, Xero) for automated financial data export and reporting.

#### Offline Mode / Resilience:

* + **Local Caching:** Implementation of local data caching on devices (tablets, KDS) to allow for continued order taking and processing even if the internet connection is temporarily lost.
  + **Offline Sync:** Automatic synchronization of data with the central database once the internet connection is restored, ensuring no data loss.
  + **Operational Continuity:** Design for minimal disruption to critical operations (ordering, kitchen display) during network outages.

#### Data Backup and Disaster Recovery:

* + **Automated Backups:** Implementation of automated, regular backups of all critical data (database, media files) to secure, off-site storage.
  + **Point-in-Time Recovery:** Capability for point-in-time recovery to restore data to a specific moment, minimizing data loss in case of corruption or accidental deletion.
  + **Disaster Recovery Plan:** A documented disaster recovery plan outlining procedures for restoring service and data in the event of a major system failure or catastrophic event.