Lazy Pizza Milestone #3 Requirements

This document describes the requirements for the first development milestone (Milestone #3) of the Lazy Pizza app.

The goal of this milestone is to implement the user authentication flow (phone number and verification code, option to continue without signing in, error state) and to update the Orders History screen for authenticated users.

The mockups define the app's appearance and behavior. Feel free to decide how you implement specific things (e.g., how you display a specific loading progress or how you specify error messages).

You can find the Lazy Pizza **mockups here**:

> https://www.figma.com/design/7EgItcq6KfY1Ct9H6uOitQ/Lazy-Pizza?node-id=4-6262

Milestone #3 Goal

- Add a user icon in the top bar (login/logout states).
- Implement the **Authentication screen** with verification code flow.
- Add the *"Continue without signing in"* option.
- Implement **error state** for wrong verification code.
- Update the **Orders History screen** for the authenticated user state.

🔐 Authentication

The main focus of this milestone is implementing **user authentication**. Sign-in is done via phone number with SMS code verification.

Why Authentication Matters

- Enables personalized user experience (order history, profile management, etc.).
- Protects user data.
- A standard feature of any real-world e-commerce application.

> 📌 For implementing authentication, we recommend using **Firebase Authentication**. Other solutions are also possible, but keep in mind the time constraints: building a custom backend for SMS verification can require significantly more effort and resources.

Comparison of Approaches: Firebase vs Custom Backend

In modern mobile app development, there are two common approaches to organizing the backend: using ready-made solutions such as **Firebase** or building a **custom backend**. Both options have their pros and cons, which should be considered when choosing an architecture.

Firebase

✅ Great to quickly get started and prototype

✅ Easy to scale

❗ Can become incredibly expensive when scaled (you pay for users, regardless of how profitable your app is - and small bugs causing plenty of requests can create cost spikes)

❗ Limited control and querying potential

Custom Backend

✅ Full control over everything from DB to storage to programming language

✅ Can be maintained for relatively little money in comparison to cloud providers (however, it's typical to stick to a hybrid model by having both a custom backend and some cloud providers, e.g. for image storage to get the best of both worlds)

❗ Steeper learning curve

❗ Scalability depends on how you structure it

Recommended Solution: Firebase

- We strongly recommend using **Firebase** for this project:
  - Firebase Storage for images
  - Firebase Firestore or Realtime Database for product data
- Firebase provides excellent Android integration, real-time capabilities, and is free for development purposes.

Icons

You may use Material Design icons where appropriate. If a suitable Material icon is not available, use the custom icons provided in the Figma mockups.

In Figma, any icon or image can be **exported** by selecting the element and clicking "Export" in the **right-hand panel**. In this panel, you can also choose the desired format (PNG, SVG, etc.).

Adaptive Layouts

The app must support two breakpoints:

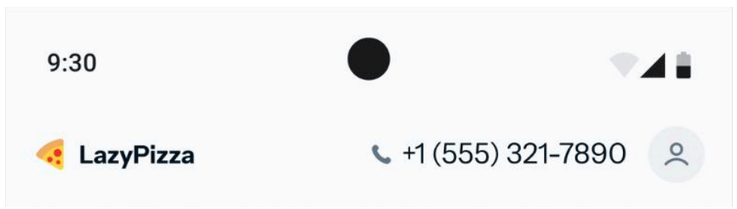
- **Up to 840 dp** → mobile layout.
- **From 840 dp and above** → wide-screen layout.

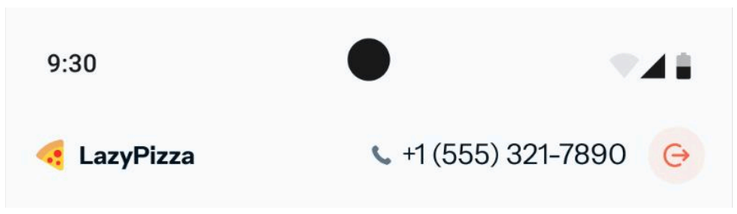
Both versions must be implemented as shown in the mockups.

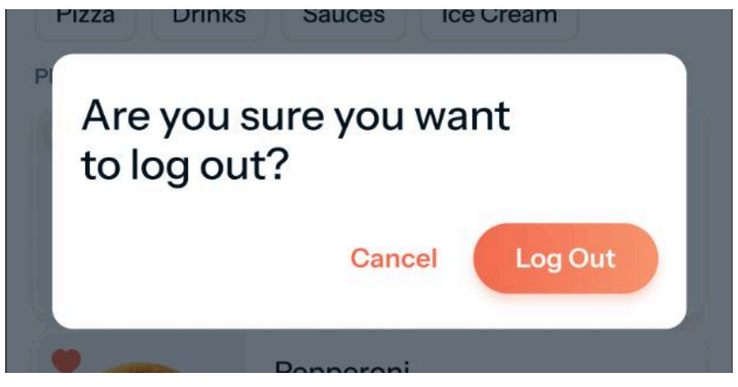
Technical Requirements

👤 User Icon in the Top Bar

- Displayed on the **main screen top bar**, aligned to the right.
- State 1 — **not signed in**:
  - Shows **a person icon** (login).
  - On tap → opens the Authentication screen.
- State 2 — **signed in**:
  - Shows **a logout icon**.
  - On tap → opens a confirmation dialog:
    - Message: “*Are you sure you want to log out?*”
    - Buttons: **Cancel / Log Out**.
    - On confirming → user is redirected to the Main screen in an unsigned state.



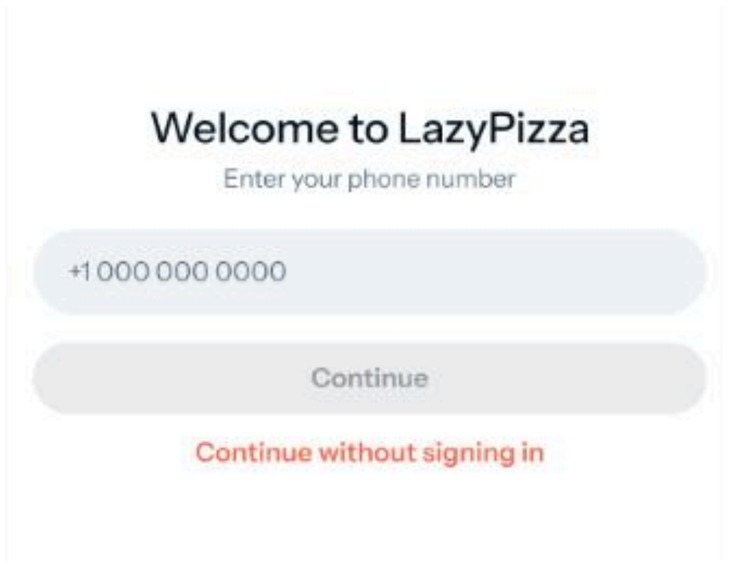


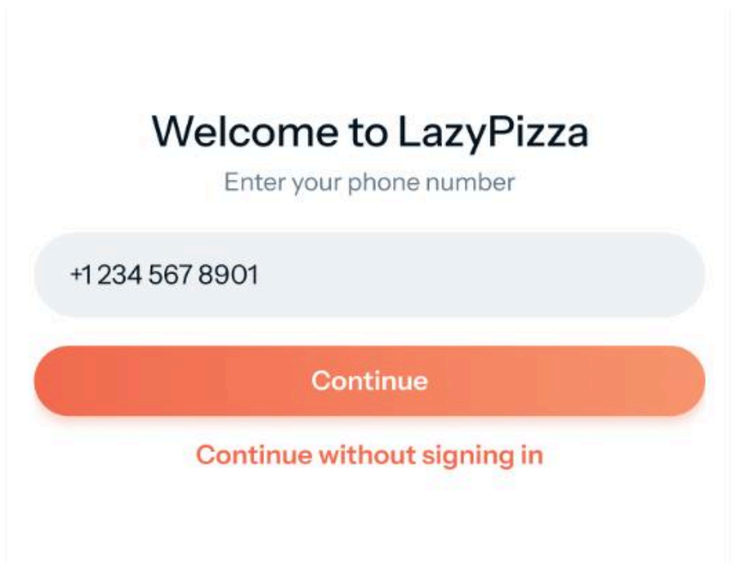


🔐 Authentication Screen

State 1 — Phone Number Input

- **Title:** *Welcome to LazyPizza*.
- **Subtitle:** *Enter your phone number*.
- **Input field:**
  - Must accept an international phone number that starts with + followed by the country code and the rest of the digits.
    - Example formats: +1 234 567 8901 (USA/Canada)
    - Placeholder (hint): displays an example number in the same format that the user must enter.
    - Only digits and the + sign are allowed.
- **Continue without signing in link:**
  - On tap → user is returned to the Main screen in an unsigned state.
- **Continue button:**
  - Must only become active when the phone number is fully entered and matches a valid international format as shown above.
  - You must adapt the validation logic to the country/region relevant for your implementation.

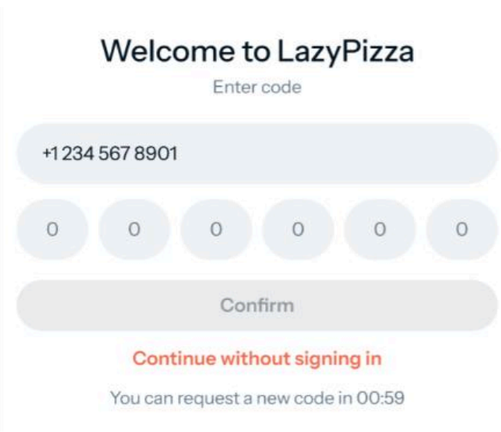




- On tap (if active):
  - The app must initiate the verification process by sending the phone number to an authentication service.
  - The service sends an SMS with a verification code to the entered number.
  - By default, the user is redirected to **State 2 (Code Input)**, where they must manually enter the code received via SMS.
  - Optionally, you may implement **automatic code retrieval** (if supported on the device and network):
    - The service can capture the incoming SMS, extract the verification code, and auto-fill it into the input fields.
- If auto-fill succeeds, the user is signed in immediately without manual input and redirected to the **signed-in state**.

> 💡 If you choose to use **Firebase Authentication:**
>
> - Firebase handles sending the SMS and provides callbacks for each stage of the flow:
>   - *onCodeSent* — SMS successfully sent to the user.
>   - *onVerificationCompleted* — automatic code retrieval succeeded; user is signed in immediately.
>   - *onVerificationFailed* — verification failed (e.g., invalid number).
>   - *onCodeAutoRetrievalTimeOut* — code auto-retrieval did not succeed within the timeout; user must enter the code manually.

State 2 — Code Input



- Subtitle changes to: *Enter code*.
- The phone number entered previously is displayed above the input fields.
- Block for entering 6 digits: **six rounded rectangular** input boxes.

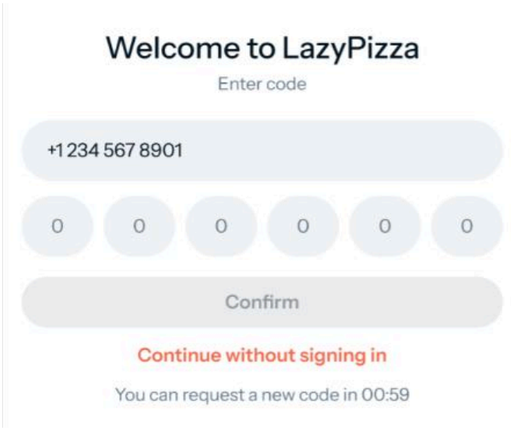
Resend flow:

- Below the input block, centered, show the message: *“You can request a new code in 00:59”*
- A **countdown timer (60 seconds)** is displayed inside this message.
- When the timer reaches 0 → replace the message with a clickable text link: *“Resend code”*
- On tap:
  - The app must request a new verification code from the chosen authentication service.
  - Reset and restart the countdown timer (message with countdown appears again).

> 💡 If you use **Firebase Authentication:**
>
> - Call the resend function with the *forceResendingToken* provided by Firebase.
> - Firebase will then send a new SMS verification code to the same phone number.

Automatic code retrieval (if supported):

- If the verification service automatically receives the incoming SMS:
  - The entered code fields are auto-filled.
  - The app proceeds with authentication immediately.
  - Optionally, a loading indicator (e.g. circular progress) may appear during the sign-in process.



- If automatic retrieval fails, the timer continues normally until it reaches 0, then Resend code becomes available.

Manual code entry (OTP-style, if automatic retrieval is unavailable):

- By default, each box shows a gray placeholder digit 0.
- When the user taps on the **first box**:
  - Placeholder 0 is replaced by a blinking cursor.
  - Only a single digit (0–9) can be entered.
- After typing a digit:
  - The box is filled, and focus **automatically moves** to the next box.
- When deleting (Backspace):
  - The current digit is cleared.
  - Focus **automatically returns** to the previous box.
- Focus flow:
  - User cannot skip boxes or tap directly into later fields.
  - Input always starts from the first box and continues sequentially.
- Additional UX detail:
  - If a user re-focuses on a filled box, the digit is automatically selected so it can be quickly replaced.

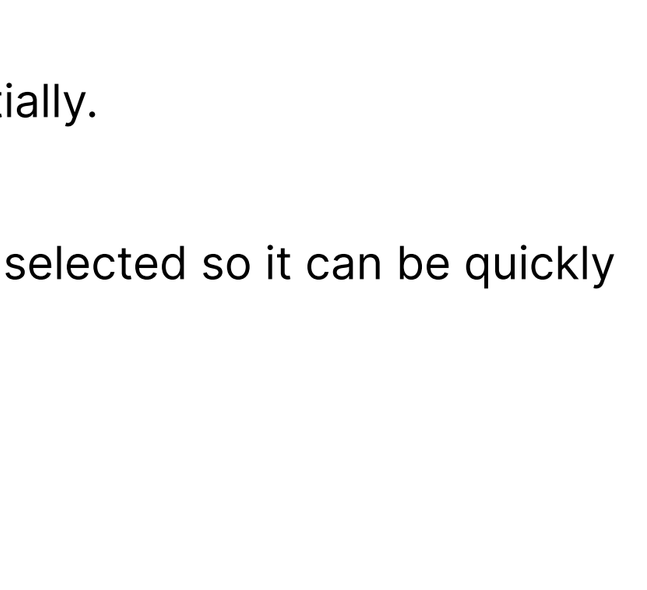
Confirm button:

- Inactive until all 6 boxes are filled.
- Becomes active once all 6 digits are entered.

State 3 — Successful Confirmation

If the code is correct:

- The user is marked as authenticated.
- Redirect to the **Main screen** (signed-in state).
- The top bar now displays the **logout icon**.



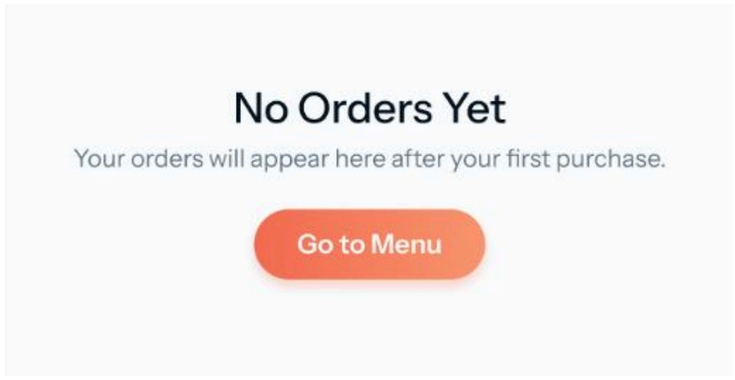
State 4 — Wrong Code

If the entered code is incorrect:

- Input fields are highlighted in orange.
- Error message displayed below: *Incorrect code. Please try again.*

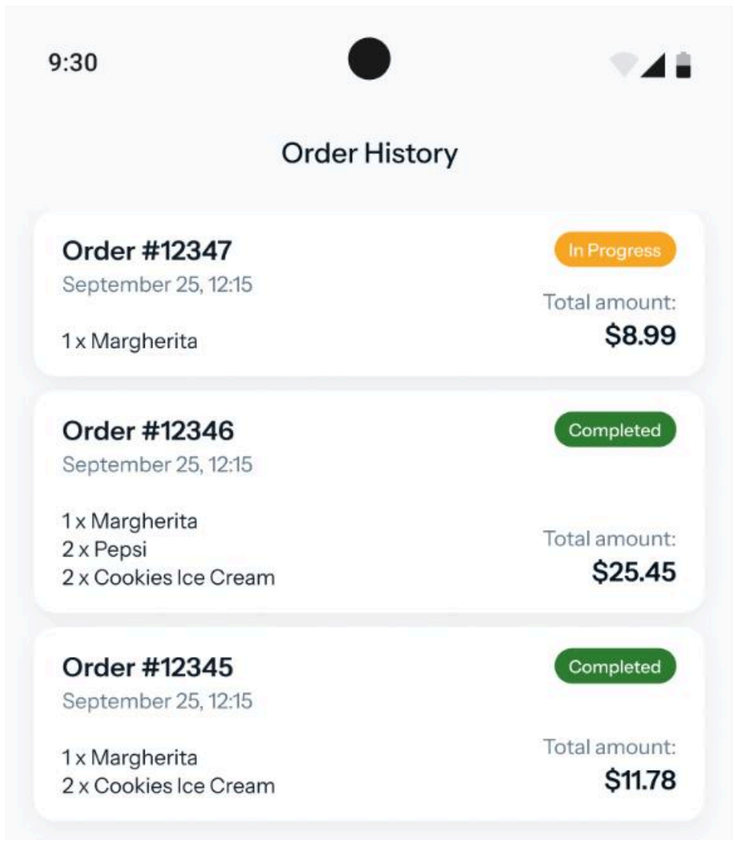
📜 Orders History Screen

Empty State



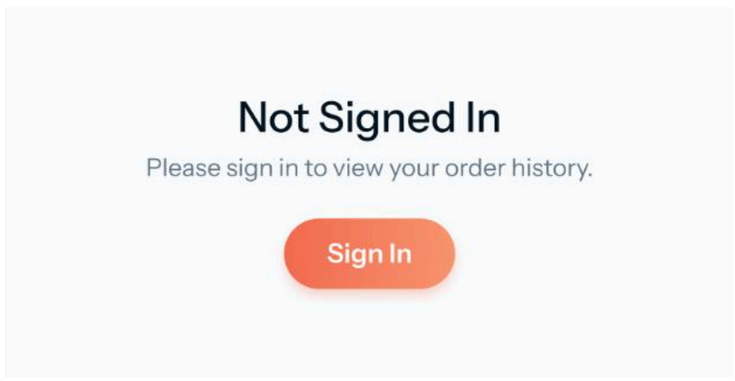
- Title: *“No Orders Yet”*.
- Subtitle: *“Your orders will appear here after your first purchase.”*
- Button: **Go to Menu** → redirects to the main screen, allowing the user to place their first order.

Filled State



- Orders are displayed as a vertical list of cards (most recent first).
- Each order card includes:
  - Order number (e.g., #12345).
  - Date and time of the order.
  - Short summary of items (e.g., Margherita, Pepsi, Garlic Sauce...).
  - Total price of the order.
  - Small status label (Completed / Canceled).

Unauthorized State



If the user is not signed in and attempts to access Orders History:

- **Title:** *“Not signed in”* — centered, with 120 dp padding from the top bar.
- **Subtitle:** *“Please sign in to view your order history.”* — centered, displayed below the title.
- **Sign In button:** centered; on tap → navigates to the Authentication screen.

🧺 Cart Persistence and Session Behavior

This section defines how the cart behaves in both guest and authenticated user modes. The goal is to provide a simple, predictable, and secure user experience when switching between sessions.

Local Storage

- The cart is stored **locally** — for example, using Room or DataStore.
- Cart data remains available after the app restarts.
- A **single local cart** is used in guest mode and is not tied to any account.
- For an authenticated user, a separate cart is created and exists only during the active session.

Guest Mode

- In guest mode, the user can browse the menu, add items, and modify the cart contents.
- The guest cart is stored locally and **persists between app restarts.**
- It is cleared only when the user:
  a. Places an order or manually removes all items.
  b. Signs in to an account — in this case, the guest cart is transferred to the user account.

Guest → Sign In

- If a user adds items without signing in and then logs in, the current guest cart is **transferred** to their account.
- This ensures that the user retains the items they selected before signing in.

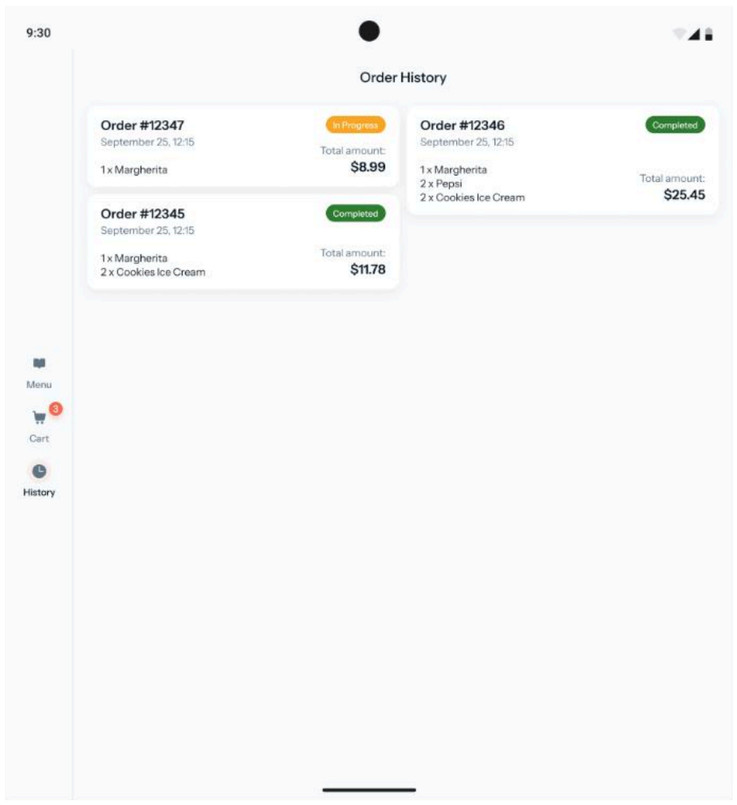
Authenticated User → Sign Out

- When the user signs out, the cart is **completely cleared**, since it is considered part of the active session.
- This behavior is intuitive: signing out ends all temporary actions, including the cart state.
- After signing out, the app returns to **guest mode** with an empty cart.

> 💡 In this version of the app, the cart uses **local storage** (e.g., Room or DataStore).
>
> Remote cart storage or synchronization can provide benefits, such as restoring carts across devices or after reinstallation, but this adds significant implementation complexity and is **outside the core scope of requirements**.
>
> Developers who wish to extend functionality may implement such logic as an optional improvement.

📏 Screen Adaptation

- **Up to 840 dp:**
  - Mobile layout (vertical stacking of elements).
- **From 840 dp and above:**
  - Authentication screen elements are centered with clean spacing.
  - Orders History screen → **two-column staggered grid**; each card adapts its height to content. Use *LazyVerticalStaggeredGrid* if convenient.



🔗 Useful Links for This Challenge

- Create a Splash Screen
- UX With Material3
- Full Guide to Material3 Theming
- The Full Jetpack Compose Responsive UI Crash Course
- How to EASILY Implement a Lazy Staggered Grid
- How to Create a Lazy Column With Categories
- How to Save & Restore the Scroll Position of a LazyColumn Persistently
- Stateful vs. Stateless Composables
- State Hoisting in Compose
- Managing State in Jetpack Compose (Codelab)
- Add shadows in Compose

📦 Application Items List

Pizza Menu

- **Margherita** — $8.99

  Ingredients: Tomato sauce, mozzarella, fresh basil, olive oil
- **Pepperoni** — $9.99

  Ingredients: Tomato sauce, mozzarella, pepperoni
- **Hawaiian** — $10.49

  Ingredients: Tomato sauce, mozzarella, ham, pineapple
- **BBQ Chicken** — $11.49

  Ingredients: BBQ sauce, mozzarella, grilled chicken, onion, corn
- **Four Cheese** — $11.99

  Ingredients: Mozzarella, gorgonzola, parmesan, ricotta
- **Veggie Delight** — $9.79

  Ingredients: Tomato sauce, mozzarella, mushrooms, olives, bell pepper, onion, corn
- **Meat Lovers** — $12.49

  Ingredients: Tomato sauce, mozzarella, pepperoni, ham, bacon, sausage
- **Spicy Inferno** — $11.29

  Ingredients: Tomato sauce, mozzarella, spicy salami, jalapeños, red chili pepper, garlic
- **Seafood Special** — $13.99

  Ingredients: Tomato sauce, mozzarella, shrimp, mussels, squid, parsley
- **Truffle Mushroom** — $12.99

  Ingredients: Cream sauce, mozzarella, mushrooms, truffle oil, parmesan

Drinks

1. Mineral Water — $1.49
2. 7-Up — $1.89
3. Pepsi— $1.99
4. Orange Juice — $2.49
5. Apple Juice — $2.29
6. Iced Tea (Lemon) — $2.19

Sauces

1. Garlic Sauce — $0.59
2. BBQ Sauce — $0.59
3. Cheese Sauce — $0.89
4. Spicy Chili Sauce — $0.59

Ice Cream

1. Vanilla Ice Cream — $2.49
2. Chocolate Ice Cream — $2.49
3. Strawberry Ice Cream — $2.49
4. Cookies Ice Cream — $2.79
5. Pistachio Ice Cream — $2.99
6. Mango Sorbet — $2.69

Extra Toppings

- Bacon — $1.00
- Extra Cheese — $1.00
- Corn — $0.50
- Tomato — $0.50
- Olives — $0.50
- Pepperoni — $1.00
- Mushrooms — $0.50
- Basil — $0.50
- Pineapple — $1.00
- Onion — $0.50
- Chili Peppers — $0.50
- Spinach — $0.50