Software Design Document

PIDDZ Pizza Delivery Application

Prepared by:

Group 4 - PIDDZ Team

Ishrak Mulla, Prachish Pandey, Daniel Araujo,
Zeyu Zhang, Djoulie Saint Louis

Table of Contents

1. Introduction

1.1 Purpose

This Software Design Document describes the architecture and detailed design of the PIDDZ Pizza Ordering System. The document provides comprehensive information about the system structure, component interactions, data models, and design decisions that guide the implementation. This document serves as a technical blueprint for developers working on the system and ensures consistency across the development team.

1.2 Scope

The PIDDZ Pizza Ordering System is a web-based application designed to facilitate online pizza ordering for customers and order management for business owners. This system consists of customer features such as menu browsing, custom pizza building, shopping cart management, and order checkout. Additionally, the system includes a business dashboard that enables restaurant staff to view incoming orders, update order status, and manage the fulfillment process. The application is built using JavaScript, HTML, and CSS, with Firebase providing backend data storage and real-time synchronization capabilities.

1.3 Document Conventions

Throughout this document, technical terms and component names are used consistently to maintain clarity. File names and code references appear in their exact case sensitive forms as they exist in the codebase. The document follows a systemic approach, beginning with high level architectural concepts and progressively diving into detailed component designs. Diagrams accompany textual descriptions to provide visual representations of system structure and data flow.

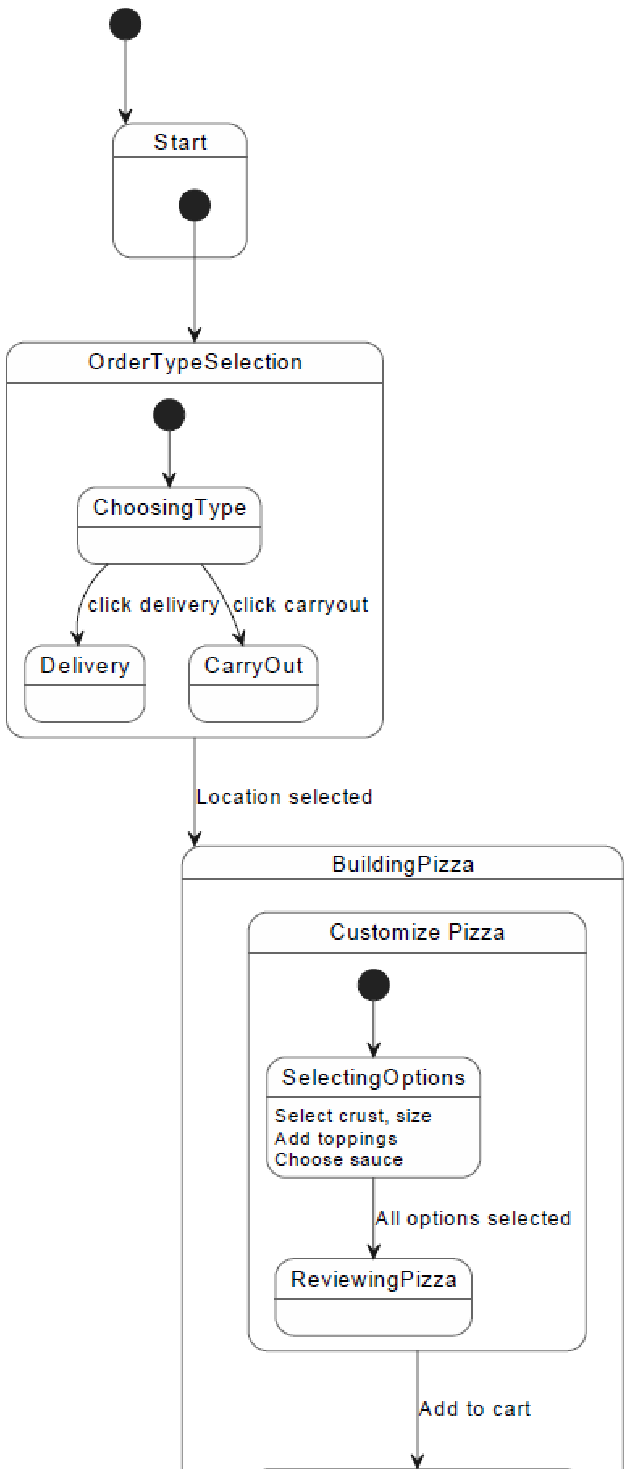
2. Design Diagrams

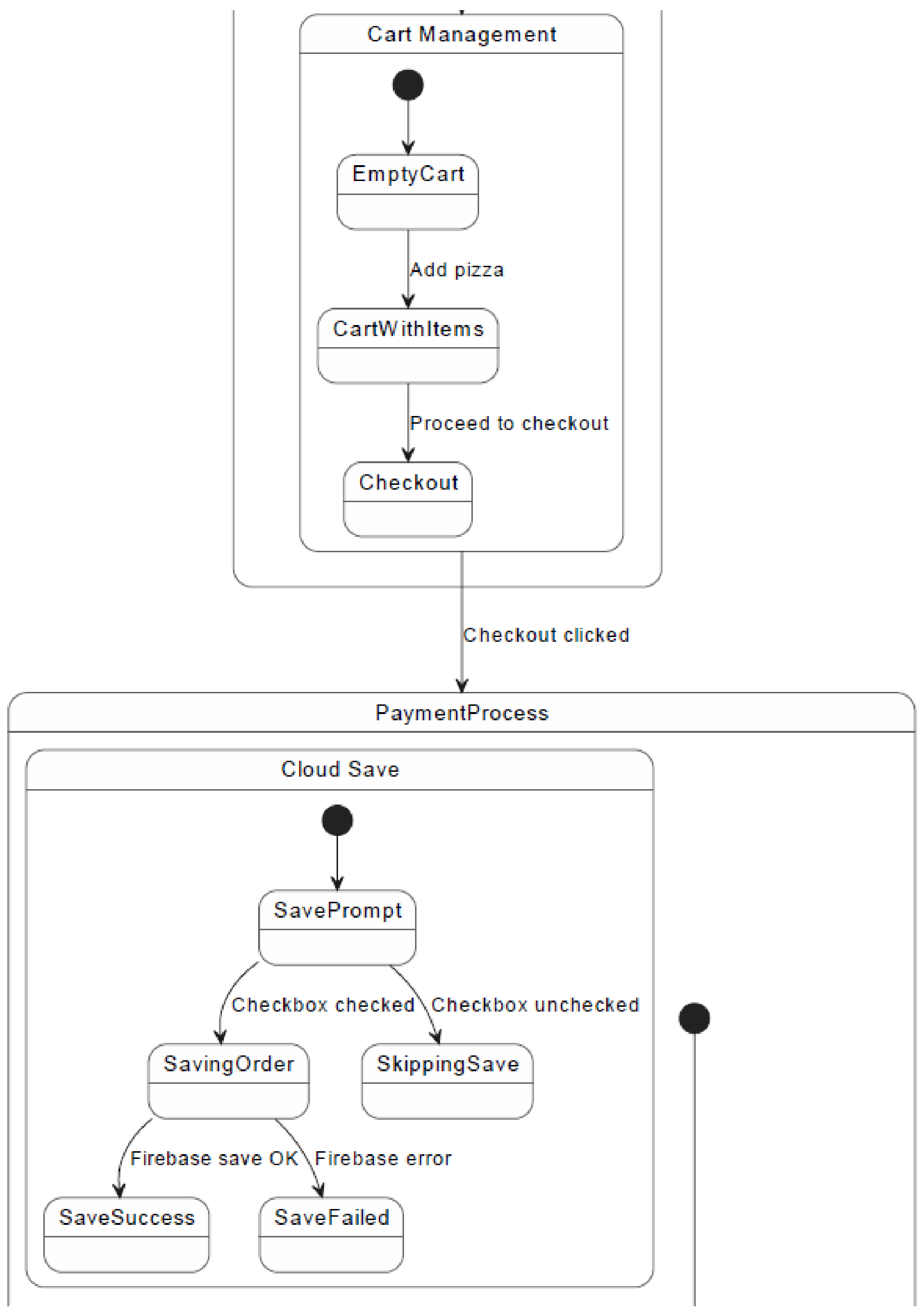
This section presents the key design diagrams that illustrate the architecture, behavior, and structure of the PIDDZ Pizza Ordering System. These diagrams provide visual representations that complement the detailed textual descriptions found throughout this document.

2.1 State Chart Diagram

The state chart diagram illustrates the various states and transitions within the pizza ordering system. It shows how the system moves through different states from initial order type selection through pizza customization, cart management, and final payment processing, including the optional cloud save functionality for orders.

The whole image will be attached separately for clarity.




Cart Management
EmptyCart
Add pizza
CartWithItems
Proceed to checkout
Checkout
Checkout clicked
PaymentProcess
Cloud Save
SavePrompt
Checkbox checked
Checkbox unchecked
SavingOrder
SkippingSave
Firebase save OK
Firebase error
SaveSuccess
SaveFailed

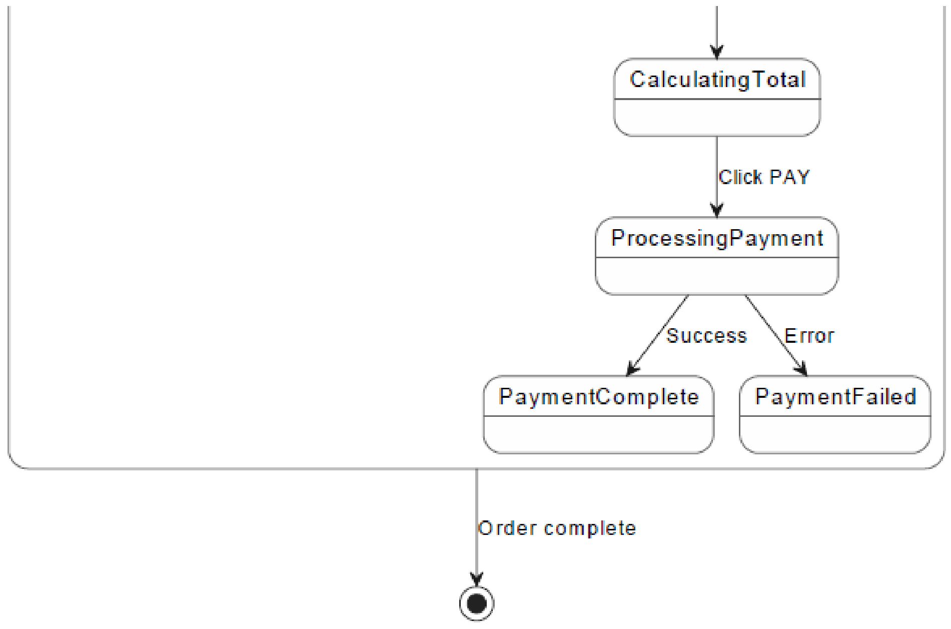


Figure 2.1: State Chart Diagram

2.2 Sequence Diagram

The sequence diagram depicts the temporal flow of interactions between the user and various system components throughout the complete ordering process. It shows the detailed message exchanges between components including the index page, menu page, pizza builder, storage mechanisms, checkout module, and Firebase helper, demonstrating how data flows through the system from initial selection to final order placement.

The whole image will be attached separately for clarity.

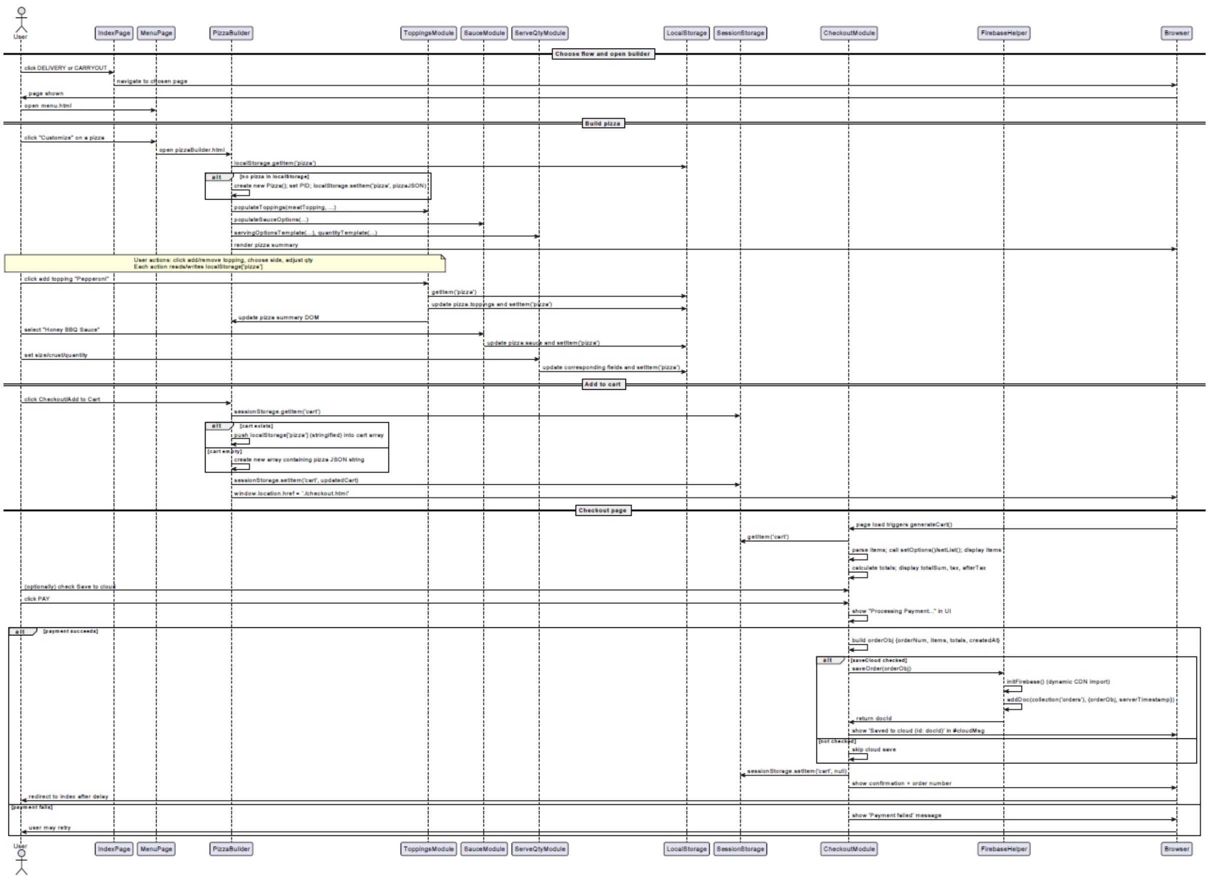

User
IndexPage
MenuPage
PizzaBuilder
ToppingsModule
SauceModule
ServeQtyModule
LocalStorage
SessionStorage
CheckoutModule
FirebaseHelper
Browser
Choose flow and open builder
click DELIVERY or CARRYOUT
navigate to chosen page
page shown
open menu.html
Build pizza
click "Customize" on a pizza
open pizzaBuilder.html
localStorage.getItem('pizza')
alt
[no pizza in localStorage]
create new Pizza(), set PID; localStorage.setItem('pizza', pizzaJSON)
populateToppings(meatTopping, ...)
populateSauceOptions(...)
servingOptionsTemplate(...), quantityTemplate(...)
render pizza summary
User actions: click add/remove topping, choose side, adjust qty
Each action reads/writes localStorage['pizza']
click add topping "Pepperoni"
getItem('pizza')
update pizza.toppings and setItem('pizza')
update pizza summary DOM
select "Honey BBQ Sauce"
update pizza.sauce and setItem('pizza')
set size/crust/quantity
update corresponding fields and setItem('pizza')
Add to cart
click Checkout/Add to Cart
sessionStorage.getItem('cart')
alt
[cart exists]
push localStorage['pizza'] (stringified) into cart array
[cart empty]
create new array containing pizza JSON string
sessionStorage.setItem('cart', updatedCart)
window.location.href = './checkout.html'
Checkout page
page load triggers generateCart()
getItem('cart')
parse items; call setOptions()/setList(); display items
calculate totals; display totalSum, tax, afterTax
(optionally) check Save to cloud
click PAY
show "Processing Payment..." in UI
alt
[payment succeeds]
build orderObj {orderNum, items, totals, createdAt}
alt
[saveCloud checked]
saveOrder(orderObj)
initFirebase() (dynamic CDN import)
addDoc(collection('orders'), {orderObj, serverTimestamp})
return docId
show 'Saved to cloud (id: docId)' in #cloudMsg
[not checked]
skip cloud save
sessionStorage.setItem('cart', null)
show confirmation + order number
redirect to index after delay
[payment fails]
show 'Payment failed' message
user may retry

Figure 2.2: Sequence Chart Diagram

2.3 Class Diagram

The class diagram presents the structural organization of the system, showing the Pizza model class along with all page modules and utility modules. It illustrates the relationships between components, their attributes, methods, and dependencies, including how different modules interact with local storage and session storage to maintain application state throughout the user session.

The whole image will be attached separately for clarity.

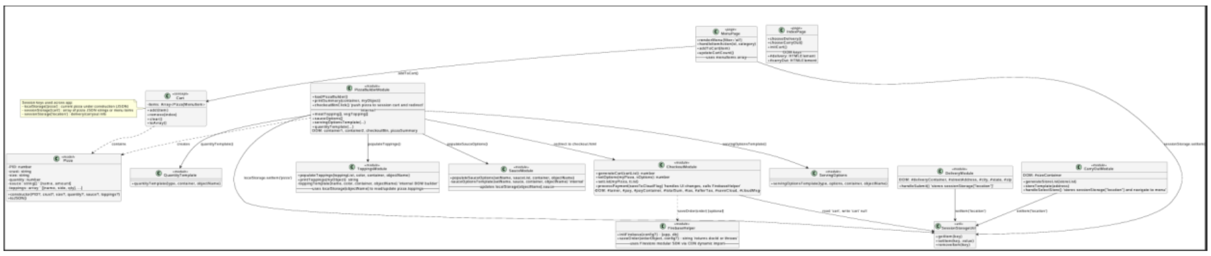


Figure 2.3: Class Diagram

3. System Overview

3.1 System Description

The PIDDZ Pizza Ordering System represents a comprehensive solution for online pizza delivery operations. The system operates through a client-side web application that provides distinct experiences for customers and business users. Customers interact with an interface that guides them through the ordering process, from initial order type selection through menu browsing, pizza customization, cart management, and final checkout. Business users access a separate dashboard interface that displays a real time order information and provides controls for managing order fulfillment. The system architecture separates concerns effectively, with presentation logic, business logic, and data persistence each handled by dedicated components.

3.2 System Context

The system operates within a standard web browser environment and communicates with Firebase cloud services for data persistence and real time synchronization. Users access the application through common web browsers including Chrome, Firefox, Safari, and Edge. The application requires an internet connection to synchronize order data with Firebase, though local storage mechanisms provide temporary data persistence during active sessions. The system integrates with Firebase Realtime Database, which handles all persistent data storage and enables real time updates across multiple connected clients. This cloud-based approach eliminates the need for traditional server infrastructure while maintaining robust data synchronization capabilities.

3.3 Key Features

The system implements several core features that define its functionality. The order type selection feature allows customers to choose between delivery and carryout options at the start of their ordering session. The menu browsing system presents available menu items organized by category, with filtering capabilities that enable customers to view specific item types such as pizzas, sides, drinks, and desserts. The pizza builder provides an interactive interface where customers can customize their pizzas by selecting size, crust type, sauce, and toppings with real-time price calculation.

The shopping cart functionality maintains customer selections throughout their session, displaying item counts and enabling navigation to checkout. The checkout process collects customer information and presents a complete order summary with tax calculations before final submission. The business dashboard offers visibility into incoming orders, presenting comprehensive order details and enabling status updates through an intuitive interface. The system employs Firebase for persistent storage, ensuring order data remains available across sessions and enabling real time synchronization between customers and business users.

4. High-Level System Architecture

4.1 Architectural Style

The PIDDZ Pizza Ordering System employs a client side Model-View-Controller architectural pattern implemented entirely in JavaScript. This architecture separates data representation, user interface presentation, and application logic into distinct concerns. The system follows a modular design philosophy where each functional area resides in dedicated JavaScript modules that handle specific responsibilities. This separation of concerns facilitates maintenance, testing, and future enhancements while keeping the codebase organized and comprehensible.

4.2 System Layers

The architecture comprises three primary layers that work together to deliver system functionality. The **presentation layer** encompasses all HTML files and CSS stylesheets that define the user interface. This layer includes **index.html** for the landing page, menu.html for browsing available items, **pizzaBuilder.html** for pizza customization, **checkout.html** for order finalization, and **dashboard.html** for business order management. Each HTML file establishes the structural foundation for its corresponding interface, while CSS files provide visual styling and responsive design capabilities.

The **business logic layer** contains JavaScript modules that implement application functionality. Core modules include **menu.js** for rendering menu items and handling category filtering, **pizzaBuilder.js** for managing pizza customization logic, **checkout.js** for processing cart contents and calculating totals, and **dashboard.js** for displaying and managing orders in the business interface. Supporting modules such as **pizzaClass.js** define data structures, while template modules like itemTemplate.js, toppings.js, **sauceOptions.js, servingOptions.js**, and **quantityTemplate.js** provide reusable components for building dynamic interfaces.

The **data layer** manages information which is persistent through multiple storage mechanisms. Browser session storage maintains shopping cart contents during active user sessions, ensuring cart data persists as customers navigate between pages. **Browser local storage** temporarily holds pizza customization data during the building process. **Firebase Realtime Database** serves as the persistent backend, storing completed order information and enabling real-time synchronization across multiple clients. The **firebase-config.js and firebase-service.js** modules encapsulate all Firebase interactions, providing a clean interface for other components to access database functionality.

4.3 Component Interactions

Components within the system interact through well-defined patterns that maintain loose coupling while enabling necessary communication. User interactions trigger event handlers in presentation-layer components, which then invoke business logic functions to process the actions. Business logic components manipulate data structures and update storage mechanisms accordingly. When data changes occur, components refresh their visual representations to reflect the updated state. The Firebase integration enables bidirectional data flow, with the application both writing order data to the database and receiving real-time updates when order statuses change. This architecture supports independent development and testing of components while ensuring they work together cohesively in the complete system.

5. Detailed Component Design

5.1 Landing Page Component

The landing page component, implemented in **index.html and index.js**, serves as the initial entry point for customer interactions. This component presents users with two primary ordering options presented as large, clickable interface elements. The delivery option initiates an ordering flow that will collect customer address information during checkout, while the carryout option indicates the customer intends to pick up their order from the restaurant location.

The implementation uses event listeners attached to the option elements that respond to user clicks by storing the selected order type in session storage and navigating to the appropriate next page. This design ensures the order type selection that persists throughout the session and influences subsequent checkout processes. The visual presentation employs flexbox layout to center the options vertically and horizontally, creating a balanced and accessible interface that works across different screen sizes.

5.2 Menu Browsing Component

The menu browsing component consists of menu.html, menu.js, and menu.css working together to present available menu items to customers. The component maintains an array of menu item objects, each containing properties for item identification, name, category, description, and price. The **renderMenu** function generates HTML dynamically based on the current filter selection, creating a grid layout of item cards that display item information and action buttons.

Category filtering enables customers to view all items or focus on specific categories such as pizzas, sides, drinks, or desserts. The implementation uses event delegation to handle filter button clicks efficiently, updating the active filter state and re-rendering the menu display accordingly. Each menu item card includes an action button whose behavior depends on the item category. For pizza items, the button text reads "Customize" and clicking it navigates to the pizza builder interface. For other items, the button reads "Add to Cart" and clicking it directly adds the item to the shopping cart.

The cart counter badge displays prominently in the interface, showing the current number of items in the shopping cart. This counter updates dynamically as items are added, providing immediate feedback to customers about their cart contents. The badge itself is clickable, allowing customers to proceed directly to checkout when they have items in their cart. The component handles edge cases gracefully, such as displaying an appropriate message when customers attempt to view an empty cart.

5.3 Pizza Builder Component

The pizza builder component implements the custom pizza creation interface through **pizzaBuilder.html, pizzaBuilder.js**, and supporting modules. This component provides the most complex user interaction in the system, enabling customers to configure every aspect of their pizza order. The component initializes by checking for an existing pizza in local storage or creating a new Pizza object if none exists. This approach supports the scenario where customers return to customize a previously configured pizza.

The interface organizes customization options into logical sections. Size selection presents three options represented by radio buttons for small, medium, and large pizzas. Crust selection offers choices between hand-tossed, New York style, and pan crust options. Sauce selection uses a button group interface where customers choose from tomato sauce, honey BBQ sauce, garlic parmesan sauce, alfredo sauce, and ranch.

The topping selection demonstrates a double section approach with separate areas for meat toppings and vegetable toppings, each containing checkboxes that customers can select to add toppings to their pizza.

The component implements price calculation that updates as customers make selections. The base price depends on the selected size, with small pizzas starting at eight dollars and ninety-nine cents, medium pizzas at ten dollars and ninety-nine cents, and large pizzas at twelve dollars and ninety-nine cents. Additional toppings have extra charges that accumulate in the price displayed. The quantity selector enables customers to order multiple identical pizzas, with the price display reflecting the total cost for all pizzas in the quantity.

The pizza summary section provides a continuous visual representation of the customer's current selections, displaying the chosen size, crust, sauce, and toppings. This summary updates dynamically as customers modify their selections, helping them verify their choices before adding the pizza to their cart. When customers click the checkout button, the component serializes the configured Pizza object and adds it to the session storage cart array before navigating to the checkout page.

5.4 Shopping Cart and Checkout Component

The checkout component, implemented **in checkout.html and checkout.js**, handles the final stages of the ordering process. This component retrieves the shopping cart contents from session storage and processes each item to generate a detailed order summary. The component distinguishes between customized pizza items and simple menu items, applying appropriate logic to calculate costs and format displays for each type.

For each item in the cart, the component creates a visual card that displays the item's details and the calculated price. Pizza items show the size, crust type, and quantity along with a detailed list of sauce and toppings. The price calculation for pizzas considers the base size cost multiplied by the quantity ordered. The component sums all item costs to calculate a subtotal, then applies a seven percent (according to MA) tax rate to determine the final total amount. These calculations display clearly in separate rows showing food cost, tax amount, and total with tax.

The order submission process begins when customers click the pay button. The component disables the button and displays a processing message to prevent duplicate submissions. After a brief simulated processing delay, the component generates a random six-digit order number and displays a confirmation message. The component then clears the shopping cart from session storage and automatically redirects customers back to the landing page after three seconds. This flow provides clear feedback about order submission while gracefully concluding the ordering session.

5.5 Business Dashboard Component

The business dashboard component provides restaurant staff with comprehensive order management capabilities through **dashboard.html and dashboard.js.** This component connects to Firebase Realtime Database using the **listenToOrders** function, which establishes a continuous connection that receives real-time updates whenever order data changes. This architecture ensures the dashboard always displays current information without requiring manual refreshes.

The dashboard displays orders as individual cards arranged in a responsive grid layout. Each order card contains comprehensive information including a truncated order identifier, timestamp showing how long ago the order was placed, complete item details with customizations, delivery address, total cost with tax, and current status. The component formats timestamps intelligently, showing recent orders as "just now" or "X minutes ago" for easy comprehension of order recency.

Order status management implements a workflow progression from pending through preparing, ready, and finally delivered. Each status displays with distinct visual styling, and orders in transitional states show action buttons that enable staff to advance the order to its next status. When staff click a status update button, the component calls the **updateOrderStatus** function in **firebase-service.js**, which writes the new status to Firebase. This change immediately propagates to all connected dashboard clients through Firebase's real-time synchronization. The dashboard filters out delivered orders older than one hour to maintain a focused view of active orders requiring attention.

6. Data Architecture and Storage

6.1 Data Models

The system employs several key data structures that represent domain concepts and facilitate data management. The Pizza class, defined in pizzaClass.js, encapsulates all properties of a customizable pizza order. This class includes a pizza identifier property for distinguishing between multiple pizzas, crust type property defaulting to hand-tossed, size property defaulting to medium, quantity property defaulting to one, sauce property stored as an array containing sauce type and amount, and toppings property stored as an array of topping arrays where each topping array contains the topping name, distribution type, and amount.

Menu items follow a simpler structure defined as plain JavaScript objects. Each menu item object contains an id property for unique identification, name property for the item's display name, category property indicating whether the item is a pizza, side, drink, or dessert, description property providing item details, and price property storing the base cost as a numeric value. This straightforward structure accommodates both pizza items that lead to the builder and simple items that add directly to the cart.

Order objects stored in Firebase contain comprehensive information about completed customer orders. These objects include an **orderId** property containing a unique identifier generated during order submission, timestamp property recording when the order was created, status property tracking the order through its fulfillment workflow, items array containing serialized representations of all ordered items, location array containing customer address information, and **totalWithTax** property storing the final calculated order cost. This structure supports both immediate order processing and historical order tracking.

6.2 Storage Mechanisms

The system employs a storage strategy that balances immediate responsiveness with long-term persistence. Browser session storage serves as the primary mechanism for maintaining shopping cart state during active user sessions. The cart array persists in session storage as a JSON-serialized string, enabling the data to survive page navigation while automatically clearing when the browser session ends. This approach ensures customers don't lose their selections when moving between menu, builder, and checkout pages while preventing abandoned carts from persisting indefinitely.

Browser local storage temporarily holds pizza configuration data during the customization process. When customers navigate to the pizza builder, the component stores the working Pizza object in local storage, enabling the configuration to persist if customers navigate away and return. This storage clears automatically when customers complete their pizza and add it to the cart or when they start a fresh order. The distinction between session storage for the cart and local storage for the current pizza provides appropriate persistence characteristics for each use case.

Firebase Realtime Database provides persistent storage for completed orders and serves as the source of truth for order data shared across multiple clients. The firebase-service.js module implements all database interactions through dedicated functions that abstract Firebase API calls. When customers complete checkout, their order data writes to Firebase under a unique order identifier. The database structure organizes orders as a collection where each order's unique identifier serves as its database key. This approach enables efficient querying and real-time listening for changes.

6.3 Data Flow Patterns

Data flows through the system following predictable patterns that maintain consistency and enable proper synchronization. Customer interactions create data that begins in temporary client-side storage and progresses toward persistent backend storage as orders reach completion. When customers browse the menu and add items to their cart, the **menu.js** component updates the session storage cart array and increments the cart counter display. Pizza customization creates a Pizza object that resides in local storage until the customer adds it to their cart, at which point it serializes and joins the session storage cart array. During checkout, the component reads the cart contents from session storage, calculates totals, and upon order submission, packages all order information into a comprehensive order object that writes to Firebase. The Firebase write operation triggers real-time updates to all connected dashboard clients, which immediately display the new order information without requiring any refreshing action from the business users.

7. User Interface Design

7.1 Design Principles

The user interface design follows principles that prioritize clarity, accessibility, and visual hierarchy. The system employs a consistent color scheme based on green tones that convey food-related themes. Interactive elements use button styling that clearly indicates their interactive nature such as rounded corners, shadow effects, and hover state changes. The design implements responsive layouts using CSS Flexbox and Grid that automatically adapt to different screen sizes, ensuring the application functions effectively on both desktop and mobile devices. Typography choices emphasize readability with adequate font sizes and appropriate spacing between text elements. The interface provides immediate feedback for user actions through visual changes, animations, and confirmation messages that help users understand the results of their interactions.

7.2 Page-Specific Designs

The landing page design centers two large option buttons vertically and horizontally within the viewport, creating an unambiguous initial decision point. The buttons employ generous padding and clear labeling to maximize usability and accessibility. The menu page organizes items in a responsive grid that adjusts column count based on available screen width, ensuring efficient space utilization across device types. Each menu item card displays with consistent formatting including a category badge, item name, description, price, and action button. The pizza builder interface employs a single-column layout on mobile devices that stacks customization sections vertically, while larger screens present a two-column layout that positions the pizza summary alongside customization options.

The checkout page presents cart items in individual cards with rounded corners and subtle shadows that create visual separation. The page uses a three-section layout with cart items at the top, cost breakdown in the middle, and payment controls at the bottom. The dashboard interface employs a card-based layout where each order appears as a self-contained unit displaying all relevant information. Cards use color-coded status badges that enable quick visual scanning of order states. The grid layout adapts responsively, showing multiple columns on wide screens and collapsing to a single column on narrow screens.

7.3 Interactive Elements

Interactive elements throughout the system provide clear visual feedback that enhances user experience. Buttons implement hover effects that slightly scale the button and adjust background colors, creating an immediate response to mouse movement that confirms the element's interactive nature. Active filter buttons on the menu page display with distinct styling that clearly indicates the current filter selection. Form inputs in the customization interface highlight when focused, drawing attention to the current input area. The cart counter badge animates when its value changes, briefly enlarging to draw attention to the update. Order status buttons in the dashboard employ color coding that associates with their status types, with preparing buttons using orange tones, ready buttons using green tones, and delivered buttons using blue tones.

8. Technology Stack and Tools

8.1 Frontend Technologies

The application frontend builds entirely on standard web technologies without relying on frameworks or libraries for core functionality. HTML5 provides the structural foundation for all pages, using semantic elements that enhance accessibility and search engine optimization. CSS3 handles all presentation aspects including layout, styling, and responsive design. The stylesheets employ modern CSS features such as Flexbox for one-dimensional layouts, Grid for two-dimensional layouts, custom properties for maintaining consistent design tokens, and media queries for responsive breakpoints.

JavaScript implements all application logic and interactivity without depending on frameworks like React or Vue. This approach keeps the codebase straightforward and reduces complexity for developers who need to understand and maintain the system. The JavaScript modules use ES6 features including import and export statements for modular organization, arrow functions for concise function syntax, template literals for string interpolation, destructuring for clean variable assignment, and async/await for handling asynchronous operations with Firebase.

8.2 Backend Services

Firebase provides all backend functionality through its suite of cloud services. Firebase Realtime Database serves as the primary data store, offering a NoSQL document database that enables real-time data synchronization. The database architecture uses JSON-based storage where data organizes hierarchically under collection paths. Firebase Authentication would handle user authentication in future implementations requiring secure access controls. Firebase Hosting deploys the static web application files, serving HTML, CSS, and JavaScript to users through a content delivery network that ensures fast loading times across geographic regions. The Firebase SDK integrates into the application through JavaScript modules that establish database connections and provide methods for reading and writing data.

8.3 Development Tools

The development environment employs several tools that support efficient development workflows. Visual Studio Code serves as the primary code editor, offering features such as syntax highlighting, intelligent code completion, integrated debugging, and Git integration. The browser developer tools built into Chrome, Firefox, Safari, and Edge enable debugging JavaScript, inspecting HTML and CSS, monitoring network requests, and analyzing performance. Git provides version control capabilities that track code changes, facilitate collaboration among team members, and maintain code history. GitHub hosts the remote repository and provides features for issue tracking, pull request reviews, and continuous integration workflows. The Firebase Console offers a web-based interface for managing database contents, monitoring usage, and configuring service settings.

9. Design Patterns and Principles

9.1 Modularity and Separation of Concerns

The system architecture embraces modularity as a core principle, organizing functionality into distinct modules that each handle specific responsibilities. Each JavaScript file focuses on a particular aspect of the application, such as menu rendering, pizza building, or checkout processing. This separation enables developers to work on individual features without affecting unrelated parts of the system. The modular structure also facilitates testing, as developers can verify individual modules in isolation before integrating them into the complete application. Supporting template modules extract reusable interface patterns into dedicated files that multiple components can utilize, promoting code reuse and maintaining consistency across the application.

9.2 Data Encapsulation

The Pizza class demonstrates the object-oriented principle of encapsulation by bundling related data properties within a single structure that represents the pizza domain concept. This encapsulation provides a clear interface for working with pizza data throughout the application. Components that need to create or manipulate pizza objects interact with the class constructor and properties rather than managing loose collections of variables. This approach reduces coupling between components and ensures pizza data maintains a consistent structure regardless of where it appears in the application. The class design includes sensible default values for all properties, enabling simple object creation while supporting customization through constructor parameters.

9.3 Event-Driven Architecture

The system employs event-driven patterns that decouple user interactions from response logic. Components attach event listeners to interactive elements, specifying callback functions that execute when specific events occur. This pattern enables flexible behavior modification without changing the underlying HTML structure. For example, filter buttons in the menu component trigger event handlers that update display state and re-render the menu without requiring page navigation. The event delegation pattern appears in components that manage dynamic content, where a single event listener on a parent element handles events for multiple child elements. This approach reduces the number of active listeners and works correctly even when child elements change dynamically.

9.4 Service Layer Pattern

The firebase-service.js module implements a service layer that abstracts Firebase operations behind a clean interface. Rather than having multiple components interact directly with Firebase APIs, they call functions exposed by the service module. This abstraction provides several benefits including centralized error handling, consistent data formatting, and the ability to modify Firebase interactions in one location without affecting multiple components. The service layer exposes functions such as getAllOrders, updateOrderStatus, and listenToOrders that provide clear semantic interfaces for common database operations. Components using these functions need not understand Firebase-specific concepts or API details.

10. Team Member Contributions

10.1 Ishrak Mulla

Ishrak Mulla has made substantial contributions across multiple areas of the project, demonstrating versatility in both frontend and backend development. He developed the comprehensive menu browsing system, implementing the menu.html layout, menu.js logic for dynamic rendering and filtering, and menu.css styling for the responsive grid layout. His work on the menu includes the complete category filtering system that enables customers to view items by type, the dynamic cart counter that updates in real-time, and the integration between menu item selection and subsequent pages. He also contributed significantly to the pizza builder component, working on the pizzaBuilder.html structure, portions of pizzaBuilder.js logic, and creating several supporting modules including toppings.js for topping selection interfaces and quantityTemplate.js for the quantity selector. Beyond code implementation, Ishrak took responsibility for creating project documentation, includes contributing to the Software Requirements Specification document, the Software Project Management Plan, and the Software Configuration Management Plan. He established the overall project structure and ensured consistent coding standards across team contributions. His backend integration work included setting up the initial Firebase configuration and establishing patterns for data flow between the frontend and Firebase services.

10.2 Prachish Pandey

Prachish Pandey focused his efforts on implementing critical frontend functionality that bridges multiple components. He developed substantial portions of the shopping cart system, implementing the cart management logic that maintains item collections in session storage throughout the user's session. His work includes the cart counter display that appears across multiple pages, providing customers with continuous visibility into their cart contents. He contributed to the checkout interface implementation, working on portions of checkout.js that handle cart item display, price calculations, and the order submission flow. Prachish also implemented various user interface components that appear throughout the application, ensuring consistent styling and behavior across different pages. He worked on form validation logic that ensures customers provide required information during checkout. His frontend development included responsive design elements that ensure the application displays correctly across different device sizes, implementing CSS media queries and flexible layouts. He contributed to the project's testing efforts by identifying edge cases and verifying that component interactions work correctly across different usage scenarios.

10.3 Daniel Araujo

Daniel Araujo served as the primary integration specialist for the project, ensuring that independently developed components work together cohesively in the complete application. He established the GitHub repository structure, implementing a branching strategy that enables parallel development while maintaining code stability. Daniel created the navigation flow logic that connects the landing page, menu, pizza builder, and checkout pages, ensuring data transfers correctly between components. He implemented the order type selection functionality on the landing page, including both the user interface elements and the logic that stores the selection for later use during checkout. His integration work extended to the Firebase configuration, where he validated the firebase-config.js settings and ensured proper initialization across all components that require database access. Daniel also worked on the delivery and carryout page variants, implementing the specific flows for each order type. He contributed to code organization efforts, refactoring components to improve modularity and reduce duplication. His work on the index.js file ensures proper initialization of the landing page and correct routing based on user selections. Daniel also participated in code reviews, providing feedback on pull requests and helping maintain code quality standards across team contributions.

10.4 Zeyu Zhang

Zeyu Zhang contributed to the development of the checkout and order confirmation systems. He worked on implementing portions of the checkout interface that display order summaries and collect customer information. His contributions include work on the form structures that gather delivery addresses and contact details during the checkout process. Zeyu participated in testing activities, verifying that the checkout flow functions correctly under various scenarios. He contributed to user interface refinements, suggesting improvements to layout and presentation that enhance the customer experience. His involvement extended to documentation efforts, helping ensure technical specifications accurately reflect the implemented system. Zeyu also assisted with integration testing, verifying that data flows correctly from the cart through checkout to final order submission.

10.5 Djoulie Saint Louis

Djoulie Saint Louis focused her contributions on visual design and user interface styling throughout the application. She developed CSS styling that creates the consistent visual appearance across all pages, implementing the color scheme, typography choices, and spacing standards that define the application's look and feel. Her work includes the responsive design elements that ensure the interface adapts appropriately to different screen sizes, using CSS Flexbox and Grid layouts effectively. Djoulie created the visual styling for interactive elements, implementing hover effects, focus states, and transition animations that enhance user experience. She worked on the modal dialog styling used in the pizza customization interface, ensuring the overlay and content area display correctly across different browsers and devices. Her contributions include the card-based layout designs used in the menu and dashboard components, implementing the shadows, borders, and spacing that create visual hierarchy. Djoulie also participated in accessibility improvements, ensuring adequate color contrast ratios and proper focus indicators for keyboard navigation.

11. GitHub Repository

The complete project source code, documentation, and related artifacts are available in the team's GitHub repository. The repository maintains comprehensive version history showing the evolution of the codebase throughout the development process. Each team member's contributions appear in the commit history, providing transparency into individual efforts and collaborative work. The repository structure organizes code into logical directories, with separate areas for HTML files, JavaScript modules, CSS stylesheets, and documentation. The README file provides setup instructions for developers who want to run the application locally or deploy it to Firebase Hosting. Issue tracking within the repository documents known bugs, feature requests, and technical tasks requiring attention. Pull requests demonstrate the code review process used by the team to maintain quality standards before merging changes into the main branch.

GitHub Repository URL: https://github.com/DanDaMan2121/PIDDZ.git