CS426 - Mobile Device Application Development

MIDTERM PROJECT

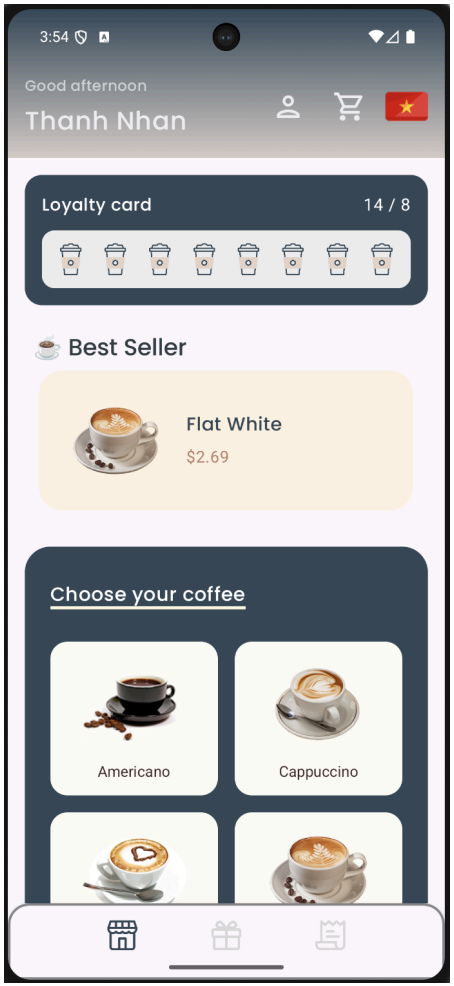
Report

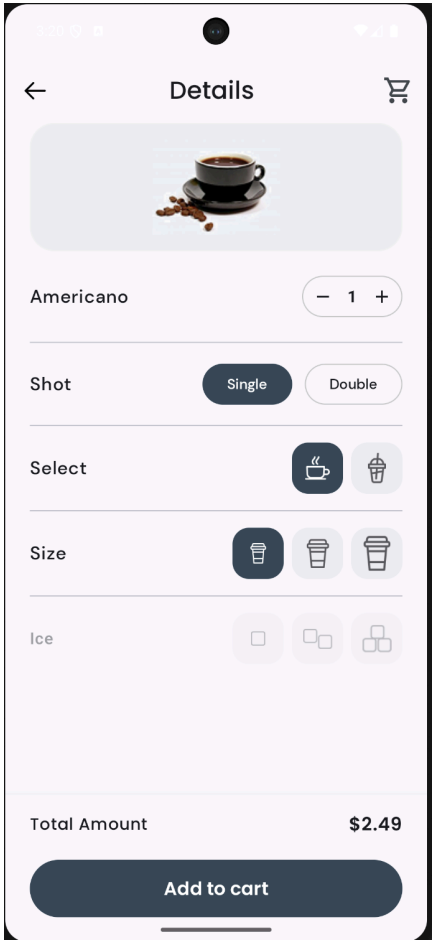
Student Information

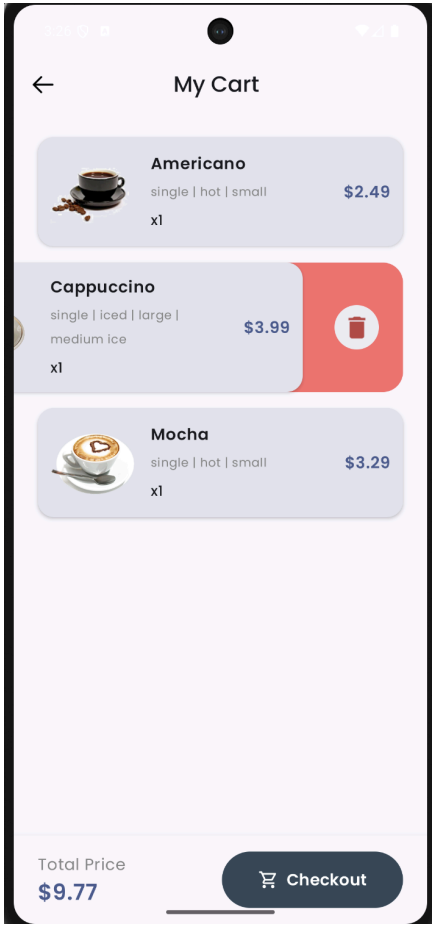
- Name: Nguyen Thanh Nhan
- Student ID: 23125014

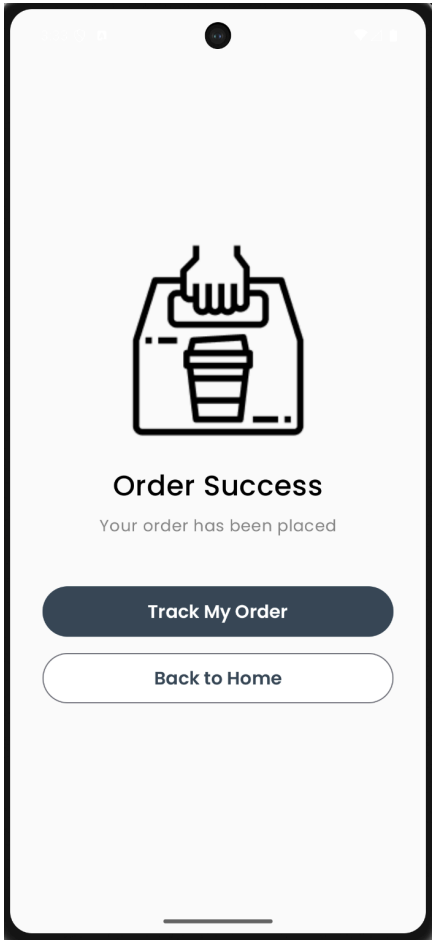
Project Self Assessment

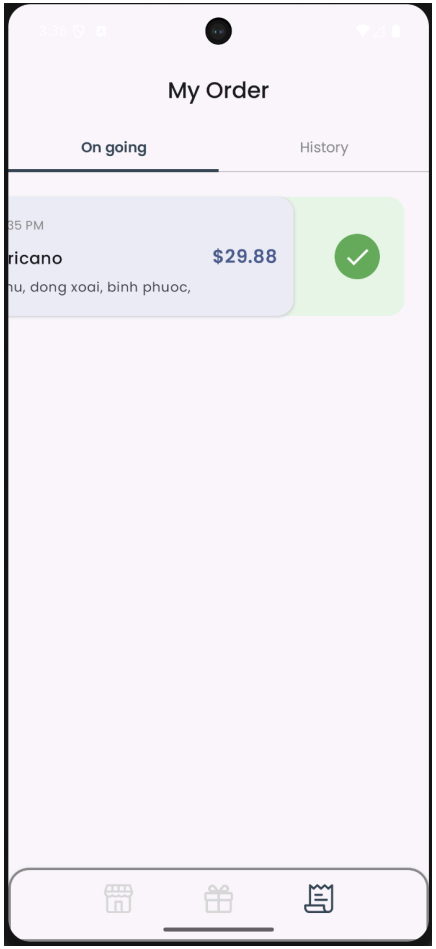
- ☐ **Self-assess your score in the evaluation column**
- ☐ **Replace the app design images with your app screenshots**

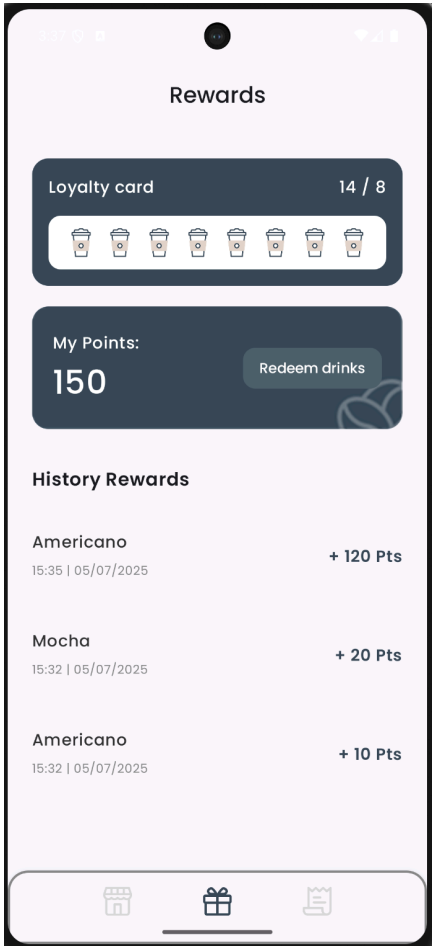
Home Screen		Self-Assessment Score	Maximum Score
	**UI Implementation**: Implement the fundamental UI layout for the home screen.	1	1
	Header Component: Integrate the specified header element.	2	2
	Bottom Navigation Bar: Implement a functional bottom navigation controller.	3	3
	Loyalty Card View: Display the user's loyalty card status.	3	3
	Coffee List View: Populate and render a `ListView` or `RecyclerView` of available coffee products.	3	3
	Navigation Intent : Implement an `on-click` listener for list items that navigates the user to the corresponding product "**Details**" screen	3	3

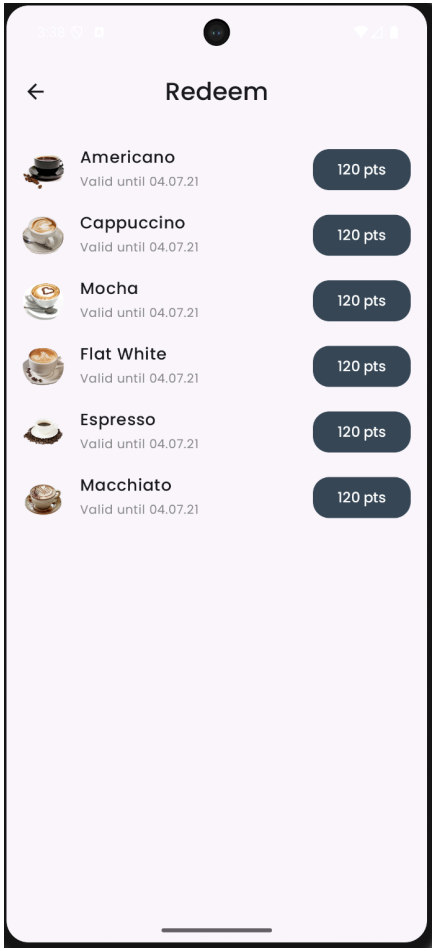
Details			
	**Product Customization Interface**: Implement UI controls to allow users to customize product options (e.g., shot, size, ice).	3	3
	Add to Cart Functionality: On "**Add to Cart**" button press, persist the customized item to the cart's state and trigger a navigation event to the "**My Cart**" screen.	3	3
	Cart Preview: Implement a method (e.g., icon click) to view the current items in the cart without navigating away from the details screen	3	3
	Dynamic Price Calculation: Ensure the total amount dynamically updates in the UI in real-time as the user modifies item quantity or custom options.	3	3
	Back Navigation: Implement a `Maps back` or `on-press back` event to return to the **Home Screen**.	1	1

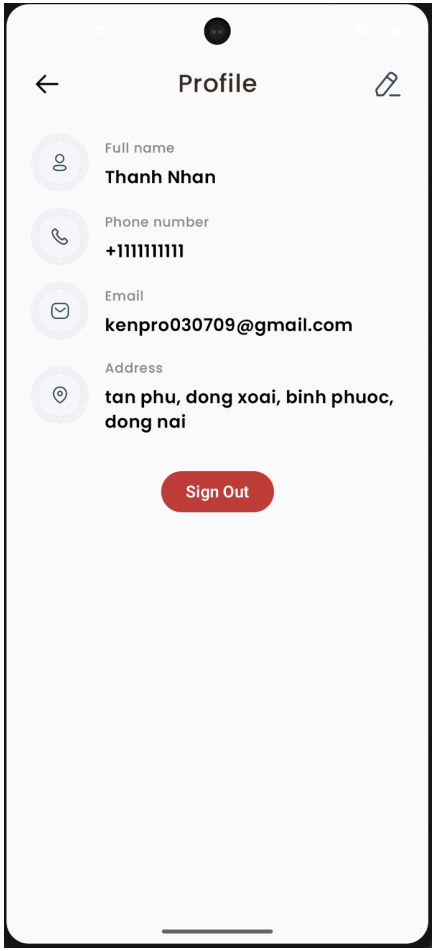
My Cart			
	**UI Implementation**: Implement the basic layout for the cart screen	1	1
	Cart Item Rendering: Display a `ListView` or `RecyclerView` of all coffee items added to the cart, reflecting their selected customizations.	7	7
	Total Price Display Compute and display the aggregate total price of all items in the cart.	3	3
	Gesture-Based Item Removal: Implement gesture detection (e.g., `on-swipe-left`) to remove items from the cart.	3	3
	Checkout Navigation: On "**Checkout**" button press, navigate to the "**Order Success**" screen.	1	1

Order Success			
	**Order Success UI**: Implement the layout for the order success confirmation screen.	1	1
	Track Order Navigation: On "Track My Order" press, navigate to the "My Orders" screen.	1	1

My Orders			
	**My Orders UI**: Implement the basic layout for the order history screen.	1	1
	Order History Display: Render a list segregating ongoing and completed/past orders.	3	3
	Order Status Transition: Implement an event handler (e.g., on-click, on-swipe) to transition an order's state from "ongoing" to "history".	3	3

Rewards			
	**Rewards Screen UI:** Implement the primary layout for the Rewards screen.	1	1
	Loyalty Stamp Logic: For each completed order, increment the loyalty card stamp count by one, up to a maximum of eight.	3	3
	Loyalty Card Reset: Implement an event (e.g., on-click) to reset the stamp count to zero upon reaching eight stamps.	3	3
	Points Calculation & Display: Implement logic to award reward points based on the total monetary value of each order and display them in a list.	3	3
	Total Points Aggregation: Display the sum of all accumulated reward points.	2	2

Redeem Rewards			
	**Points Redemption**: On a user action, allow the exchange of accumulated points for a product, and correctly decrement the total points.	3	3

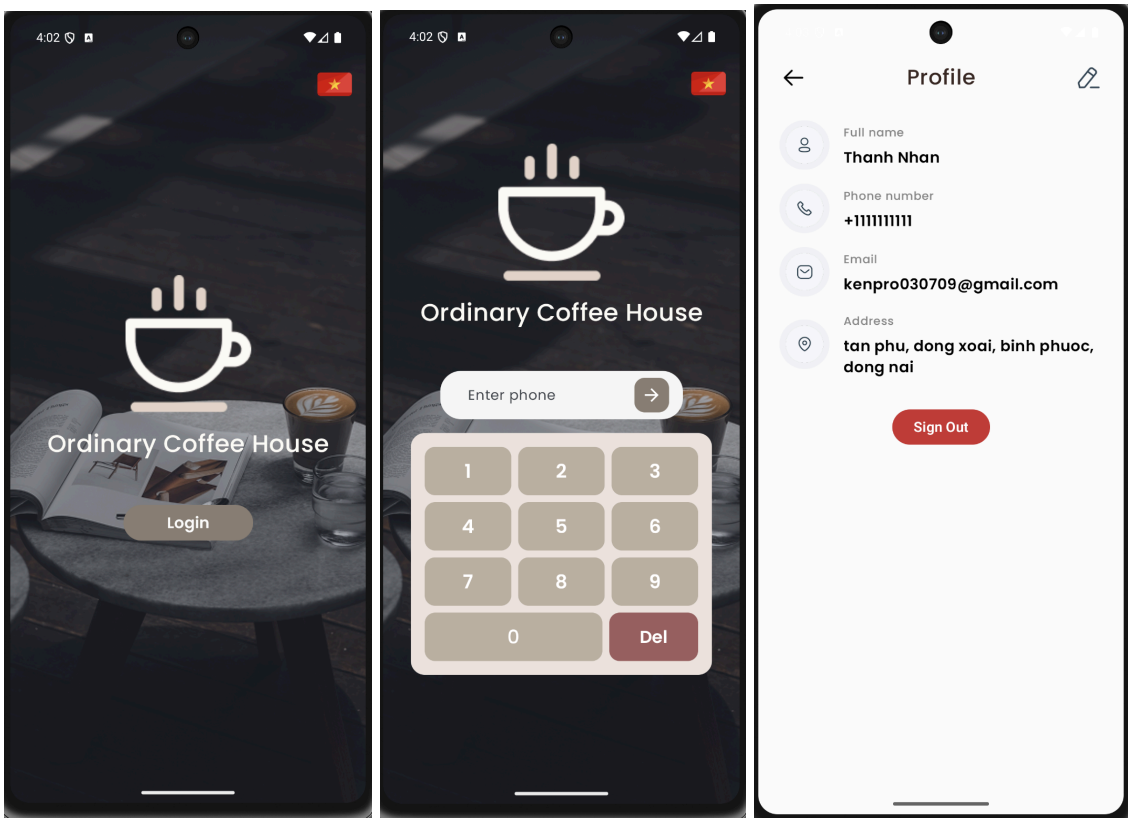
Profile			
	**UI Implementation**: Implement the user profile screen layout.	3	3
	Profile Editing Functionality: Enable an edit mode via an icon press, allowing for modification of user profile data.	3	3

General application requirements			
	State & Lifecycle Management: Implement robust state management to handle the application lifecycle (e.g., `onPause`, `onResume`, `onStop`) and preserve data across configuration changes.	12	12
	Data Persistence & Initialization: Implement a data persistence strategy (e.g., SharedPreferences, Room, SQLite) and handle the initial seeding of required application data.	6	6
	User-Defined Features: Implement novel features or requirements beyond the scope of this document.	**50**	**50**
Total	**Min(150, Total score of all requirements above)**	**141**	**141**
Additional Criteria	**Code Quality, Report & Demo (±10 Points):** The final score may be adjusted by up to 10 points based on the quality, clarity, and organization of the source code, the project report, and the video demonstration.	**+10**	**[-10,+10]**
Your score: Min(10,(Total + Additional Criteria) / 15)			

Development Retrospective

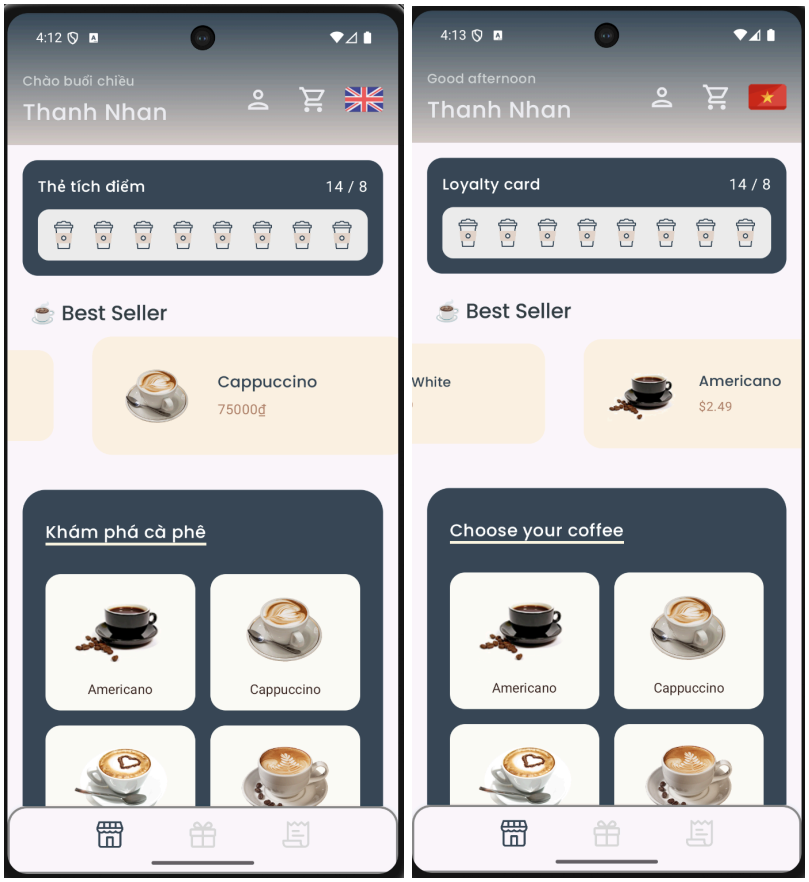
A. Content of New Features

1. Log in Log out



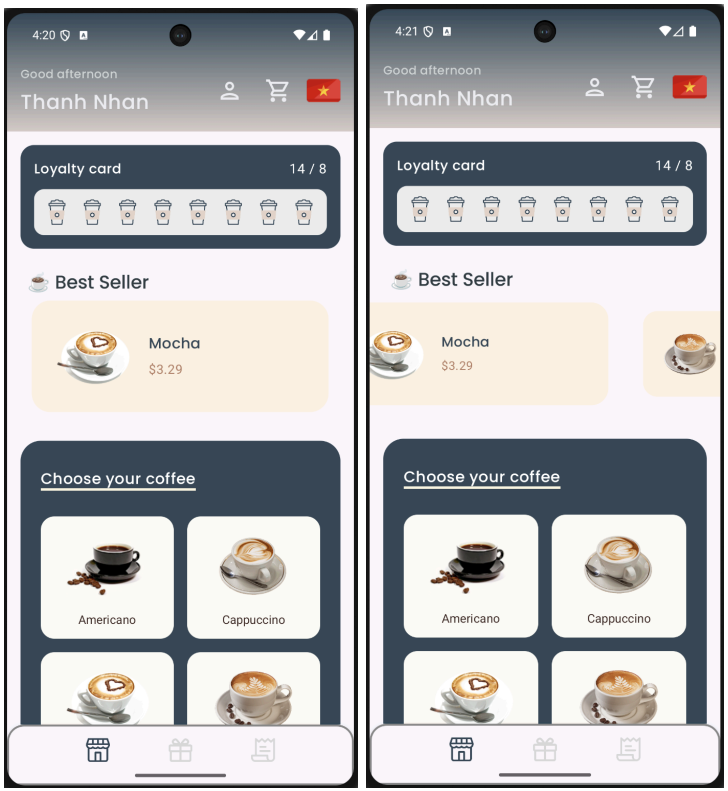
This feature allows users to log in using their phone number. After entering a valid phone number and passing verification, the user is directed to the Home screen, where personalized content is displayed. The app retrieves and shows the user's profile information (such as name, email, and address) in the Profile section. From there, users can choose to sign out. This feature ensures secure and personalized access, protects user privacy, and supports safe usage across shared devices.

2. Two languages



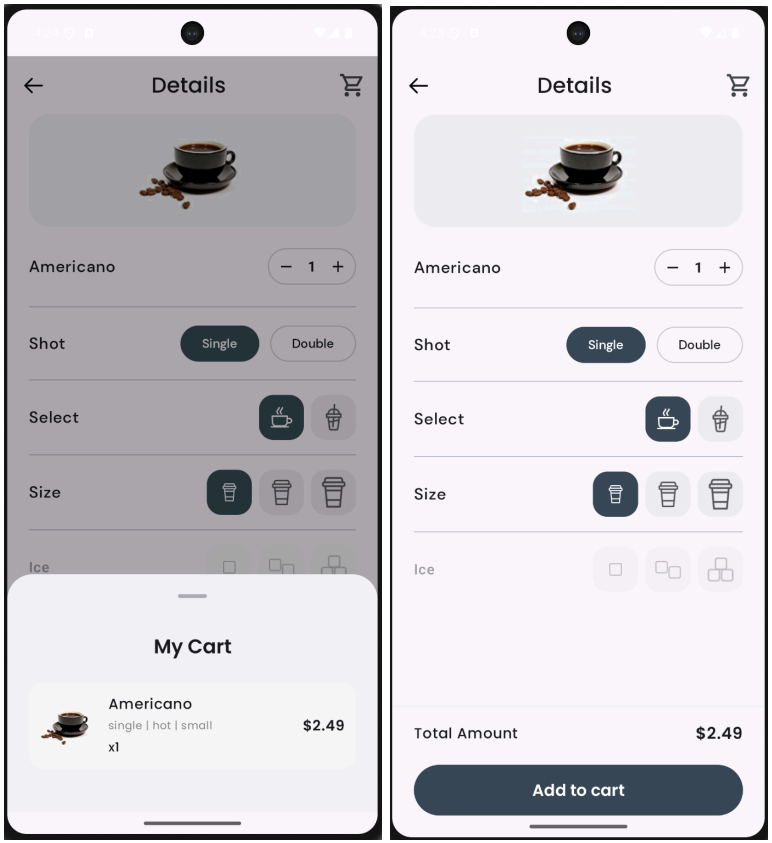
This feature allows users to switch between Vietnamese and English by tapping the flag icon in the top-right corner of the screen. All UI texts, such as greetings, section titles, and coffee descriptions, are dynamically translated based on the selected language. This enhances user accessibility, especially for bilingual users or foreign customers, and improves the overall user experience by providing a localized interface.

3. Best seller



This feature displays a horizontal carousel of the most popular coffee items. Users can manually swipe through the items with smooth animations, or simply wait as the carousel automatically scrolls every 2 seconds. The combination of manual and auto-scrolling ensures that users can easily discover the best-selling products in an engaging and accessible way.

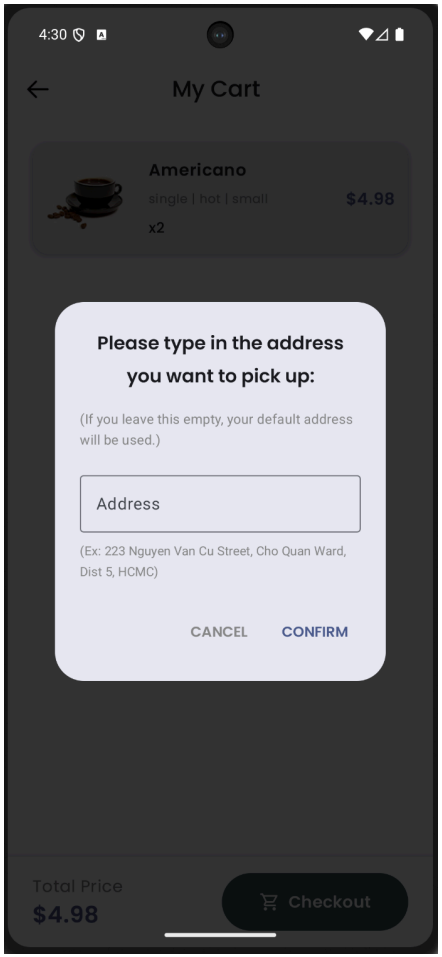
4. In detail screen



The detail screen allows users to customize their drinks with options such as shot type, serving method (hot or cold), and cup size. By tapping the cart icon in the top right corner, users can access a cart preview, which provides a quick overview of their selected items without navigating to the full cart screen.

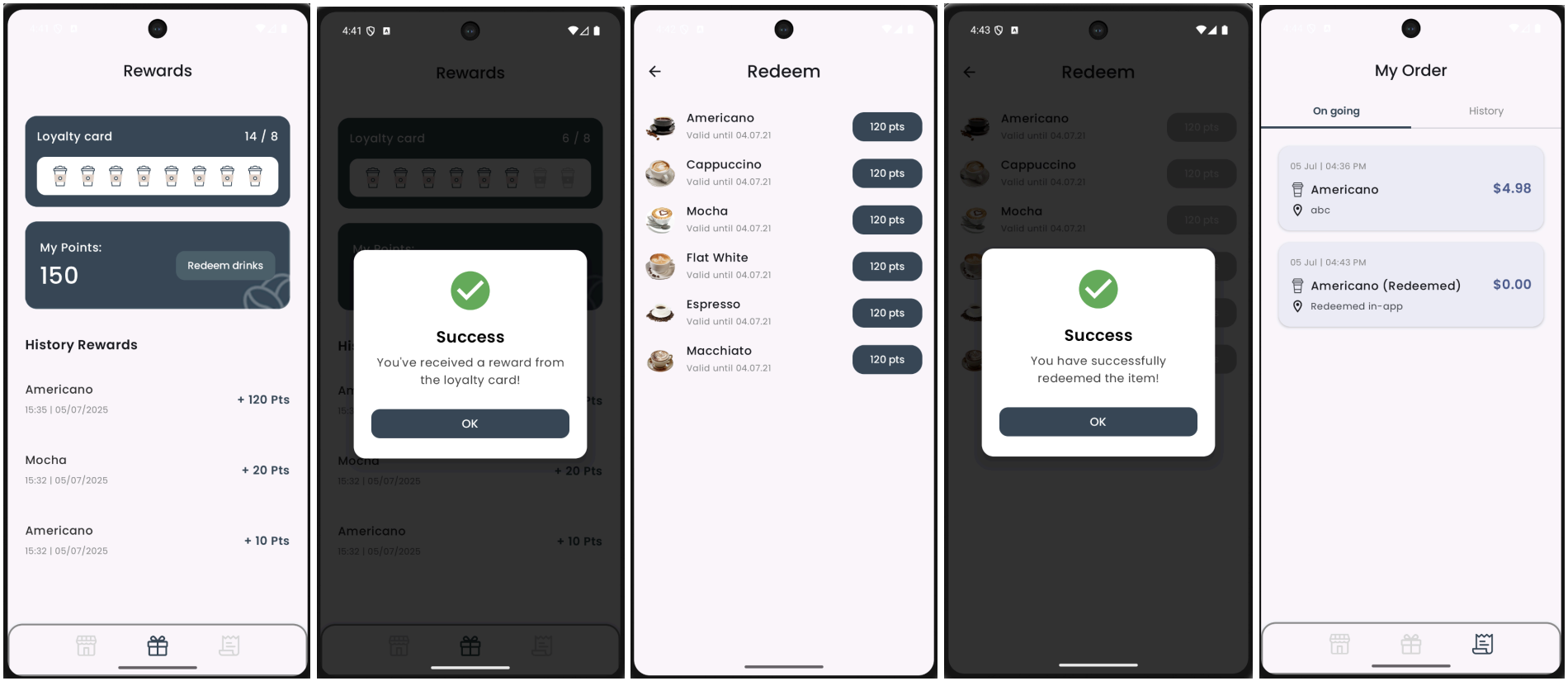
Additionally, the app includes smart logic handling: when a hot drink is selected, the ice options are automatically disabled to prevent invalid combinations. This ensures a smoother and more accurate ordering experience.

5. In my_cart screen



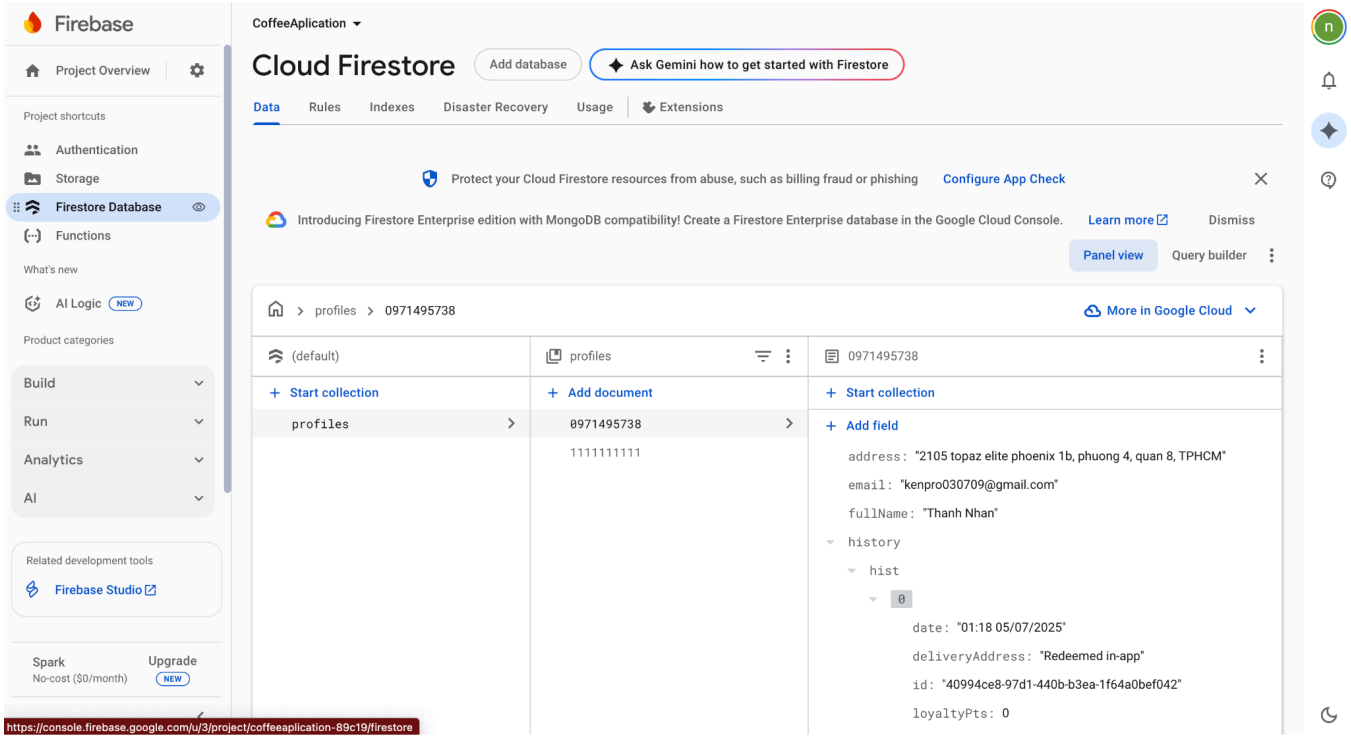
On the My Cart screen, when the user taps the Checkout button, a dialog appears prompting them to enter a pickup address. If the input is left blank, the app will automatically use the user's default address stored in their profile. This dialog ensures that users clearly specify their desired pickup location before placing an order, offering both flexibility and convenience. The feature helps prevent miscommunication and ensures a smoother ordering process, especially for users with multiple delivery locations.

6. **In Reward Screen**



In the Reward screen, when the user taps on the loyalty card and has collected 8 stamps, they will automatically receive a reward. A success dialog is shown to confirm the redemption, and 8 stamps are deducted from the card. Additionally, in the Redeem drinks section, if the user has enough points (e.g. 120 pts), the corresponding drink option becomes active (highlighted). When the user taps to redeem, points are automatically deducted, and a basic version of the selected drink is added to their ongoing orders with a price of $0. This feature provides a smooth and interactive reward experience, encouraging customer engagement and loyalty.

7. Firebase Realtime Database



The application uses Firebase Firestore to store and manage user data, including profiles, order history, loyalty points, and addresses. To reduce delays caused by fetching data directly from Firebase on every screen, a centralized ViewModel is initialized during login and persists throughout the session until logout. This approach helps minimize loading time, reduces redundant Firebase reads, and ensures a smoother and faster user experience across the entire app.

B. Techniques Applied

1. Firebase Firestore

Firebase Firestore was used as the cloud-based backend for storing and retrieving essential user data, including profiles, order history, loyalty points, and addresses. A custom `PhoneProfileManager` class was created to manage Firestore access, with the user's phone number acting as the document ID. All Firestore operations—such as fetching or updating user information—were implemented asynchronously using Kotlin coroutines within ViewModels to ensure smooth UI interactions. During development, I learned how to replace traditional callback listeners with coroutine-based methods, improving code readability and performance. Additionally, minimizing redundant reads was crucial to reducing latency and ensuring a responsive user experience.

2. ViewModel and State Management

ViewModels played a central role in maintaining consistent application state across multiple screens. Key ViewModels included `ProfileViewModel` for managing user data, `CartViewModel` for cart operations, `DetailsViewModel` for customization, and `LanguageViewModel` for language settings. These ViewModels used `MutableStateFlow` to expose observable states, which were collected in composable functions using `collectAsState()`. This ensured that UI components reacted automatically to state changes. A key challenge was managing ViewModel scope correctly to persist data across navigation, and ensuring separation of concerns when handling shared data like cart updates or user profiles.

3. Navigation and Animated Transitions

The application implemented screen navigation using `AnimatedNavHost` from the Accompanist Navigation library. This allowed for custom transition animations such as horizontal sliding when moving between screens like Home, Details, Cart, and Profile. Routes were defined with parameters to support dynamic navigation, for instance navigating to a drink's details via `details/{drinkName}`. Special attention was given to handling back navigation and ensuring that the animation transitions were context-sensitive. Through this, I gained experience in managing complex navigation flows and resolving backstack inconsistencies using `popBackStack()`.

4. Jetpack Compose UI

All user interface components were developed using Jetpack Compose, enabling a declarative and state-driven approach to UI design. Core layouts were built using composables such as `Column`, `Row`, `LazyRow`, `Box`, and `Dialog`, combined with `Modifier` for styling, alignment, padding, and responsiveness. State was lifted into ViewModels, allowing the UI to react dynamically to user interactions. This composable architecture required mastering recomposition behavior and managing performance during real-time updates. Designing scalable and adaptive layouts for various screen sizes was a particularly valuable skill gained through this process.

5. Animation and Auto Scrolling

Visual interactivity was enhanced using animations, particularly in the Best Seller carousel. This feature utilized `HorizontalPager` and `pagerState` to implement horizontal scrolling, with focused items enlarged using scale effects applied via `graphicsLayer` and `lerp()`. Auto-scrolling was achieved using `LaunchedEffect` and coroutine delays, creating a dynamic and engaging browsing experience. One of the challenges here was synchronizing automatic page scrolling with manual swipes to avoid conflicts and lag. I also learned to calculate visual focus using `currentPageOffsetFraction` to fine-tune animation responsiveness.

6. Multi-language Support

To support both English and Vietnamese, the app implemented a language switching system using `LocaleHelper`, `SharedPreferences`, and `LanguageViewModel`. Users could toggle the language at any time by tapping the flag icon, which would update the app's locale and trigger UI recomposition. This approach eliminated the need to restart the app and ensured a seamless bilingual experience. One key challenge was ensuring that all visible strings were updated consistently, which required thorough use of string resources and correct state propagation across composables.

C. Implementation Experience

Throughout the development of this project, I encountered a number of challenges and valuable learning moments that significantly shaped my understanding of modern Android development.

One of the initial issues I faced was the latency when accessing user data from Firebase. Every time a screen needed user information, there was a noticeable delay. To address this, I decided to implement a persistent `ViewModel` to cache the data after login, reducing the number of redundant calls to Firestore. This greatly improved app performance and user experience. During this phase, I initially confused `remember` and `rememberSaveable`, which led to several hard-to-trace bugs because the state was not behaving as expected across recompositions.

Another highlight was customizing the screen transition animations. Rather than relying on the default navigation behavior, I created sliding transitions between screens using `AnimatedNavHost`. This made the navigation feel

smoother and more dynamic. However, it also introduced additional complexity—deciding when to slide left versus right depending on the navigation direction required logical control and careful backstack handling.

From a database perspective, I learned that assigning unique IDs to each order or reward was critical. At one point, I mistakenly reused the same document ID, which caused data to overwrite and break order tracking.

The Best Seller carousel was one of the most technically intensive and rewarding components. Initially, I tried to build it using a basic scrollable row and manual state handling, but the implementation was buggy—users had to tap at just the right moment to navigate to the detail page. Later, switching to `HorizontalPager` greatly simplified the logic and resolved interaction issues. Implementing auto-scroll, snapping, and scaling animations took multiple iterations but eventually produced a smooth and engaging user interface.

In summary, this project helped me improve my understanding of Jetpack Compose, Firebase integration, state management, and UI animation. It also reinforced the importance of debugging composable behavior, thinking carefully about state lifecycles, and designing smooth user interactions.

References

1. **Firebase SDK (Auth, Realtime DB):** Google's backend-as-a-service platform, used for authentication, data persistence, and file storage.
   - https://firebase.google.com
2. **Material Components:** Google's design system components (BottomNavigationView, CardView, TextInputLayout, etc.).
   - https://m3.material.io
3. **GitHub Repository**: Project source code: https://github.com/nahn-apcs/The-Code-Cup
4. **Demo Video:** Walkthrough of application functionality: Link