Nikunj Pradhan
Group 39
Assignment 2

Table Of Contents

Part 1: Use Case Identification

Use Case #1: Quick Taste Quiz Onboarding

Related User Stories: A1
Actor(s): Ava (primary)
Brief Description: A new user completes a brief taste quiz to seed immediate recommendations.
Priority: High

Use Case #2: Save Favorites

Related User Stories: A2
Actor(s): Ava (primary)
Brief Description: User taps a heart to add a drink to a favorites list for quick access later.
Priority: High

Use Case #3: Gentle Similar Alternatives

Related User Stories: A3
Actor(s): Ava (primary)
Brief Description: User requests drinks similar to a selected item to discover close flavor matches.
Priority: High

Use Case #4: Budget Filter

Related User Stories: A4
Actor(s): Ava (primary)
Brief Description: User limits browse results to drinks within a chosen price tier ($/$$/$$$).
Priority: Medium

Use Case #5: Sweetness Slider

Related User Stories: A5
Actor(s): Ava (primary)
Brief Description: User adjusts a sweetness level to exclude overly sweet options from results.
Priority: High

Use Case #6: Recently Viewed

Related User Stories: A6
Actor(s): Ava (primary)
Brief Description: User opens a history list of previously viewed drinks to compare options.
Priority: Medium

Use Case #7: “Not For Me” Dismiss

Related User Stories: A8
Actor(s): Ava (primary)
Brief Description: User marks a drink as “not for me,” reducing similar items in future recommendations.
Priority: Medium

Use Case #8: Caffeine Range Filter

Related User Stories: L1
Actor(s): Leo (primary)
Brief Description: User sets an allowable caffeine range (mg) so results respect daily targets.
Priority: High

Use Case #9: Nutrition Snapshot

Related User Stories: L2
Actor(s): Leo (primary)
Brief Description: User views a concise panel showing sugar, calories, and caffeine on drink pages.
Priority: High

Use Case #10: Explain-My-Recs

Related User Stories: L3
Actor(s): Leo (primary)
Brief Description: User taps “Why?” to see the top 2–3 factors used to recommend a drink.
Priority: High

Use Case #11: Allergy/Ingredient Exclusions

Related User Stories: L4
Actor(s): Leo (primary)
Brief Description: User excludes specific ingredients to avoid allergens or unwanted components.
Priority: High

Use Case #12: Decaf/Energy Modes

Related User Stories: L5
Actor(s): Leo (primary)

Brief Description: User switches quick mode toggles (Decaf, Low-Caf, Energy) to change recommendation context.
Priority: Medium

Use Case #13: Time-of-Day Sensitivity

Related User Stories: L6
Actor(s): Leo (primary)
Brief Description: Evening usage reduces high-caffeine recommendations to protect sleep.
Priority: Medium

Use Case #14: Preference Profile Editing

Related User Stories: L7
Actor(s): Leo (primary)
Brief Description: User edits taste and health preferences so future recommendations adapt accordingly.
Priority: High

Use Case #15: Add/Edit Drink Catalog Entries

Related User Stories: M1
Actor(s): Maya (Admin) (primary)
Brief Description: Admin creates or updates drink entries with ingredients and required flags.
Priority: High

Use Case #16: Duplicate Detection

Related User Stories: M2
Actor(s): Maya (Admin) (primary)
Brief Description: Admin reviews and merges system-suggested duplicate drink records.
Priority: Medium

Use Case #17: Mandatory Safety Flags Validation

Related User Stories: M3
Actor(s): Maya (Admin) (primary)
Brief Description: Admin verifies that caffeine/alcohol safety flags are present before publishing.
Priority: High

Use Case #18: Age Gate for Alcohol

Related User Stories: M4
Actor(s): Any User (primary), Age Verification Service (secondary)

Brief Description: The System restricts alcohol content visibility until the user's age is verified.
Priority: High

Use Case #19: Tag Consistency Report

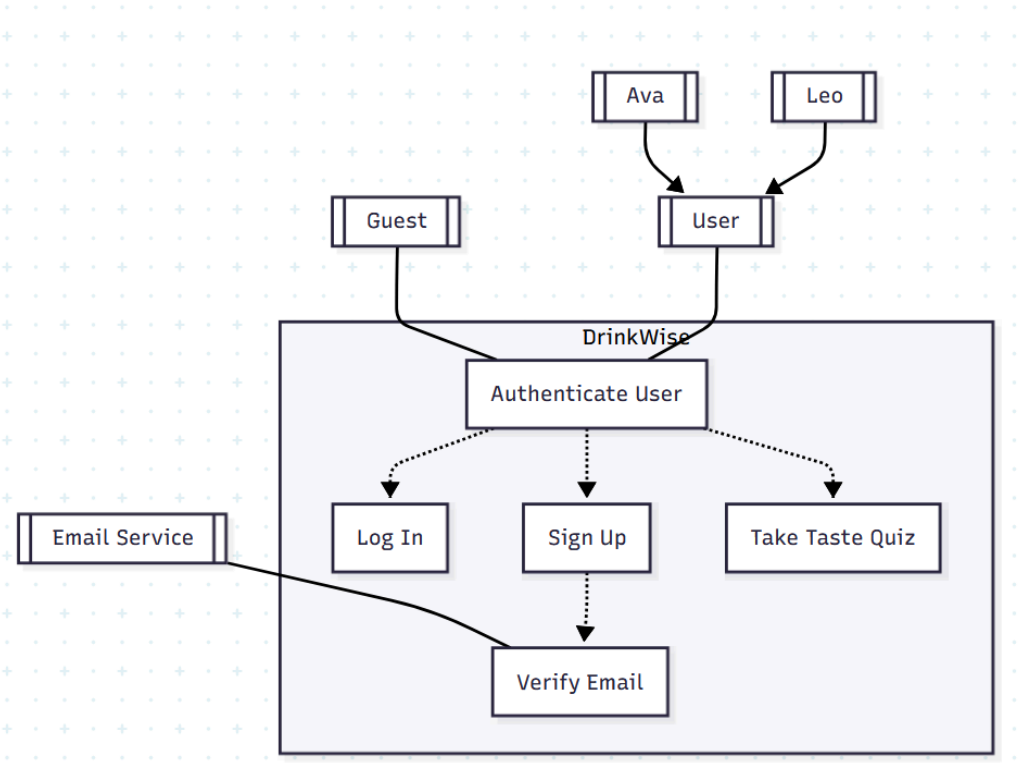
Related User Stories: M5
Actor(s): Maya (Admin) (primary)
Brief Description: Admin runs a weekly report to find missing or inconsistent tags in the catalog.
Priority: Medium

Use Case #20: Accessibility Checker

Related User Stories: M6
Actor(s): Maya (Admin) (primary)
Brief Description: Admin triggers checks for alt text and contrast on catalog images to improve accessibility.
Priority: Medium

Part 2: Use Case Diagrams

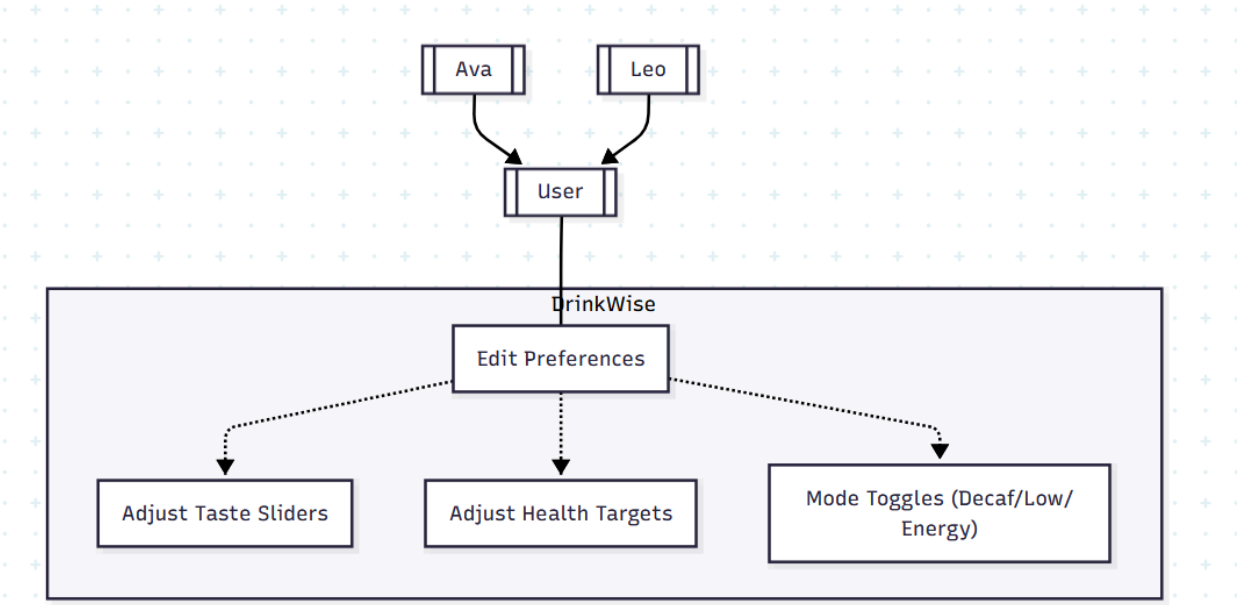
1. Authentication



Mermaid:

```
flowchart TB
  %% Actors
  guest[[Guest]]
  ava[[Ava]]
  leo[[Leo]]
  User[[User]]
  ava --> User
  leo --> User
  emailsvc[[Email Service]]

  %% System
  subgraph useCases["DrinkWise"]
    UC_auth["Authenticate User"]
    UC_login["Log In"]
    UC_signup["Sign Up"]
    UC_verify["Verify Email"]
    UC_quiz["Take Taste Quiz"]
  end

  guest --- UC_auth
  User --- UC_auth
  emailsvc --- UC_verify

  UC_auth -.-> UC_login:::incl
  UC_auth -.-> UC_signup:::incl
  UC_auth -.-> UC_quiz:::incl
  UC_signup -.-> UC_verify:::ext
```

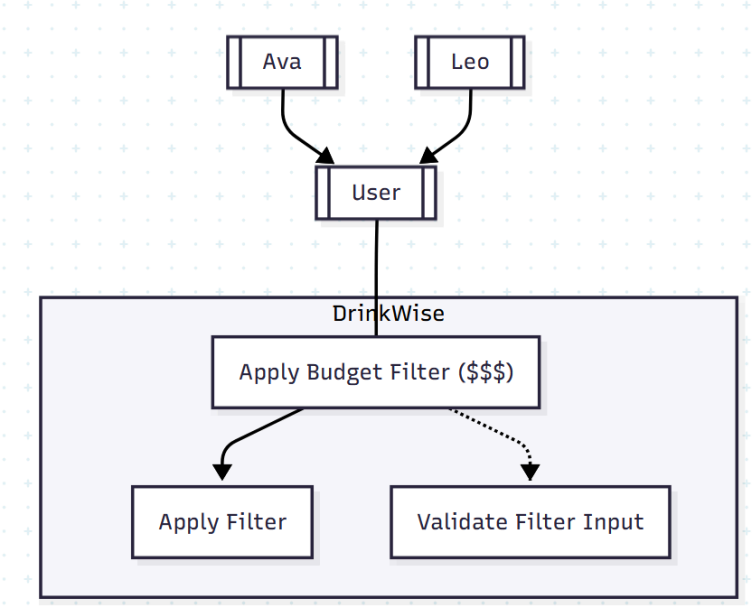
2. Editing Preferences



Mermaid:

```
flowchart TB
  ava[[Ava]]
  leo[[Leo]]
  User[[User]]
  ava --> User
  leo --> User

  subgraph useCases["DrinkWise"]
    UC_pref["Edit Preferences"]
    UC_taste["Adjust Taste Sliders"]
    UC_health["Adjust Health Targets"]
    UC_modes["Mode Toggles (Decaf/Low/Energy)"]
  end

  User --- UC_pref
  UC_pref -.-> UC_taste:::incl
  UC_pref -.-> UC_health:::incl
  UC_pref -.-> UC_modes:::ext
```

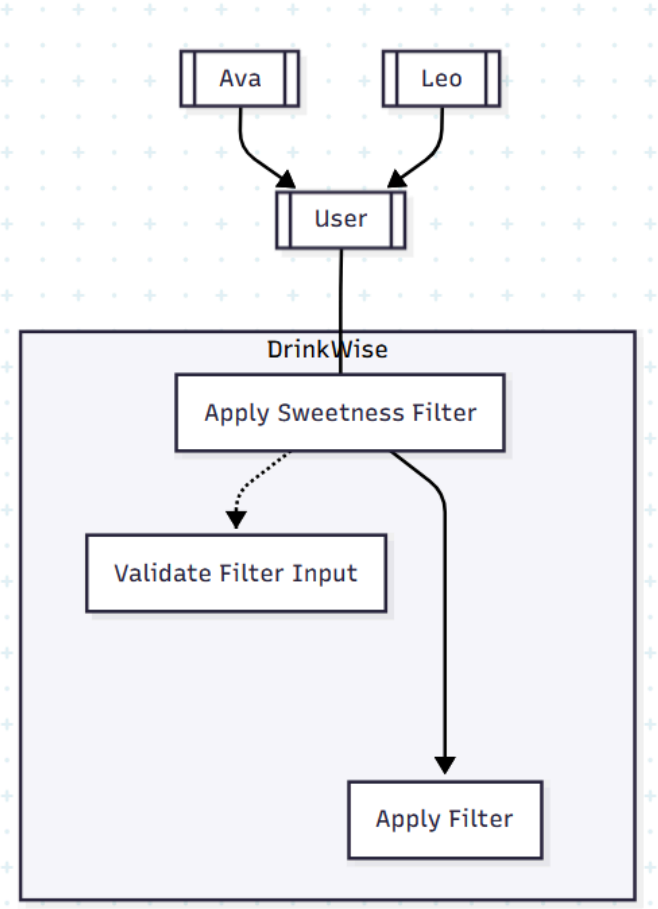
3. Budget Filter



Mermaid:

```
flowchart TB
  ava[[Ava]]
  leo[[Leo]]
  User[[User]]
  ava --> User
  leo --> User

  subgraph useCases["DrinkWise"]
    UC_budget["Apply Budget Filter ($$$)"]
    UC_filterBase["Apply Filter"]
    UC_validate["Validate Filter Input"]
  end

  User --- UC_budget
  UC_budget --> UC_filterBase
  UC_budget -.-> UC_validate:::incl
```

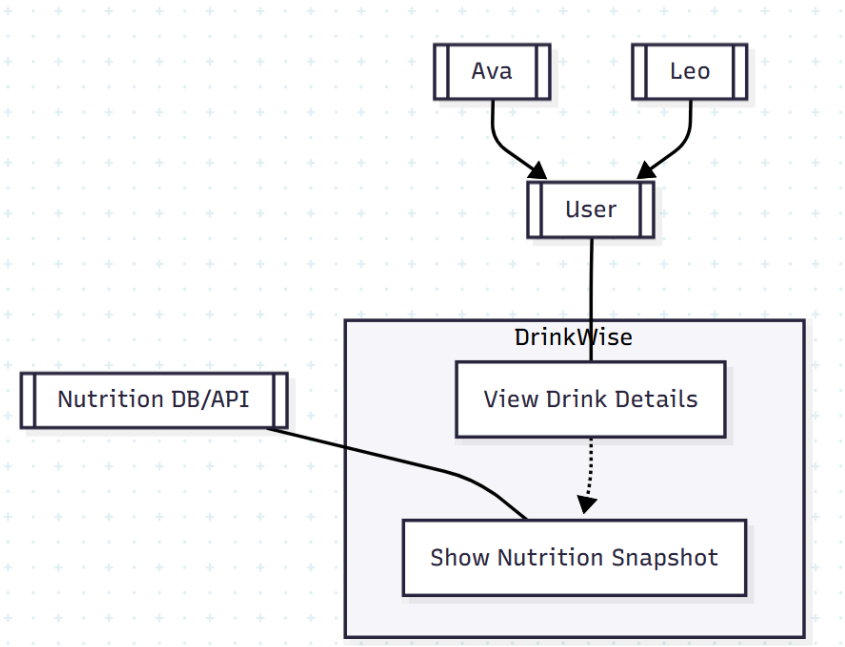
4. Sweetness Filter



Mermaid:

```
flowchart TB
  ava[[Ava]]
  leo[[Leo]]
  User[[User]]
  ava --> User
  leo --> User

  subgraph useCases["DrinkWise"]
    UC_sweet["Apply Sweetness Filter"]
    UC_filterBase["Apply Filter"]
    UC_validate["Validate Filter Input"]
  end

  User --- UC_sweet
  UC_sweet ----> UC_filterBase
  UC_sweet -.-> UC_validate:::incl
```

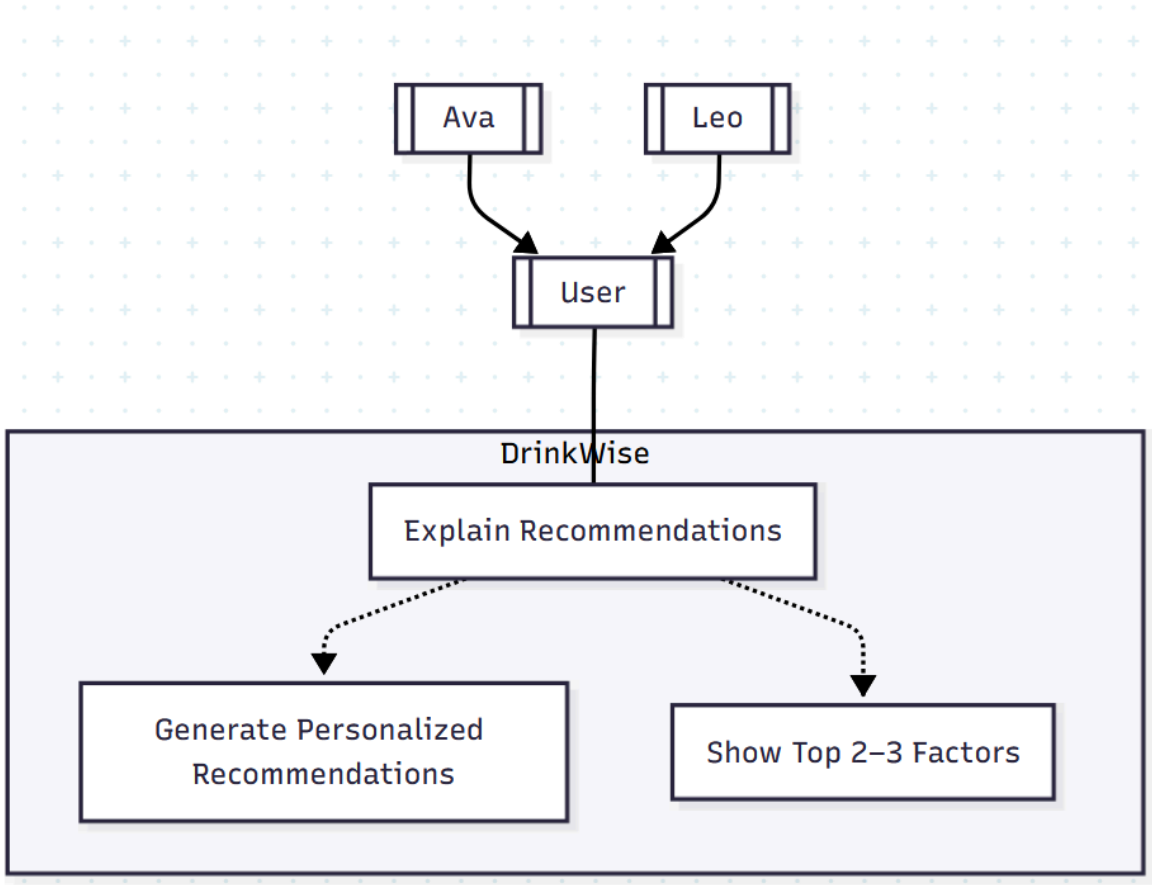
5. Nutrition Snapshot



Mermaid:

```
flowchart TB
  ava[[Ava]]
  leo[[Leo]]
  User[[User]]
  ava --> User
  leo --> User
  nutrdb[[Nutrition DB/API]]

  subgraph useCases["DrinkWise"]
    UC_view["View Drink Details"]
    UC_nutri["Show Nutrition Snapshot"]
  end

  User --- UC_view
  nutrdb --- UC_nutri
  UC_view -.-> UC_nutri:::incl
```

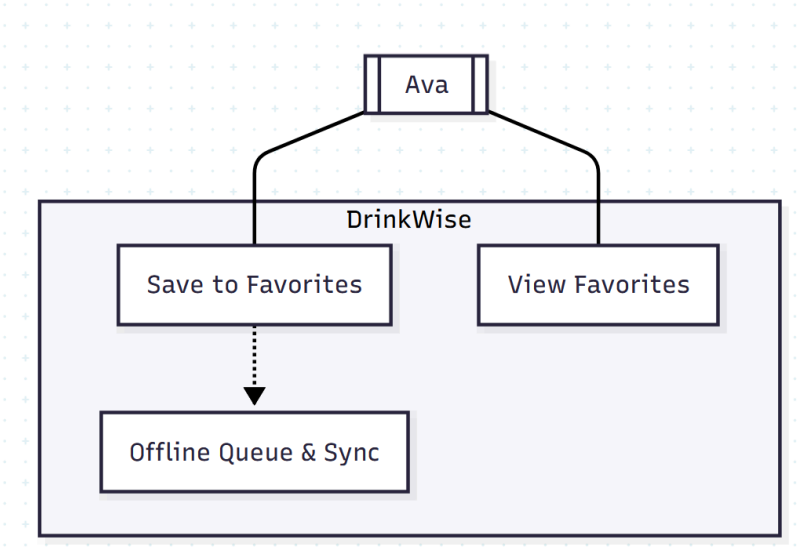
6. Recommendations Transparency



Mermaid:

```
flowchart TB
  ava[[Ava]]
  leo[[Leo]]
  User[[User]]
  ava --> User
  leo --> User

  subgraph useCases["DrinkWise"]
    UC_explain["Explain Recommendations"]
    UC_recs["Generate Personalized Recommendations"]
    UC_why["Show Top 2–3 Factors"]
  end

  User --- UC_explain
  UC_explain -.-> UC_recs:::incl
  UC_explain -.-> UC_why:::incl
```

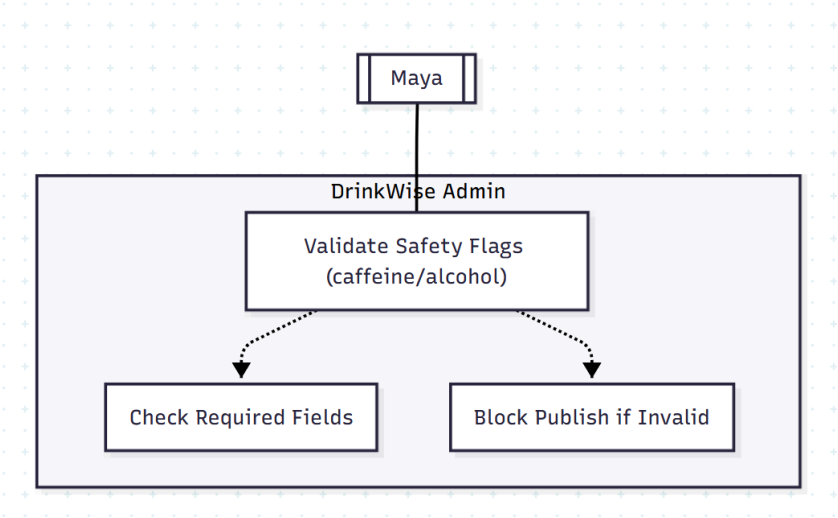
7. Favorites



Mermaid:

```
flowchart TB
  ava[[Ava]]

  subgraph useCases["DrinkWise"]
    UC_fav["Save to Favorites"]
    UC_list["View Favorites"]
    UC_offsync["Offline Queue & Sync"]
  end

  ava --- UC_fav
  ava --- UC_list
  UC_fav -.-> UC_offsync:::ext
```

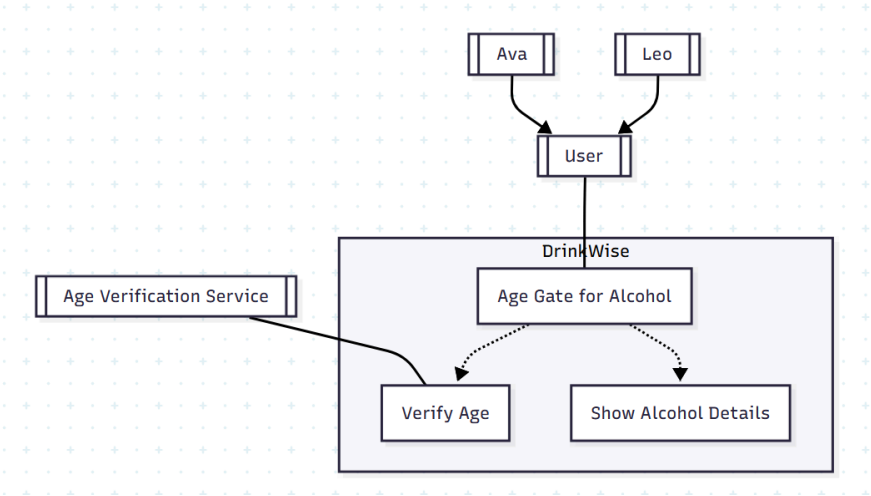
8. Safety Flags



Mermaid:

```
flowchart TB
  maya[[Maya]]

  subgraph useCases["DrinkWise Admin"]
    UC_safety["Validate Safety Flags (caffeine/alcohol)"]
    UC_required["Check Required Fields"]
    UC_block["Block Publish if Invalid"]
  end

  maya --- UC_safety
  UC_safety -.-> UC_required:::incl
  UC_safety -.-> UC_block:::incl
```

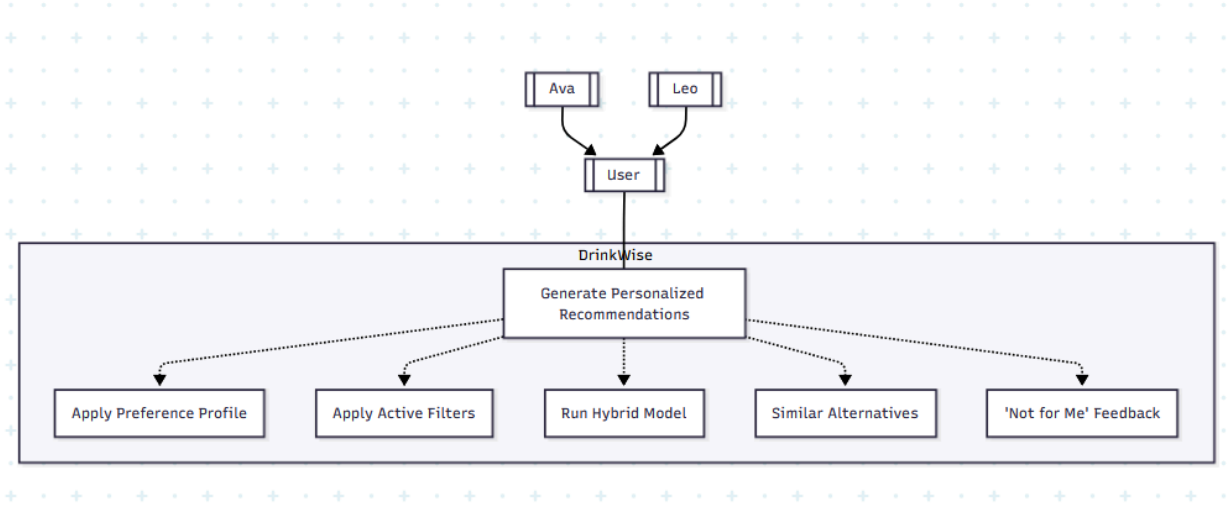
9. Age Gate



Mermaid:

```
flowchart TB
  ava[[Ava]]
  leo[[Leo]]
  User[[User]]
  ava --> User
  leo --> User
  agesvc[[Age Verification Service]]

  subgraph useCases["DrinkWise"]
    UC_age["Age Gate for Alcohol"]
    UC_verify["Verify Age"]
    UC_showAlc["Show Alcohol Details"]
  end

  User --- UC_age
  agesvc --- UC_verify

  UC_age -.-> UC_verify:::incl
  UC_age -.-> UC_showAlc:::ext
```

10. Personalized Recommendations

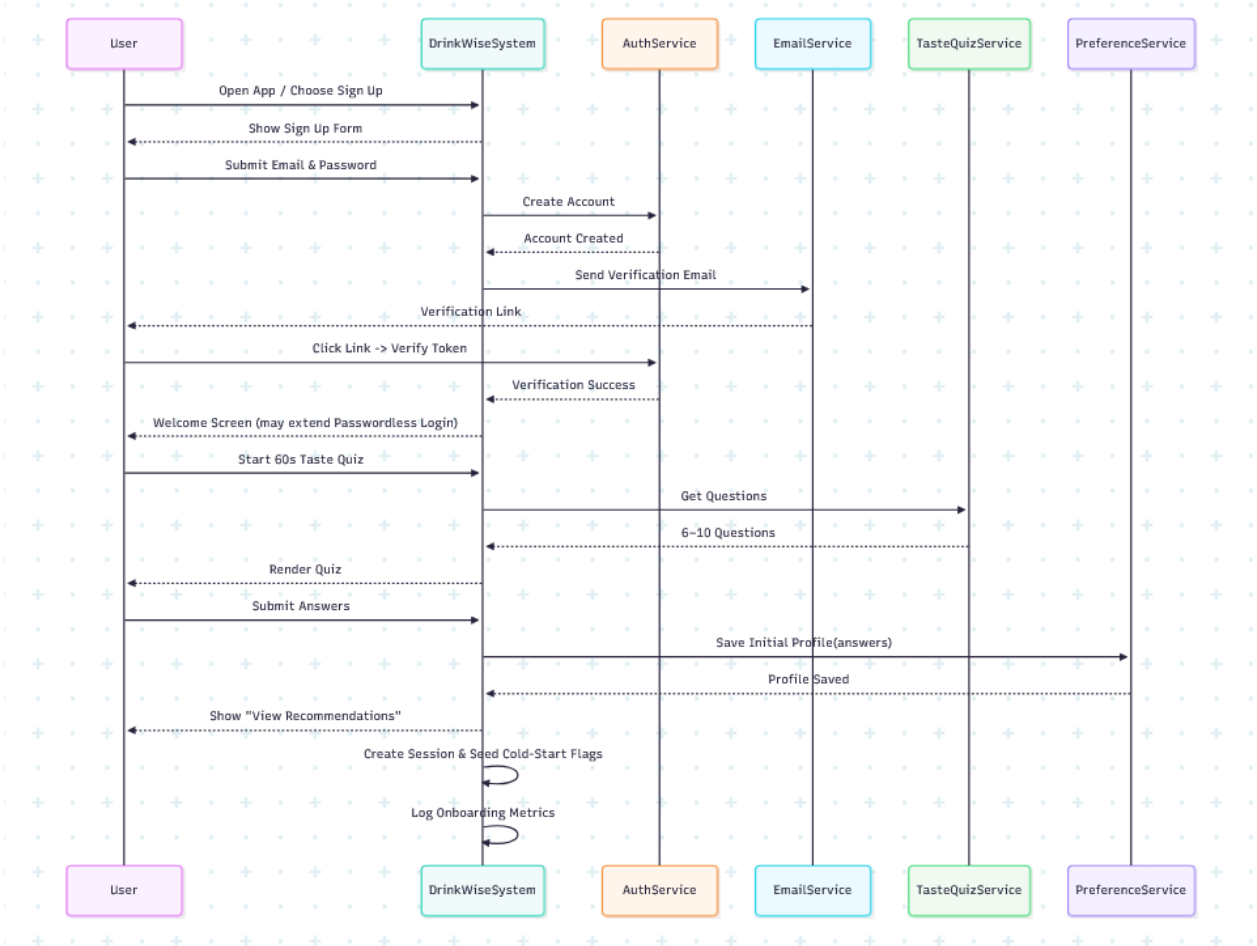


Mermaid:

```
flowchart TB
  ava[[Ava]]
  leo[[Leo]]
  User[[User]]
  ava --> User
  leo --> User

  subgraph useCases["DrinkWise"]
    UC_recs["Generate Personalized Recommendations"]
    UC_profile["Apply Preference Profile"]
    UC_filters["Apply Active Filters"]
    UC_model["Run Hybrid Model"]
    UC_sim["Similar Alternatives"]
    UC_notme["'Not for Me' Feedback"]
  end

  User --- UC_recs
  UC_recs -.-> UC_profile:::incl
  UC_recs -.-> UC_filters:::incl
  UC_recs -.-> UC_model:::incl
  UC_recs -.-> UC_sim:::incl
  UC_recs -.-> UC_notme:::ext
```

Part 3: Sequence Diagrams

1. Authenticate & Take Taste Quiz




```
sequenceDiagram
    participant User
    participant DrinkWiseSystem
    participant AuthService
    participant EmailService
    participant TasteQuizService
    participant PreferenceService

    %% Step 1: Sign Up
    User->>DrinkWiseSystem: Open App / Choose Sign Up
    DrinkWiseSystem-->>User: Show Sign Up Form
    User->>DrinkWiseSystem: Submit Email & Password
```

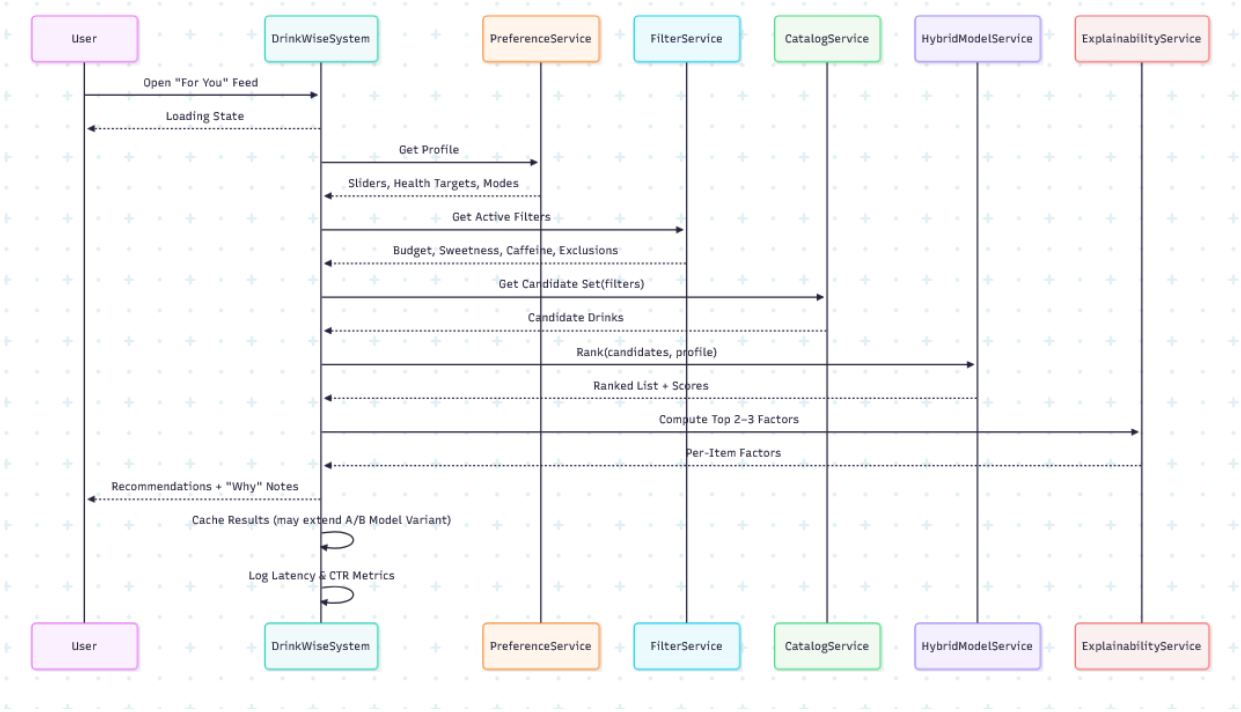
```
%% Step 2: Account Creation & Verification
DrinkWiseSystem->>AuthService: Create Account
AuthService-->>DrinkWiseSystem: Account Created
DrinkWiseSystem->>EmailService: Send Verification Email
EmailService-->>User: Verification Link
User->>AuthService: Click Link -> Verify Token
AuthService-->>DrinkWiseSystem: Verification Success
DrinkWiseSystem-->>User: Welcome Screen (may extend Passwordless Login)

%% Step 3: Taste Quiz
User->>DrinkWiseSystem: Start 60s Taste Quiz
DrinkWiseSystem->>TasteQuizService: Get Questions
TasteQuizService-->>DrinkWiseSystem: 6–10 Questions
DrinkWiseSystem-->>User: Render Quiz
User->>DrinkWiseSystem: Submit Answers

%% Step 4: Save Initial Preferences
DrinkWiseSystem->>PreferenceService: Save Initial Profile(answers)
PreferenceService-->>DrinkWiseSystem: Profile Saved
DrinkWiseSystem-->>User: Show "View Recommendations"

%% Step 5: System Updates
DrinkWiseSystem->>DrinkWiseSystem: Create Session & Seed Cold-Start Flags
DrinkWiseSystem->>DrinkWiseSystem: Log Onboarding Metrics
```

Description:
A new user signs up, verifies their email, completes the 60-second taste quiz, and the system saves an initial preference profile. This flow establishes a secure account and seeds personalization used by later recommendations.

Interactions:
1. Create Account followed by email verification ensures account authenticity before personalization.
2. Quiz questions are fetched from a dedicated service to keep onboarding lightweight and maintainable.
3. Quiz answers are transformed into an initial profile via PreferenceService for immediate recommendation use.
4. System self-updates create a session and log onboarding metrics for product analytics.

Design Decisions:
Email verification is handled by a separate service to decouple delivery from identity management. Taste quiz and preference persistence are split to keep onboarding modular and enable future A/B testing on quiz content independent of profile storage.

2. Generate Personalized Recommendations




```
sequenceDiagram
    participant User
    participant DrinkWiseSystem
    participant PreferenceService
    participant FilterService
    participant CatalogService
    participant HybridModelService
    participant ExplainabilityService

    %% Step 1: Request Recs
    User->>DrinkWiseSystem: Open "For You" Feed
    DrinkWiseSystem-->>User: Loading State

    %% Step 2: Gather Inputs
    DrinkWiseSystem->>PreferenceService: Get Profile
    PreferenceService-->>DrinkWiseSystem: Sliders, Health Targets, Modes
    DrinkWiseSystem->>FilterService: Get Active Filters
    FilterService-->>DrinkWiseSystem: Budget, Sweetness, Caffeine, Exclusions

    %% Step 3: Candidate & Ranking
    DrinkWiseSystem->>CatalogService: Get Candidate Set(filters)
    CatalogService-->>DrinkWiseSystem: Candidate Drinks
```

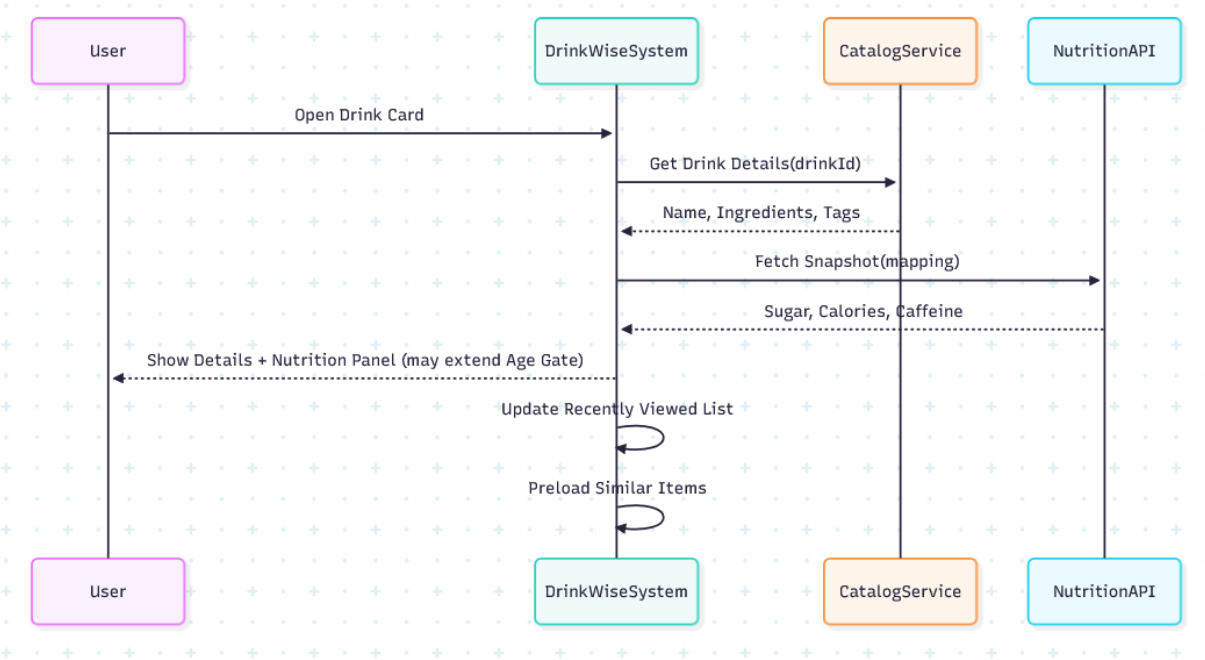
```
DrinkWiseSystem->>HybridModelService: Rank(candidates, profile)
HybridModelService-->>DrinkWiseSystem: Ranked List + Scores

%% Step 4: Explanations
DrinkWiseSystem->>ExplainabilityService: Compute Top 2–3 Factors
ExplainabilityService-->>DrinkWiseSystem: Per-Item Factors

%% Step 5: Return & Update
DrinkWiseSystem-->>User: Recommendations + "Why" Notes
DrinkWiseSystem->>DrinkWiseSystem: Cache Results (may extend A/B Model Variant)
DrinkWiseSystem->>DrinkWiseSystem: Log Latency & CTR Metrics
```

Description:
A signed-in user opens the “For You” feed. The system gathers the user’s preference profile and active filters, narrows candidates, ranks them with a hybrid model, and returns recommendations with brief explanations.

Interactions:
1. Controller composes inputs from PreferenceService and FilterService before any ranking.
2. CatalogService reduces the search space to a candidate set for performance.
3. HybridModelService produces a scored, ranked list; explanations are computed afterward per item.
4. Results are cached and metrics logged to improve latency and monitor quality.

Design Decisions:
Isolation of the ranking logic in HybridModelService enables fast iteration on algorithms without touching orchestration. Explanations are computed by a dedicated service to keep the model service lean and to allow multiple explanation strategies.

3. View Drink Details + Nutrition Snapshot



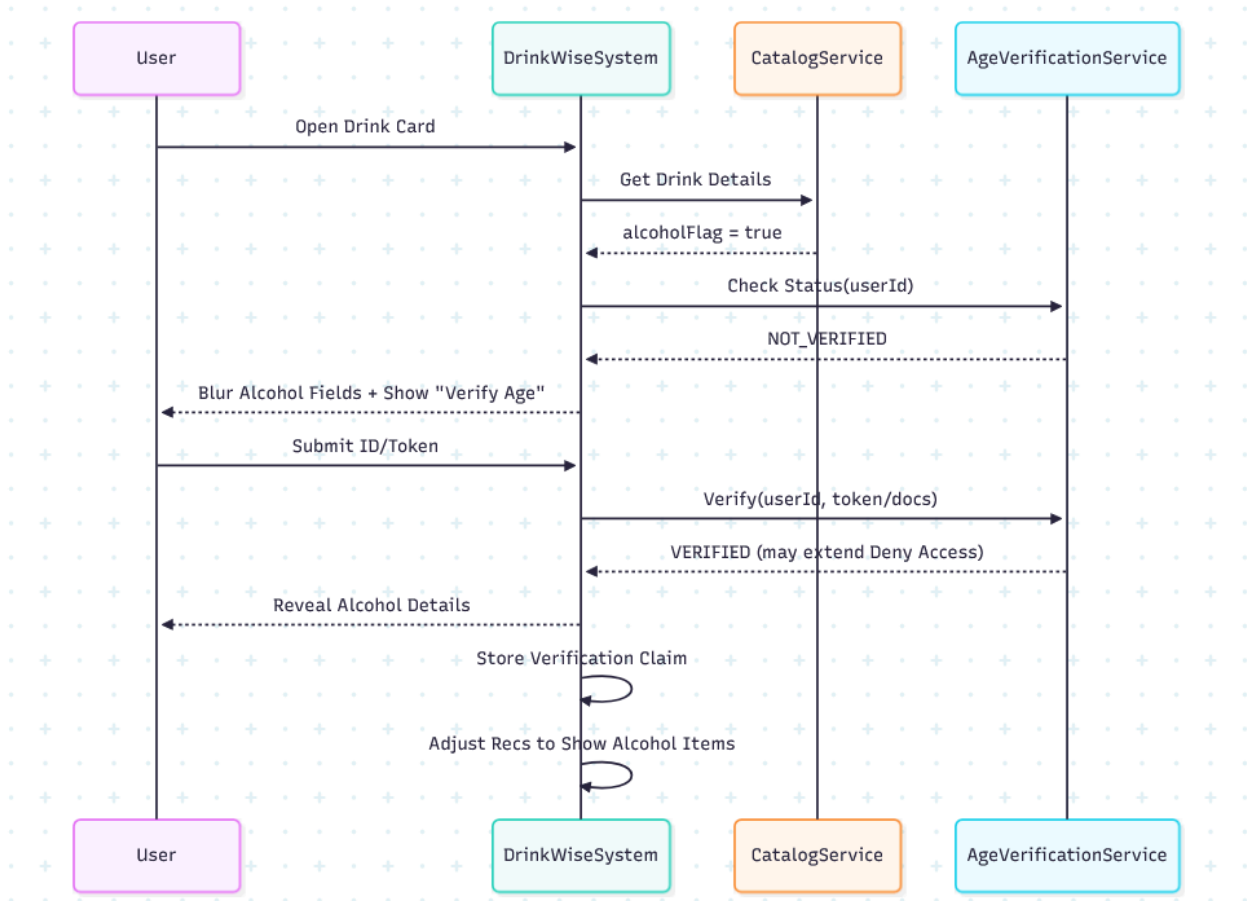

```
sequenceDiagram
  participant User
  participant DrinkWiseSystem
  participant CatalogService
  participant NutritionAPI

  %% Step 1: Open Drink
  User->>DrinkWiseSystem: Open Drink Card
  DrinkWiseSystem->>CatalogService: Get Drink Details(drinkId)
  CatalogService-->>DrinkWiseSystem: Name, Ingredients, Tags

  %% Step 2: Nutrition Panel
  DrinkWiseSystem->>NutritionAPI: Fetch Snapshot(mapping)
  NutritionAPI-->>DrinkWiseSystem: Sugar, Calories, Caffeine

  %% Step 3: Render & Update
  DrinkWiseSystem-->>User: Show Details + Nutrition Panel (may extend Age Gate)
  DrinkWiseSystem->>DrinkWiseSystem: Update Recently Viewed List
  DrinkWiseSystem->>DrinkWiseSystem: Preload Similar Items
```

Description:
When a user opens a drink card, the app fetches canonical details from the catalog and augments them with a concise nutrition snapshot from an external API. The UI composes both sources into the final view.

Interactions:
1. CatalogService is the single source of truth for drink metadata.
2. NutritionAPI augments details with sugar, calories, and caffeine values.
3. UI updates a “recently viewed” list and preloads similar items to improve perceived speed.

Design Decisions:
Separating catalog reads and nutrition lookups allows caching each independently and degrades gracefully if the nutrition API is slow. Preloading likely next views (similar items) improves responsiveness without blocking the details page.

4. Age Gate on Alcohol Details




```
sequenceDiagram
  participant User
  participant DrinkWiseSystem
  participant CatalogService
  participant AgeVerificationService

  %% Step 1: Detect Alcohol
```

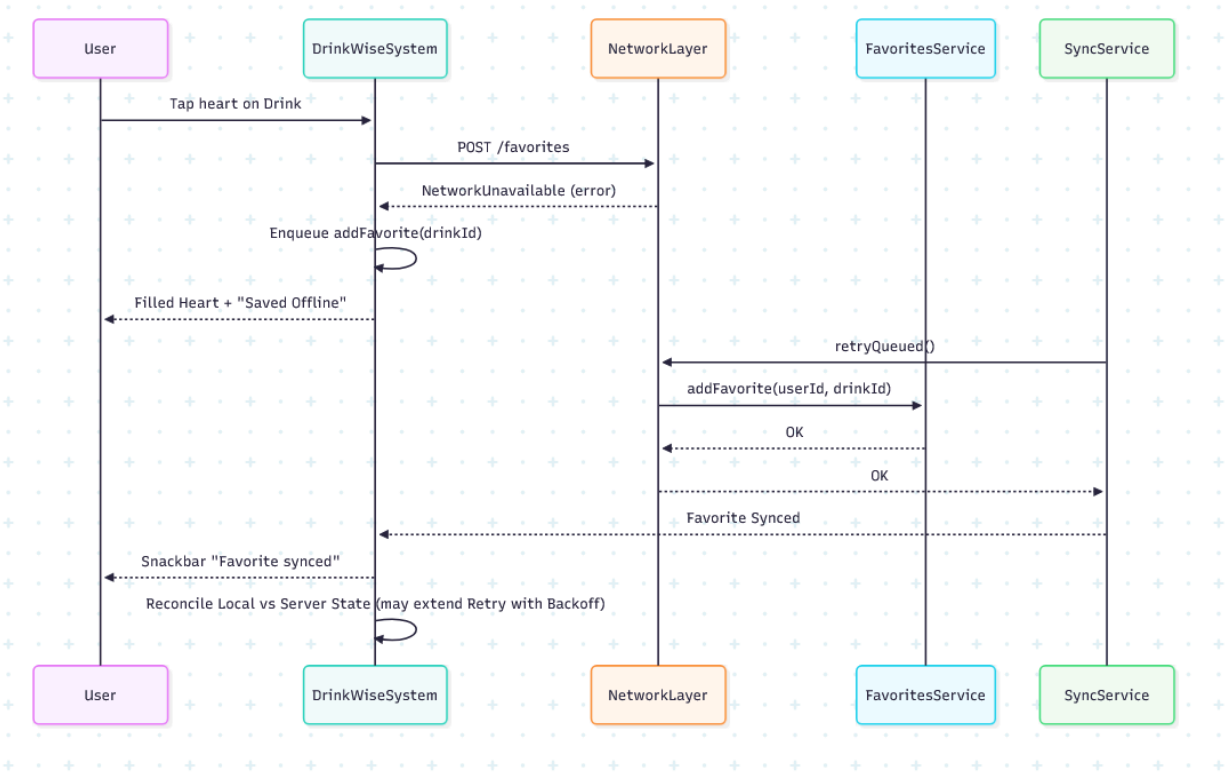
```
User->>DrinkWiseSystem: Open Drink Card
DrinkWiseSystem->>CatalogService: Get Drink Details
CatalogService-->>DrinkWiseSystem: alcoholFlag = true

%% Step 2: Check Verification
DrinkWiseSystem->>AgeVerificationService: Check Status(userId)
AgeVerificationService-->>DrinkWiseSystem: NOT_VERIFIED
DrinkWiseSystem-->>User: Blur Alcohol Fields + Show "Verify Age"

%% Step 3: Verify
User->>DrinkWiseSystem: Submit ID/Token
DrinkWiseSystem->>AgeVerificationService: Verify(userId, token/docs)
AgeVerificationService-->>DrinkWiseSystem: VERIFIED (may extend Deny Access)
DrinkWiseSystem-->>User: Reveal Alcohol Details

%% Step 4: System Updates
DrinkWiseSystem->>DrinkWiseSystem: Store Verification Claim
DrinkWiseSystem->>DrinkWiseSystem: Adjust Recs to Show Alcohol Items
```

Description:
If a drink is flagged as alcohol and the user is not age-verified, the system blurs alcohol fields and prompts verification. Upon successful verification, the full details are revealed.

Interactions:
1. Catalog signals alcohol content; UI responds with a verification check.
2. Unverified users see blurred content and a call to verify.
3. Successful verification immediately unlocks the alcohol details and updates policy state.

Design Decisions:
Gating at the UI after an authoritative status check prevents accidental disclosure while keeping the flow responsive. Persisting the verification claim lets downstream services (e.g., recommendations) honor eligibility without repeated checks.

5. Save Favorite with Offline Queue




```
sequenceDiagram
  participant User
  participant DrinkWiseSystem
  participant NetworkLayer
  participant FavoritesService
  participant SyncService

  %% Step 1: Favorite Action
  User->>DrinkWiseSystem: Tap heart on Drink
  DrinkWiseSystem->>NetworkLayer: POST /favorites
  NetworkLayer-->>DrinkWiseSystem: NetworkUnavailable (error)

  %% Step 2: Enqueue & Feedback
  DrinkWiseSystem->>DrinkWiseSystem: Enqueue addFavorite(drinkId)
  DrinkWiseSystem-->>User: Filled Heart + "Saved Offline"

  %% Step 3: Background Sync
  SyncService->>NetworkLayer: retryQueued()
  NetworkLayer->>FavoritesService: addFavorite(userId, drinkId)
  FavoritesService-->>NetworkLayer: OK
  NetworkLayer-->>SyncService: OK
```

%% Step 4: Confirm & Update
SyncService-->>DrinkWiseSystem: Favorite Synced
DrinkWiseSystem-->>User: Snackbar "Favorite synced"
DrinkWiseSystem->>DrinkWiseSystem: Reconcile Local vs Server State (may extend Retry with Backoff)

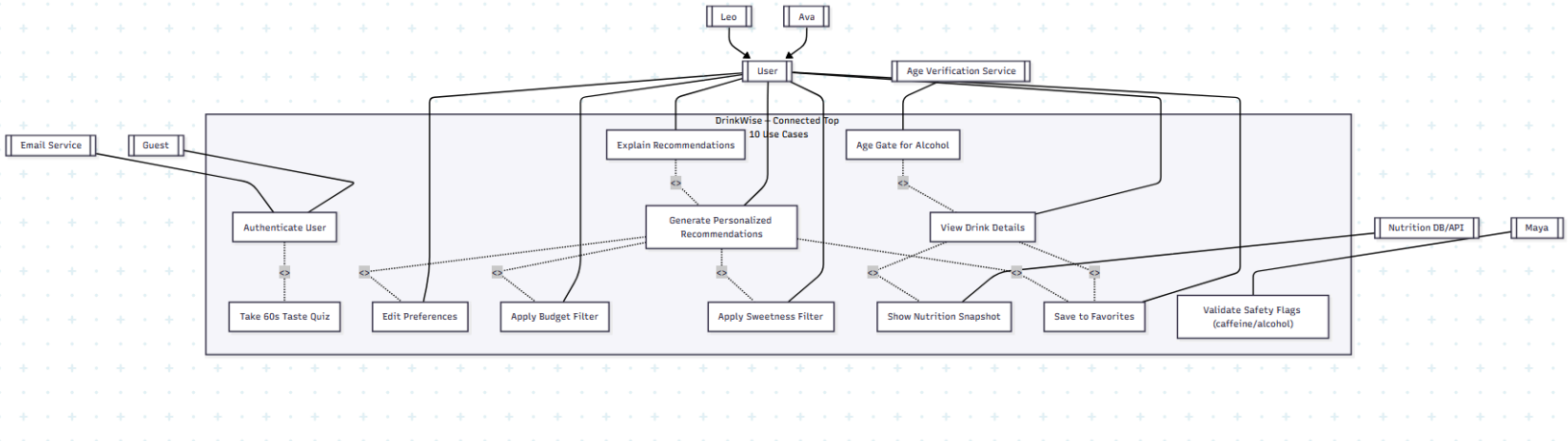
Description:
When a user favorites a drink while offline, the action is enqueued locally and shown optimistically. A background sync retries and persists the favorite once the network is available.

Interactions:
1. A network failure triggers local enqueue of the favorite action.
2. The UI updates optimistically to maintain user momentum.
3. Background SyncService reliably drains the queue and confirms success.
4. Final reconciliation ensures local and server states remain consistent.

Design Decisions:
Optimistic UI plus a durable client-side queue preserves responsiveness in poor connectivity. Separating NetworkLayer from SyncService enables standardized retries/backoff across multiple features, not just favorites.

Part 4: Mermaid Code Already Attached

Part 5: Complete System Use Case Diagram



Code:

```
flowchart TB
  %% ====== Actors & Generalization ======
  guest[[Guest]]
  ava[[Ava]]
  leo[[Leo]]
  User[[User]]
  ava --> User
  leo --> User
  admin[[Maya (Admin)]]
  nutrdb[[Nutrition DB/API]]
  agesvc[[Age Verification Service]]
  emailsvc[[Email Service]]

  %% ====== System Boundary ======
  subgraph System["DrinkWise — Connected Top 10 Use Cases"]
    UC_auth["Authenticate User"]
    UC_quiz["Take 60s Taste Quiz"]
    UC_editPref["Edit Preferences"]
    UC_budget["Apply Budget Filter ($/$$/$$$)"]
    UC_sweet["Apply Sweetness Filter"]
    UC_view["View Drink Details"]
    UC_nutri["Show Nutrition Snapshot"]
    UC_explain["Explain Recommendations"]
    UC_fav["Save to Favorites"]
    UC_safety["Validate Safety Flags (caffeine/alcohol)"]
    UC_age["Age Gate for Alcohol"]
    UC_recs["Generate Personalized Recommendations"]
  end

  %% ====== Actor Associations ======
  guest --- UC_auth
  emailsvc --- UC_auth
  User --- UC_editPref
  User --- UC_budget
  User --- UC_sweet
  User --- UC_view
  User --- UC_explain
  User --- UC_fav
  User --- UC_recs
  admin --- UC_safety
  nutrdb --- UC_nutri
  agesvc --- UC_age

  %% ====== Key Relationships (include/extend/generalization) ======
```

```
%% Auth includes Quiz (as per Part 2 diagram #1)
UC_auth -.<<include>>.- UC_quiz

%% View includes Nutrition (diagram #5)
UC_view -.<<include>>.- UC_nutri

%% Recs includes profile, filters, model, similar (condensed to major drivers here)
UC_recs -.<<include>>.- UC_editPref
UC_recs -.<<include>>.- UC_budget
UC_recs -.<<include>>.- UC_sweet

%% Explain depends on already-computed recommendations (diagram #6)
UC_explain -.<<include>>.- UC_recs

%% Favoriting can originate from viewing or from seeing recs (diagrams #7 & #10)
UC_view -.<<extend>>.- UC_fav
UC_recs -.<<extend>>.- UC_fav

%% Alcohol details only shown when gate passes (diagram #9)
UC_age -.<<extend>>.- UC_view
```

Contributions:

I contributed to Part #3 and worked together with my team to develop Parts 4 and 5. My other teammates completed Parts #1 and #2, and we worked together on Parts #4 and #5. We each contributed 33%.