**Summary of the Two Transcripts on GSSP Architecture and Functional Capabilities**

**1. Overview**

The sessions provided a detailed explanation of the **GSSP (Group Supplemental Services Portal)**, its architecture, functional capabilities, and backend processes. The portal is designed for end users (employees) to manage their benefits, claims, and preferences for various insurance products.

**2. Key Functional Capabilities**

**Products Supported**

* **Core Products**: Disability, Absence, Dental, Vision.
* **Voluntary Benefits (VB)**: Critical Illness, Group Accident, Hospital Indemnity, Health Screening Benefit (HSB), Paid Time, Legal Insurance.

**Portal Features**

1. **Landing Page**:
   * Displays dynamic quick links based on the user's products (e.g., ID card for dental users).
2. **My Accounts**:
   * Shows the user's benefits and product participation details.
3. **Claim Center**:
   * Consolidated view of claims across all products.
   * Includes claim history, associated claims, and downloadable claim summaries.
4. **Documents and Forms**:
   * Displays correspondence and forms related to the user's products.
5. **Profile and Preferences (PNP)**:
   * Allows users to update preferences, manage profiles, and link accounts.
6. **Customer Support and Message Center**:
   * AEM-driven content for support and notifications.

**3. Backend Architecture**

**Login and Authentication**

* **MIS (MetLife Identity Services)**:
  + Handles user authentication.
* **USD Login**:
  + Routes users to the appropriate portal (e.g., GSSP, employer portal).
  + Performs **Check Member Profile (CMP)** to validate user details (e.g., e-consent, security questions).

**Hydration Flow**

* **Purpose**: Prefetch and cache user data for faster page loads.
* **Steps**:
  1. **Context Call**:
     + Triggered by USD Login to fetch user profile and product information.
  2. **Profile Service**:
     + Retrieves user roles, preferences (via EDPM), and product participation details (via UIS and Profile DB).
  3. **Accounts and Claims Services**:
     + Fetches benefits and claims data from respective systems (e.g., UCS for dental, entity for ANH).
  4. **View Services**:
     + Combines SOR (System of Record) data with AEM content.
     + Caches the hydrated data in MongoDB for quick UI rendering.

**Caching and Data Management**

* **MongoDB Collections**:
  + **US Online**: Stores raw SOR data.
  + **US Customer Web**: Stores hydrated data (SOR + AEM content).
* **TTL (Time-to-Live)**:
  + Cached data is purged after a set duration (e.g., 24 hours) or when the user logs out.
* **Event-Driven Architecture**:
  + RabbitMQ is used to emit events (e.g., profile ready, accounts ready) for parallel processing.

**4. Claims and Accounts Processing**

**Accounts Service**

* Splits user data into threads for each product (e.g., dental, disability, ANH).
* Fetches product-specific details (e.g., coverage, certificates).
* Emits "accounts ready" events for further processing.

**Claims Service**

* Fetches claims data from respective systems (e.g., entity for ANH, UCS for dental).
* Emits events for each claim to hydrate claim cards.

**View Services**

* **Page View Service**:
  + Handles common UI elements (e.g., headers, filters).
* **Account View Service**:
  + Hydrates account-specific cards.
* **Claim View Service**:
  + Hydrates claim-specific cards.

**5. Technical Challenges and Improvements**

* **Legacy Systems**:
  + Transition from Meta Online to USD Login improved routing and CMP processes.
* **Performance Issues**:
  + Initial event-driven architecture faced latency issues as more products were onboarded.
  + Modernization efforts (e.g., Cosmos for data storage) have improved performance.
* **Redundant Data Fetching**:
  + Prefetching all data, even if unused, increases load times and complexity.

**6. Key Takeaways**

* **Dynamic User Experience**:
  + The portal adapts to the user's products and preferences.
* **Backend Optimization**:
  + Hydration flow and caching ensure faster page loads.
* **Future Enhancements**:
  + Redesign efforts are underway for better UI and streamlined backend processes.

**7. Outstanding Items**

* **ANH Redesign**:
  + Upcoming sessions will focus on changes specific to ANH claims and accounts.
* **Documentation**:
  + Additional details on MongoDB collections and backend flows will be shared in the OneNote repository.

**Conclusion**

The sessions provided a comprehensive overview of GSSP's architecture and functional capabilities, highlighting its user-centric design, backend processes, and technical challenges. The focus on caching, event-driven architecture, and dynamic content ensures a seamless user experience while addressing scalability and performance issues.

**Detailed Explanation of the Hydration Flow in GSSP**

The **hydration flow** in GSSP (Group Supplemental Services Portal) is a backend process designed to prefetch, process, and cache user-specific data from multiple systems of record (SORs) and content sources. The goal is to ensure that the portal loads quickly and provides a seamless user experience by reducing the need for repeated backend calls during navigation.

**1. Purpose of Hydration Flow**

* **Prefetch Data**: Gather all necessary user data (e.g., profile, benefits, claims) at the time of login.
* **Cache Data**: Store the processed data in MongoDB collections for quick access during the user's session.
* **Reduce Latency**: Minimize backend calls during navigation by preparing the data in advance.
* **Merge Data**: Combine raw SOR data with AEM (Adobe Experience Manager) content to create a unified view for the UI.

**2. Steps in the Hydration Flow**

**Step 1: Context Call**

* **Trigger**: The hydration flow begins when the user logs in and USD Login triggers a **context call** to the **User Profile Service**.
* **Purpose**: The context call provides minimal user information (e.g., MetLife ID, login ID) to initiate the data-fetching process.

**Step 2: Profile and Product Information**

* **User Profile Service**:
  + Fetches user roles, preferences, and basic profile details.
  + Key systems involved:
    - **IBSC**: Retrieves user roles and unique identifiers (e.g., MetLife ID).
    - **EDPM**: Fetches delivery preferences (e.g., email, mail) and profile details (e.g., e-consent, terms acceptance).
    - **Product Holdings**: Retrieves product participation details (e.g., policies, coverage) from **Profile DB**.
  + **Output**: Emits a "profile ready" event and stores raw SOR data in the **US Online** MongoDB collection.

**Step 3: Benefits Information**

* **Account Service**:
  + Fetches benefits-related data for each product the user participates in.
  + Splits the process into parallel threads for each product (e.g., dental, disability, ANH).
  + Key systems involved:
    - **UCS/Nova**: Fetches benefits data for dental products.
    - **Entity**: Fetches policy and certificate details for ANH products.
    - **Other Systems**: Fetches data for disability and absence management.
  + **Output**: Emits "accounts ready" events for each product and stores the data in the **US Online** collection.

**Step 4: Claims Information**

* **Claim Service**:
  + Fetches claims-related data for each product.
  + Key systems involved:
    - **Entity**: Fetches claims for ANH products.
    - **UCS**: Fetches claims for dental products.
    - **Other Systems**: Fetches claims for disability and absence management.
  + **Output**: Emits "claims ready" events for each claim and stores the data in the **US Online** collection.

**Step 5: Data Merging and Caching**

* **View Services**:
  + Combines raw SOR data (from **US Online**) with AEM content (e.g., labels, templates).
  + Hydrates the data into a unified format for the UI.
  + Stores the hydrated data in the **US Customer Web** MongoDB collection.
  + Key View Services:
    - **Account View Service**: Hydrates account-related data (e.g., benefits cards).
    - **Claim View Service**: Hydrates claim-related data (e.g., claim cards).
    - **Page View Service**: Hydrates common page elements (e.g., headers, filters).

**3. Data Flow in MongoDB**

* **US Online Collection**:
  + Stores raw SOR data fetched from external systems.
  + Example: Policy details, claim details, user roles.
* **US Customer Web Collection**:
  + Stores hydrated data (SOR + AEM content) for UI rendering.
  + Example: Fully prepared account and claim cards with labels and templates.

**4. Event-Driven Architecture**

* **RabbitMQ**:
  + Used to emit events (e.g., "profile ready," "accounts ready," "claims ready") during the hydration flow.
  + Enables parallel processing of data for multiple products and claims.
* **Parallel Threads**:
  + Each product (e.g., dental, disability, ANH) is processed in its own thread to improve performance.

**5. Caching and TTL (Time-to-Live)**

* **Session-Based Caching**:
  + Data is cached in MongoDB for the duration of the user's session.
  + Data is purged when the user logs out or after a period of inactivity (e.g., 30 minutes).
* **TTL for Collections**:
  + Cached data is automatically deleted after a set duration (e.g., 24 hours for some collections, 4 hours for others).
  + Ensures that stale data is not served to users.

**6. Benefits of Hydration Flow**

* **Improved Performance**:
  + Reduces backend calls during navigation by prefetching and caching data.
* **Seamless User Experience**:
  + Ensures that pages load quickly with minimal latency.
* **Scalability**:
  + Parallel processing and event-driven architecture allow the system to handle multiple products and claims efficiently.

**7. Challenges**

* **Data Redundancy**:
  + Prefetching all data, even if unused, can lead to unnecessary processing and storage.
* **Complexity**:
  + Handling multiple products and claims with different SORs and templates increases system complexity.
* **Latency in Event Processing**:
  + RabbitMQ events may face delays, requiring fallback mechanisms (e.g., hack calls).

**8. Future Enhancements**

* **Selective Prefetching**:
  + Fetch only the data relevant to the user's immediate actions to reduce overhead.
* **Redesign for ANH**:
  + Simplify the hydration flow for ANH products by reducing dependencies on legacy systems and templates.

**Conclusion**

The hydration flow is a critical component of GSSP's architecture, enabling fast and seamless user interactions by prefetching, processing, and caching data. While it provides significant performance benefits, ongoing enhancements are needed to address challenges like data redundancy and system complexity.