Bulky Items Pilot Solution Proposal

Executive Summary

As an implementation of NodeJS leveraging TypeScript and cloud hosting services, the Bulky Items Pilot demonstrates an integrated pipeline from development to production.

Composability and Extendibility / Separation of Concerns : Bulky Items Pilot will leverage multiple services and applications to provide complete functionality. By separating these business logic needs into services such portions can be easily understood and decomposed. This allows for future development of systems that extend functionality while keeping such changes limited within their own contexts and without compromising the rest of the system.

Quality Assurance and Testability : Unit tests and Integration tests are essential to the quality and validated execution of the system as a whole and as individual components.

Maintainability and Risk Tolerance : Each service is self-contained and therefor, deployable within its own context, this allows for deployment of updates in a way that does not compromise the system as a whole. Maintainability is ensured by keeping services small enough for a single developer to fully understand and implement changes.

Design and Use Considerations

Considering both implementations for the Frontend Applications to customers and internal members, such application will also be expected to function in an iframe and standalone environment. A Single Page Application implementation is recommended. This will be accomplished by creating separate artifacts of both the Customer and Internal portals, however, which share common functionality through a shared library. This implementation relies on JavaScript to be enabled within the browser and does not supply fallback functionality. For customers there will be no sessions or client-saved identifiers, in the case of a user refreshing the host browser page, no (in-memory) saved information from API Services would be sent until client dual-factor identifies, at which point, they would continue from where they had left off. For Internal CSR a short-term token is distributed and expected to be kept and included during transmission until expiration.

Architecture Decisions

Small codebase, few dependencies, horizontal scalability. To ensure sensitive data remains private we will be leveraging HTTPS only (http auto upgrade requests) connections through the publicly facing services. Internal/Private network applications will only accept connections originating from within the internal VPC (AWS security groups). Data pertaining to Bulky Item Pickup Requests will be written to a SQL database with corresponding internal network rules. Private Network APIs require authentication before PII requests are honored, present in both the request headers and accompanying payload as an API issued transient (time limited) token. To ensure data privacy on customer / shared systems, no cookies, local storage, or long-term identifiers will be issued, and no information will be kept between client sessions. Transient PII (including auth tokens) will be stored in the Private Network and will be removed upon successful/initiated outcomes or transient tokens expiring. Leveraging AWS S3 for static asset serving and API Gateway as method of Service Discovery.

Workflow

1. Infrastructure A: API Gateway
   1. Route incoming requests to internal Utility & Data Access Lambda
2. Infrastructure B: AWS S3
   1. Public access to static assets
   2. File and HTML template serving through HTTPS
3. Frontend Applications
   1. Customer Facing Portal
      1. HTTPS requests to API Gateway
      2. No long term, session, or cookie storage of PII
      3. Walkthrough Wizard
   2. Internal Customer Support Portal
      1. HTTPS requests to API Gateway
      2. Walkthrough Wizard (ext)
      3. Authentication short-term tokens kept in client
      4. Daily Logs (sort by date, print by type/all)
      5. Close Work Orders (sort by date, search by work order #id, mark notes, extra charges)
4. Lambda A: Utility & Data Access Lambda
   1. Inbound Internal traffic rule, non-accessible to public
   2. Communication with GIS
   3. Database Communication for Storing and Managing TS Tracking of BIP Requests
   4. (Read-Remove) Authentications for Data Retrieval
5. Lambda B: Auth Lambda
   1. Inbound Internal traffic rule, non-accessible to public
   2. Authenticate identity requests through Utiligy (dual-factor), Generated Auth Token
   3. Authenticate CSR identity requests through Okta, Generated Auth Token
   4. (Read-Write-Remove) Transient Auth Token allocation for Data Retrieval / PII requests
6. Database A: TS RDBMS Request Database
   1. Inbound traffic rules only from VPC originated requests
   2. Implement tables for storing Customer and Internally generated requests
   3. Storage of Authentication Tokens and associated Pickup PII
7. Mock Service A: Notification Manager
   1. Receive Notification requests but not act upon them
8. Continuous Unit / Integration Testing
9. Staging and Deployment Considerations
10. QA and Use Case Validation
11. Production Deployment