OneInc

Long-running Job

Contents

Executive Summary

This solution is designed to manage and process long-running and heavy tasks in a secure, scalable, and reliable way. The architecture follows the C4 model to clearly define the system context, containers, components, and deployment structure.

The system includes a web application built with React, backend services in .NET Core, and Azure cloud services such as Azure Functions, Azure Queue Storage, and API Management. It uses event-driven architecture with REST APIs and azure functions to ensure flexibility and easy scaling.

The solution is designed to process messages regardless of execution time, ensuring strong performance and reliability. The user interface will be compatible with modern browsers and previous versions.

The system is divided into clear modules:

- Encode API to create and manage long-running jobs.
- Durable Functions to orchestrate and manage job execution.
- Azure Functions to process background tasks.
- A React-based web interface for users.

The project will follow a Time and Materials (T&M) model using Agile Scrum with two-week sprints. The delivery includes four phases: Discovery and Design, Implementation, Testing, and Deployment. Automated testing and security validation will ensure quality before production release.

Overall, this solution provides a modern, secure, and scalable cloud-based platform that supports long-running job processing with high availability, strong security, and flexible delivery.

Solution Approach

Architecture Overview

The solution will adopt the C4 model for clarity:

System Context: Long-running job which processes heavy tasks.

Containers: Web UI, API Gateway, application services, database (storage table), cache (Redis), and integration middleware (storage queue).

Components: Within each service, core modules manage the long-running job lifecycle.

In this sense, the solution will provide:

- Scalability & Performance: Event-driven architecture with REST APIs and Azure functions that ensure modularity and horizontal scaling.
- Security: SSO with OAuth2/SAML, role-based access, and audit trails.
- Compliance: WCAG 2.0 accessibility, coding standards, and enterprise guidelines.

Architecture Significant Requirements ASRs

- The application should have three nines of availability.
    - Quality Attributes: Availability.
    - Constraints: None
- The system should process the messages regardless of the execution time.
    - Quality Attributes: Performance.
    - Constraints: None.
- The UI should have backward compatibility across all browsers and their previous versions.
    - Quality Attributes: Compatibility.
    - Constraints: None.
- To ensure that all requests are made by authenticated users and to enhance security and usability, the system must implement Single Sign-On (SSO). This will provide a centralized authentication mechanism, reducing the need for multiple credentials across services and ensuring consistent access control.
    - Quality Attributes: Security, Usability.
    - Constraints: None.
- The system must support near-real-time notifications for critical business events or system errors. This is essential to ensure responsiveness and enable rapid incident response, minimizing potential business impact.
    - Quality Attributes: Reliability.
    - Constraints: None.

Functional Decomposition

- Encode API: Module to create and manage long-running jobs.
- Long-running durable function: Module to orchestrate new executions according to the new request from the queue.
- Long-running functions: Component to process the long-running executions.
- Web app UI: Interface that provides all the features that are required for the users.

Integration

Authentication: Claims-based SSO, compatible with enterprise identity provider.

Technology Stack

Frontend: React + modern UI frameworks.

Backend: .NET Core 10.

Databases: Storage Account – Table Storage.

Infra: Azure Functions, Durable function actions, API Management, Azure Queue Storage, Azure SignalR Service.

Delivery Approach

Engagement Model

This project will follow a Time and Materials (T&M) approach, allowing flexibility to adapt to evolving requirements and priorities identified during the Agile iterations. The T&M model ensures that the customer only pays for the actual effort spent, promoting transparency and continuous collaboration throughout the delivery.

Methodology

We will follow an Agile Scrum approach with 2-week sprints, ensuring transparency, incremental delivery, and early feedback.

Phases

1. **Discovery & Design**
- Architecture & UX prototypes.

2. **Implementation**
- Develop product.
- Integrate SSO, APIs, and notifications.

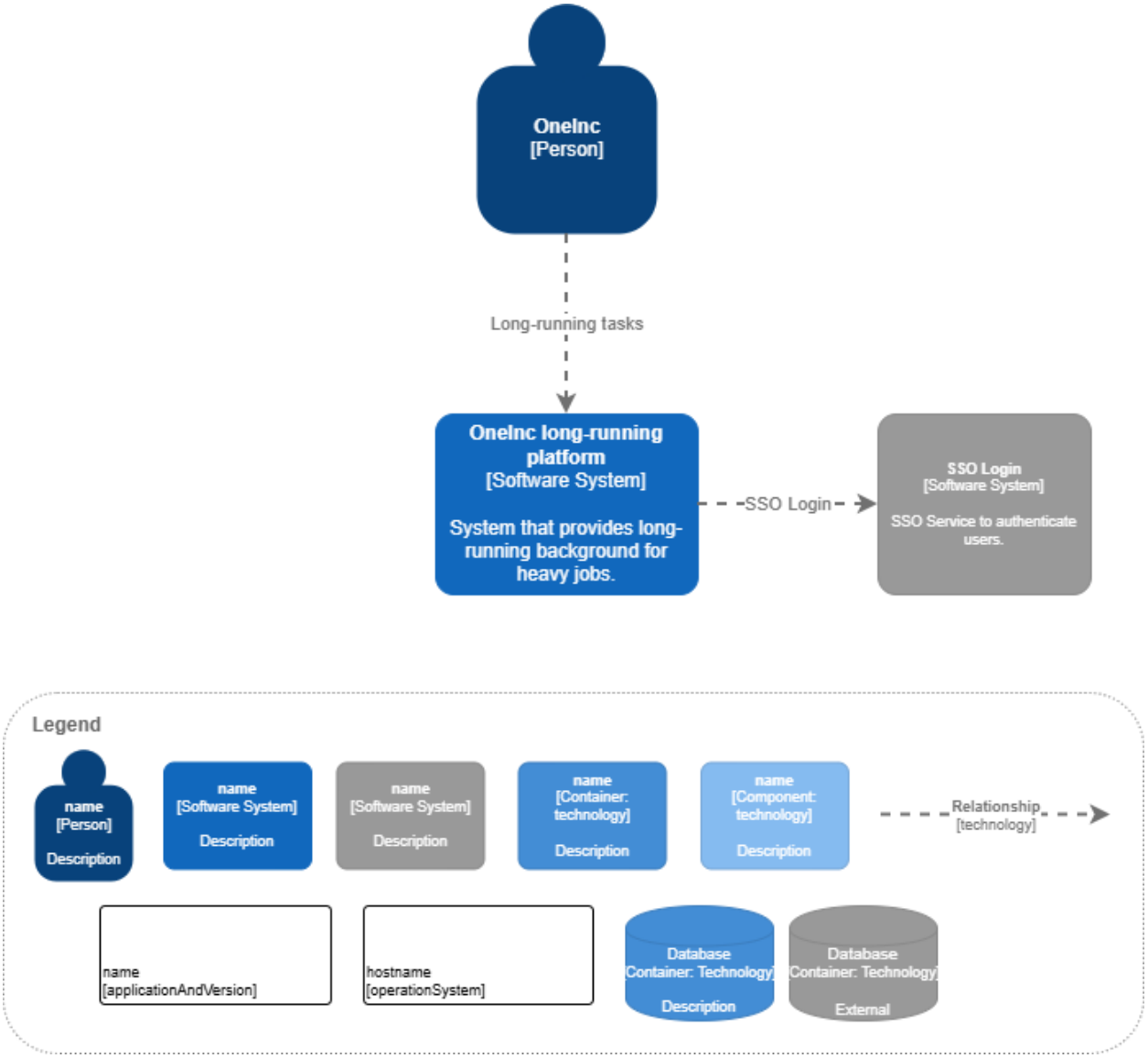
3. **Testing**
- Automated testing (unit, integration, performance).
- Security

4. **Deployment & Transition**
- Rollout to production, monitoring setup, and knowledge transfer.

Context Diagram

This diagram gives a high-level view of the system. The platform connects with the SSO Login System, which is used to authenticate users securely.

This view explains the system's main actors, its boundaries, and how it interacts with external systems.

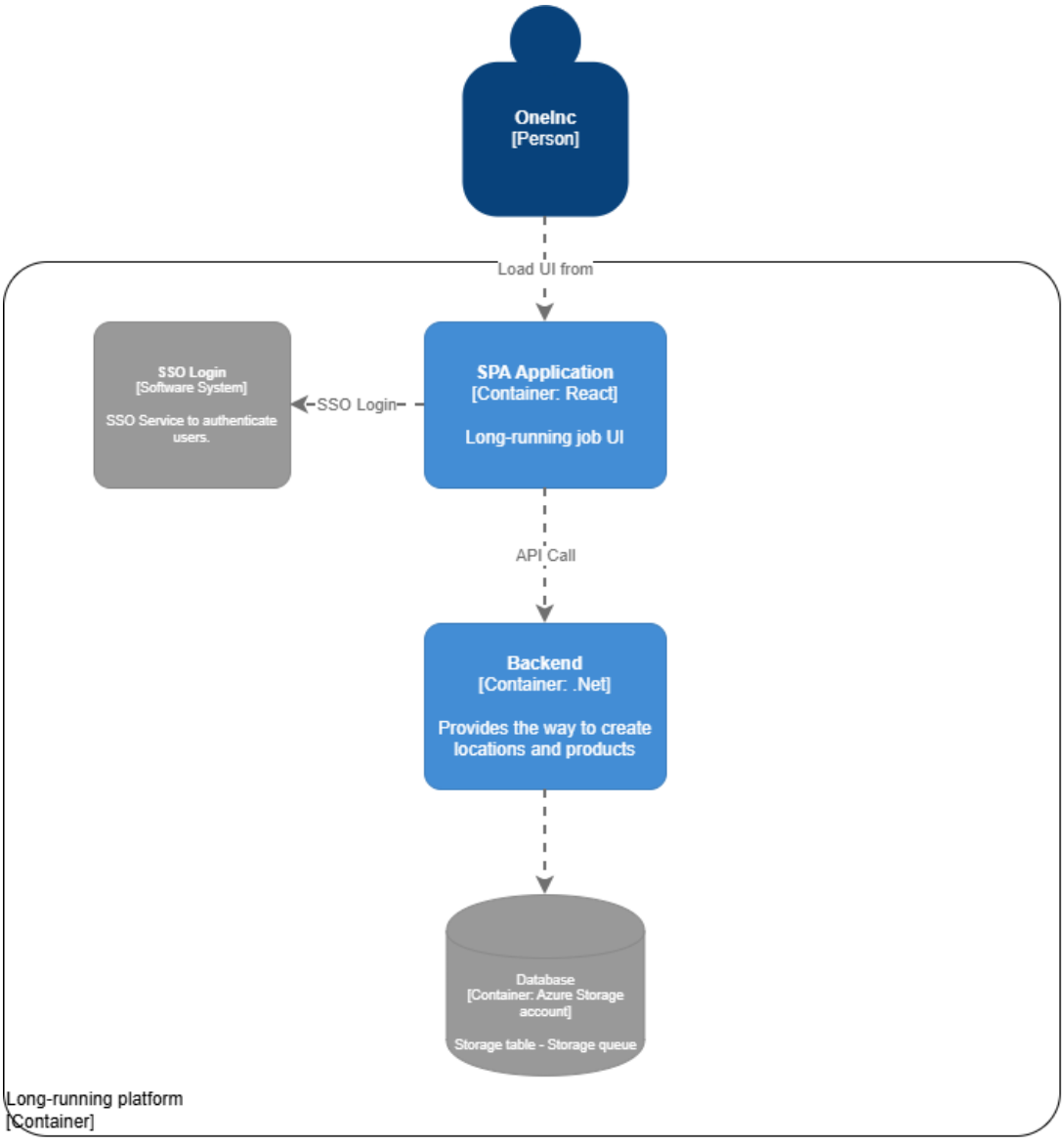


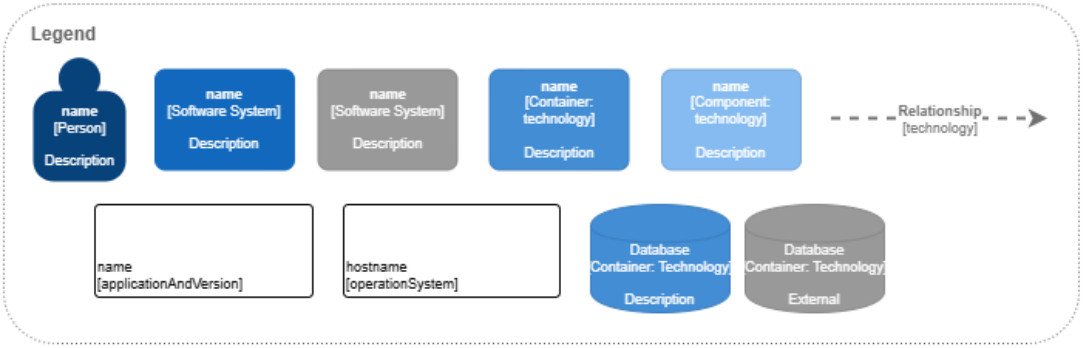
Container Diagram

This diagram divides the system into several containers:

- SPA Application (React): The web interface that users see and interact with.
- Backend (.NET): Manages creation and updates of long-running jobs.
- Databases (Azure storage account): Store information for long-running jobs and manage queue.
- SSO Login System: External service that provides secure login for users.

The Company (Client) accesses the SPA Application, which calls the backend APIs to perform actions. This diagram shows how the system is organized in layers user interface, business logic, and data making it easier to scale and maintain.


OneInc
[Person]
Load UI from
SSO Login
[Software System]
SSO Service to authenticate users.
SSO Login
SPA Application
[Container: React]
Long-running job UI
API Call
Backend
[Container: .Net]
Provides the way to create locations and products
Database
[Container: Azure Storage account]
Storage table - Storage queue
Long-running platform
[Container]


Legend
name
[Person]
Description
name
[Software System]
Description
name
[Software System]
Description
name
[Container: technology]
Description
name
[Component: technology]
Description
Relationship
[technology]
name
[applicationAndVersion]
hostname
[operationSystem]
Database
Container: Technology
Description
Database
Container: Technology
External

Component Diagram

This diagram shows the main components inside the front-end and back-end systems and how they connect.

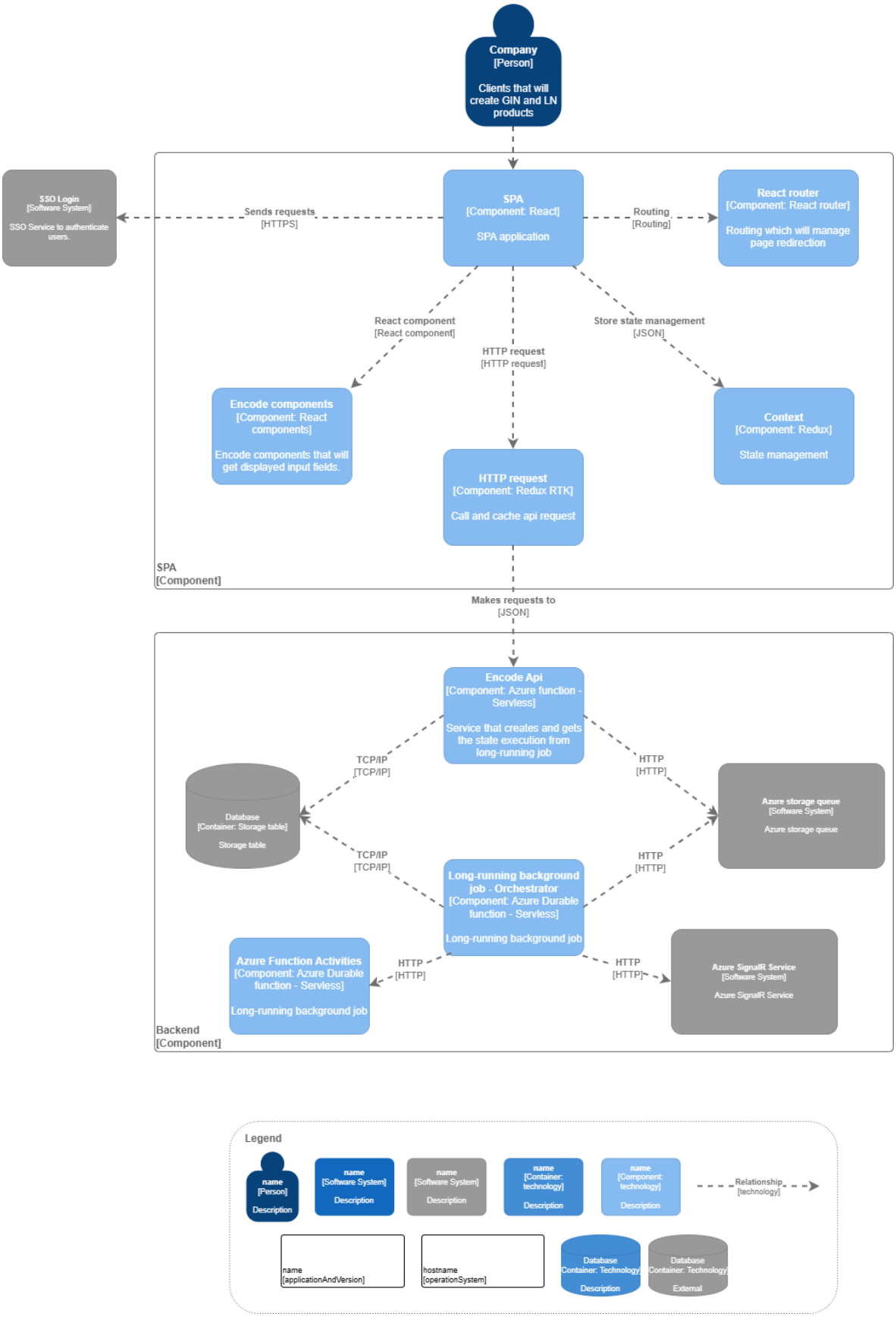
Front-End (SPA Application):

- SPA Component (React): Manages the web pages and routes.
- Encode components: Let users create and edit identifiers.
- HTTP Request Layer (React Query): Sends and receives data from APIs.
- Context (React Context/Redux): Manages shared data and session state.
- SSO Login: External component that handles user authentication.

Back-End services:

- Encode API (.NET): Creates and manages long-running jobs.
- Long-running Orchestrator (Azure Functions): Get new messages and start a new execution.
- Long-running Activities (Azure Functions): Functions that will process the messages.
- Databases (Azure storage table): Store the state of each execution.
- Azure Service Queue: Messages to be processed.
- Azure SignalR Service: Service to push live updates to the UI.

This diagram shows how the internal parts work together, helping to keep the system modular, flexible, and easy to extend.


Company
[Person]
Clients that will create GIN and LN products
SSO Login
[Software System]
SSO Service to authenticate users.
Sends requests
[HTTPS]
SPA
[Component: React]
SPA application
Routing
[Routing]
React router
[Component: React router]
Routing which will manage page redirection
React component
[React component]
HTTP request
[HTTP request]
Store state management
[JSON]
Encode components
[Component: React components]
Encode components that will get displayed input fields.
HTTP request
[Component: Redux RTK]
Call and cache api request
Context
[Component: Redux]
State management
SPA
[Component]
Makes requests to
[JSON]
Encode Api
[Component: Azure function - Serviless]
Service that creates and gets the state execution from long-running job
TCP/IP
[TCP/IP]
HTTP
[HTTP]
Database
[Container: Storage table]
Storage table
Azure storage queue
[Software System]
Azure storage queue
TCP/IP
[TCP/IP]
HTTP
[HTTP]
Long-running background job - Orchestrator
[Component: Azure Durable function - Serviless]
Long-running background job
Azure Function Activities
[Component: Azure Durable function - Serviless]
Long-running background job
HTTP
[HTTP]
HTTP
[HTTP]
Azure SignalR Service
[Software System]
Azure SignalR Service
Backend
[Component]
Legend
name
[Person]
Description
name
[Software System]
Description
name
[Software System]
Description
name
[Container: technology]
Description
name
[Component: technology]
Description
Relationship
[technology]
name
[applicationAndVersion]
hostname
[operationSystem]
Database
Container: Technology
Description
Database
Container: Technology
External

Deployment Diagram

This diagram shows how the web application and its services are deployed and connected in the cloud using Microsoft Azure.

1. Front door and WAF

- The request is first routed through the Front Door service (e.g., CDN/Global Load Balancer), which provides global entry point routing, SSL termination, and performance optimization.
- A Web Application Firewall (WAF) inspects incoming traffic to protect against common web vulnerabilities and attacks (e.g., SQL injection, XSS, OWASP Top 10 threats).
- The WAF enforces security policies, rate limiting, IP filtering, and bot protection before forwarding validated traffic to the backend application services.

2. User Access and Authentication

- The user opens the web application through a browser.
- The user signs in using SSO (Single Sign-On), which provides secure and centralized authentication.
- After signing in, the user can use the web app to send and receive data.

3. Gateway and API Management

- The Gateway receives all requests from the web app and sends them to the right services.
- APIM (Azure API Management) controls and manages all the APIs. It helps with security, routing, and monitoring of API calls.

4. Encode API

- Create and manage long-running job tasks.
- A Redis Cache is used to speed up data access and reduce load on the databases.

5. Storage Queue and Azure Functions

- The Storage Queue acts as a message broker that allows services to communicate in a reliable and asynchronous way.
- It sends messages and events to different Azure Functions, such as:

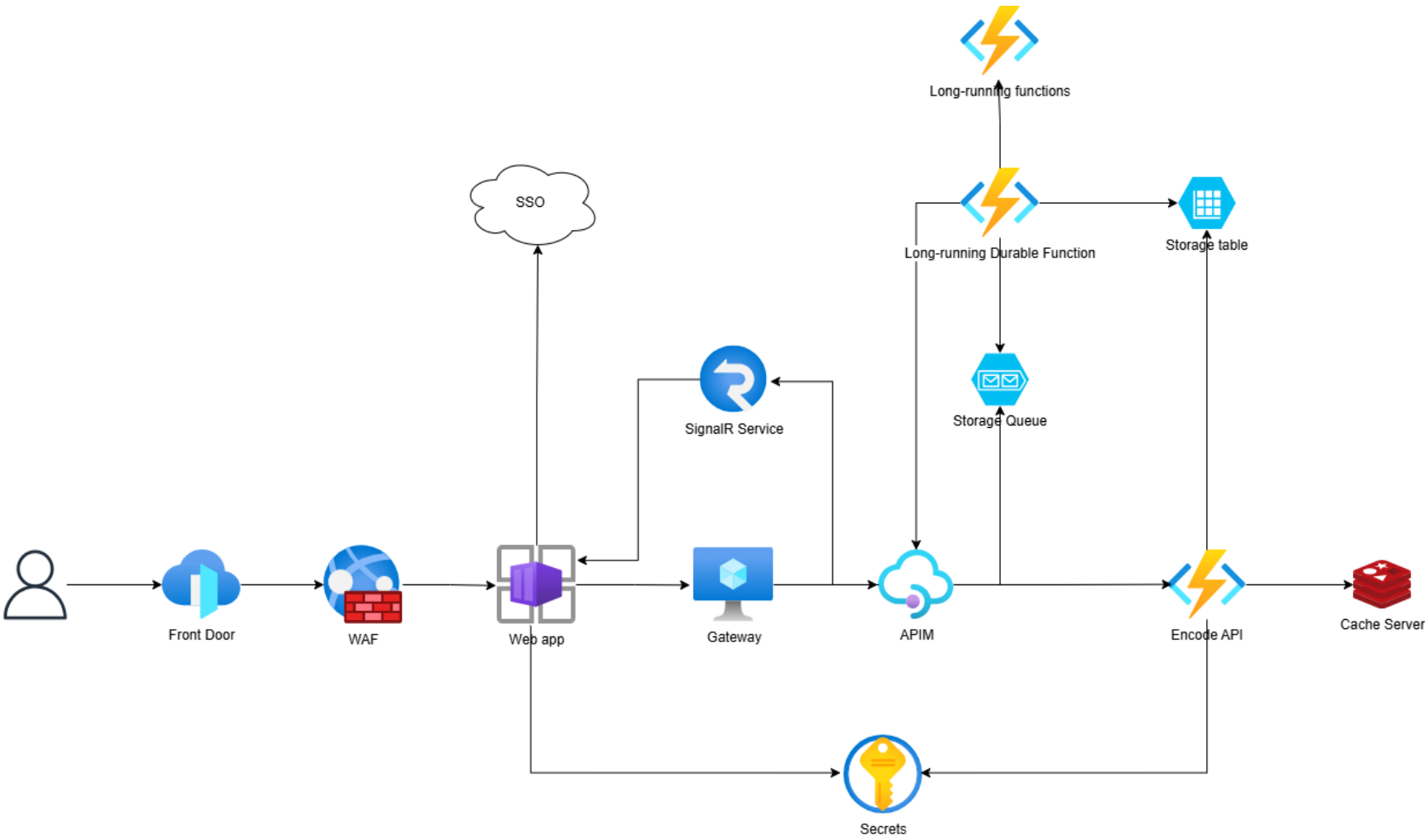
    - Azure durable functions.
- These functions process incoming messages and perform automated or event-driven tasks.

6. Databases

- Storage table that manages long-running job data.

7. Azure SignalR Service

- Service that receives the latest state and pushes live updates about long-running jobs.


Long-running functions
SSO
Long-running Durable Function
Storage table
SignalR Service
Storage Queue
Front Door
WAF
Web app
Gateway
APIM
Encode API
Cache Server
Secrets

Dependencies

- The Single Sign-On (SSO) system.

Risks

- If SSO or Identity Management is delayed, integration and testing may be late.
- Security issues may happen if external libraries are not updated.

Assumptions

- Users will use browsers listed in the compatibility requirements.
- Requirements will evolve, but changes will follow a formal change management process.
- No message ordering required (FIFO algorithm).
- There is no relational data or complex joins.