



UNIVERSIDAD TÉCNICA DE AMBATO
FACULTAD DE INGENIERÍA EN SISTEMAS ELECTRÓNICA E INDUSTRIAL
CARRERA DE TECNOLOGÍAS DE LA INFORMACIÓN
CICLO ACADÉMICO: SEPTIEMBRE – ENERO 2026



Tema:

Integración de Sistemas de Datos y Desarrollo de Dashboards para Análisis de Ventas

Unidad de Organización Curricular:

PROFESIONAL

Nivel y Paralelo:

Octavo "A"

Alumnos participantes:

Bryan David Coello Macias

Gabriel Alejandro Ocampo Tixi

Mateo Nicolas Palate Dominguez

Asignatura:

Integración de Sistemas

Docente:

Ing. Jose Ruben Caiza Caizabuan



2.1 Objetivos

General:

Implementar una solución integral de inteligencia de negocios que abarque desde la extracción, transformación y carga (ETL) de datos heterogéneos hasta su visualización en dashboards interactivos, integrando seguridad mediante autenticación externa.

Específicos:

- Configurar un entorno de control de versiones distribuido utilizando Git para la gestión colaborativa del código fuente del Backend, Frontend y procesos ETL.
- Desarrollar procesos ETL con Pentaho Data Integration para la ingesta y limpieza de datos provenientes de archivos CSV y bases de datos transaccionales hacia un Data Warehouse en PostgreSQL.
- Implementar una capa de seguridad y gestión de identidad utilizando Keycloak desplegado en un entorno virtualizado.
- Construir una interfaz de usuario web con Angular para la visualización de indicadores clave de rendimiento (KPIs) relacionados con ventas, clientes y productos.

2.2 Modalidad

Presencial.

2.3 Instrucciones

- Clonar el repositorio del proyecto y configurar las variables de entorno necesarias para la conexión con la base de datos y el servidor de autenticación.
- Desplegar la base de datos PostgreSQL y el servidor Keycloak en una máquina virtual (VirtualBox) asegurando la conectividad de red con la máquina host.
- Ejecutar los scripts de creación del Data Warehouse y poblar las tablas de dimensiones y hechos utilizando las transformaciones de Pentaho (.ktr y .kjb).
- Iniciar los servicios de Backend (Node.js/Express) y Frontend (Angular) para verificar la integración de los componentes.
- Documentar el proceso de desarrollo, los desafíos técnicos encontrados y las soluciones implementadas.

2.4 Listado de equipos, materiales y recursos

- Equipos: Computadoras personales con capacidad de virtualización.
- Software: Git, Pentaho Data Integration (PDI), PostgreSQL, Keycloak, Node.js, Angular CLI, VirtualBox, Visual Studio Code.
- Recursos: Conexión a internet, Repositorio de código fuente (GitHub).

TAC:

- ☒ Plataformas educativas
- ☐ Simuladores y laboratorios virtuales
- ☐ Aplicaciones educativas
- ☒ Recursos audiovisuales
- ☐ Gamificación
- ☒ Inteligencia Artificial

Otros (Especifique): Herramientas de control de versiones (Git), Herramientas de Virtualización.



2.5 Actividades por desarrollar

- Configuración del Entorno y Repositorio: Inicialización del repositorio Git y estructuración de directorios para Backend, Frontend, ETL y documentación.
- Despliegue de Infraestructura Base: Instalación y configuración de PostgreSQL y Keycloak en un servidor virtualizado (VirtualBox).
- Diseño e Implementación del Data Warehouse: Modelado del esquema en estrella (Star Schema) e implementación de tablas de dimensiones y hechos en PostgreSQL.
- Desarrollo de Procesos ETL: Creación de transformaciones en Pentaho para la extracción de datos desde fuentes CSV (clientes, productos, ventas) y APIs externas (clima), su limpieza y carga en el DW.
- Desarrollo de Backend y Seguridad: Implementación de API REST segura que consume datos del DW y valida tokens de autenticación contra Keycloak.
- Desarrollo de Frontend y Dashboards: Construcción de la aplicación Angular con componentes gráficos para visualizar tendencias de ventas y métricas de negocio.

2.6 Resultados obtenidos

Como resultado de la ejecución del proyecto, se obtuvo una solución de Inteligencia de Negocios (BI) funcional e integrada. A continuación, se describen los subsistemas y componentes resultantes:

Arquitectura del Repositorio y Control de Versiones (Git)

Se consolidó un repositorio centralizado bajo el control de versiones Git, estructurado para separar claramente las responsabilidades del sistema. La estructura final del proyecto (proyecto-integrador-datascience-ventas) incluye:

- Backend: Lógica de servidor y API REST.
- Frontend: Interfaz de usuario en Angular.
- ETL: Procesos de integración de datos con Pentaho.
- SQL/Files: Scripts de base de datos y fuentes de datos planas (CSV).

Ingeniería de Datos y Procesos ETL (Pentaho)

Se desarrollaron y ejecutaron exitosamente flujos de trabajo en Pentaho Data Integration (PDI), logrando la ingesta de datos heterogéneos.

Orquestación Maestra: Se implementó un Job principal (master_etl.kjb) que coordina la ejecución secuencial de las transformaciones, asegurando la integridad referencial.

Transformaciones Específicas: Se crearon transformaciones individuales (.ktr) para procesar archivos CSV de diversas fuentes:

- Datos Transaccionales: Carga de Ventas (04_load_ventas.ktr).
- Datos Maestros: Clientes, Productos y Sucursales.
- Datos Externos: Se integró exitosamente información meteorológica mediante la carga del archivo clima.csv (05_load_clima.ktr) y datos de redes sociales (06_load_redes_sociales.ktr).

Limpieza y Normalización: Los flujos incluyen pasos para el manejo de nulos, estandarización de fechas y formatos numéricos antes de la inserción en el Data Warehouse.



Diseño e Implementación del Data Warehouse

Se desplegó un Data Warehouse (DW) en PostgreSQL, alojado en un servidor virtualizado, siguiendo un modelo dimensional (Esquema en Estrella):

Tablas de Dimensiones: Se poblaron las tablas `dim_cliente`, `dim_producto`, `dim_sucursal` y `dim_tiempo`, las cuales permiten el filtrado y segmentación de los datos.

Tablas de Hechos: Se generaron las tablas `fact_ventas` y `fact_redes_sociales`, que almacenan las métricas cuantitativas y claves foráneas, permitiendo el cruce de información entre el rendimiento de ventas y factores externos como el clima.

Seguridad y Gestión de Identidad (Keycloak & VirtualBox)

Se diseñó e implementó una arquitectura de seguridad robusta y desacoplada, utilizando Keycloak como Proveedor de Identidad (IdP) alojado en un servidor externo virtualizado (VirtualBox). Esta infraestructura garantiza la autenticación centralizada y la autorización segura mediante estándares modernos.

Virtualización y Aislamiento de Servicios

Para simular un entorno de producción realista, se desplegó el servidor Keycloak y la base de datos PostgreSQL en una máquina virtual independiente (VirtualBox), separando la capa de persistencia y seguridad de la capa de aplicación. Se configuró la red virtual en modo "Puente" (Bridge) o "Anfitrión" (Host-only), garantizando la conectividad segura mediante IP estáticas entre la máquina host (donde se ejecutan Backend/Frontend) y los servicios virtualizados.

Automatización del Entorno (Carpeta Script)

Se implementó una estrategia de "Infraestructura como Código" (IaC) mediante scripts de Shell (Bash) ubicados en el directorio `Script/` del repositorio, lo que permite el aprovisionamiento desatendido y reproducible del entorno:

Script de Instalación (`instalar_todo.sh`): Se desarrolló un script de aprovisionamiento que automatiza la instalación de las dependencias base en el servidor virtual, incluyendo el entorno de ejecución Java (JDK), el motor de base de datos PostgreSQL y la descarga/descompresión del servidor Keycloak. Esto elimina los errores humanos asociados a la instalación manual de paquetes.

Script de Configuración (`configurar_keycloak.sh`): Se creó un script de automatización lógica que interactúa con la Keycloak Admin CLI (`kcadm.sh`). Al ejecutarse, este script realiza las siguientes acciones secuenciales sin intervención humana:

- Inicia el servicio de Keycloak y se autentica contra el realm Master.
- Crea un nuevo **Realm** específico para el proyecto (aislado del entorno administrativo).
- Define y configura los **Clientes (Clients)** para el Backend (bearer-only o confidencial) y el Frontend (public), estableciendo automáticamente las *Valid Redirect URIs* y *Web Origins* necesarias para el CORS.
- Genera la estructura de **Roles** (ej. admin, usuario) y crea usuarios de prueba con sus respectivas credenciales y asignación de roles.



Implementación de OIDC y OAuth2

La seguridad de la aplicación se basó en el protocolo OpenID Connect (OIDC) sobre OAuth 2.0. El flujo de autenticación implementado delega el inicio de sesión a Keycloak, el cual emite Tokens JWT (JSON Web Tokens) firmados criptográficamente.

- Frontend: Intercepta la navegación no autorizada y redirige al usuario al portal de login de Keycloak. Gestiona el ciclo de vida del token (almacenamiento y renovación).
- Backend: Actúa como *Resource Server*, validando la firma del token JWT en cada petición HTTP contra la clave pública del Realm, asegurando que solo peticiones con un token válido y activo tengan acceso a los endpoints de datos.

Control de Acceso Basado en Roles (RBAC)

Gracias a la configuración automatizada, se estableció un control de acceso granular. El Backend inspecciona los "claims" del token JWT para verificar los roles del usuario, permitiendo, por ejemplo, que solo usuarios con el rol admin puedan acceder a endpoints sensibles o de escritura en el Data Warehouse, mientras que el rol usuario mantiene acceso de solo lectura a los dashboards.

Desarrollo del Backend (API REST)

Se desarrolló una capa de servicios (Backend) robusta y escalable utilizando el entorno de ejecución Node.js y el framework Express, diseñada para actuar como el intermediario seguro entre el Data Warehouse y la interfaz de usuario. La implementación se rigió por los principios de arquitectura RESTful y modularidad.

Arquitectura Modular y Separación de Responsabilidades

Para garantizar la mantenibilidad y escalabilidad del código, se estructuró el proyecto siguiendo el patrón de diseño MVC (Modelo-Vista-Controlador) adaptado a servicios API:

- Enrutamiento (/routes): Se desacopló la definición de las rutas (endpoints) de la lógica de negocio. Archivos como `api.routes.js` y `dashboardRoutes.js` gestionan exclusivamente la recepción de peticiones HTTP (GET, POST) y las redirigen al controlador correspondiente.
- Controladores (/controllers): Se implementaron módulos como `dataController.js` y `api.controllers.js`, donde reside la lógica de procesamiento. Estos controladores son responsables de validar la entrada, invocar las consultas a la base de datos y formatear la respuesta JSON estandarizada que consume el Frontend.
- Configuración (/config): Se centralizaron las configuraciones críticas, como los parámetros de conexión a Keycloak (`keycloak-config.js`) y las variables de entorno, evitando valores "hardcodeados" en el código fuente.

Gestión Eficiente de Conexiones a Datos (PostgreSQL)

La interacción con el Data Warehouse PostgreSQL se optimizó mediante la implementación de un Pool de Conexiones utilizando la librería `pg` (`node-postgres`).

- Estrategia de Pooling: En lugar de abrir y cerrar una conexión por cada petición del cliente (lo cual es costoso en términos de rendimiento), se configuró un pool que mantiene un conjunto de conexiones reutilizables activas.
- Consultas Analíticas: El backend no se limita a operaciones CRUD simples; ejecuta consultas SQL complejas (agregaciones, JOINS entre tablas de hechos y dimensiones) directamente sobre el Data Warehouse para pre-procesar los datos que alimentan los gráficos de ventas y clima, reduciendo la carga de procesamiento en el navegador del cliente.



Middleware de Seguridad y Validación de Identidad

La seguridad se integró transversalmente en el flujo de petición-respuesta mediante el uso de Middlewares en Express.

- Integración con Keycloak: Se implementó el adaptador keycloak-connect dentro de la configuración del servidor (app.js). Este middleware intercepta cada solicitud entrante a las rutas protegidas (/api/*).
- Validación de Tokens (JWT): El backend verifica criptográficamente la firma del Bearer Token enviado en los encabezados HTTP. Si el token no es válido, ha expirado o no ha sido emitido por el Realm confiable del servidor Keycloak virtualizado, la petición es rechazada con un código de estado 401 Unauthorized antes de llegar a la capa de datos.
- Gestión de CORS: Se configuró el middleware cors para permitir solicitudes de recursos cruzados (Cross-Origin Resource Sharing) exclusivamente desde el dominio donde se aloja el Frontend (Angular), previniendo el uso no autorizado de la API desde orígenes desconocidos.

Endpoints y Lógica de Negocio

Se expuso una API RESTful documentada que provee los datos necesarios para los indicadores clave de rendimiento (KPIs):

- Endpoint de Resumen de Ventas: Procesa la tabla de hechos fact_ventas para devolver totales agrupados por tiempo y sucursal.
- Endpoint de Clima y Tendencias: Cruza la información de ventas con la tabla dim_tiempo y los datos meteorológicos importados, permitiendo al frontend visualizar correlaciones entre temperatura/lluvia y el volumen de ventas.
- Manejo de Errores: Se implementó un manejador global de errores que captura excepciones en las consultas SQL o lógica interna, devolviendo mensajes de error controlados al cliente sin exponer detalles sensibles de la infraestructura o la base de datos (Stack Traces).

Configuración del Entorno (.env)

Siguiendo las mejores prácticas de The Twelve-Factor App, toda la configuración sensible (credenciales de base de datos, URLs de Keycloak, puertos de escucha) se extrajo del código fuente y se gestionó a través de variables de entorno cargadas mediante la librería dotenv. Esto permite desplegar el mismo código en desarrollo y producción simplemente cambiando el archivo de configuración, sin tocar el código base.

Desarrollo del Frontend y Visualización de Datos (Angular)

Se construyó una interfaz de usuario moderna, reactiva y escalable bajo el paradigma de Single Page Application (SPA), utilizando el framework Angular. La aplicación actúa como la capa de presentación que consume los datos procesados del Data Warehouse a través del Backend seguro.

Arquitectura de Componentes y Modularidad

El desarrollo se estructuró siguiendo una arquitectura basada en componentes, lo que favorece la reutilización de código y la mantenibilidad. La estructura del directorio src/app refleja una separación lógica de responsabilidades:



Módulos de Página (/pages): Se implementaron vistas dedicadas para cada dominio funcional.

- Dashboard (/dashboard): Componente principal encargado de la orquestación visual de los indicadores. Se compone de archivos dashboard.html (plantilla), dashboard.ts (lógica) y dashboard.css (estilos), encapsulando la presentación de los KPIs de ventas y clima.
- Gestión de Clientes (/clientes): Interfaz para la visualización y administración de la dimensión de clientes.

Servicios (/services): Se aisló la lógica de comunicación con el Backend mediante servicios inyectables, específicamente dashboard.service.ts. Este patrón permite que los componentes permanezcan ligeros, delegando la obtención de datos a servicios especializados.

Integración de Datos y Programación Reactiva (RxJS)

La comunicación con la API REST del Backend se gestionó utilizando el módulo HttpClient de Angular.

- Consumo de API: El servicio dashboard.service.ts centraliza las peticiones HTTP (GET, POST) hacia los endpoints expuestos por el servidor Node.js.
- Manejo de Asincronía: Se implementó el patrón Observer mediante la librería RxJS. Los componentes se suscriben a los Observables retornados por el servicio, permitiendo una actualización dinámica y asíncrona de la interfaz de usuario tan pronto como el Backend retorna los datos del Data Warehouse, sin bloquear la experiencia del usuario.

Enrutamiento y Navegación (app.routes.ts)

Se configuró el enrutador de Angular (Router) en el archivo app.routes.ts para gestionar la navegación dentro de la aplicación sin recargas de página completas. Esto garantiza una experiencia de usuario fluida y rápida, permitiendo transiciones instantáneas entre la vista de "Dashboard" y la de "Clientes", manteniendo el estado de la sesión activo.

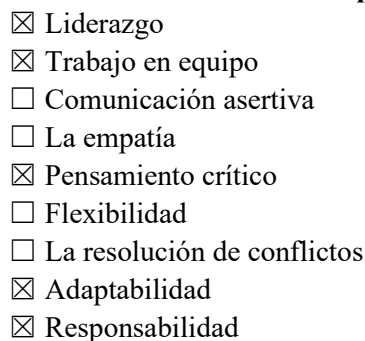
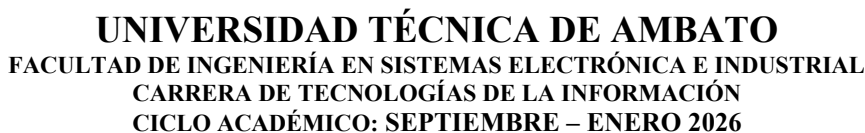
Visualización de KPIs y Dashboards

En el componente Dashboard, se implementó la lógica de visualización para transformar los datos crudos (JSON) recibidos del Backend en información gráfica interpretable.

- Renderizado Dinámico: Utilizando las directivas estructurales de Angular (*ngFor, *ngIf), el HTML del dashboard (dashboard.html) se renderiza dinámicamente en base a los datos recibidos, mostrando tablas de resumen y tarjetas de métricas que responden a los cambios en el Data Warehouse.
- Integración de Gráficos: Se preparó la interfaz para alojar librerías de visualización, mapeando los datos de ventas cruzados con los datos climáticos para evidenciar correlaciones visuales en el frontend.

Configuración de Entorno y Despliegue

Siguiendo las mejores prácticas de DevOps para frontend, las configuraciones específicas del entorno (como la URL base de la API y la configuración del cliente de Keycloak) se abstraeron en los archivos de entorno (src/environments/environment.ts). Esto permite que el proceso de construcción (npm build) genere artefactos optimizados para producción o desarrollo sin modificar el código fuente de los componentes.



- La utilización de Git como sistema de control de versiones fue fundamental para mantener un historial coherente de cambios y permitir el trabajo paralelo en los módulos de ETL, Backend y Frontend sin conflictos de integración mayores.
- La arquitectura separada, con la base de datos y el servicio de identidad (Keycloak) en un entorno virtualizado, simuló eficazmente un entorno de producción real, demostrando la viabilidad de sistemas distribuidos seguros.
- El proceso ETL implementado con Pentaho demostró ser robusto para la normalización de datos heterogéneos, garantizando que la información visualizada en los dashboards sea confiable y consistente para la toma de decisiones.

- Se recomienda automatizar el despliegue de los contenedores (Dockerización) tanto del Frontend como del Backend para simplificar la configuración en nuevos entornos de desarrollo, reduciendo la dependencia de configuraciones manuales en la máquina local.
- Para futuras iteraciones, sería beneficioso implementar un proceso de carga incremental (CDC) en los flujos ETL para optimizar el rendimiento y evitar el procesamiento redundante de datos históricos que no han sufrido cambios.
- Es aconsejable configurar alertas automáticas en Pentaho que notifiquen a los administradores en caso de fallos en la carga de datos o inconsistencias en los archivos de origen.

Sign in to your account

VENTAS-REALM

Sign in to your account

Username or email

Password

Sign in

Ilustración 1: Dashboard Login



UNIVERSIDAD TÉCNICA DE AMBATO

FACULTAD DE INGENIERÍA EN SISTEMAS ELECTRÓNICA E INDUSTRIAL

CARRERA DE TECNOLOGÍAS DE LA INFORMACIÓN

CICLO ACADÉMICO: SEPTIEMBRE – ENERO 2026

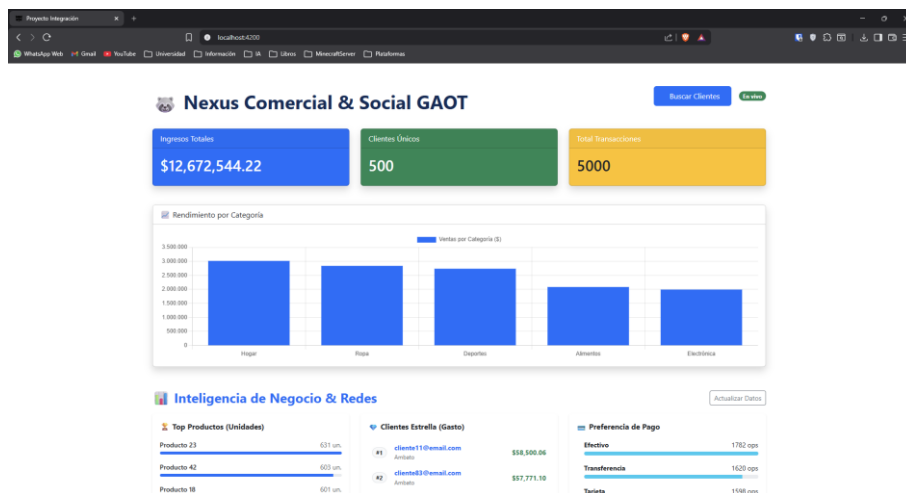


Ilustración 2: Dashboard01

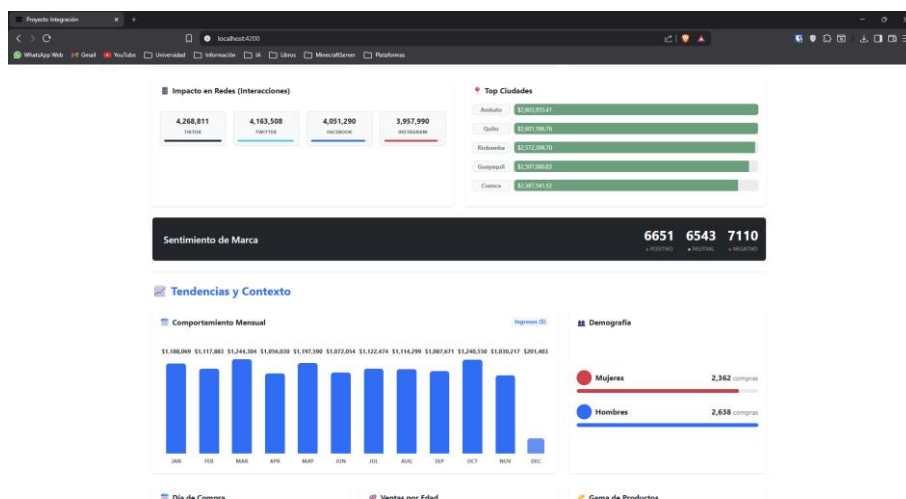


Ilustración 3: Dashboard02

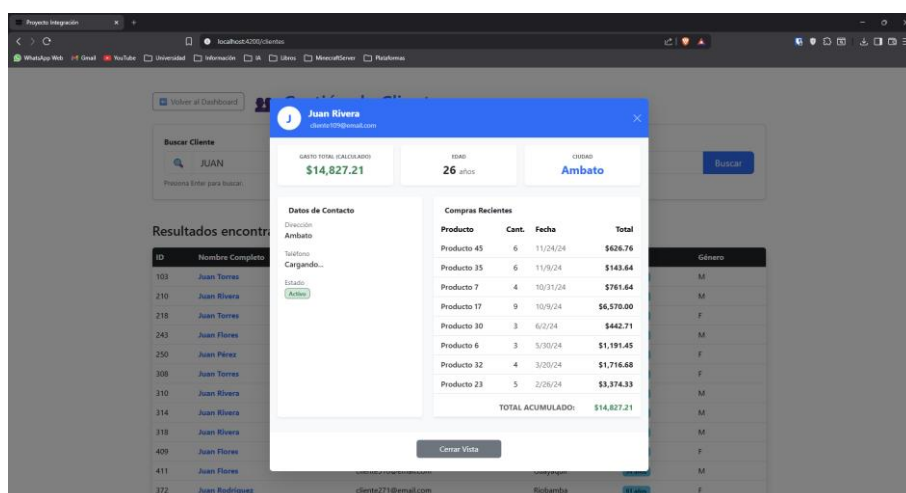


Ilustración 4: Dashboard Clientes



2.11 Referencias bibliográficas

- [1]. Brown, E. (2019). Web Development with Node and Express: Leveraging the JavaScript Stack (2ª ed.). O'Reilly Media.
- [2]. Chacon, S., & Straub, B. (2014). Pro Git (2ª ed.). Apress.
- [3]. Freeman, A. (2024). Pro Angular 16 (6ª ed.). Manning Publications.
- [4]. Kimball, R., & Ross, M. (2013). The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling (3ª ed.). John Wiley & Sons.
- [5]. Obe, R. O., & Hsu, L. S. (2017). PostgreSQL: Up and Running (3ª ed.). O'Reilly Media.
- [6]. Roldán, M. C. (2013). Pentaho Data Integration Beginner's Guide (2ª ed.). Packt Publishing.
- [7]. Thorgersen, S., & Silva, P. I. (2021). Keycloak - Identity and Access Management for Modern Applications. Packt Publishing.