



UNIVERSIDAD TÉCNICA ESTATAL DE QUEVEDO

Facultad de ciencias de la ingeniería



ESTUDIANTE:

CHICA VALFRE VALESKA SOFIA

LETURNE PLUAS JHON BYRON

CARRERA:

Ingeniería de software

CURSO:

7to Software "A"

ASIGNATURA:

Aplicaciones distribuidas

DOCENTE:

Ing. Guerrero Ulloa Gleiston Cicerón, MSc

TEMA:

Desarrollo de Aplicación en Capas

Contenido

1. Introducción	4
2. Firebase.....	5
2.1. Firebase como plataforma de desarrollo de aplicaciones en tiempo real.	5
2.2. Uso de Firebase Authentication para autenticación de usuarios	5
2.3. Firebase Realtime Database para almacenamiento y sincronización de datos.....	6
2.4. Firebase Cloud Functions para la lógica de servidor sin servidor.....	7
3. Herramientas de Desarrollo de Microsoft Azure	8
3.1. Azure como plataforma de servicios en la nube.....	8
3.2. Azure App Service para la implementación de aplicaciones web y móviles	9
3.3. Azure Functions para la creación de funciones sin servidor	9
3.4. Azure SQL Database para el almacenamiento de datos	10
4. Amazon Web Services (AWS).....	11
4.1. AWS como plataforma de servicios en la nube.....	11
4.2. AWS Lambda para la ejecución de código sin servidor.....	12
4.3. AWS DynamoDB para el almacenamiento de datos NoSQL	13
4.4. AWS Cognito para la autenticación y autorización de usuarios.....	13
5. Practica (Firebase)	14
5.1. Tecnologías utilizadas	14
5.2. Diagrama de ejemplo.....	15
5.3. Firebase datos	15
5.3.1. Colección productos.....	15
5.3.2. Colección Usuario	15
5.4. Organización del ejemplo.....	16
5.5. Dependencias.....	17
5.6. Obtención de servicio de FireBase	17
5.7. Paquete principal	20
5.7.1. Utilización de Firebase Java y Registro aplicación.....	20
5.7.2. Messages	22
5.7.3. Encriptación de contraseñas	22
5.8. Modelos	22
5.8.1. Modelo usuario	22
5.8.2. Modelo producto	23
5.9. Repositorio	23
5.9.1. Insertar y actualizar	24

5.9.2.	Lectura de datos	25
5.9.3.	Busqueda de datos.....	25
5.9.4.	Eliminación de documentos	26
5.10.	Servicios.....	26
5.10.1.	Servicio Usuario.....	27
5.10.2.	Servicio productos.....	27
5.11.	Controladores.....	29
5.11.1.	Controlador Usuario.....	29
5.11.2.	Controlador Producto	30
5.12.	Vista Cliente	32
5.12.1.	Login	32
5.12.2.	Registro usuario	33
5.12.3.	Registro productos	33
5.13.	Interfaces.....	35
5.14.	GitHub ejemplo.....	37
6.	Conclusión	37
7.	Bibliografía	38

Ilustraciones

Ilustración 1	Diagrama de ejemplo de usuarios y productos	15
Ilustración 2	Colección productos.....	15
Ilustración 3	Colección usuario.....	15
Ilustración 4	Organización de los paquetes (API).....	16
Ilustración 5	Pagina principal FiresBase.....	18
Ilustración 6	Creación de un proyecto Obtenida de:	18
Ilustración 7	Pasos para la creación del proyecto	19
Ilustración 8	Firestore Database.....	19
Ilustración 9	Obtención de clave privada.....	20
Ilustración 10	Archivo JSON (Clave Privada).....	20
Ilustración 11	Login	35
Ilustración 12	Registro usuario	35
Ilustración 13	Productos registrados	36
Ilustración 14	Formulario productos.....	36
Ilustración 15	GitHub.....	37

Tablas

Tabla 1	Herramientas utilizadas	14
---------	-------------------------------	----

1. Introducción

En el vasto universo de la tecnología de la información, ¿cómo logramos construir aplicaciones robustas, escalables y fáciles de mantener? Una de las respuestas a esta interrogante reside en el modelo de Desarrollo de Aplicación en Capas, una metodología que ha cobrado especial relevancia en los últimos años debido al crecimiento exponencial de las necesidades empresariales y la complejidad de los sistemas informáticos. Este enfoque, que divide la aplicación en niveles lógicos o capas, cada uno con una responsabilidad específica, ha demostrado ser una solución eficaz para enfrentar los desafíos del desarrollo de software en el siglo XXI [1].

El presente trabajo tiene como objetivo explorar en profundidad el paradigma del Desarrollo de Aplicación en Capas, analizando sus principios fundamentales, la estructura que lo compone y las ventajas que ofrece para el desarrollo de aplicaciones complejas. Asimismo, se realizará una comparación con otros modelos de desarrollo para destacar sus particularidades y entender en qué contextos resulta más beneficioso su uso. A través de este análisis, se busca no solo comprender la importancia y aplicabilidad de este modelo en el escenario actual de la tecnología de la información, sino también proporcionar una guía para aquellos desarrolladores y arquitectos de software que buscan optimizar sus procesos de desarrollo y asegurar la calidad y sostenibilidad de sus aplicaciones. En un mundo donde la adaptabilidad y eficiencia son claves para el éxito, el Desarrollo de Aplicación en Capas emerge como una estrategia esencial para enfrentar los retos del futuro [2].

El Desarrollo en Capas ha demostrado traer múltiples beneficios a la construcción de aplicaciones empresariales modernas. Al dividir un sistema en capas lógicas e independientes, se promueve la separación de conceptos, la reutilización de código y el bajo acoplamiento entre módulos [2]. Esto facilita el mantenimiento, testing y escalamiento de aplicaciones complejas a lo largo del tiempo. A pesar de agregar cierta complejidad inicial, un buen diseño en capas potencia la flexibilidad y productividad a mediano y largo plazo. Por estas razones, continúa siendo un paradigma dominante en la ingeniería de software de hoy [3].

En el documento, compartiremos varias definiciones de distintos servicios en la nube y exploraremos cómo interactúan entre sí. Además, se presentará un ejemplo con Firebase para comprender mejor su funcionamiento y cómo gestiona los datos.

2. Firebase

2.1. Firebase como plataforma de desarrollo de aplicaciones en tiempo real.

Firebase es una plataforma integral de Google que permite a los desarrolladores crear aplicaciones móviles de alta calidad con capacidades en tiempo real. Gracias a sus servicios como base de datos en tiempo real, autenticación, alojamiento en la nube y mensajería, Firebase posibilita la sincronización instantánea de datos entre usuarios y proporciona una infraestructura lista para usar, optimizada para el desarrollo ágil de apps interactivas [4].

Con Firebase los desarrolladores pueden enfocarse en crear experiencias de usuario innovadoras sabiendo que cuentan con una plataforma sólida que maneja todo el trabajo pesado de backend y escalamiento. La plataforma ofrece una base de datos NoSQL en tiempo real llamada Firebase Realtime Database que permite la sincronización bidireccional de los datos, de modo que se actualizan al instante en todos los clientes conectados [4].

También proporciona un sistema de autenticación lista para usar con proveedores sociales y por email/contraseña para que los usuarios puedan iniciar y cerrar sesión en tiempo real. Cuenta con servicios de alojamiento en la nube para contenido multimedia generado por usuarios, funciones backend ejecutadas en respuesta a eventos de la base de datos, hosting para desplegar y escalar apps web de forma automática, y mensajería push para notificaciones fiables a usuarios [5].

2.2. Uso de Firebase Authentication para autenticación de usuarios

Firebase Authentication es una solución de gestión de identidades y autenticación de usuarios desarrollada por Google para aplicaciones web y móviles. Ofrece una manera segura y fácil de autenticar a los usuarios, soportando una amplia variedad de proveedores de identidad, como Google, Facebook, Twitter, y GitHub, así como autenticación por correo electrónico y contraseña, y autenticación por número de teléfono [6].

El servicio simplifica el proceso de implementación de sistemas de autenticación, al manejar tareas comunes como la verificación de correos electrónicos, la recuperación de contraseñas y el inicio de sesión con proveedores de identidad de terceros. Esto permite a los desarrolladores concentrarse en la creación de las características únicas de sus aplicaciones, mientras confían en Firebase para la gestión segura de la autenticación de usuarios [6].

Firebase Authentication también se integra estrechamente con otros servicios de Firebase, como Firestore, Firebase Realtime Database, y Cloud Functions, permitiendo una

sincronización y gestión de datos eficiente y segura. Esto facilita la creación de experiencias de usuario personalizadas, basadas en los datos de autenticación de los usuarios [6], [7].

Además, Firebase Authentication proporciona una consola de administración fácil de usar, donde los desarrolladores pueden gestionar los usuarios de sus aplicaciones, revisar las estadísticas de autenticación y configurar los métodos de autenticación. Esta consola simplifica la administración de usuarios y ayuda a los desarrolladores a mantener el control sobre el acceso a sus aplicaciones [7].

La seguridad es una prioridad en Firebase Authentication, que utiliza prácticas de seguridad robustas para proteger la información de los usuarios. Esto incluye el uso de HTTPS para todas las comunicaciones, el almacenamiento seguro de contraseñas y la conformidad con estándares de seguridad internacionales. Esto asegura que la información de autenticación de los usuarios se maneje de manera segura en todo momento [7].

2.3. Firebase Realtime Database para almacenamiento y sincronización de datos

Firebase Realtime Database es un servicio de base de datos alojado en la nube que permite a los desarrolladores de aplicaciones web y móviles almacenar y sincronizar datos entre los usuarios en tiempo real [8]. Es una base de datos NoSQL que almacena datos en formato JSON y proporciona una sincronización en tiempo real con cada cliente conectado. Esto significa que cuando los datos se actualizan, se transmiten inmediatamente a todos los dispositivos conectados, ofreciendo una experiencia de usuario fluida y reactiva [9].

Una de las características clave de Firebase Realtime Database es su capacidad para trabajar en aplicaciones offline. Los datos a los que acceden los usuarios se almacenan localmente en el dispositivo, permitiendo que las aplicaciones funcionen incluso sin una conexión a internet. Una vez que el dispositivo se reconecta, la base de datos sincroniza automáticamente los cambios con el estado actual del servidor, garantizando que los datos estén siempre actualizados [8].

La seguridad y la protección de datos son aspectos fundamentales en Firebase Realtime Database. Los desarrolladores pueden definir reglas de seguridad detalladas en un lenguaje de configuración basado en JSON para controlar el acceso y las operaciones permitidas en la base de datos. Estas reglas de seguridad aseguran que los datos sean accesibles solo para los usuarios autorizados, protegiendo la privacidad y la integridad de los datos [8].

La integración con otros servicios de Firebase es otra ventaja importante. Por ejemplo, se puede usar junto con Firebase Authentication para crear sistemas de autenticación seguros y personalizados, o con Cloud Functions para ejecutar código backend en respuesta a eventos en la base de datos sin necesidad de administrar servidores [9].

La implementación de Firebase Realtime Database es relativamente sencilla, lo que permite a los desarrolladores comenzar rápidamente a trabajar en sus proyectos sin preocuparse por la infraestructura de backend. La base de datos es escalable, lo que significa que puede manejar aplicaciones desde las más pequeñas hasta las de mayor tráfico, ajustando los recursos automáticamente según las necesidades [9].

2.4. Firebase Cloud Functions para la lógica de servidor sin servidor

Firebase Cloud Functions es un servicio de computación en la nube que permite a los desarrolladores ejecutar la lógica del lado del servidor en un entorno completamente administrado, sin la necesidad de administrar servidores [10]. Este enfoque de "computación sin servidor" permite a los desarrolladores escribir funciones individuales que se ejecutan en respuesta a eventos generados por otros servicios de Firebase y HTTPS. Estas funciones pueden realizar tareas como el procesamiento de datos en segundo plano, la ejecución de lógica compleja en respuesta a eventos de la base de datos, el envío de notificaciones push, y mucho más, sin necesidad de preocuparse por la infraestructura de alojamiento [11].

La arquitectura sin servidor de Cloud Functions ofrece varias ventajas. Primero, la escalabilidad es automática; Firebase aloja y gestiona las funciones, escalándolas en respuesta a la demanda [11], [12]. Esto significa que las funciones pueden manejar desde unos pocos hasta millones de invocaciones sin necesidad de configuración adicional. Segundo, los desarrolladores pagan solo por los recursos que utilizan, lo que hace que Cloud Functions sea una opción eficiente en términos de costos para muchas aplicaciones [13].

Cloud Functions se integra profundamente con otros servicios de Firebase y Google Cloud, lo que permite a las funciones reaccionar a cambios en Firestore, Firebase Realtime Database, Firebase Authentication, Google Analytics for Firebase, y eventos de almacenamiento en Firebase Cloud Storage. Por ejemplo, una función puede dispararse automáticamente cuando se agrega un nuevo documento a Firestore o cuando un usuario sube un archivo a Cloud Storage [13].

La implementación de la lógica del lado del servidor mediante Cloud Functions facilita la creación de aplicaciones más seguras y escalables. Al mover la lógica y las operaciones

sensibles al servidor, los desarrolladores pueden mantener el control sobre las operaciones que se realizan en sus datos, asegurando que solo se ejecuten acciones validadas y autorizadas [14].

Desarrollar con Firebase Cloud Functions también implica una experiencia de desarrollo más ágil. Los desarrolladores pueden escribir funciones en JavaScript o TypeScript, aprovechando el ecosistema de Node.js para acceder a una amplia gama de bibliotecas y herramientas. Además, Firebase proporciona herramientas para probar funciones localmente y desplegarlas fácilmente a la nube, simplificando el proceso de desarrollo y despliegue [14].

3. Herramientas de Desarrollo de Microsoft Azure

3.1. Azure como plataforma de servicios en la nube

Azure, desarrollada por Microsoft, es una de las plataformas de servicios en la nube líderes en el mercado, ofreciendo una amplia gama de servicios de computación en la nube para ayudar a las empresas a construir, administrar y desplegar aplicaciones en una red global gestionada por Microsoft [15]. Proporciona soluciones en varias categorías, incluyendo computación, redes, bases de datos, almacenamiento, inteligencia artificial (IA), Internet de las Cosas (IoT), seguridad y mucho más. Azure permite a las organizaciones aprovechar la escalabilidad, la flexibilidad y la eficiencia de la nube para innovar y optimizar sus operaciones [16].

Azure brinda una suite completa de servicios que incluyen opciones de computación como Máquinas Virtuales, Azure Kubernetes Service para contenedores y Azure Functions para ejecución sin servidor, adaptándose a diferentes necesidades de desarrollo. Ofrece almacenamiento seguro y escalable con Azure Blob Storage, File Storage, y Queue Storage, así como bases de datos avanzadas como Azure SQL Database y Cosmos DB para aplicaciones de alta demanda [16]. En redes, Azure Virtual Network y DNS mejoran la seguridad y rendimiento. Para IA y aprendizaje automático, Azure AI y Machine Learning permiten incorporar inteligencia avanzada en aplicaciones. La seguridad es robusta con Azure Active Directory y Security Center, mientras que herramientas como Azure Synapse Analytics y IoT Hub facilitan la integración y análisis de datos, consolidando a Azure como una plataforma integral para soluciones empresariales en la nube [17].

Azure se destaca por su capacidad para integrarse con productos existentes de Microsoft, como Office 365 y SharePoint, proporcionando una experiencia cohesiva para las empresas que ya dependen de herramientas de Microsoft. Además, su compromiso con la apertura y la flexibilidad significa que Azure soporta una variedad de sistemas operativos, lenguajes de programación, frameworks, bases de datos y dispositivos [18].

3.2. Azure App Service para la implementación de aplicaciones web y móviles

Azure App Service, proporcionado por Microsoft Azure, es una plataforma como servicio (PaaS) diseñada para simplificar el desarrollo, implementación y escalado de aplicaciones web y móviles. Este servicio permite a los desarrolladores construir aplicaciones robustas sin preocuparse por la gestión de la infraestructura [19]. Soporta múltiples lenguajes y frameworks, incluidos .NET, Java, Node.js, PHP, y Python, facilitando la integración continua con plataformas como GitHub y Azure DevOps para un flujo de trabajo de desarrollo ágil [20].

Para el desarrollo de aplicaciones móviles, Azure App Service ofrece funcionalidades clave como autenticación de usuarios, almacenamiento de datos offline y notificaciones push. Estas características permiten crear experiencias móviles interactivas sin la complejidad de manejar el backend. Además, la plataforma asegura la escalabilidad automática de las aplicaciones, permitiendo ajustar recursos fácilmente para manejar picos de tráfico, lo que garantiza un rendimiento óptimo bajo cualquier carga de trabajo [20].

La seguridad es una prioridad en Azure App Service, ofreciendo autenticación integrada, conexiones seguras SSL/TLS y aislamiento de red a través de Azure Virtual Network. La integración con Azure Active Directory facilita la implementación de políticas de seguridad y control de acceso. Además, la plataforma se integra sin problemas con otros servicios de Azure, como bases de datos y almacenamiento, extendiendo la funcionalidad de las aplicaciones con las extensiones disponibles en Azure Marketplace [21].

Azure App Service también proporciona entornos de staging para pruebas y desarrollo, permitiendo a los equipos desplegar y testear aplicaciones en entornos aislados antes de su lanzamiento [21]. Esta característica es esencial para adoptar prácticas de integración y entrega continuas (CI/CD), mejorando el ciclo de vida del desarrollo de software. En resumen, Azure App Service es una solución flexible y potente para el desarrollo e implementación de aplicaciones en la nube, ofreciendo escalabilidad, seguridad y una amplia gama de integraciones para optimizar el desarrollo de aplicaciones [22].

3.3. Azure Functions para la creación de funciones sin servidor

Azure Functions es un servicio de computación sin servidor ofrecido por Microsoft Azure que permite a los desarrolladores ejecutar pequeños fragmentos de código, o "funciones", en respuesta a varios eventos, sin la necesidad de preocuparse por la infraestructura de servidor subyacente. Este modelo de ejecución basado en eventos es ideal para tareas como el

procesamiento de datos en tiempo real, la integración de sistemas, la automatización de flujos de trabajo y la creación de APIs [23].

Una de las principales ventajas de Azure Functions es su capacidad para escalar automáticamente. Dependiendo del volumen de solicitudes, Azure puede asignar más recursos para manejar aumentos en la carga de trabajo, y luego reducir esos recursos cuando la demanda disminuye, optimizando así los costos. Los desarrolladores solo pagan por el tiempo de ejecución de sus funciones, lo que hace de Azure Functions una solución costo-efectiva para muchos escenarios de uso [23], [24].

Azure Functions soporta una amplia gama de lenguajes de programación, incluyendo C#, Java, JavaScript, TypeScript, Python, y PowerShell. Esto permite a los desarrolladores utilizar el lenguaje con el que se sientan más cómodos o que mejor se adapte a la tarea específica. Además, Azure Functions se integra fácilmente con otros servicios de Azure y recursos externos, permitiendo a las funciones interactuar con bases de datos, servicios de almacenamiento, servicios de terceros y más. El servicio ofrece también un modelo de desarrollo flexible, con opciones para desarrollar y probar funciones localmente en el propio equipo del desarrollador antes de desplegarlas en la nube. Esto facilita un ciclo de desarrollo rápido y eficiente. Además, Azure Functions proporciona herramientas para monitorear el rendimiento y la ejecución de las funciones, ayudando a los desarrolladores a diagnosticar y solucionar problemas rápidamente [24].

3.4. Azure SQL Database para el almacenamiento de datos

Azure SQL Database es un servicio de base de datos relacional completamente gestionado proporcionado por Microsoft Azure. Basado en la última versión del Motor de Base de Datos SQL Server de Microsoft, este servicio ofrece una solución escalable, segura y de alta disponibilidad para el almacenamiento y la gestión de datos [25]. Azure SQL Database está diseñado para simplificar la administración de bases de datos, automatizando tareas de mantenimiento rutinarias como el ajuste del rendimiento, las copias de seguridad y la alta disponibilidad, lo que permite a los desarrolladores y administradores de bases de datos centrarse en aspectos más estratégicos de su trabajo [26].

Una característica clave de Azure SQL Database es su capacidad para escalar automáticamente los recursos según las necesidades de la aplicación, sin tiempo de inactividad. Esto significa que los recursos pueden ser ajustados dinámicamente para manejar picos de carga de trabajo, lo que asegura que las aplicaciones mantengan un rendimiento óptimo sin incurrir en costos

innecesarios por recursos no utilizados. Además, el servicio ofrece modelos de precios flexibles que se adaptan a diferentes patrones de uso, lo que hace que sea una opción económica para empresas de todos los tamaños [27].

En términos de seguridad, Azure SQL Database proporciona un conjunto robusto de características, incluyendo el cifrado de datos en reposo y en tránsito, autenticación avanzada y políticas de seguridad granulares. También ofrece protección avanzada contra amenazas, lo que ayuda a detectar y mitigar posibles vulnerabilidades y ataques SQL. Azure SQL Database soporta el desarrollo de aplicaciones modernas con requerimientos complejos de procesamiento y análisis de datos. Se integra fácilmente con otras herramientas y servicios de Azure, como Azure Functions para la lógica de aplicación sin servidor y Azure Logic Apps para la automatización del flujo de trabajo, proporcionando una plataforma cohesiva para el desarrollo de soluciones empresariales [28].

El servicio también facilita la migración de bases de datos existentes a la nube, ofreciendo herramientas y servicios que simplifican el proceso de traslado de bases de datos locales a Azure SQL Database. Esto permite a las organizaciones aprovechar las ventajas de la nube, como la escalabilidad, la flexibilidad y la reducción de costos de infraestructura, sin comprometer el rendimiento o la seguridad de sus datos [29].

4. Amazon Web Services (AWS)

4.1. AWS como plataforma de servicios en la nube

Amazon Web Services (AWS) es una de las plataformas de servicios en la nube más completas y ampliamente adoptadas en el mundo, ofrecida por Amazon.com. AWS proporciona una gama diversa de servicios y soluciones de infraestructura en la nube que permiten a las empresas, desde startups hasta corporaciones y agencias gubernamentales, desplegar aplicaciones y manejar datos a gran escala [30]. Con AWS, los usuarios pueden acceder a recursos de computación, almacenamiento, bases de datos, análisis, redes, móviles, herramientas de desarrollo, herramientas de gestión, IoT, seguridad y aplicaciones empresariales, lo que les permite escalar y crecer sin la necesidad de invertir en hardware físico [31].

AWS se destaca por su escalabilidad y flexibilidad, ofreciendo a los usuarios la capacidad de ajustar rápidamente los recursos a medida que cambian sus necesidades, garantizando al mismo tiempo altos niveles de disponibilidad y redundancia [31]. La red global de AWS abarca múltiples zonas geográficas denominadas "regiones" y "zonas de disponibilidad", lo que

permite a los usuarios desplegar aplicaciones y datos en ubicaciones específicas para reducir la latencia y cumplir con los requisitos de residencia de datos [32].

En términos de seguridad, AWS proporciona una infraestructura robusta diseñada para mantener la seguridad y la integridad de los datos. Los usuarios pueden aprovechar un amplio conjunto de herramientas de seguridad, políticas y funciones para gestionar el acceso, proteger sus infraestructuras y cumplir con diversas regulaciones de cumplimiento [30].

AWS también fomenta la innovación al ofrecer servicios avanzados en áreas como la inteligencia artificial (IA), el aprendizaje automático, el Internet de las Cosas (IoT) y la analítica de datos. Estos servicios permiten a los usuarios explorar nuevas oportunidades y crear soluciones complejas sin la necesidad de experticia especializada en cada una de estas tecnologías [30].

4.2. AWS Lambda para la ejecución de código sin servidor

AWS Lambda es un servicio de computación sin servidor ofrecido por Amazon Web Services (AWS) que permite a los desarrolladores ejecutar código en respuesta a eventos sin necesidad de provisionar o administrar servidores [31]. Con Lambda, puedes cargar tu código y el servicio automáticamente lo ejecuta, escalando la infraestructura de computación según sea necesario. Lambda soporta múltiples lenguajes de programación, como Node.js, Python, Java, Go, y .NET Core [32].

El modelo de precios de Lambda se basa en el número de solicitudes y la duración de la ejecución del código, lo que permite a los usuarios pagar solo por los recursos que consumen. Lambda puede ser disparado por eventos de otros servicios de AWS, como cambios en archivos en Amazon S3, actualizaciones en tablas de DynamoDB, solicitudes HTTP a través de Amazon API Gateway, entre otros, facilitando la creación de arquitecturas orientadas a eventos y aplicaciones altamente escalables y eficientes en cuanto a costos [32].

Este servicio elimina la necesidad de administrar infraestructura de servidor, permitiendo a los desarrolladores enfocarse en escribir código para procesar datos, integrar sistemas y crear aplicaciones ricas en lógica sin preocuparse por el aprovisionamiento de servidores, mantenimiento, o escalabilidad de la infraestructura subyacente. AWS Lambda es ideal para tareas como procesamiento de archivos, ejecución de lógica de backend para aplicaciones web y móviles, integraciones de sistemas, y automatización de procesos [33].

4.3. AWS DynamoDB para el almacenamiento de datos NoSQL

AWS DynamoDB es un servicio de base de datos NoSQL ofrecido por Amazon Web Services (AWS) diseñado para proporcionar rendimiento a escala con baja latencia y alta disponibilidad. Es una base de datos completamente administrada, lo que significa que AWS se encarga del mantenimiento, las copias de seguridad, y la escalabilidad, permitiendo a los desarrolladores centrarse en el desarrollo de la aplicación sin preocuparse por la infraestructura de la base de datos. DynamoDB es capaz de manejar grandes volúmenes de datos y soportar picos de tráfico, haciéndolo ideal para aplicaciones móviles, web, juegos, tecnología publicitaria, IoT, y muchos otros casos de uso que requieren un acceso rápido y confiable a los datos. Ofrece un modelo de datos clave-valor y documentos, lo que permite a los desarrolladores almacenar y recuperar cualquier cantidad de datos, y servir cualquier nivel de solicitud de tráfico [33].

Una de las características clave de DynamoDB es su capacidad para escalar automáticamente. La capacidad de la base de datos se ajusta automáticamente en respuesta a los patrones de tráfico en tiempo real, asegurando un rendimiento óptimo y evitando costos innecesarios. Además, DynamoDB ofrece modelos de consistencia ajustables, transacciones, y cifrado en reposo, mejorando la seguridad y la flexibilidad de la gestión de datos. DynamoDB se integra bien con otros servicios de AWS, como Lambda para la creación de aplicaciones orientadas a eventos, y Amazon S3 para el almacenamiento de grandes cantidades de datos no estructurados. Además, soporta triggers, lo que permite a los desarrolladores ejecutar funciones personalizadas automáticamente en respuesta a cambios en los datos, facilitando la creación de aplicaciones reactivas y en tiempo real [34].

4.4. AWS Cognito para la autenticación y autorización de usuarios

AWS Cognito es un servicio de Amazon Web Services (AWS) que proporciona autenticación, autorización y gestión de usuarios para aplicaciones web y móviles a escala. Con Cognito, los desarrolladores pueden implementar rápidamente un sistema de autenticación seguro que soporta el inicio de sesión a través de proveedores de identidad social como Google, Facebook, y Amazon, así como a través de proveedores de identidad empresarial y de inicio de sesión con nombre de usuario y contraseña. Una de las principales características de AWS Cognito es su capacidad para manejar la gestión de usuarios y sesiones de forma segura, permitiendo a los desarrolladores enfocarse en la lógica principal de la aplicación en lugar de en los detalles de implementación de la seguridad. Cognito facilita la creación de flujos de autenticación y autorización, incluyendo la verificación de correos electrónicos y teléfonos, la recuperación de contraseñas y la sincronización de datos de usuario a través de dispositivos [34].

El servicio se divide en dos componentes principales: User Pools y Identity Pools. User Pools actúa como un directorio de usuarios que proporciona operaciones de registro e inicio de sesión, mientras que Identity Pools permite otorgar a los usuarios identidades temporales para acceder a otros servicios de AWS, facilitando la creación de experiencias de usuario personalizadas y seguras. AWS Cognito es especialmente útil para desarrolladores que buscan implementar soluciones de autenticación y autorización robustas sin la complejidad de administrar la infraestructura de seguridad. Ofrece escalabilidad, facilidad de uso y una integración profunda con el ecosistema de AWS, lo que lo convierte en una opción atractiva para aplicaciones de cualquier tamaño que requieran gestión de identidades y acceso [35].

5. Practica (Firebase)

Este ejemplo se enfoca en que los usuarios puedan registrarse e iniciar sesión. Además, se les brinda la opción de registrar productos, proporcionando detalles como el nombre del producto y el proveedor. A través de este caso práctico, mostramos cómo usar el servicio Firebase para manejar la información en una aplicación, permitiendo así entender mejor su funcionamiento. A continuación, presentamos un diagrama que muestra cómo se llevan a cabo los diferentes procesos, junto con las herramientas utilizadas para crear este ejemplo.

5.1. Tecnologías utilizadas

Tabla 1 Herramientas utilizadas
Realizado por LETURNE PLUAS JHON BYRON & CHICA VALFRE VALESKA SOFIA

Tecnologías utilizadas				
Frontend	Backend	Servidores	Versiones de java	Base de datos
HTML	Java	Apache Tomcat	JDK 19	FiresBase
CSS	SpringBoot		JDK 11	
Bootstrap	Firebase admin			
Jquery				
AngularJS				

5.2. Diagrama de ejemplo

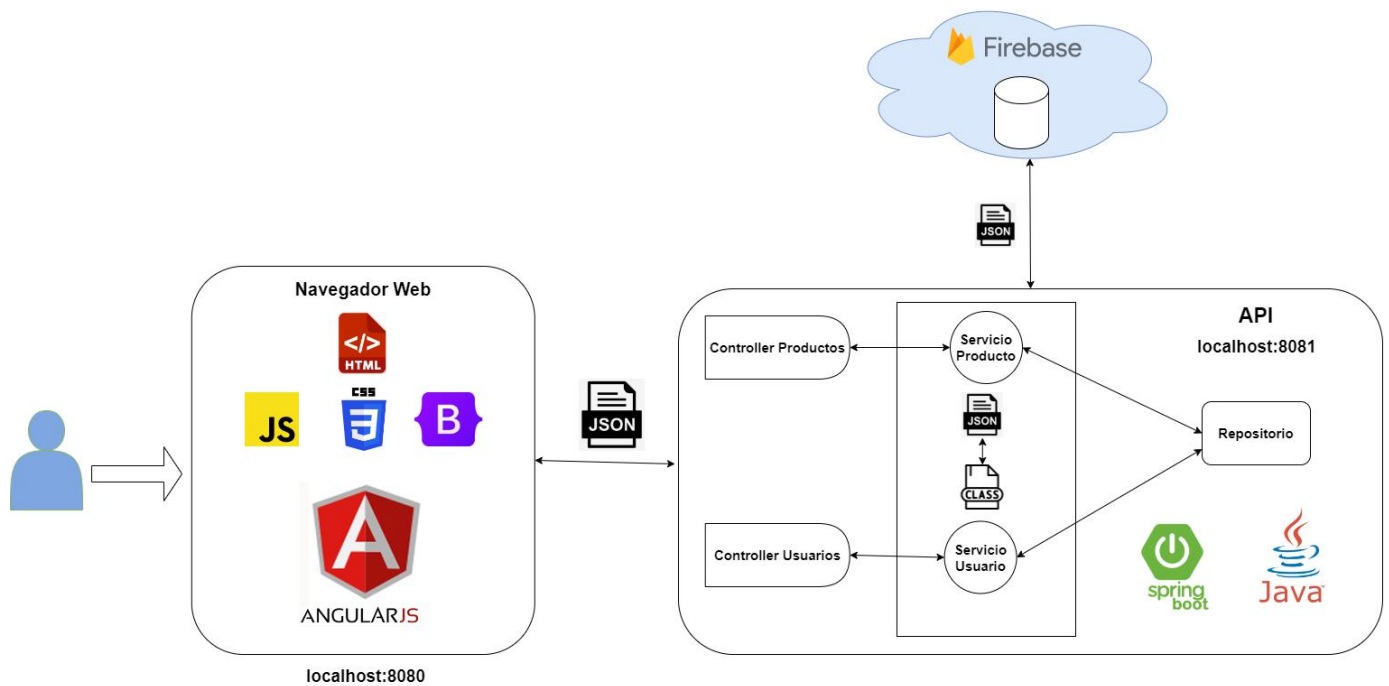


Ilustración 1 Diagrama de ejemplo de usuarios y productos
Realizado por LETURNE PLUAS JHON BYRON & CHICA VALFRE VALESKA SOFIA

5.3. Firebase datos

5.3.1. Colección productos

(default)	productos	KxYwIpmyUJZNL91au9mp
+ Iniciar colección	+ Agregar documento	+ Iniciar colección
productos >	KxYwIpmyUJZNL91au9mp >	+ Agregar campo
usuario	L35ByhMi9o4cdS0g65ks	descripcion: "el mejor producto"
	OkDDbgiiUhasKvM0a99X	descuento: 1
	SVE4gPTdoWKe1R52l85I	empresa: "S.A"
		nombre: "jhonh"
		precio_unitario: 1

Ilustración 2 Colección productos
Realizado por LETURNE PLUAS JHON BYRON & CHICA VALFRE VALESKA SOFIA

5.3.2. Colección Usuario

(default)	usuario	MF21f3vaEnluB8bdan1g
+ Iniciar colección	+ Agregar documento	+ Iniciar colección
productos	MF21f3vaEnluB8bdan1g >	+ Agregar campo
usuario >	ddAs19AAvij04aXx5wr8	contrasenia: "5994471abb01112afcc18159f6cc74b4f511b99806d"
	yqxS4S94pKD3chNnokIb	usuario: "test"

Ilustración 3 Colección usuario
Realizado por LETURNE PLUAS JHON BYRON & CHICA VALFRE VALESKA SOFIA

5.4. Organización del ejemplo

Creamos un ejemplo en Spring Boot donde utilizamos la aplicación como una API para enviar datos al cliente. En la implementación de la API, incorporamos bibliotecas que facilitan la comunicación entre Firebase y Java.

El ejemplo se estructura de la siguiente forma:

Paquete principal: en este se definen configuraciones y registro de la aplicación de firebase así también como mensajes que se utilizan para el control de errores, además de agregar encriptación de contraseñas.

Controller: Ayuda a la comunicación del Backend con el Frontend.

Modelos: Contiene las clases necesarias para gestionar los datos dentro del controlador y servicios. Es importante, ya que facilita la deserialización de los datos provenientes de Firebase.

Repositorio: Maneja diversos procesos como la inserción o eliminación de datos. Aquí se definen todas las acciones que facilitan la comunicación entre la API y Firebase.

Servicio: Se encarga de la comunicación entre el controlador y el repositorio, implementando los procesos del repositorio para que este aplique la lógica necesaria y asegure un manejo adecuado de la información.

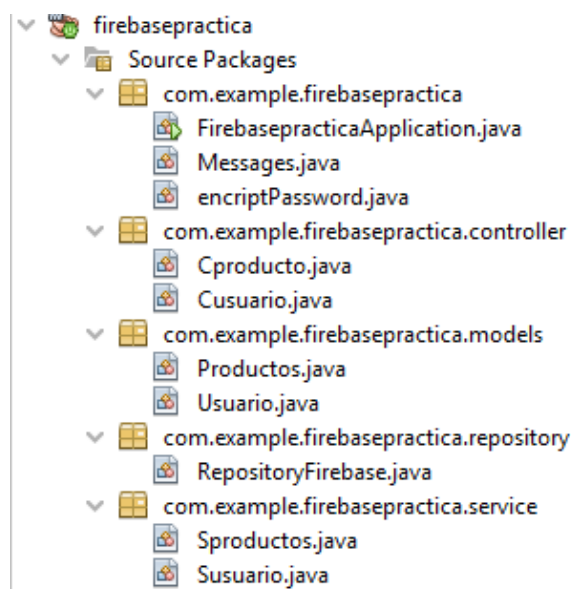


Ilustración 4 Organización de los paquetes (API)
Realizado por LETURNE PLUAS JHON BYRON & CHICA VALFRE VALESKA SOFIA

5.5. Dependencias

Dependencias que se utilizaron para la elaboración del ejemplo.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-devtools</artifactId>
  <scope>runtime</scope>
  <optional>true</optional>
</dependency>
<dependency>
  <groupId>org.postgresql</groupId>
  <artifactId>postgresql</artifactId>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <optional>true</optional>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>com.google.firebase</groupId>
  <artifactId>firebase-admin</artifactId>
  <version>9.2.0</version>
</dependency>
```

5.6. Obtención de servicio de FireBase

Para la obtención del servicio de Firebase nos dirigimos a la siguiente pagina <https://firebase.google.com/> en la cual podremos crear la cuenta para utilizar este servicio. A continuación, presentaremos los pasos necesarios para poder utilizar el servicio de Firebase.

1. Luego damos clic al botón de “Comienza Ahora” y nos pedirá crear una cuenta o iniciar con una existente.



Ilustración 5 Pagina principal FiresBase
Obtenida de: <https://firebase.google.com/>

2. Luego de haber creado la cuenta e iniciado con la misma nos pedirá crear un proyecto esto es necesario para obtener las credenciales de conexión.

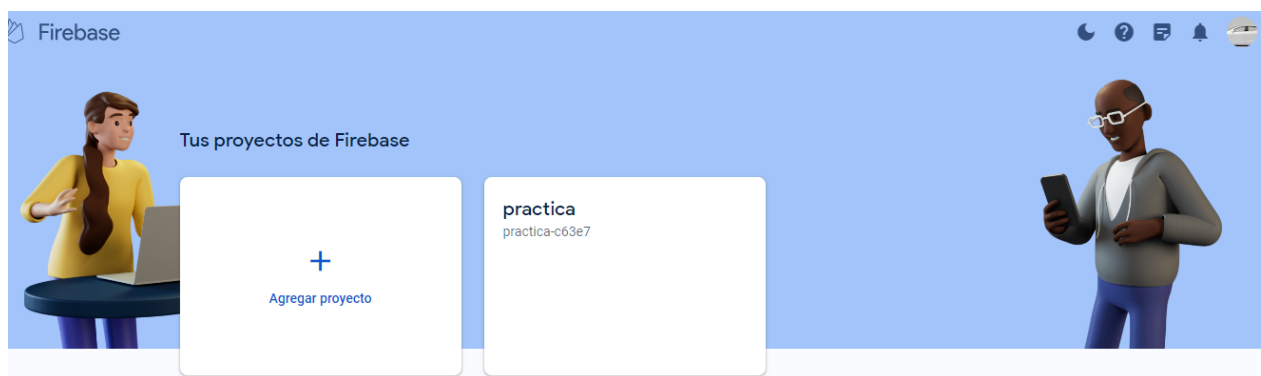


Ilustración 6 Creación de un proyecto Obtenida de:
<https://firebase.google.com/>

3. Para iniciar la creación del proyecto, hacemos clic en "Agregar proyecto". A continuación, se nos solicitará ingresar un nombre para el proyecto. Luego, se nos dará la opción de activar Google Analytics de forma opcional. Finalmente, se nos ofrecerá la opción de crear el proyecto. Es importante señalar que, si no seleccionamos Google Analytics, el proyecto se creará directamente.

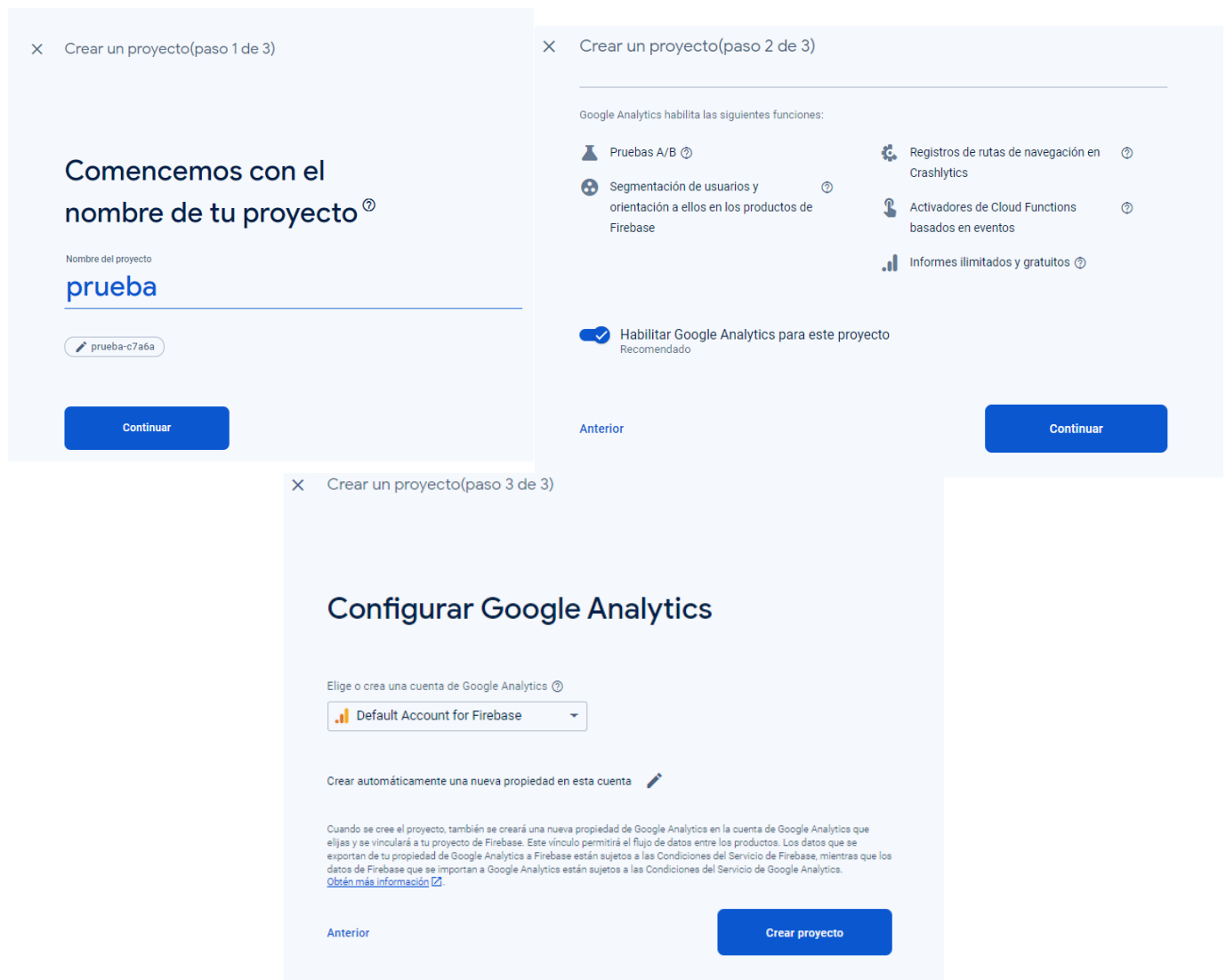


Ilustración 7 Pasos para la creación del proyecto
Obtenida de: <https://firebase.google.com/>

4. Finalmente, cuando se cree el proyecto entramos y presionamos “Firestore Database” y ya tendremos creado el proyecto correctamente.

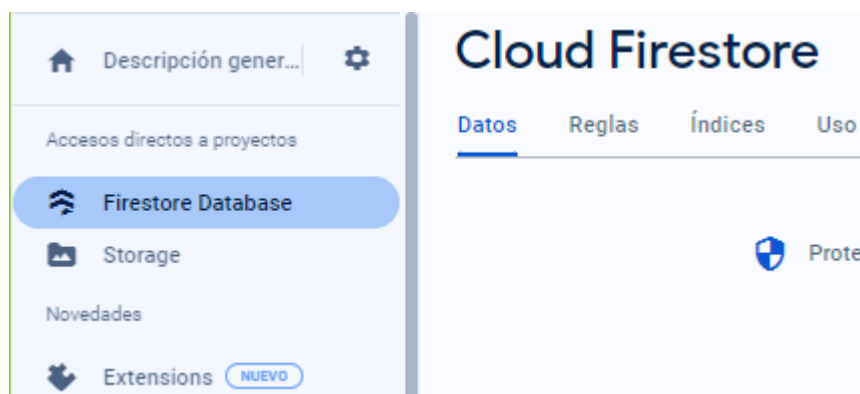


Ilustración 8 Firestore Database
Obtenida de: <https://firebase.google.com/>

5. Para generar la clave privada y utilizar el servicio desde cualquier lenguaje nos dirigimos hacia las configuraciones del proyecto y nos dirigimos a la sección de “Cuentas de servicio”. Luego seleccionamos el lenguaje de programación en el cual queremos utilizarlo y finalmente generamos la clave privada.

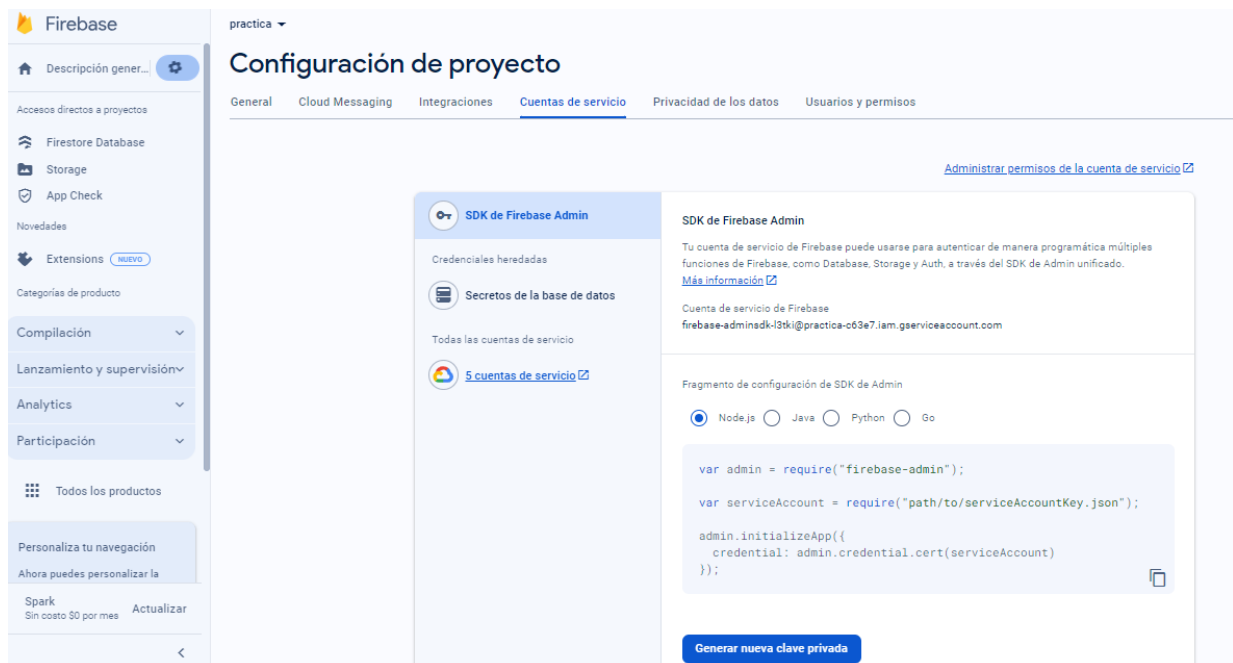


Ilustración 9 Obtención de clave privada
Obtenida de: <https://firebase.google.com/>

6. Finalmente nos generada un archivo en formato JSON el cual contendrá información personal y la clave privada para la conexión al servicio.

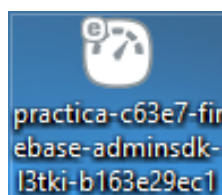


Ilustración 10 Archivo JSON (Clave Privada)
Realizado por LETURNE PLUAS JHON BYRON & CHICA VALFRE VALESKA SOFIA

5.7. Paquete principal

5.7.1. Utilización de Firebase Java y Registro aplicación

Al iniciar la aplicación Spring Boot, necesitamos configurar el inicio de Firebase, haciendo referencia al archivo JSON que previamente creamos con la clave privada. Es esencial tener en cuenta que el registro de la aplicación debe realizarse solo una vez durante la ejecución de la

API. Si este proceso se realiza incorrectamente, no será posible utilizar el servicio, ya que solo se permite un registro único.

```
@SpringBootApplication
public class FirebasepracticaApplication {

    public static Boolean registrado = true;

    public static void main(String[] args) throws IOException {
        SpringApplication.run(FirebasepracticaApplication.class, args);
        if (FirebaseApp.getApps().size() <= 0) {
            initFiresbase();
        }
    }

    public static void initFiresbase() throws IOException {
        InputStream serviceAccount = null;
        try {
            serviceAccount = new
FileInputStream(System.getProperty("user.dir") +
"\\src\\main\\resources\\autenticacion.json");
            GoogleCredentials credentials =
GoogleCredentials.fromStream(serviceAccount);
            FirebaseOptions options = new FirebaseOptions.Builder()
                .setCredentials(credentials)
                .build();
            FirebaseApp.initializeApp(options);

        } catch (FileNotFoundException ex) {
            Logger.getLogger(RepositoryFirebase.class.getName()).log(Level.SEVERE, null, ex);
        } finally {
            try {
                serviceAccount.close();
            } catch (IOException ex) {
                Logger.getLogger(RepositoryFirebase.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    }
}
```

Para lograr esto, establecemos una condición que nos indique si la aplicación ya ha sido registrada. Si esta condición es falsa, la aplicación se registrará utilizando el método **initFirebase**. En este proceso de registro, es necesario proporcionar la ruta del archivo JSON al objeto **FileInputStream**. Luego, este archivo se procesará mediante el método

GoogleCredentials y, finalmente, se registrará a través de **FirebaseApp.initializeApp(options)**.

5.7.2. Messages

También se definen mensajes que se los utilizara para la comunicación con el cliente y los modelos.

```
public class Messages
{
    public static String usuario_inexistente="El usuario no esta registrado en el sistema";
    public static String error="error";
    public static String error_datos_incompletos="Los datos estan incompletos";
    public static String sucessInsert="Sucess";
    public static String sucessInsertdescripcion="Transaccion completada exitosamente";
    public static String INSERT="insert";
    public static String UPDATE="update";
}
```

5.7.3. Encriptación de contraseñas

También se define una clase la cual ayuda a la encriptación de las contraseñas de los usuarios.

```
public class encryptPassword
{
    public String encryptPasswordUser(String pwd)
    {
        return DigestUtils.sha256Hex(pwd);
    }
}
```

5.8. Modelos

En estas, se establecen los atributos que contendrá cada modelo, así como el nombre de la colección y los atributos que se utilizarán para Firebase. Es importante tener en cuenta que la colección se asocia con el nombre de la tabla.

5.8.1. Modelo usuario

```
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public class Usuario
{
```

```

    public static String coleccionUsuario="usuario";

    public static String PusuarioK="usuario";
    public static String PcontraseniaK="contrasenia";

    String id;
    String usuario;
    String contrasenia;
    LocalDateTime fecha_creacion=LocalDateTime.now();
}

```

5.8.2. Modelo producto

```

@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public class Productos
{
    public static String coleccionProductos="productos";

    public static String PnombreK="nombre";
    public static String PempresaK="empresa";
    public static String Pprecio_unitarioK="precio_unitario";
    public static String PdescuentoK="descuento";
    public static String PproveedorK="proveedor";
    public static String PdescripcionK="descripcion";

    String id;
    String nombre;
    String empresa;
    Double precio_unitario;
    Double descuento;
    String proveedor;
    String descripcion;
    LocalDateTime fecha_creacion=LocalDateTime.now();
}

```

5.9. Repositorio

Se centra en la comunicación con Firebase, incorporando métodos para insertar, eliminar, leer, modificar y buscar datos. Es importante tener en cuenta que para que estos procesos funcionen, la aplicación debe haber sido registrada previamente.

5.9.1. Insertar y actualizar

Firebase utiliza colecciones y mapas para registrar y actualizar documentos. Es importante destacar que la colección se refiere al nombre de la tabla o documento en el contexto de este servicio. Para la inserción de datos, primero se crea una referencia con Firestore, que contiene los métodos necesarios para la comunicación. Luego, mediante `collection`, se hace referencia a la colección donde se desea insertar el dato, y mediante `document`, se especifica el nombre del identificador único que tendrá el documento. Es relevante mencionar que si no se le proporcionan parámetros de entrada a `document()`, se agregará un identificador único alfanumérico automáticamente al documento.

Por otro lado, al referirse a la colección y al mismo identificador único que se crea, al enviar un nuevo mapa con los atributos del documento, solo se actualizarán los datos asociados a ese identificador.

```
public void FirebaseInsertOrUpdate(String coleccion, Map<String, Object>
data,String documento)
{
    try
    {
        Firestore fs = FirestoreClient.getFirestore();
        if(fs!=null)
        {
            DocumentReference docRef = null;
            if(documento==null)
                docRef=fs.collection(coleccion).document();
            else
                docRef=fs.collection(coleccion).document(documento);
            ApiFuture<WriteResult> result = docRef.set(data);
            System.out.println("Update time : " +
result.get().getUpdateTime());
            return;
        }
        System.out.println(this.errorConeccion);
    }
    catch(Exception ex)
    {
        System.out.println("Error: "+ex.getMessage());
    }
}
```


5.9.2. Lectura de datos

Para obtener todos los datos asociados a una colección, de manera similar, se obtiene una referencia a Firestore y luego a la colección de la cual se desea extraer datos. Posteriormente, estos datos se guardarán en una lista de tipo QueryDocumentSnapshot, la cual se puede recorrer para obtener la información necesaria.

```
public List<QueryDocumentSnapshot> FirebaseRead(String coleccion)
{
    try
    {
        Firestore fs = FirestoreClient.getFirestore();
        if(fs!=null)
        {
            ApiFuture<QuerySnapshot> query = fs.collection(coleccion).get();
            QuerySnapshot querySnapshot = query.get();
            List<QueryDocumentSnapshot> documents =
querySnapshot.getDocuments();
            return documents;
        }
        System.out.println(this.errorConeccion);
    }
    catch(Exception ex)
    {
        System.out.println("Error: "+ex.getMessage());
    }
    return null;
}
```

5.9.3. Búsqueda de datos

Para buscar datos, es necesario especificar la colección, el campo y el valor que se desea encontrar en los documentos. Utilizamos la función llamada whereEqualTo, que requiere el campo y el valor a buscar como parámetros de entrada. Al igual que el método anterior, devuelve una lista con todas las coincidencias encontradas.

```
public List<QueryDocumentSnapshot> buscar(String coleccion,String campo,String
valor)
{
    try {
        Firestore fs = FirestoreClient.getFirestore();
        ApiFuture<QuerySnapshot> qs =
fs.collection(coleccion).whereEqualTo(campo, valor).get();
        List<QueryDocumentSnapshot> documents = qs.get().getDocuments();
        return documents;
    } catch (InterruptedException ex) {
```

```

        Logger.getLogger(RepositoryFirebase.class.getName()).log(Level.SEVERE,
null, ex);
    } catch (ExecutionException ex) {
        Logger.getLogger(RepositoryFirebase.class.getName()).log(Level.SEVERE,
null, ex);
    }
    return null;
}

```

5.9.4. Eliminación de documentos

Para eliminar datos en Firebase, solo es necesario especificar la colección y el ID del documento que se desea eliminar. Utilizando el método `delete()`, se elimina el documento a través de la referencia obtenida anteriormente.

```

public void eliminar(String coleccion,String id)
{
    try
    {
        Firestore fs = FirestoreClient.getFirestore();
        DocumentReference docRef = fs.collection(coleccion).document(id);
        docRef.delete();
        System.out.println("Documento eliminado");
    }
    catch(Exception ex)
    {
        System.out.println(ex.getMessage());
        Logger.getLogger(RepositoryFirebase.class.getName()).log(Level.SEVERE,
null, ex);
    }
}

```

5.10. Servicios

En esta sección se definen las operaciones que llevará a cabo cada servicio que se implemente en el software. En esta parte, se emplean los métodos previamente creados en el repositorio, pero el servicio los adapta según sus propias preferencias o necesidades. Para lograrlo, es necesario instanciar un repositorio en el servicio y utilizarlo tanto en los métodos de inserción, eliminación o búsqueda de datos.

También se puede observar que en estos servicios se encargan de deserializar los datos provenientes de Firebase, parseándolos a los modelos creados previamente para facilitar el manejo de los datos de manera más cómoda. Esto se logra mediante la función `document.toObject(Clase.class)` que ayuda en este proceso.

5.10.1. Servicio Usuario

```
public class Susuario
{
    RepositoryFirebase rf;

    public Susuario()
    {
        rf=new RepositoryFirebase();
    }

    public int insert(Map<String, Object> data)
    {
        try {
            Object valor=data.get(Usuario.PusuarioK);
            if(buscar(valor.toString()).size()<=0){
                rf.FirebaseInsertOrUpdate(Usuario.coleccionUsuario,
data,null);
            }
        } catch (InterruptedException ex) {
            Logger.getLogger(Susuario.class.getName()).log(Level.SEVERE, null,
ex);
        } catch (ExecutionException ex) {
            Logger.getLogger(Susuario.class.getName()).log(Level.SEVERE, null,
ex);
        }
        return 0;
    }

    public List<Usuario> buscar(String valor) throws InterruptedException,
ExecutionException
    {
        List<QueryDocumentSnapshot> documents
=rf.buscar(Usuario.coleccionUsuario, Usuario.PusuarioK,valor);
        Usuario u;
        List<Usuario> lstusuario=new ArrayList<>();
        for (DocumentSnapshot document : documents) {
            u=document.toObject(Usuario.class);
            lstusuario.add(u);
        }
        return lstusuario;
    }
}
```

5.10.2. Servicio productos

```
public class Sproductos
{
    RepositoryFirebase rf;
```

```

public Sproductos()
{
    rf=new RepositoryFirebase();
}

public void insert(Map<String, Object> data)
{
    rf.FirebaseInsertOrUpdate(Productos.coleccionProductos, data,null);
}

public void update(Map<String, Object> data,String iddocumento)
{
    rf.FirebaseInsertOrUpdate(Productos.coleccionProductos,
data,iddocumento);
}

public List<Productos> read()
{
    List<QueryDocumentSnapshot>
documents=rf.FirebaseRead(Productos.coleccionProductos);
    Productos p;
    List<Productos> lstproductos=new ArrayList<>();

    for(QueryDocumentSnapshot document : documents)
    {
        System.out.println(document.getId());
        p=document.toObject(Productos.class);
        p.setId(document.getId());
        lstproductos.add(p);
    }
    return lstproductos;
}

public List<Productos> buscar(String campo,String valor) throws
InterruptedException, ExecutionException
{
    List<QueryDocumentSnapshot> documents
=rf.buscar(Productos.coleccionProductos, campo,valor);
    Productos p;
    List<Productos> lstproduc=new ArrayList<>();
    for (DocumentSnapshot document : documents) {
        System.out.println(document.getId());
        lstproduc.add(document.toObject(Productos.class));}
    return lstproduc;
}

public Integer eliminar(String id)
{

```

```

        try
        {
            rf.eliminar(Productos.coleccionProductos,id);
            return 1;
        }
        catch(Exception ex)
        {

        }
        return 0;
    }
}

```

5.11. Controladores

Dentro de esta se definen los procesos como inserción, actualización o eliminación de datos que se comunicaran con el usuario y la API. Hay que tener en cuenta de configurar correctamente los CrossOrigin para que el cliente no tenga problemas al comunicarse con la API.

5.11.1. Controlador Usuario

Aquí se muestra un ejemplo de cómo la función de usuario obtiene los datos de la vista mediante @RequestParam, además de cómo implementa el servicio para llevar a cabo el proceso deseado.

```

@RestController()
@RequestMapping("/api/usuario")
@CrossOrigin("*")
public class Cusuario
{
    @PostMapping("/login")
    public ResponseEntity<?> login(@RequestParam(name="usuario") String
usuarioF,
                                     @RequestParam(name="contrasenia") String
contraseniaF)
    {
        Map<String,Object> response=new HashMap();
        Susuario sp=new Susuario();
        try
        {
            if(usuarioF!="" && contraseniaF!="")
            {
                List<Usuario> lstusuario=sp.buscar(usuarioF);
                if(lstusuario.size(>0)
                {
                    Usuario usu=lstusuario.get(0);

```

```

        encryptPassword ep=new encryptPassword();
        if(usuarioF.equals(usu.getUsuario()) &&
ep.encryptPasswordUser(contraseniaF).equals(usu.getContrasenia()))
        {
            response.put(Messages.sucessInsert, true);
            return new
ResponseEntity<Map<String,Object>>(response,HttpStatus.OK);
        }else
        {
            response.put(Messages.sucessInsert, false);
            return new
ResponseEntity<Map<String,Object>>(response,HttpStatus.OK);
        }
    }
    else
    {
        response.put(Messages.error,
Messages.usuario_inexistente);
        return new
ResponseEntity<Map<String,Object>>(response,HttpStatus.NOT_ACCEPTABLE);
    }
}
else
{
    response.put(Messages.error,
Messages.error_datos_incompletos);
    return new
ResponseEntity<Map<String,Object>>(response,HttpStatus.NOT_ACCEPTABLE);
}
}
catch(Exception ex)
{
    response.put(Messages.error, ex.getMessage());
    return new
ResponseEntity<Map<String,Object>>(response,HttpStatus.NOT_ACCEPTABLE);
}
}
}

```

5.11.2. Controlador Producto

En este caso, al igual que el método anterior, se presenta un ejemplo de inserción de productos. En este caso, la función recibe directamente un objeto. Para lograrlo, el cliente debe enviar un JSON con nombres coincidentes a los atributos de la clase, permitiendo así una correcta asociación y creación del objeto mediante la instancia. Posteriormente, se crea un mapa con los

datos obtenidos y se procede a realizar la inserción. Cabe destacar que es necesario haber instanciado previamente un servicio para su correcta utilización.

```
@RestController()
@RequestMapping("/api/producto")
@CrossOrigin("*")
public class Cproducto
{
    @PostMapping("/insertar")
    public ResponseEntity<?> insertar(Productos productos)
    {
        Map<String,Object> response=new HashMap();
        Sproductos sp=new Sproductos();
        try
        {
            if(productos.getNombre()!="" &&
productos.getPrecio_unitario()!=null && productos.getDescuento()!=null
            && productos.getEmpresa()!="" &&
productos.getDescripcion()!=null && productos.getProveedor()!=null)
            {
                Map<String,Object> data=new HashMap();
                data.put(Productos.PnombreK, productos.getNombre());
                data.put(Productos.PempresaK, productos.getEmpresa());
                data.put(Productos.Pprecio_unitarioK,
productos.getPrecio_unitario());
                data.put(Productos.PdescuentoK, productos.getDescuento());
                data.put(Productos.PproveedorK, productos.getProveedor());
                data.put(Productos.PdescripcionK, productos.getDescripcion());
                sp.insert(data);

                return new
ResponseEntity<Map<String,Object>>(data,HttpStatus.OK);
            }else
            {
                response.put(Messages.error,
Messages.error_datos_incompletos);
                return new
ResponseEntity<Map<String,Object>>(response,HttpStatus.NOT_ACCEPTABLE);
            }
        }
        catch(Exception ex)
        {
            response.put(Messages.error, ex.getMessage());
            return new
ResponseEntity<Map<String,Object>>(response,HttpStatus.NOT_ACCEPTABLE);
        }
    }
}
```

5.12. Vista Cliente

Para implementar el consumo de la API, se empleó AngularJS y Ajax, facilitando así la tarea de enviar y recibir datos de los servicios de manera más sencilla. En esta sección, no es necesario abordar los procesos relacionados con Firebase, ya que estos son manejados por el backend.

5.12.1. Login

```
angular.module("app",[])
.controller("loginController", function ($scope, $http)
{
    $scope.url='http://localhost:8081/api/usuario';
    $scope.usuario="";
    $scope.contrasenia="";

    $scope.login=(form)=> {
        console.log(form);
        if($scope.usuario!="" && $scope.contrasenia!=""){
            let formData=new FormData();
            formData.append("usuario",form.usuario.$viewValue);
            formData.append("contrasenia",form.contrasenia.$viewValue);
            $.ajax({
                method:"POST",
                url:$scope.url+"/login",
                processData:false,
                contentType: false,
                data:formData,
                beforeSend: function (xhr) {
                    console.log("cargando...");
                },
                success: function (data) {
                    console.log(data);
                    if(data.Sucess==true)
                        location.href="app.html#!/";
                    else
                        message_toast("Usuario o contraseña incorrecta");
                },
                error: function (objXMLHttpRequest)
                {
                    message_toast(objXMLHttpRequest.responseJSON.error);
                    console.log("error: ", objXMLHttpRequest);
                }
            });
        }
        else{
            message_toast("Error campos vacios");
        }
    }
});
```


5.12.2. Registro usuario

```
angular.module("app",[])
.controller("signController", function ($scope, $http)
{
    $scope.url='http://localhost:8081/api/usuario';
    $scope.usuario="";
    $scope.contrasenia="";

    $scope.signup=(form)=> {
        console.log(form);
        if($scope.usuario!="" && $scope.contrasenia!=""){
            let formData=new FormData();
            formData.append("usuario",form.usuario.$viewValue);
            formData.append("contrasenia",form.contrasenia.$viewValue);
            $.ajax({
                method:"POST",
                url:$scope.url+"/insertar",
                processData:false,
                contentType: false,
                data:formData,
                beforeSend: function (xhr) {
                    console.log("cargando...");
                },
                success: function (data) {
                    location.href="/viewFiresbase/"
                },
                error: function (objXMLHttpRequest)
                {
                    message_toast(objXMLHttpRequest.responseJSON.error);
                    console.log("error: ", objXMLHttpRequest);
                }
            });
        }else{
            message_toast("Error campos vacios");
        }
    }
});
```

5.12.3. Registro productos

```
angular.module("app")
.controller("rest_controller_productos", function ($scope, $http) {

    $scope.productos=undefined;
    $scope.url='http://localhost:8081/api/producto';

    $scope.id="";
```

```

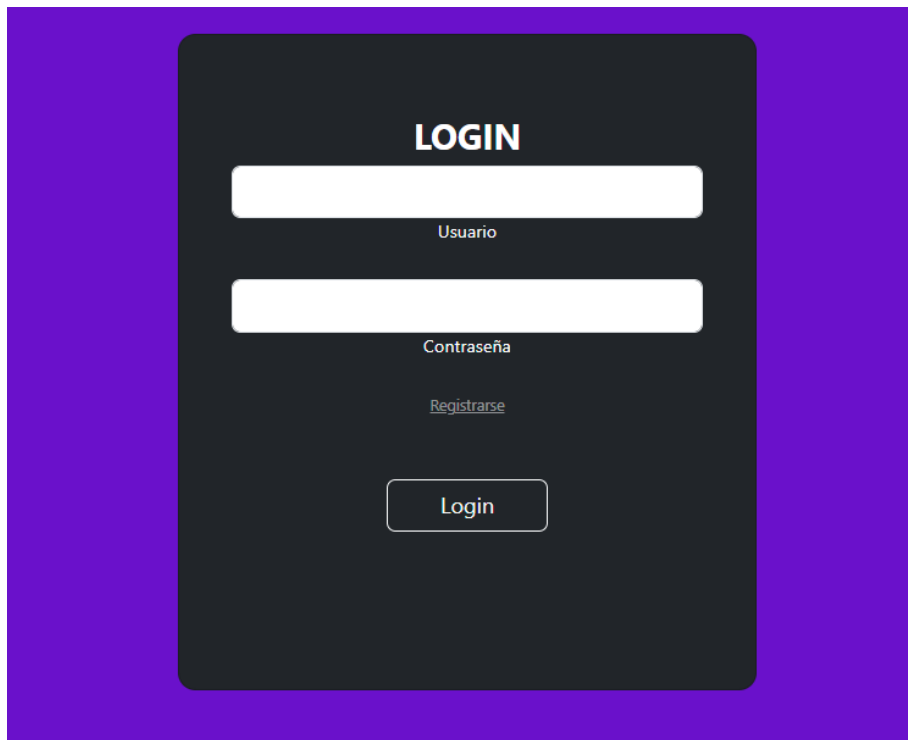
$scope.nombre="";
$scope.empresa="";
$scope.precio_unitario="";
$scope.descuento="";
$scope.proveedor="";
$scope.descripcion="";

$scope.actualizar_guardar=(form)=> {
    console.log(form);

    let formData=new FormData();
    if(form.id.$viewValue!="")
        formData.append("id",form.id.$viewValue);
    formData.append("nombre",form.nombre.$viewValue);
    formData.append("empresa",form.empresa.$viewValue);
    formData.append("precio_unitario",form.precio_unitario.$viewValue);
    formData.append("descuento",form.descuento.$viewValue);
    formData.append("proveedor",form.proveedor.$viewValue);
    formData.append("descripcion",form.descripcion.$viewValue);
    $.ajax({
        method:"POST",
        url:$scope.url+"/"+(form.id.$viewValue=="?"?"insertar":"actualizar"
    ),
        processData:false,
        contentType: false,
        data:formData,
        beforeSend: function (xhr) {
            console.log("cargando...");
        },
        success: function (data) {
            $scope.$apply(function()
            {
                consumer_rest();
                if(form.id.$viewValue=="")
                    message_toast("Información insertada correctamente");
                else
                    message_toast("Información actualizada
correctamente");
            });
            console.log(data);
        },
        error: function (objXMLHttpRequest)
        {
            message_toast(objXMLHttpRequest.responseJSON.error);
            console.log("error: ", objXMLHttpRequest);
        }
    });
}
});

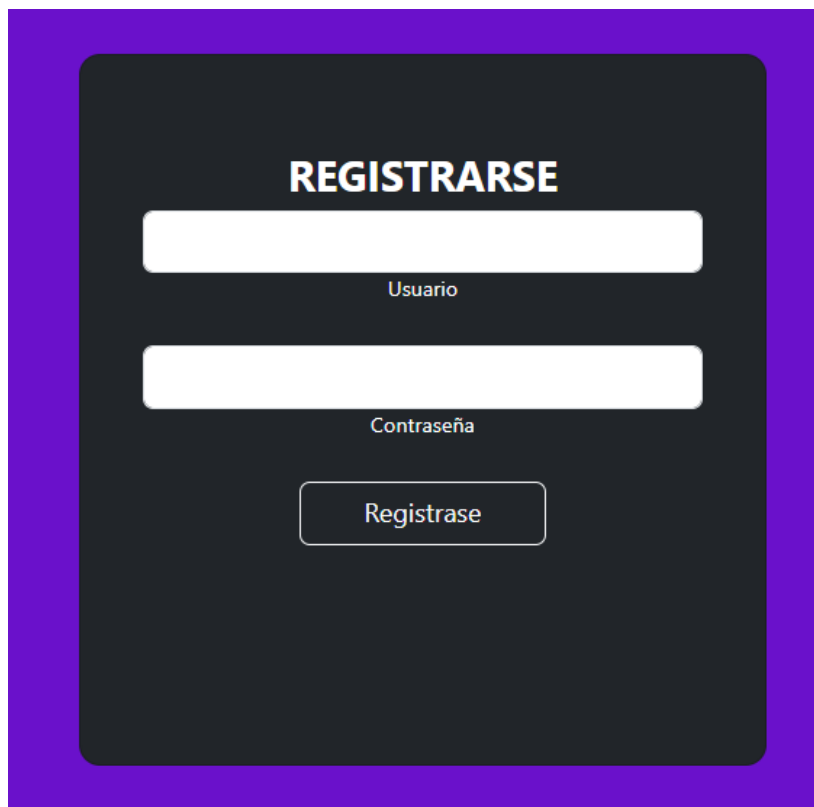
```

5.13. Interfaces




A login interface mockup featuring a dark gray rounded rectangle centered on a vibrant purple background. The rectangle is titled "LOGIN" in bold white uppercase letters. Below the title are two white rectangular input fields. The first field is labeled "Usuario" and the second is labeled "Contraseña" in small white text. Below the password field is a white text link that reads "Registrarse". At the bottom of the form is a white rounded rectangular button with the text "Login".

Ilustración 11 Login
Realizado por LETURNE PLUAS JHON BYRON & CHICA VALFRE VALESKA SOFIA



A register user interface mockup featuring a dark gray rounded rectangle centered on a vibrant purple background. The rectangle is titled "REGISTRARSE" in bold white uppercase letters. Below the title are two white rectangular input fields. The first field is labeled "Usuario" and the second is labeled "Contraseña" in small white text. Below the password field is a white rounded rectangular button with the text "Regístrate".

Ilustración 12 Registro usuario
Realizado por LETURNE PLUAS JHON BYRON & CHICA VALFRE VALESKA SOFIA



PRODUCTOS

Nombre producto....

Buscar

Nombre	Empresa	Precio unitario	Descuento	Proveedor	Descripción	Editar	Eliminar
jhonh	S.A	1	1	YO MISMO	el mejor producto	Editar	Eliminar
jhonh	S.A	1	1	YO MISMO	el mejor producto	Editar	Eliminar
jhonh	S.A	1	1	YO MISMO	el mejor producto	Editar	Eliminar
fcdfdf	petsi:V	1	1	Jhon s.ajj	samsung j7	Editar	Eliminar

Ilustración 13 Productos registrados
Realizado por LETURNE PLUAS JHON BYRON & CHICA VALFRE VALESKA SOFIA

Producto

Nombre

Ingrese la nombre de producto

Empresa

Ingrese nombre de la empresa

Precio unitario

Ingrese precio unitario

Descuento

Ingrese descuento

Proveedor

Ingrese proveedor

Descripcion

Ingrese desripcion

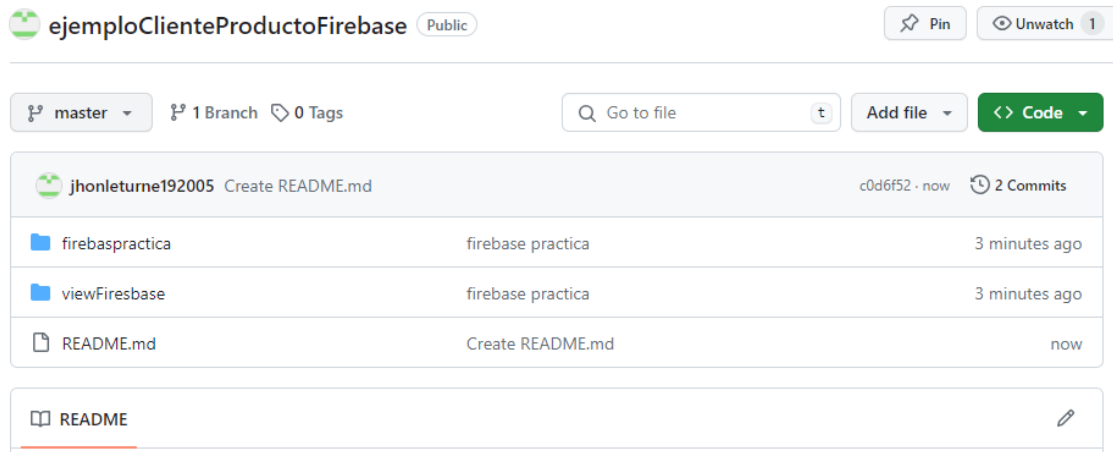
Guardar

Ilustración 14 Formulario productos
Realizado por LETURNE PLUAS JHON BYRON & CHICA VALFRE VALESKA SOFIA

5.14. GitHub ejemplo

Se podrá descargar todos los archivos necesarios para ejecutar el ejemplo anteriormente creado.

Enlace: <https://github.com/jhonleturne192005/ejemploClienteProductoFirebase.git>



*Ilustración 15 GitHub
Realizado por LETURNE PLUAS JHON BYRON & CHICA VALFRE VALESKA SOFIA*

6. Conclusión

El enfoque de desarrollo de aplicaciones en capas, al dividir una aplicación en componentes independientes, proporciona una estructura organizada y modular. Al implementar capas claramente definidas, como frontend y backend, se facilita el mantenimiento, la escalabilidad y la colaboración en el desarrollo de software. Con Firebase, la capa backend puede gestionar eficientemente la lógica de la aplicación y la interacción con la base de datos, mientras que la capa frontend se centra en la presentación de datos al usuario, permitiendo un desarrollo más estructurado y eficiente.

A través de este ejemplo, podemos observar cómo se gestionan diversos procesos para el manejo de datos con Firebase. Es importante destacar que la lógica de Firebase se centra en documentos, donde cada documento representa atributos y la colección equivale a una tabla. Al subdividir la lógica de la aplicación, obtenemos un mayor control ante posibles cambios en el futuro, además de contar con un código más organizado, accesible y fácil de mantener.

7. Bibliografia

- [1] A. Wang, “Discussion on the Application of Layering Technology in the Development of Computer Software,” *J Phys Conf Ser*, vol. 1748, no. 2, p. 022003, Jan. 2021, doi: 10.1088/1742-6596/1748/2/022003.
- [2] Luo Qiqiang and Chen Shuang, “The Application of Layering Technology in Computer Software Development,” in *2018 7th International Conference on Advanced Materials and Computer Science (ICAMCS 2018)*, Clausius Scientific Press, 2018. doi: 10.23977/icamcs.2018.090.
- [3] W. Zhuxian, G. Xingmin, and F. Peng, “The Application of Layering Technology in Computer Software Development,” in *2017 International Conference on Robots & Intelligent System (ICRIS)*, IEEE, Oct. 2017, pp. 329–333. doi: 10.1109/ICRIS.2017.89.
- [4] P. Späth, “Firebase,” in *Pro Android with Kotlin*, Berkeley, CA: Apress, 2022, pp. 363–366. doi: 10.1007/978-1-4842-8745-3_10.
- [5] L. Moroney, “Firebase App Indexing,” in *The Definitive Guide to Firebase*, Berkeley, CA: Apress, 2017, pp. 189–201. doi: 10.1007/978-1-4842-2943-9_10.
- [6] P. R. Saraf, “A Review on Firebase (Backend as A Service) for Mobile Application Development,” *Int J Res Appl Sci Eng Technol*, vol. 10, no. 1, pp. 967–971, Jan. 2022, doi: 10.22214/ijraset.2022.39958.
- [7] P. R. Saraf, “A Review on Firebase (Backend as A Service) for Mobile Application Development,” *Int J Res Appl Sci Eng Technol*, vol. 10, no. 1, pp. 967–971, Jan. 2022, doi: 10.22214/ijraset.2022.39958.
- [8] A. Bahtiar Semma, M. Ali, M. Saerozi, M. Mansur, and K. Kusriani, “Cloud computing: google firebase firestore optimization analysis,” *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 29, no. 3, p. 1719, Mar. 2023, doi: 10.11591/ijeecs.v29.i3.pp1719-1728.
- [9] Md. S. A. Khan, A. R. Farabi, and A. Iqbal, “What Do Firebase Developers Discuss About? An Empirical Study on Stack Overflow Posts,” in *Proceedings of the 9th International Conference on Networking, Systems and Security*, New York, NY, USA: ACM, Dec. 2022, pp. 63–74. doi: 10.1145/3569551.3569558.
- [10] R. Patnaik, R. Pradhan, S. Rath, C. Mishra, and L. Mohanty, “Study on Google Firebase for Real-Time Web Messaging,” 2021, pp. 461–469. doi: 10.1007/978-981-15-5971-6_50.
- [11] L. Moroney, “The Firebase Realtime Database,” in *The Definitive Guide to Firebase*, Berkeley, CA: Apress, 2017, pp. 51–71. doi: 10.1007/978-1-4842-2943-9_3.
- [12] S. De Carli, “Introduction to Firebase,” in *Build Mobile Apps with SwiftUI and Firebase*, Berkeley, CA: Apress, 2023, pp. 17–42. doi: 10.1007/978-1-4842-9452-9_2.

- [13] R. Patnaik, R. Pradhan, S. Rath, C. Mishra, and L. Mohanty, "Study on Google Firebase for Real-Time Web Messaging," 2021, pp. 461–469. doi: 10.1007/978-981-15-5971-6_50.
- [14] H. Gupta, D. Roy, A. Garg, H. Agarwal, and T. Dudeja, "Firebase Cloud: Web-Based Streaming Platform," in *2022 Fifth International Conference on Computational Intelligence and Communication Technologies (CCICT)*, IEEE, Jul. 2022, pp. 526–532. doi: 10.1109/CCICT56684.2022.00098.
- [15] M. Copeland, J. Soh, A. Puca, M. Manning, and D. Gollob, "Microsoft Azure and Cloud Computing," in *Microsoft Azure*, Berkeley, CA: Apress, 2015, pp. 3–26. doi: 10.1007/978-1-4842-1043-7_1.
- [16] P. Mazumdar, S. Agarwal, and A. Banerjee, "Microsoft Azure Storage," in *Pro SQL Server on Microsoft Azure*, Berkeley, CA: Apress, 2016, pp. 35–52. doi: 10.1007/978-1-4842-2083-2_3.
- [17] O. Poppe *et al.*, "Seagull: an infrastructure for load prediction and optimized resource allocation," *Proceedings of the VLDB Endowment*, vol. 14, no. 2, pp. 154–162, Oct. 2020, doi: 10.14778/3425879.3425886.
- [18] T. Redkar and T. Guidici, "Windows Azure Platform Overview," in *Windows Azure Platform*, Berkeley, CA: Apress, 2011, pp. 1–47. doi: 10.1007/978-1-4302-3564-4_1.
- [19] D. Subbarao, B. Raju, F. Anjum, C. venkateswara Rao, and B. M. Reddy, "Microsoft Azure active directory for next level authentication to provide a seamless single sign-on experience," *Appl Nanosci*, vol. 13, no. 2, pp. 1655–1664, Feb. 2023, doi: 10.1007/s13204-021-02021-0.
- [20] S. K, A. X. K, D. Davis, and N. Jayapandian, "Internet of Things and Cloud Computing Involvement Microsoft Azure Platform," in *2022 International Conference on Edge Computing and Applications (ICECAA)*, IEEE, Oct. 2022, pp. 603–609. doi: 10.1109/ICECAA55415.2022.9936126.
- [21] N. Vermeir, "Microsoft Azure," in *Introducing .NET 6*, Berkeley, CA: Apress, 2022, pp. 221–257. doi: 10.1007/978-1-4842-7319-7_8.
- [22] P. Wiewiura, M. Malawski, and M. Piwowar, "Distributed Execution of Dynamically Defined Tasks on Microsoft Azure," 2016, pp. 291–301. doi: 10.1007/978-3-319-32149-3_28.
- [23] M. Copeland, J. Soh, A. Puca, M. Manning, and D. Gollob, "Overview of Microsoft Azure Services," in *Microsoft Azure*, Berkeley, CA: Apress, 2015, pp. 27–69. doi: 10.1007/978-1-4842-1043-7_2.
- [24] R. Barga, V. Fontama, and W. H. Tok, "Introducing Microsoft Azure Machine Learning," in *Predictive Analytics with Microsoft Azure Machine Learning*, Berkeley, CA: Apress, 2014, pp. 21–42. doi: 10.1007/978-1-4842-0445-0_2.
- [25] B. Nemade, S. Moorthy, and O. Kadam, "Cloud computing," in *Proceedings of the International Conference & Workshop on Emerging Trends in Technology - ICWET*

- '11, New York, New York, USA: ACM Press, 2011, p. 1361. doi: 10.1145/1980022.1980341.
- [26] A. Mishra, “Azure SQL from Java Applications,” in *Microsoft Azure for Java Developers*, Berkeley, CA: Apress, 2022, pp. 137–159. doi: 10.1007/978-1-4842-8251-9_7.
 - [27] M. Kromer, “Introduction to Azure Data Factory,” in *Mapping Data Flows in Azure Data Factory*, Berkeley, CA: Apress, 2022, pp. 13–26. doi: 10.1007/978-1-4842-8612-8_2.
 - [28] O. Poppe *et al.*, “Moneyball: proactive auto-scaling in Microsoft Azure SQL database serverless,” *Proceedings of the VLDB Endowment*, vol. 15, no. 6, pp. 1279–1287, Feb. 2022, doi: 10.14778/3514061.3514073.
 - [29] Z. Barać and D. Scott-Raynsford, “The Journey to Hyperscale Architecture in Azure SQL,” in *Azure SQL Hyperscale Revealed*, Berkeley, CA: Apress, 2023, pp. 3–36. doi: 10.1007/978-1-4842-9225-9_1.
 - [30] M. Zadka, “Amazon Web Services,” in *DevOps in Python*, Berkeley, CA: Apress, 2022, pp. 187–200. doi: 10.1007/978-1-4842-7996-0_13.
 - [31] O. Mustafa, “Overview of Amazon Web Services,” in *A Complete Guide to DevOps with AWS*, Berkeley, CA: Apress, 2023, pp. 1–35. doi: 10.1007/978-1-4842-9303-4_1.
 - [32] S. Hashemipour and M. Ali, “Amazon Web Services (AWS) – An Overview of the On-Demand Cloud Computing Platform,” 2020, pp. 40–47. doi: 10.1007/978-3-030-60036-5_3.
 - [33] V. D. A. Kumar, V. D. A. Kumar, H. Divakar, and R. Gokul, “Cloud enabled media streaming using Amazon Web Services,” in *2017 IEEE International Conference on Smart Technologies and Management for Computing, Communication, Controls, Energy and Materials (ICSTM)*, IEEE, Aug. 2017, pp. 195–198. doi: 10.1109/ICSTM.2017.8089150.
 - [34] S. Krause, “Tutorial: Hands on Introduction to Amazon Web Services,” in *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, IEEE, Dec. 2013, pp. xxx–xxx. doi: 10.1109/UCC.2013.16.
 - [35] Md. S. Shams, S. M. A. Ali, A. M. Sattar, and M. Kr. Ranjan, “Study and Analysis of Components and Impact of AWS on Cloud Computing,” *RECENT TRENDS IN PARALLEL COMPUTING*, 2023, doi: 10.37591/rtpc.v10i1.515.