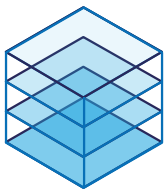


MISO

Maestría en Ingeniería de Software





Hoja de Trabajo

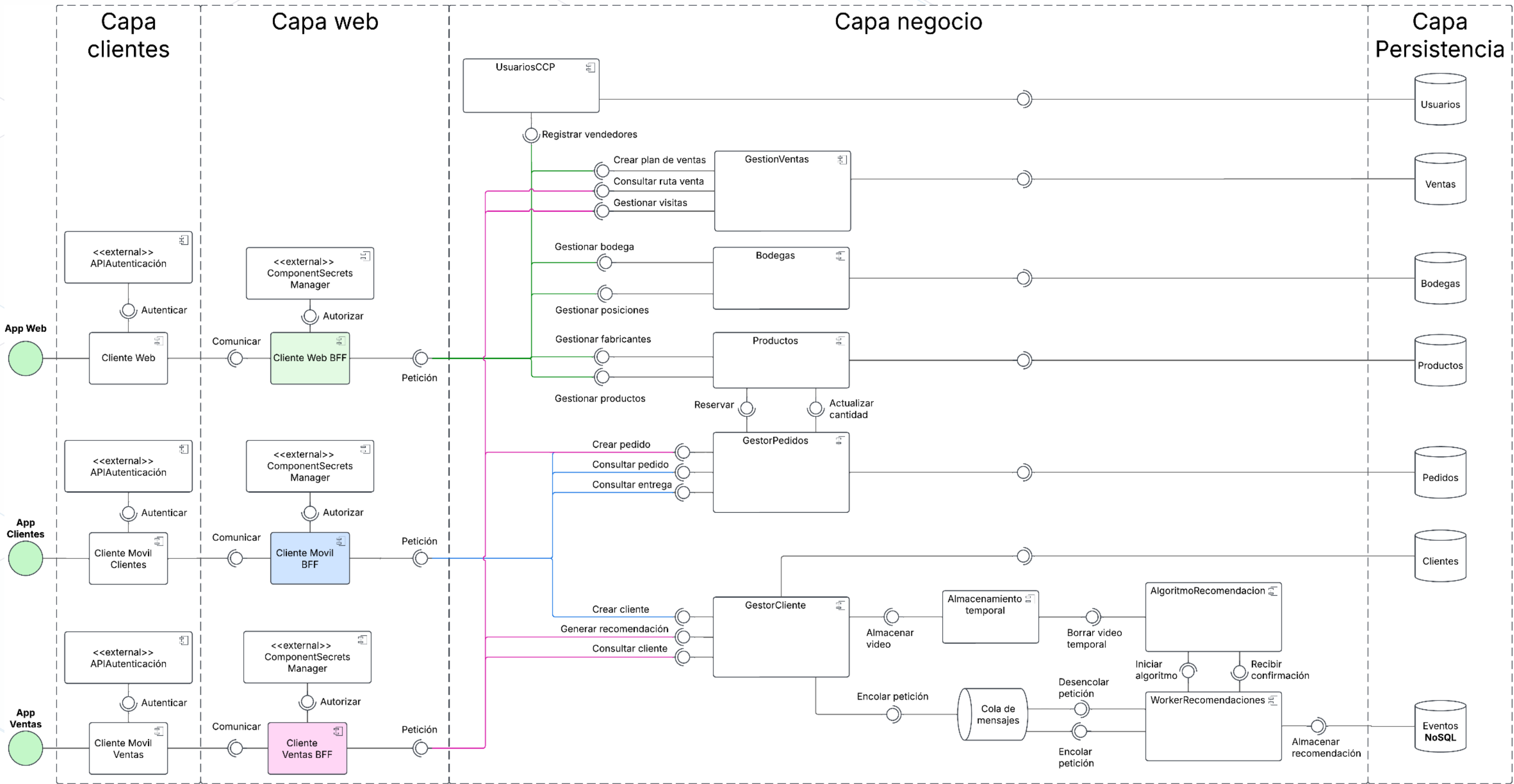
Semana 4 – Diseño detallado y experimentación

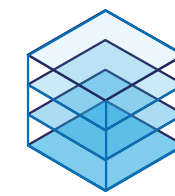


Vista Funcional – (Estilo Microservicios)

Proyecto	CCP Supply	ID	VC-001	Elaboración	Grupo 7	Versión	1.0
Vista	Funcional		Modelo	Componentes			

Convención
Estilo microservicios
 Interfaz
 Estímulo
 Componente
 Base de datos





Vista Funcional – (Estilo Microservicios)

Proyecto	CCP Supply	ID	VC-001	Elaboración	Grupo 7	Versión	1.0
Vista	Funcional		Modelo	Componentes			

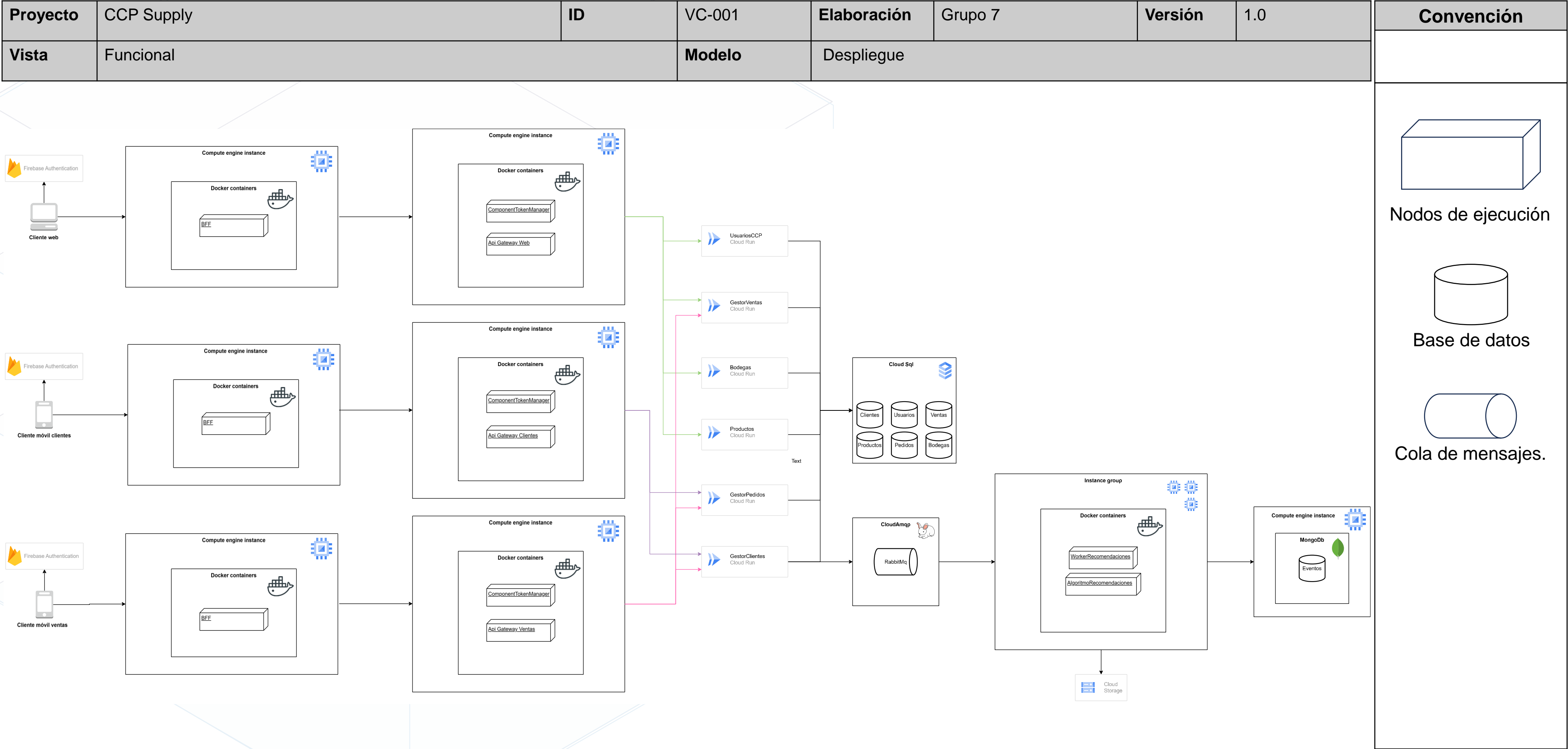
Patrones y tácticas de arquitectura utilizadas

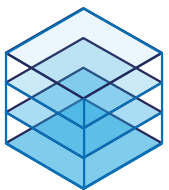
- Microservicios
- Backend For Frontend (BFF)
- Autenticación
- Autorización
- Colas de mensajería

Razonamiento sobre las principales decisiones de arquitectura tomadas en este modelo

- Por medio del estilo de microservicios se espera incrementar la disponibilidad del sistema y tener lógicas de negocio más focalizadas en los componentes asociados, mejorando la facilidad de modificación
- El patrón BFF se escogió en contraposición al patrón de API Gateway para reducir el tamaño de los diferentes aplicativos a realizar. En este caso particular, reducirá la dependencia de un único componente, aumentará la disponibilidad y permitirá redistribuir adecuadamente las cargas
- Se introdujo un servicio externo de autenticación. Este permitirá mantener, por medio del estándar OAuth2, mantener la identidad de los usuarios, tener trazabilidad de sus actividades y delegar roles y permisos. Así mismo, este servicio reducirá la carga de código y es más seguro que un servicio in-house
- Se utilizará un servicio cloud para el almacenamiento y uso de secrets, el cual permitirá autorizar a los componentes para realizar sus debidas operaciones
- Para el manejo del procesamiento de videos para la generación de recomendaciones se implementará una cola de mensajería que permitirá activar un worker que realizará el proceso de forma asíncrona. De esta forma se reduce el tiempo de respuesta y al implementar la lógica “exactly once”, se garantizará el procesamiento de todos los videos

Vista de despliegue





Vista de despliegue

Proyecto	CCP Supply	ID	VC-001	Elaboración	Grupo 7	Versión	1.0
Vista	Funcional		Modelo	Despliegue			

Patrones y tácticas de arquitectura utilizadas

- Microservicios.
- Cliente-servidor

Razonamiento sobre las principales decisiones de arquitectura tomadas en este modelo

- El despliegue de la solución se hará principalmente utilizando la nube y los servicios de GCP.
- Para el proceso de autenticación de cada área del proyecto se utilizarán los servicios de Firebase authenticator, esto nos ayudara principalmente a acelerar el proceso de desarrollo y gestionar una misma herramienta para la parte web y móvil.
- Los diferentes servicios serán desplegados en un compute engine y cada uno en su propio container. Con esto en mente, se debe tener en cuenta la configuración de la VPC para la interconexión de todos los servicios (compute engines, bases de datos, sistema de encolamiento, entre otros.).
- Debido a la complejidad del sistema de recomendaciones para los clientes (basándonos en videos tomados por los vendedores), se utilizará una base de datos No relacional con el fin de tener mayor flexibilidad en el proceso de desarrollo del modelo y los objetos a persistir en una base de datos.

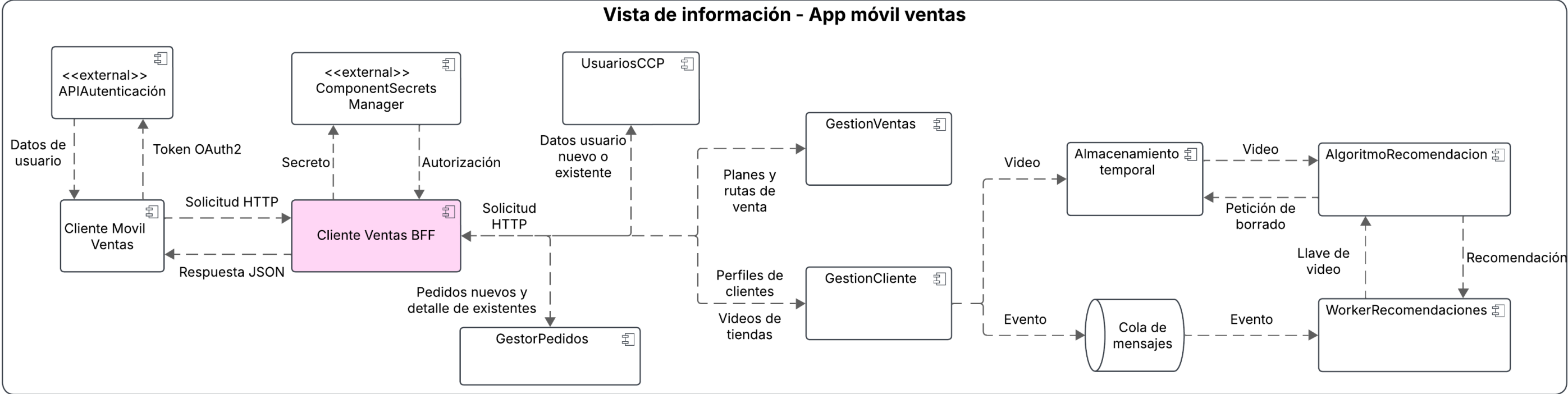
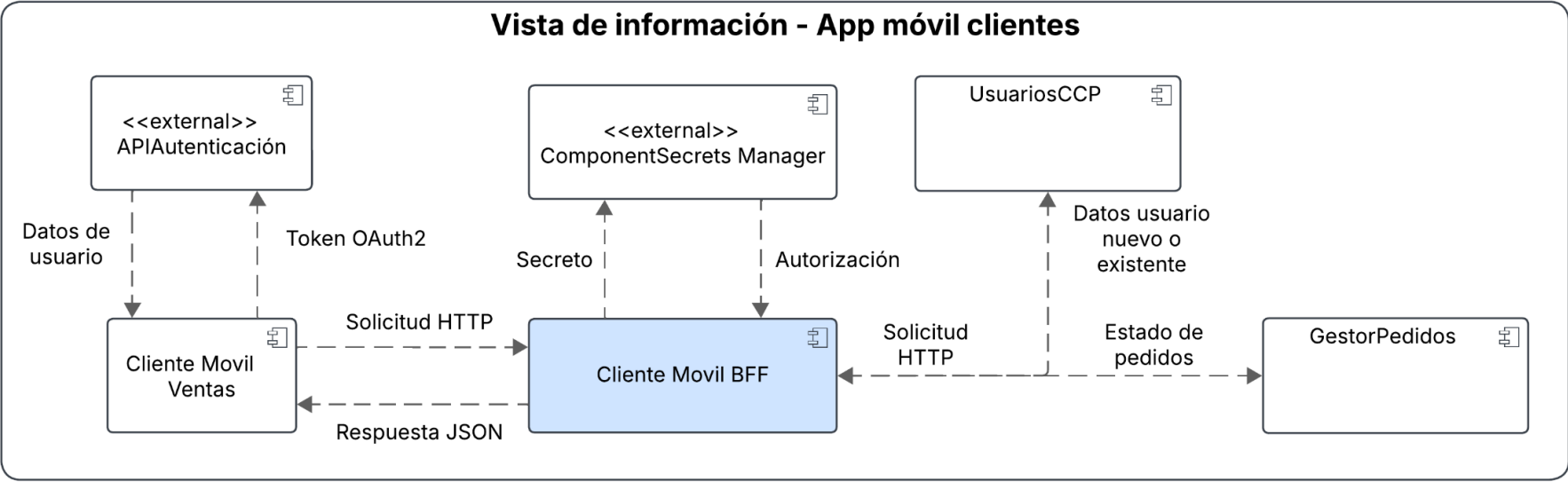
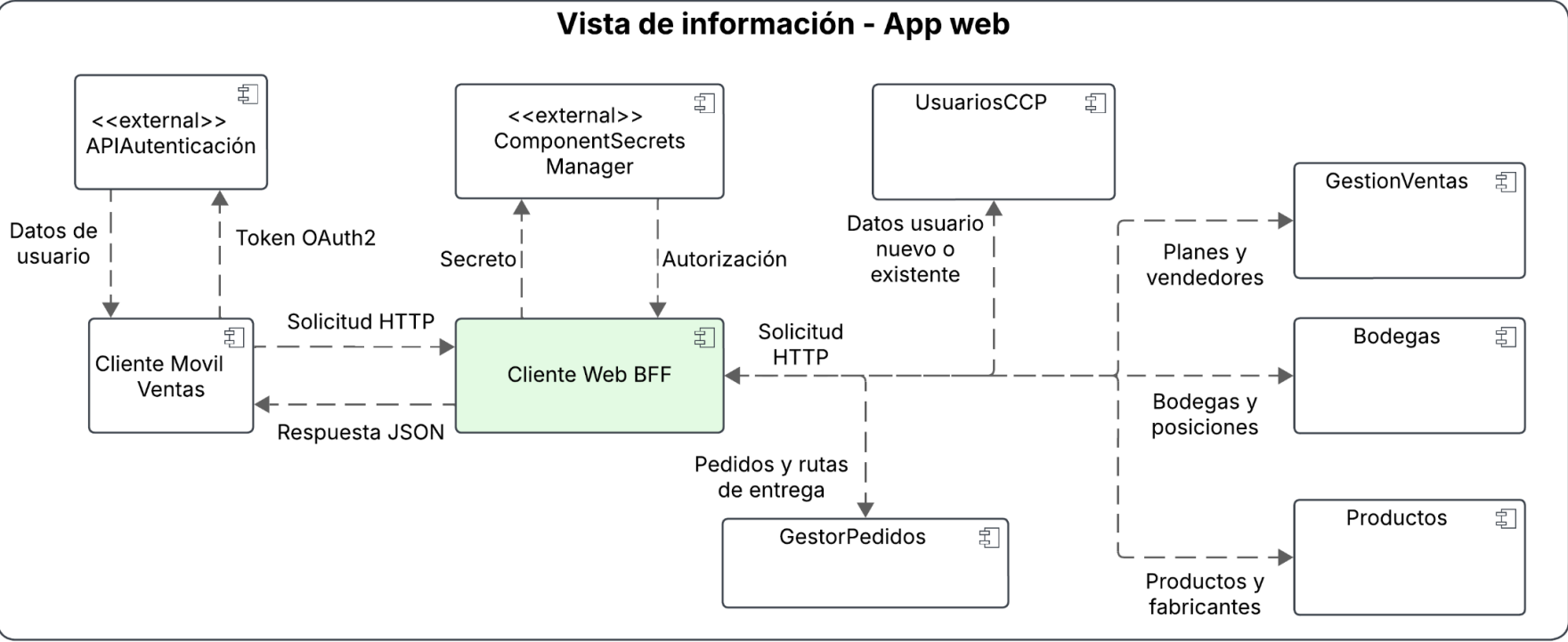
Vista de Información

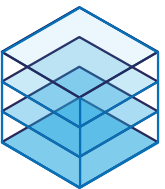


Proyecto	CCP Supply	ID	VC-001	Elaboración	Grupo 7	Versión	1.0
Vista	Información		Modelo	Información			

Convención

Componente





Vista de Información

Proyecto	SportApp	ID	VC-004	Elaboración		Versión	1.1
Vista	Información		Modelo	Información			

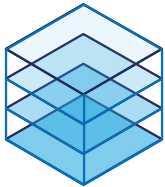
Patrones y tácticas de arquitectura utilizadas

- Backend For Frontend (BFF)
- Colas de eventos


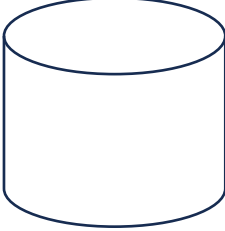
Razonamiento sobre las principales decisiones de arquitectura tomadas en este modelo

- Para el manejo del procesamiento de videos para la generación de recomendaciones se implementará una cola de mensajería que permitirá activar un worker que realizará el proceso de forma asíncrona. Esto, en conjunto con una lógica exactly once, muestra que el sistema será resiliente a fallos ya que podrá procesar más adelante los videos cargados cuándo el sistema esté disponible, y tener un espacio eficiente al eliminarlos del almacenamiento temporal enlazado a él
- El patrón BFF muestra en este caso como el sistema distribuido se parte en partes más pequeñas. Por tal razón, cada uno deberá manejar menos tipos de datos, aumentando significativamente la facilidad de modificación

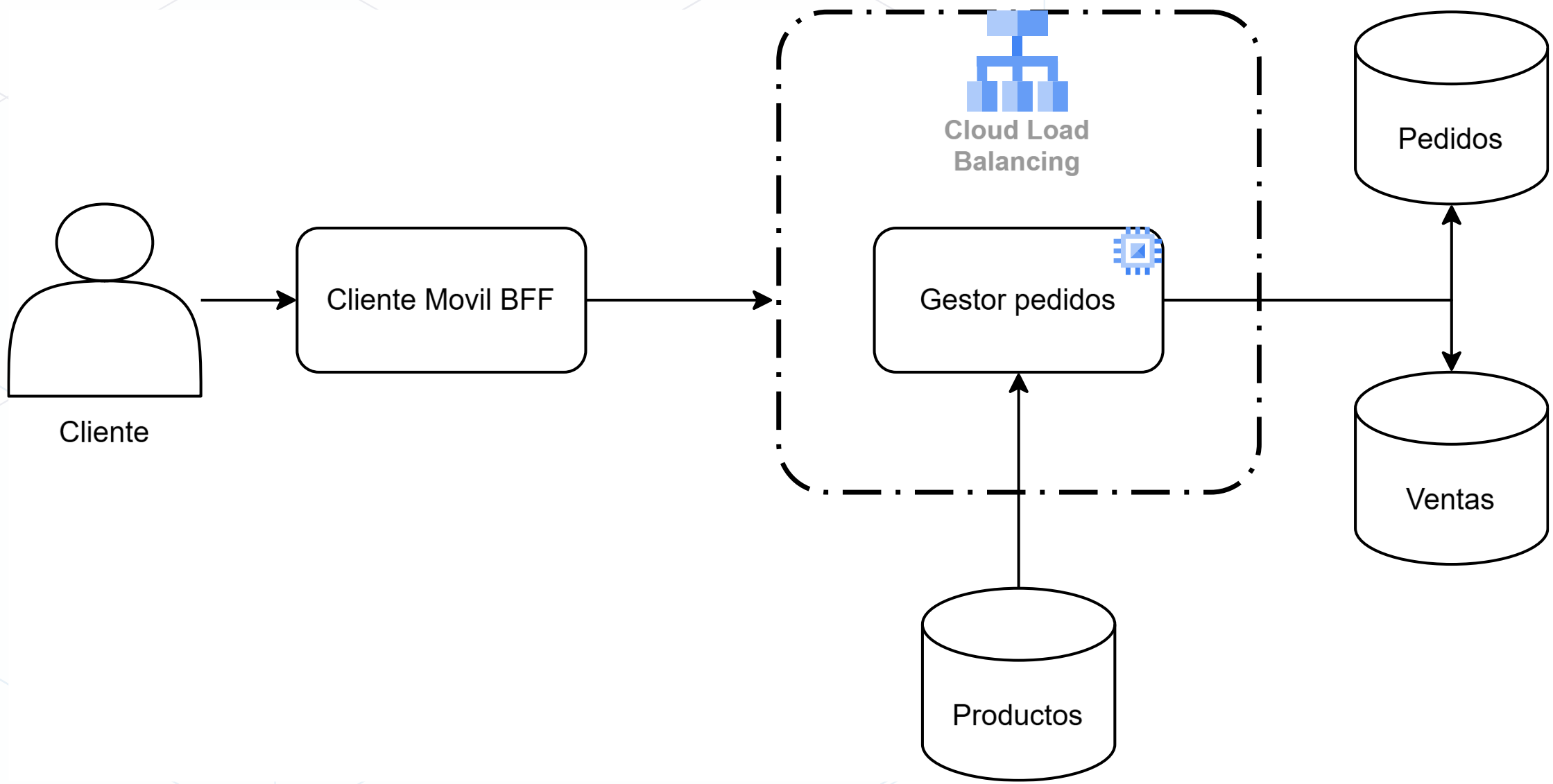
Vista de Información

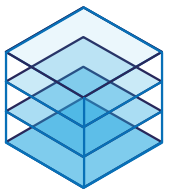


Proyecto	CCP Supply	ID	VC-001	Elaboración	Grupo 7	Versión	1.0
Vista	Concurrencia		Modelo	Concurrencia			

Convención
<div></div> <div>Componente</div>
<div></div> <div>Base de datos</div>

Vista de concurrencia de la operación de escalamiento para la creación de pedidos





Vista de Información

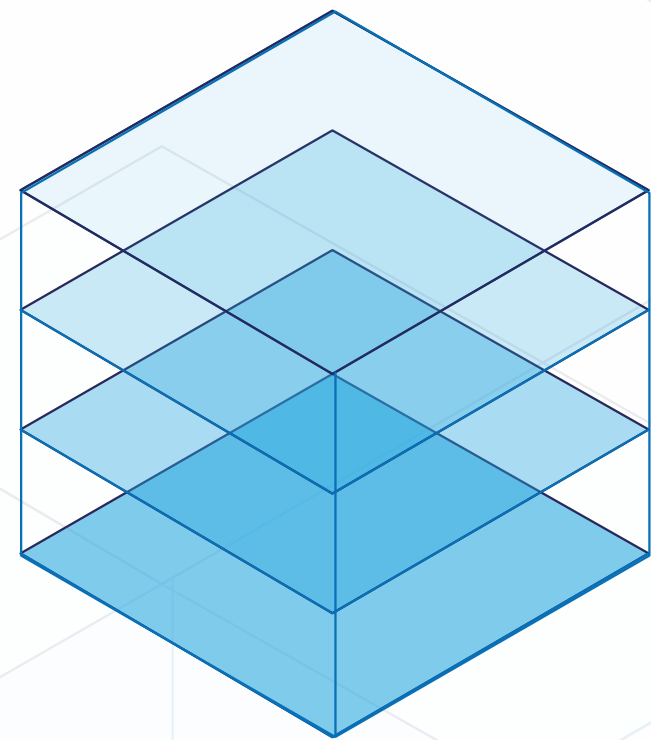
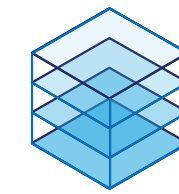
Proyecto	SportApp	ID	VC-004	Elaboración		Versión	1.1
Vista	Concurrencia		Modelo				

Patrones y tácticas de arquitectura utilizadas

- Backend For Frontend (BFF)
- Load balancer

Razonamiento sobre las principales decisiones de arquitectura tomadas en este modelo

- Como uno de los requisitos de calidad principales en nuestra arquitectura es la escalabilidad, y dado que vamos a hacer un experimento relacionado a ello. Desarrollamos a un alto nivel el segmento del diagrama completo de arquitectura para el componente de realización de pedidos. Esto con el fin de adelantar los experimentos de arquitectura, documentar la hipótesis y probarla para este ciclo.
- El principal componente que debe escalar y al cual se evaluara su escalamiento es el Gestor de pedidos.
- Se busca probar el patrón BFF en un ambiente de alta concurrencia, evaluar cómo se comporta el balanceador de cargar y el BFF y evaluar los límites del BFF en este tipo de condiciones.
- El backend de la aplicación independientemente de ser móvil o web estará alojada en la nube. En este caso las peticiones serán hechas directamente a la nube.

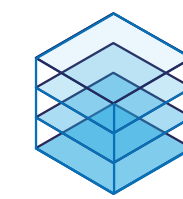


MISO

Maestría en Ingeniería de Software

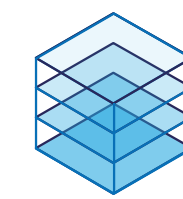
Diseño de experimento #1

Titulo del experimento	#1 – Efectos de las tácticas de seguridad en los requisitos de latencia de la plataforma
Propósito del experimento	Determinar cómo afecta la latencia la implementación de las tácticas de seguridad propuestas durante la creación de fabricantes en caso de alta demanda.
Resultados esperados	La creación de un fabricante ocurre: <ul style="list-style-type: none"> - Escenario sin táctica de seguridad en tiempo menor a 2 segundos. - Escenario con táctica de seguridad en tiempo mayor a 3 segundos. - Escenario con táctica de seguridad y manejo de latencia en tiempo menor a 3 segundos.
Recursos requeridos	Python, Javascript, Flask, 1 instancia de GCP App Service, Locust, 1 app en Firebase
Elementos de arquitectura involucrados	ASR #EC001 Componentes: Productos Vistas: Funcional, información y concurrencia Puntos de sensibilidad: <ul style="list-style-type: none"> • Las capas de seguridad agregadas afectan de forma importante la latencia del sistema. • Combinar tácticas de seguridad y de latencia permiten llevar a cabo la creación en menos de 3 segundos.
Esfuerzo estimado	Se estiman 22 horas hombre para la configuración de la librería, el ambiente de pruebas y la ejecución de las pruebas



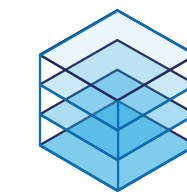
Hipótesis de diseño

Punto de sensibilidad	Las capas de seguridad agregadas afectan de forma importante la latencia del sistema.
Historia de arquitectura asociada	#EC001
Nivel de incertidumbre	85%

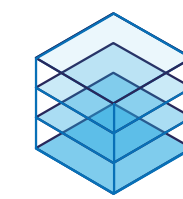


Hipótesis de diseño

Punto de sensibilidad	Combinar tácticas de seguridad y de latencia permiten llevar a cabo la creación en menos de 3 segundos
Historia de arquitectura asociada	#EC001
Nivel de incertidumbre	80%



Estilos de Arquitectura asociados al experimento	Análisis (Atributos de calidad que favorece y desfavorece)
Microservicios	Favorece: Disponibilidad, Escalabilidad, Facilidad de modificación, Seguridad Desfavorece: Latencia, Simplicidad, Costo, Facilidad para realizar pruebas



Tácticas de Arquitectura asociadas al experimento	Descripción
Autenticación de actores	Se busca utilizar un servicio externo que funcione como componente del sistema para validar la identidad de un usuario y darle acceso a los servicios y acciones de su rol.
Autorización	Se utiliza un servicio externo que se encarga de gestionar los secretos que identifican a cada componente y autoriza continuar con la operación en caso de que el secreto almacenado y el recibido sean iguales
Manejo de copias de datos	Almacenamiento de datos en el caché del cliente para disminuir el número de autenticaciones y mejorar el tiempo de respuesta

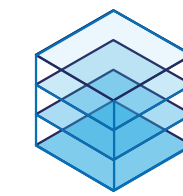


Listado de componentes (Microservicios) involucrados en el experimento

Microservicio	Propósito y comportamiento esperado	Tecnología Asociada
Productos	Contiene los métodos y la información asociadas a los productos y fabricantes	Python / Flask
Cliente web	Tiene la responsabilidad de autenticar los usuarios y permitir los accesos a los servicios según su rol.	Firebase
API Gateway	Se encarga de los enrutamientos entre el cliente y el microservicio de producto	Python / Flask

Listado de conectores involucrados en el experimento

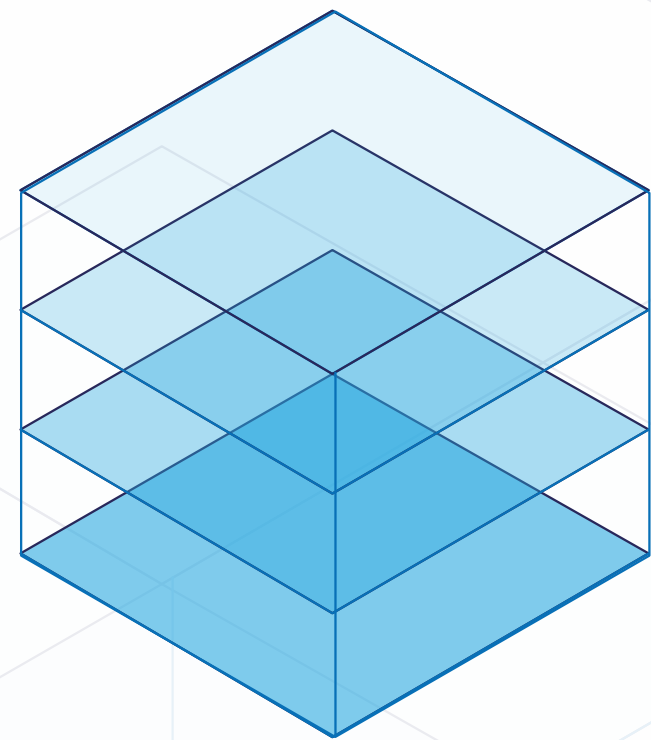
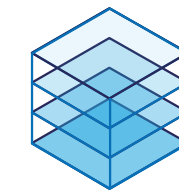
Conector	Comportamiento deseado en el experimento	Tecnología Asociada
crear_fabricante	Este conector se encuentra en el microservicio productos y se encarga de la creación de un fabricante a partir de sus datos de negocio	Python/Flask
autenticar	Este conector es llamado desde el cliente al servicio de autenticación externo y es el responsable de generar el token que autoriza la acción al usuario	Javascript/Firebase SDK



Tecnología asociada con el experimento (Desarrollo, infraestructura, almacenamiento)	Justificación
Lenguajes de programación	Python
Plataforma de despliegue	GCP
Bases de datos	PostgreSQL
Herramientas de análisis	Locust
Librerías	Firebase, SQLAlchemy, dotenv
Frameworks de desarrollo	Flask

Distribución de actividades por integrante

Integrante	Tareas a realizar	Esfuerzo Estimado
Diego Naranjo	Creación de los microservicios	7 horas
Simón Buriticá	Despliegue en GCP	5 horas
Jhonn Calderón	Configuración de Firebase	4 horas
Juan Pablo Rodríguez	Ejecución de escenarios y análisis de resultados	6 horas



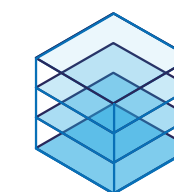
MISO

Maestría en Ingeniería de Software

Diseño de experimento #2

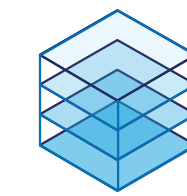
Titulo del experimento	#2 - Tiempo de escalamiento y estabilización del sistema en una situación de alta demanda de solicitud de pedidos
Propósito del experimento	Identificar, bajo los patrones de seguridad utilizados, si es posible conseguir un tiempo de escalamiento inferior a los 3 minutos requeridos, cuando el sistema supera un límite de solicitudes concurrentes.
Resultados esperados	Durante el escalamiento: <ul style="list-style-type: none"> • El tiempo completo de replicación del componente de pedidos es menor a 3 minutos • Durante la puesta a punto, no se pierden solicitudes de creación de pedido
Recursos requeridos	Python, Javascript, Flask, 1 instancia de GCP App Service, Locust, 1 app en Firebase, servicio de base de datos
Elementos de arquitectura involucrados	ASR #EC003 Componentes: Pedidos Vistas: Funcional, información y concurrencia Puntos de sensibilidad: <ul style="list-style-type: none"> • Las tácticas de seguridad reducen la capacidad del sistema para procesar solicitudes significativamente en una situación de alta demanda • Un límite máximo de 80% de uso de recursos (CPU y/o RAM) es adecuado para replicar el componente antes de perder solicitudes
Esfuerzo estimado	Se estiman 15 horas hombre para la configuración de la librería, el ambiente de pruebas y la ejecución de las pruebas

Hipótesis de diseño	
Punto de sensibilidad	Las tácticas de seguridad reducen la capacidad del sistema para procesar solicitudes significativamente en una situación de alta demanda
Historia de arquitectura asociada	#EC003
Nivel de incertidumbre	57%



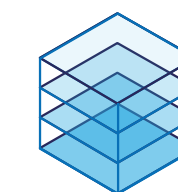
Hipótesis de diseño

Punto de sensibilidad	Un límite máximo de 80% de uso de recursos (CPU y/o RAM) es adecuado para replicar el componente antes de perder solicitudes
Historia de arquitectura asociada	#EC003
Nivel de incertidumbre	80%



Estilos de Arquitectura asociados al experimento	Análisis (Atributos de calidad que favorece y desfavorece)
Microservicios	Favorece: Disponibilidad, Escalabilidad, Facilidad de modificación, Seguridad Desfavorece: Latencia, Simplicidad, Costo, Facilidad para realizar pruebas

Tácticas de Arquitectura asociadas al experimento	Descripción
Autenticación de actores	Se busca utilizar un servicio externo que funcione como componente del sistema para validar la identidad de un usuario y darle acceso a los servicios y acciones de su rol.
Autorización	Se utiliza un servicio externo que se encarga de gestionar los secretos que identifican a cada componente y autoriza continuar con la operación en caso de que el secreto almacenado y el recibido sean iguales
Replicación	El sistema externo se debe configurar con políticas de escalamiento para atender los requisitos de concurrencia del sistema.
Manejo de copias de datos	Almacenamiento de datos en el caché del cliente para disminuir el número de autenticaciones y mejorar el tiempo de respuesta



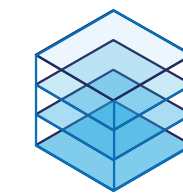
Listado de componentes (Microservicios) involucrados en el experimento

Microservicio	Propósito y comportamiento esperado	Tecnología Asociada
Pedidos	Responsable de gestionar la creación y consulta de pedidos de clientes y vendedores	Python / Flask
Cliente web	Tiene la responsabilidad de autenticar los usuarios y permitir los accesos a los servicios según su rol.	Firebase
API Gateway	Se encarga de los enrutamientos entre el cliente y el microservicio de producto	Python / Flask



Listado de conectores involucrados en el experimento

Conector	Comportamiento deseado en el experimento	Tecnología Asociada
solicitar_pedido	Este conector se encuentra en el microservicio pedidos y se encarga de la creación de una solicitud de pedido	Python/Flask
autenticar	Este conector es llamado desde el cliente al servicio de autenticación externo y es el responsable de generar el token que autoriza la acción al usuario	Javascript/Firebase SDK



Tecnología asociada con el experimento (Desarrollo, infraestructura, almacenamiento)	Justificación
Lenguajes de programación	Python
Plataforma de despliegue	GCP
Bases de datos	PostgreSQL
Herramientas de análisis	Locust
Librerías	Firebase, SQLAlchemy, dotenv
Frameworks de desarrollo	Flask

Distribución de actividades por integrante

Integrante	Tareas a realizar	Esfuerzo Estimado
Jhonn Calderón	Creación de los microservicios	3 horas
Juan Pablo Rodríguez	Configuración del entorno de replicación	5 horas
Simón Buriticá	Ejecución de escenarios de prueba	4 horas
Diego Naranjo	Análisis de resultados y ajustes de arquitectura	3 horas