

MODELO DE ARQUITECTURA PARA FRONT-END (BASADO EN MICRO-FRONTENDS) APlicado al PRODUCTO DIGITAL DE FUERZA ESPECIALIZADA DE VIVIENDA DEL BANCO DE BOGOTÁ



**FELIPE NARIÑO 20192099036
JORGE ATUESTA 20192099025
GRUPO 2**

**PROYECTO DE GRADO PARA OPTAR POR EL TÍTULO DE ESPECIALISTA
EN INGENIERÍA DE SOFTWARE**

**UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS
FACULTAD DE INGENIERÍA
BOGOTÁ
2019**

MODELO DE ARQUITECTURA PARA FRONT-END (BASADO EN MICRO-FRONTENDS) APlicado al PRODUCTO DIGITAL DE FUERZA ESPECIALIZADA DE VIVIENDA DEL BANCO DE BOGOTÁ

**FELIPE NARIÑO 20192099036
JORGE ATUESTA 20192099025
GRUPO 2**

PROYECTO DE GRADO PARA OPTAR POR EL TÍTULO DE ESPECIALISTA EN INGENIERÍA DE SOFTWARE

**DIRECTOR: ALEJANDRO PAOLO DAZA
REVISOR: JHON FREDY PARRA**

**UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS
FACULTAD DE INGENIERÍA
BOGOTÁ
2019**

TABLA DE CONTENIDO

INTRODUCCIÓN	6
PARTE I. CONTEXTUALIZACIÓN DE LA INVESTIGACIÓN	7
CAPÍTULO 1. DESCRIPCIÓN DE LA INVESTIGACIÓN	8
1.1. PLANTEAMIENTO/IDENTIFICACIÓN DEL PROBLEMA	8
1.1.2 FORMULACIÓN DEL PROBLEMA	9
1.1.3 SISTEMATIZACIÓN DEL PROBLEMA	9
1.2. OBJETIVOS DE LA INVESTIGACIÓN	10
1.2.1. OBJETIVO GENERAL	10
1.2.2. OBJETIVOS ESPECÍFICOS	10
1.3. JUSTIFICACIÓN DE LA INVESTIGACIÓN	11
1.4. HIPÓTESIS	13
1.5. MARCO REFERENCIAL	13
1.5.1 MARCO TEÓRICO	14
1.5.2 MARCO ESPACIAL	18
1.5.3 MARCO CONCEPTUAL	19
1.6. METODOLOGÍA DE LA INVESTIGACIÓN	21
1.6.1 Tipo de estudio	21
1.6.2 Método de investigación	21
1.6.3 Fuentes y técnicas de recolección de la información	21
1.6.4 Tratamiento de la información.	21
1.7. ORGANIZACIÓN DEL TRABAJO DE GRADO.	21
1.8. ESTUDIO DE SISTEMAS PREVIOS.	22
PARTE II. DESARROLLO DE LA INVESTIGACIÓN	25
CAPÍTULO 2. EVALUACIÓN DEL ESTADO ACTUAL DE LOS PROYECTOS FRONT-END (ARQUITECTURAS, TECNOLOGÍAS, LENGUAJES DE PROGRAMACIÓN) DE LOS PRODUCTOS DIGITALES DEL BANCO DE BOGOTÁ	26
CAPÍTULO 3. ESTUDIO Y ANÁLISIS DE LA CREACIÓN Y COMUNICACIÓN ENTRE COMPONENTES FRONT-END DESARROLLADOS EN ANGULAR	32
3.1. Creación de la biblioteca	32
3.2. Creación de servicio	34
3.3. Creacion de componente	34
3.4. Compilación de la biblioteca	36
3.5. Uso	37
3.6. Publicación	39
CAPÍTULO 4. ARQUITECTURA	40
4.1. Capa de negocio.	42

4.1.1. Punto de Vista de Organización.	43
4.1.2. Punto de Vista de Cooperación de Actor.	44
4.1.3. Punto de Vista de Función de Negocio.	45
4.1.4. Punto de Vista de Proceso de Negocio.	46
4.1.4. Punto de Vista de Cooperación de Proceso de Negocio.	46
4.1.5. Punto de vista de Producto.	47
4.2. Capa de Aplicación.	48
4.2.1. Punto de Vista de Comportamiento de Aplicación.	49
4.2.2. Punto de Vista de Cooperación de Aplicación.	50
4.2.3. Punto de Vista de Estructura de Aplicación.	51
4.2.4. Punto de Vista de Uso de Aplicación.	52
4.3. Capa de Tecnología.	53
4.3.1. Punto de Vista de Tecnología.	53
4.3.2. Punto de Vista de Uso de Tecnología.	54
4.3.3. Punto de Vista de Organización e Implementación.	55
4.3.4. Punto de Vista de Estructura de Información.	56
4.3.5. Punto de Vista de Realización de Servicio.	56
4.3.6. Punto de Vista de Capas.	58
4.4. Capa de Motivación.	59
4.4.1. Punto de Vista de Stakeholder.	59
4.4.2. Punto de Vista de Realización de Objetivos	60
4.4.3. Punto de Vista de Contribución de Objetivos	61
4.4.4. Punto de Vista de Principios	62
4.4.5. Punto de Vista de Realización de Requerimientos.	63
4.4.6. Punto de Vista de Motivación	64
4.4. Diagramas de componentes arquitectura Micro Front-end.	65
CAPÍTULO 5. APPLICACIÓN DE LA ARQUITECTURA LOGRADA AL PROYECTO FRONT DE LA FUERZA ESPECIALIZADA DE VIVIENDA DEL BANCO DE BOGOTÁ	69
5.1. Descripción de la biblioteca	69
5.2. Descripción de los componentes internos	70
5.1. Despliegue continuo	76
5.1. Uso	78
PARTE III. CIERRE DE LA INVESTIGACIÓN	79
CAPÍTULO 6. RESULTADOS Y DISCUSIÓN	80
CAPÍTULO 7. CONCLUSIONES	91
7.1 Verificación, contraste y evaluación de los objetivos	91
7.2 Síntesis del modelo propuesto	¡Error! Marcador no definido.
7.3 Aportes originales	92
7.4 Trabajos o publicaciones derivadas	92
CAPÍTULO 8. PROSPECTIVA DEL TRABAJO DE GRADO	92
8.1 Líneas de investigación futuras	92
8.2 Trabajos de investigación futuros	93

REFERENCIAS 94

ANEXOS 95

INTRODUCCIÓN

Uno de los objetivos de las empresas que elaboran productos de software es que estos puedan llegar a ser mantenibles, escalables y reusables para cada una de las áreas que componen la organización o para los diferentes productos que ella elabora, reduciendo tiempos de desarrollo y costos.

Un objetivo primordial para el Banco de Bogotá, específicamente de su área digital, radica en poder utilizar las tecnologías de vanguardia en el desarrollo de productos digitales y poder así llevar la banca a muchas más personas.

En este proyecto se encuentra un análisis de cada uno de las dificultades técnicas y prácticas que se tienen actualmente en los productos digitales del Banco de Bogotá, además del planteamiento de la solución mediante el uso de tecnologías basadas en microservicios para la parte front-end de los proyectos. También se busca ilustrar al lector en temas concernientes a micro front-end: su definición, características, ventajas, implementación y su aplicabilidad en un producto digital.

PARTE I. CONTEXTUALIZACIÓN DE LA INVESTIGACIÓN

CAPÍTULO 1. DESCRIPCIÓN DE LA INVESTIGACIÓN

1.1. PLANTEAMIENTO/IDENTIFICACIÓN DEL PROBLEMA

En la actualidad, en el laboratorio digital del Banco de Bogotá, se desarrollan soluciones digitales que buscan revolucionar la relación entre las personas y la banca, para esto el Banco dispone de distintas aplicaciones web, entre las cuales se encuentran soluciones para: apertura de cuenta de ahorros, solicitud de tarjeta de crédito, crédito de libre inversión, libranza y crédito de vivienda. Las aplicaciones con las que se cuenta actualmente son desarrolladas bajo una arquitectura de referencia (transversal a todos los productos) enfocada principalmente hacia el back-end, dentro de la arquitectura de referencia se especifica REST como estilo arquitectónico además de la implementación de microservicios basados en contenedores.

La arquitectura actual, se enfoca en la búsqueda del cumplimiento de atributos de calidad (mediante implementación de buenas prácticas, patrones de arquitectura y desarrollo), pero solo del lado de back-end, olvidando por completo la mantenibilidad e integración continua del componente Front-end, esto debido a que no se segregan funcionalidades y responsabilidades a múltiples componentes (principio de responsabilidad única), en lugar de esto se trabaja monolíticamente con un único componente que se encarga de presentar al usuario todas las pantallas y herramientas visuales que tiene a su disposición. Con el tiempo este único componente crece y se nutre de nuevas funcionalidades que, en cuanto al back-end dan lugar a nuevos microservicios que trabajan sinérgicamente y que a su vez cuentan con componentes que orquestan el funcionamiento organizado y sincronizado, pero que en cuanto a front-end, hacen crecer exponencialmente el único componente (monolito), dificultando la mantenibilidad del sistema, afectando tiempos en cuanto a la integración continua (pruebas y despliegue). Tener una única aplicación haciendo todas las labores front-end trae graves implicaciones en el futuro, debido a que el código crece conforme a la evolución del software. Esto complica la labor de quienes mantienen los productos, ya que, por el volumen de código, se ocupa mayor tiempo en la detección y futura corrección de errores.

Para superar la situación actual, se propone la implementación de micro front-ends de las aplicaciones, donde sus componentes puedan ser modificados sin alterar las funcionalidades con las que convive, donde sea posible realizar pequeños y rápidos despliegues, además de contar con única responsabilidad de manera que pequeñas unidades funcionales trabajen coordinadamente para lograr formar sistemas

complejos de fácil mantenimiento y corrección temprana de errores. Estas pequeñas fracciones deben poder ser desarrolladas por diferentes equipos o grupos de personas y deben ser lo suficientemente adaptables para su reutilización en distintos productos diferentes para el que fue creado inicialmente.

1.1.2 FORMULACIÓN DEL PROBLEMA

¿Cómo evitar que con el paso del tiempo, una aplicación front-end pueda degradarse a tal punto que pierda atributos de calidad y por ende afecte la integración continua de un producto del Banco de Bogotá?

1.1.3 SISTEMATIZACIÓN DEL PROBLEMA

Luego de el planteamiento anterior, nacen cuestionamientos como:

¿Cómo medir el impacto del modelo a crear en métricas de calidad?

¿Cómo afecta la segregación de nuevos componentes a la velocidad de un equipo de desarrollo y por ende a la integración continua?

¿Cómo se podrían comunicar componentes hechos en diferentes lenguajes dentro de una misma aplicación?

1.2. OBJETIVOS DE LA INVESTIGACIÓN

1.2.1. OBJETIVO GENERAL

Proponer un modelo arquitectural para las aplicaciones front-end el cual cumpla con atributos de calidad expuestos en las normas ISO-25010 e ISO-9126, mediante la aplicación de tecnologías basadas en micro front-end, con el fin de evitar el deterioro de estas dentro de los productos digitales del Banco de Bogotá.

1.2.2. OBJETIVOS ESPECÍFICOS

- Reducir la complejidad de los proyectos front-end de los productos digitales del Banco mediante la aplicación del modelo arquitectural propuesto, para contribuir al cumplimiento de atributos de calidad.
- Realizar una prueba del concepto arquitectural en una fracción de un producto actual del Banco de Bogotá, aplicando modelo micro front-end planteado con el fin de comparar su resultado de evaluación de métricas de calidad contra la aplicación ya existente.
- Aumentar la mantenibilidad de los proyectos front-end, dividiendo su arquitectura en pequeñas unidades funcionales asignadas a un equipo de trabajo específico, para poder resolver rápidamente los diferentes bugs que lleguen a presentarse.

1.3. JUSTIFICACIÓN DE LA INVESTIGACIÓN

Los productos digitales del Banco de Bogotá, en especial los que se desarrollan en el área digital presentan algunos inconvenientes en su front-end, es decir, en la parte que va de cara al usuario.

personal-loan-front			
Showing 1–30			
SUCCEEDED	CC_51_INSURANCE_CONDITIONS_PDF / deploypersonal-loan-front	8 min ago	07:42 4f14756
Rerun			
CANCELED	release-DIAN / deploypersonal-loan-front	55 min ago	03:11 c6c89a3
Rerun	Merge pull request #1302 from bancodebogota/develop-dian		
SUCCEEDED	develop-dian / deploypersonal-loan-front	1 hr ago	06:01 #1302 8010547
Rerun	Merge pull request #1301 from bancodebogota/CC_DIAN_83_desktop_disbursement_screen		
SUCCEEDED	CC_DIAN_83_desktop_disbursement_screen / deploypersonal-loan-front	1 hr ago	07:12 #1301 0a0d87e
Rerun	Adjust DIAN info styles in disbursement screen		
SUCCEEDED	develop-dian / deploypersonal-loan-front	1 hr ago	06:03 #1300 c56f67e
Rerun	Merge pull request #1300 from bancodebogota/CC_DIAN_83_Adapt_approval_screen		
SUCCEEDED	CC_DIAN_83_Adapt_approval_screen / deploypersonal-loan-front	2 hrs ago	07:43 #1300 b2179bc
Rerun	Adapt approval screen - first commit		
SUCCEEDED	develop-dian / deploypersonal-loan-front	5 hrs ago	05:47 #1300
Rerun			

Tiempos de despliegue de la aplicación front del producto de libre destino del Banco de Bogotá
Fuente: Banco de Bogotá

bdb-insurance-mngr			
Showing 1–30			
SUCCEEDED	develop / building_and_deployment	7 min ago	03:25 #53 4c611bf
Rerun	Merge pull request #52 from bancodebogota/CC_51_DOWNLOAD_PDF		
SUCCEEDED	CC_51_DOWNLOAD_PDF / building_and_deployment	1 hr ago	03:47 #52 1d5f8a4
Rerun	CC_51_DOWNLOAD_PDF: Se agrega el recurso para poder descargar las condiciones del seguro de cuota protegida según la oc...		
SUCCEEDED	develop / building_and_deployment	4 days ago	03:35 #51 227923f
Rerun	Merge pull request #51 from bancodebogota/CC_51_DOWNLOAD_PDF		
SUCCEEDED	CC_51_DOWNLOAD_PDF / building_and_deployment	4 days ago	04:53 #51 5e802c9
Rerun	CC_51_DOWNLOAD_PDF: Se agrega el recurso para poder descargar las condiciones del seguro de cuota protegida según la oc...		
SUCCEEDED	CC_43_FEE_PROTECTED_INSURANCE / building_and_deployment	1 month ago	03:21 #3365643
Rerun	container definition		
SUCCEEDED	CC_43_FEE_PROTECTED_INSURANCE / building_and_deployment	1 month ago	03:49 #3365643 5f062e4
Rerun	container definition		
SUCCEEDED	develop / building_and_deployment	1 month ago	03:28
Rerun			

Tiempos de despliegue de la aplicación back del producto de seguros libre destino del Banco de Bogotá
Fuente: Banco de Bogotá

Como se puede apreciar en las figuras expuestas, los tiempo de despliegue para la parte front de un producto digital oscilan entre 6 y 7 minutos, mientras que para una aplicación back oscilan entre 3 y 4 minutos.

De acuerdo con la revisión expuesta anteriormente se hace necesario la creación de un modelo arquitectural de una manera práctica y metodológica para el front-end de los productos del Banco, basándose en las guías que se evidencian en las ideas de los micro front-ends. Esto se realiza porque se quieren mitigar todos los problemas que se tienen con lo que el usuario final ve de la aplicación, poder escalar y realizar una mejor labor de mantenibilidad.

1.4. HIPÓTESIS

Si se tiene una mayor segregación, despliegues independientes y responsabilidades únicas en componentes a nivel front-end, entonces los productos digitales cuentan con mayor mantenibilidad, mejores tiempos de despliegue y mayor facilidad en la detección y corrección de errores.

1.5. MARCO REFERENCIAL

1.5.1 MARCO TEÓRICO

Antes de entrar a definir qué es micro front-end es necesario realizar la descripción de lo que es un microservicio. Un microservicio se define como un estilo arquitectural que compone una aplicación como una colección de servicio con bajo acoplamiento [10]. Esto es el desarrollo de una aplicación compuesta de pequeñas unidades cada una cumpliendo una función específica y comunicadas entre ellos a través de mecanismos livianos, usualmente recursos HTTP.

Adoptando la filosofía de los microservicios de tener varios componentes pequeños creados por diferentes equipos, nace el enfoque arquitectural de los micro front-ends, que busca atacar el problema del desarrollo separado entre el back y el front que se está presentando en algunas organizaciones y que conlleva a tener monolitos front-end, problemas de mantenimiento y de escalabilidad.

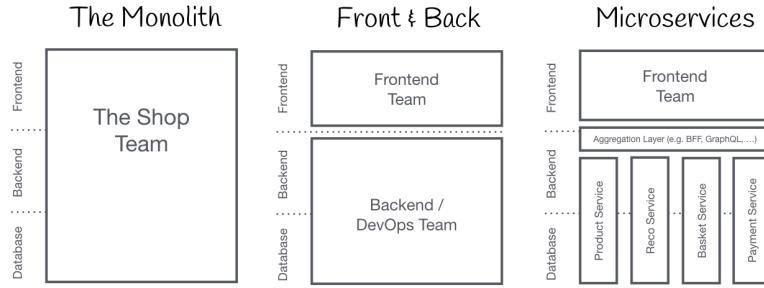
Origen de los micro front-end

La primera aparición que se conoce sobre los micro front-end fue publicada por la consultora de tecnología ThoughtWorks Technology en noviembre de 2016, la cual cuenta con un radar que indica gráficamente la cantidad de empresas que indagan, testean y adoptan las tecnologías de tendencia en el mercado.

La idea detrás de los micro frontends es pensar acerca de un sitio web o una aplicación web con una composición de muchas características, cada una desarrollada por equipos independientes y que son especialistas en un área de negocio determinada.

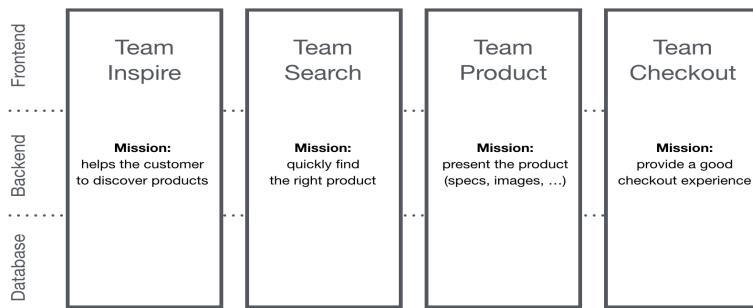
El concepto busca hacer una división vertical de cada una de las funcionalidades que tienen los front-end monolíticos y asignar cada una de ellas a un equipo independiente. [11]

A continuación, se presenta un gráfico el cual ilustra la distribución de “componentes” dependiendo si es un front-end monolítico o basado en micro front-end [18].



Front-end Monolítico Fuente: <https://micro-frontends.org/>

End-to-End Teams with Micro Frontends



Micro front-end Fuente: <https://micro-frontends.org/>

Problemas que resuelven los micro front-ends

Aplicaciones poco mantenibles:

Micro front-end permite a los equipos escalar y mejorar la entrega de servicios mediante la creación, testeo, despliegue y mantenimiento independiente de componentes (o features), es decir, se lucha en contra de la creación de front-end con grandes y complejos monolitos , dando lugar a pequeños componentes que conforman un sistema completo, allí, cada componente es independiente (y reutilizable), más fácil de escalar, de mantener, con un índice de detección y corrección de errores más eficiente[11].

Aplicación

Es positivo aplicar este enfoque en proyectos de mediana y gran escala ya que al hacerlo aporta a la facilidad del escalamiento del software. A su vez es mejor utilizarlos en entornos web, ya que el uso de herramientas basadas en javascript facilita la descomposición vertical de los monolitos que se tengan implementados en las aplicaciones.

También es posible hacer que un equipo de desarrollo pueda hacerse cargo de más de un micro frontend, ya que en ese equipo pueden existir capacidades técnicas para

analizar más de una lógica de negocio. El enfoque arquitectural de micro frontends se puede aplicar en múltiples escenarios, entre los más importantes se pueden destacar:

- Una SPA (Single Page Application o Aplicación de una sola página), la cual está compuesta de múltiples componentes desarrollados con diferentes tecnologías
- Apps pequeñas aisladas dentro de iframes
- Diferentes componentes orquestados dentro de una aplicación principal

Empresas como Spotify, IKEA, Klarna, Zalando, Up Work, and Allegro, and HelloFresh usan actualmente micro frontends en la construcción de sus aplicaciones web.

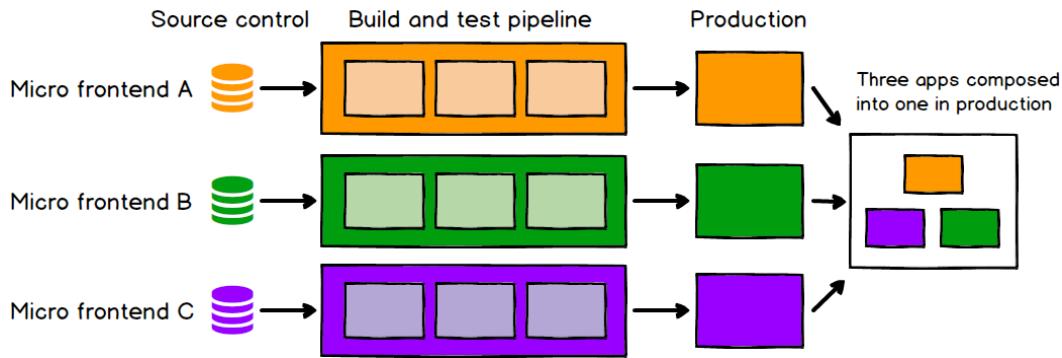
Spotify utiliza micro frontends para su aplicación de escritorio haciendo uso de iFrames para concurrir múltiples componentes en una misma vista (pantalla). Realiza esta comunicación de componentes a través de un bus de eventos que permite comunicar los elementos de la vista entre sí. [12]

Beneficios del uso de micro front-ends

Unos de los constantes temores a los que se enfrentan las empresas que tienen monolitos frontend y que lo han mantenido por muchos años, normalmente por limitaciones en tiempos de entrega, es el de tener que reescribir todo el código.[13]. Pero si se arriesga puede encontrar en la división vertical de los micro frontends una buena alternativa, ya que podrá hacer actualizaciones incrementales de su producto en lugar de mantener una aplicación que en algún momento puede fallar drásticamente.

En estos últimos años han aparecido muchas tecnologías basadas en Javascript y cada una de ellas tiene sus ventajas y desventajas. Cada equipo de desarrollo busca siempre maximizar los beneficios y minimizar los riesgos al utilizar estas tecnologías. Uno de los beneficios que presentan los micro frontends son el uso o la experimentación de estas nuevos frameworks de desarrollo.

Otro de los beneficios a resaltar dentro de los micro frontends y que es heredado de los microservicios en la capacidad de poder ser desplegados en producción de manera independiente, esto contribuye tanto al testeo como a la integración continua que se tenga planteada.



Cada microfrontend puede desplegarse a producción independientemente Fuente: <https://martinfowler.com/articles/micro-frontends.html>

Pero como la gran mayoría de las soluciones que se presentan en ingeniería de software, los micro frontends no son balas de plata y no resolverán todos los problemas. Es importante entender tanto sus beneficios como sus limitaciones. Entre las más importantes se pueden destacar: la curva de aprendizaje que deben de superar los equipos de desarrollo al enfrentarse a nuevas tecnologías y el cambio de paradigma al cual se encuentran ya habituados.

Antes de continuar, hay que detenerse evaluar las tecnologías de vanguardia que se pueden utilizar para el desarrollo de micro front-ends.

Angular:

Está compuesto de una serie de bibliotecas de código abierto que sirven para construir aplicativos web avanzados del lado del cliente. Es ideal para hacer aplicaciones de negocio y aplicaciones de gestión que se despliegan en una única página (SPA). Usa el patrón de diseño habitualmente encontrado en el desarrollo web MVC, aunque en una variante llamada a veces MV* y a veces MVVM. Esto, junto con otras herramientas disponibles en Angular permite un desarrollo ordenado, sencillo de realizar y sobre todo más fácil de mantener en un futuro. Angular está apoyado por Google y cada día más desarrolladores lo adoptan [14].

React:

React es una biblioteca JavaScript de código abierto que se utiliza para la construcción de interfaces gráficas de usuario (GUI), mantenida por Facebook, Instagram y la comunidad [15]. Esta tecnología resalta en el mercado de desarrollo de aplicaciones empresariales ya que es simple, declarativa, fácil de combinar y utiliza el patrón MVC. Además, React permite crear interfaces interactivas de manera simple y actualiza los componentes de las aplicaciones cuando la información cambia, debido al diseño de vistas simples y manejo de eventos en la aplicación.

Vue:

Es un Framework progresivo creado para construir interfaces gráficas de usuario (GUI), diferente a los frameworks monolíticos, Vue está diseñado desde cero para ser incrementalmente adoptable. La biblioteca central está enfocada solo en la capa visual, y es fácil de tomar e integrar con otras biblioteca y proyectos ya existentes. Por otra parte, Vue también es capaz de trabajar con aplicaciones Single-Page cuando se combina con herramientas modernas [16].

Web Components

El concepto de Web Components busca reutilizar y encapsular código cliente, debido a que permite encapsular código HTML, CSS y Javascript, de forma que no pueda ser afectado por el código de la página que lo importa [17]. También permite extender la funcionalidad de elementos de la página ya existentes. Los Web Components ahorran código HTML al desarrollador, ya que repite menos código y permite enviar propiedades (datos de entrada) a los componentes importados para que ofrezcan la funcionalidad y la información deseada.

Después de observar las tecnologías que ofrece el mercado; se busca, mediante el uso de micro front-ends presentar al usuario final una interfaz que ofrezca la funcionalidad requerida, pero que sea compuesta por múltiples componentes que trabajan de manera sinérgica y transparente a la vista:

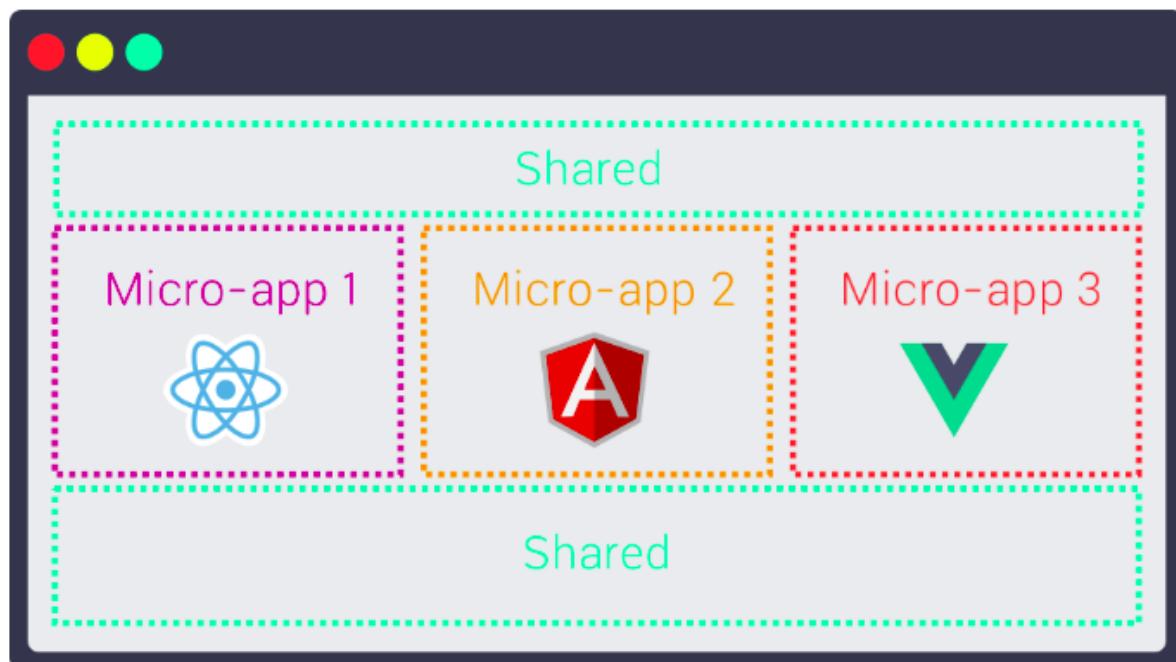


Imagen: Micro front-ends desarrollados en diferentes tecnologías

Fuente:<https://medium.com/@tomsoderlund/micro-frontends-a-microservice-approach-to-front-end-web-development-f325ebdad16>

1.5.2 MARCO ESPACIAL

El presente proyecto se desarrollará en el área Digital del Banco de Bogotá (Colombia). Área encargada de desarrollar productos digitales que buscan revolucionar la relación entre la Banca y las personas mediante soluciones tales como: creación de cuenta de ahorros, tarjeta de crédito, crédito de consumo, libranzas y créditos de Vivienda Digital. Puntualmente, para este caso de estudio se toma el producto digital dirigido a la Fuerza especializada de Vivienda Digital.

1.5.3 MARCO CONCEPTUAL

En la actualidad, el Banco de Bogotá cuenta con productos digitales que han explotado el potencial del personal técnico compuesto por equipos de desarrollo que suelen ser un grupo pequeño de unas 5-9 personas y tienen autoridad para organizar y tomar decisiones para conseguir su objetivo. Está involucrado en la estimación de esfuerzo de las tareas a desarrollar en el producto de software [1] en la arquitectura de software, entendida como la estructura o estructuras de un sistema, la cual está compuesta de elementos, propiedades y la relación entre ellos [2], y desarrollo de aplicaciones mediante el uso de tecnologías de vanguardia.

La mayor parte del esfuerzo se centraliza en la búsqueda e implementación de herramientas y tecnologías orientadas al back-end, el cual se define como: "Aplicaciones implicadas en las actividades que se realizan del lado del servidor; es decir, las tareas de bases de datos y los servidores que el usuario no visualiza directamente desde su interacción con los aplicativos"[3].

Antes de continuar, es necesario saber qué es un patrón de diseño y qué es una métrica de código limpio: un patrón de diseño describe un problema que ocurre una y otra vez en el desarrollo de software, y luego describe el corazón de la solución a dicho problema, de tal manera que esta solución se pueda usar infinitad de veces [4]. Ahora, para comprender el concepto de "Clean Code", es primordial citar al creador del lenguaje de programación Bjarne Stroustrup: "Me gusta que mi código sea elegante y eficiente. La lógica debe ser sencilla de hacer y compleja para ocultar errores, un mínimo de dependencias para facilitar el mantenimiento, completo manejo de errores y un rendimiento cercano al óptimo ... el código limpio hace una cosa bien". Dicho esto, este concepto se vuelve imprescindible para esta investigación ya que lo que se pretende no es que una unidad sea la encargada o la responsable de todas las funciones posibles, sino que, en efecto, el código esté fraccionado por funcionalidades haciendo que cada "parte" tenga una sola responsabilidad y asegurarse que la haga "bien" [5].

Después de algo más de dos años de vida del área digital del banco, el área cuenta con claros lineamientos de desarrollo (patrones de diseño y métricas de código limpio) y arquitecturas (back-end) de referencia (la misma para base para todos los

productos), donde cada producto (o aplicación) cuenta con un estilo arquitectónico REST además de la implementación de microservicios basados en contenedores. Pero ¿qué es REST?, ¿Qué son microservicios? REST (REpresentational State Transfer) se refiere a que un cliente tiene un recurso expuesto y un cliente puede "requerir" una "representación" del estado del recurso [6].

Distintas organizaciones, como Amazon, eBay y Netflix resolvieron sus problemas de grandes aplicaciones monolíticas adoptando lo que se conoce como Arquitectura basada en microservicios; es decir, en lugar de crear una única y grande ("monstruosa" según el autor), se decide la aplicación en pequeños e interconectados servicios [7], permitiendo que las aplicaciones sean independientes tanto en arquitectura, lógica y responsabilidades.

Para el presente proyecto, se propone abordar a fondo la arquitectura orientada al front, proponiendo un modelo arquitectural basado en micro front-ends. Para lograr esto, se debe realizar una investigación (y demos) de los lenguajes y tecnologías de vanguardia, desarrollo web por componentes y web components.

Este diseño y posterior desarrollo, busca que los productos digitales del banco cumplan a cabalidad con atributos de calidad presentados por la norma ISO/IEC 25000 enfocados principalmente en los que se refieren a la mantenibilidad de software: Modularidad, reusabilidad, analizabilidad, capacidad de ser modificado, capacidad de ser probado, estabilidad [8].

Otro punto por atacar durante la presente investigación se centra en la Integración continua en el desarrollo de software, esta es una práctica en el desarrollo de software en la que los miembros del equipo integran su trabajo individual de manera regular, en esta práctica se automatiza la construcción, pruebas y validación. Esta práctica ayuda a encontrar y corregir errores rápidamente, además de mejorar constantemente la calidad del software [9].

1.6. METODOLOGÍA DE LA INVESTIGACIÓN

1.6.1 Tipo de estudio

El tipo de estudio para esta investigación es descriptivo, esto debido a que se quieren identificar, clasificar y dar una solución a los problemas que se presentan en el front-end de los productos digitales del Banco de Bogotá

1.6.2 Método de investigación

El método que se va a seguir en la investigación es el inductivo ya que con base en las falencias en varios aspectos de calidad que se evidenciaron en las observaciones preliminares que tiene la arquitectura front-end en los productos digitales del Banco de Bogotá podemos hacer el modelamiento de una solución de una arquitectura basadas en micro front-end en uno de los productos digitales del banco que permita no solo mitigar dichos problemas de calidad sino que también pueda aplicarse a los demás productos digitales del banco.

1.6.3 Fuentes y técnicas de recolección de la información

Debido a que nos encontramos laborando en el Banco de Bogotá y con el permiso de la entidad financiera nuestra fuente primaria es el acceso a los códigos fuente de los productos digitales del Banco y las herramientas de monitoreo y despliegue de dichos productos, las cuales nos servirán para realizar la investigación pertinente durante el desarrollo del proyecto.

1.6.4 Tratamiento de la información.

Una vez se hayan implementado el modelo arquitectural se comparan los tiempos de despliegue antes de realizarlo y después de hacerlo, para poder sacar las conclusiones pertinentes de la efectividad del modelo en el aspecto de calidad de integración continua.

1.7. ORGANIZACIÓN DEL TRABAJO DE GRADO.

En el primer capítulo de este trabajo de grado se encuentra una evaluación de cómo es el estado de arquitectura actual de los proyectos front-end que se están desarrollando en el laboratorio digital del Banco de Bogotá.

En el segundo capítulo se explica la forma de cómo se realiza una biblioteca para front-end en el lenguaje angular 8 y su respectivo despliegue en un manejador de paquetes.

En el tercer capítulo se muestra la arquitectura actual de la solución y la arquitectura propuesta en el trabajo de grado realizado en su gran mayoría en el lenguaje de modelado de arquitectura empresarial Archimate.

En el cuarto capítulo se muestra como fue construida la biblioteca que aloja a todos los micro front-ends desarrollados, su respectivo despliegue y publicación.

Finalmente en la parte 3 se muestran los resultados obtenidos en la investigación y la discusión de los mismos, posteriormente se determinan las conclusiones basándose en los objetivos propuestos al inicio y por último se expone la prospectiva del trabajo de grado, explicando los temas que en los que se puede trabajar en investigaciones futuras.

1.8. ESTUDIO DE SISTEMAS PREVIOS.

En el laboratorio digital del banco de bogotá existen diversos productos creados para revolucionar la relación de la banca con las personas, es decir brindando la posibilidad a los clientes de crear y gestionar productos bancarios desde la web. Al día de hoy existen productos como: cuenta de ahorros, tarjeta de crédito, crédito de libranza, crédito de libre inversión, Banca móvil, Banca virtual, seguros y crédito de vivienda.

Hace poco más de tres años, se concibieron los proyectos de cuenta de ahorros y posteriormente tarjeta de crédito, siendo estos los primeros productos que brindan la posibilidad a los clientes de crear productos de manera 100% digital. En el momento de su planeación, se decide hacer uso de la tecnología AngularJs, debido a que se pretendía renderizar los contenidos y realizar las operaciones del lado del cliente, dejando atrás la gran carga que recibían los servidores cuando estos eran los encargados de la renderización y operaciones de cara al usuario. En cuanto la calidad del código, no se realizan pruebas exhaustivas del lado del frontEnd y se vé al proyecto FrontEnd como una única unidad que convive e interactúa con los componentes de BackEnd, encargados de enviar y recibir información del frontEnd para realizar actividades de persistencia y consulta a otros servicios del banco para

poder continuar con el flujo esperado por el cliente interesado en adquirir sus productos con el banco.

Con el paso del tiempo van naciendo nuevas ideas que se convierten en nuevos productos desarrollados por el laboratorio Digital del Banco de Bogotá: crédito de libranza, crédito de consumo y crédito de vivienda. Teniendo ya la experiencia obtenida en la construcción y desarrollo de cuenta de ahorros y tarjeta de crédito, se decide hacer uso de Angular versión 4, siendo más rigurosos en la correctitud del sistema, haciendo fuerte énfasis en el uso de técnicas de código limpio, cumpliendo estándares de estilo en el código y preocupándose en la escritura de pruebas unitarias.

Sin embargo, en el laboratorio siempre ha sido de mayor prioridad la aplicación de técnicas y tecnologías de punta apuntando al backEnd, debido a la importancia que tiene la escalabilidad y alta disponibilidad que deben tener los servicios para soportar a todos los clientes que concurren en la aplicación. Es decir, para el banco era más importante contar con infraestructura (cloud) y servicios más robustos que soporten y den respuesta a los clientes en tiempos muy bajos que tener un componente Frontend mejor desarrollado, porque bien o mal estaba al aire y cumplía con su deber: hacer llegar la información diligenciada por el cliente al backEnd, quién es el encargado de hacer lo demás.

En la actualidad, han nacido nuevos productos que ya incorporan herramientas como ReactJs, Angular 8, entre otras. Para el manejo de estilos se hace uso de SASS versión 4.x; debido a que se brinda especial atención a la escritura de pruebas unitarias con el fin de tener controlado el código y brindar mayor correctitud funcional, se hace uso de la herramienta Jasmine 3.X. Otra herramienta vital para brindar a los clientes un producto de alta calidad utilizada por todos los productos del Laboratorio Digital es SonarCloud, herramienta encargada de realizar reportes de: cobertura de código (porcentaje de líneas cubiertas por las pruebas unitarias), código duplicado, fragmentos de código sin utilizar, importaciones que nunca se emplean, entre otros.

En el laboratorio digital se impulsa la cultura devops, buscando despliegues automáticos usando Infraestructura como código (IaC), para hacer esto realidad del lado del frontEnd, se hace uso de la herramienta CircleCI, esta es una herramienta para la integración continua de software, esta herramienta se sincroniza con GitHub (gestor de versiones utilizado por el banco), y cada vez que recibe cambios confirmados para los diferentes ambientes(pruebas, preproducción y producción), dispara una serie de "jobs" (actividades secuenciales programadas).

En cuanto a la arquitectura empleada por los productos, se siguen los lineamientos y arquitectura de referencia establecida para todos los productos (web) del laboratorio digital:

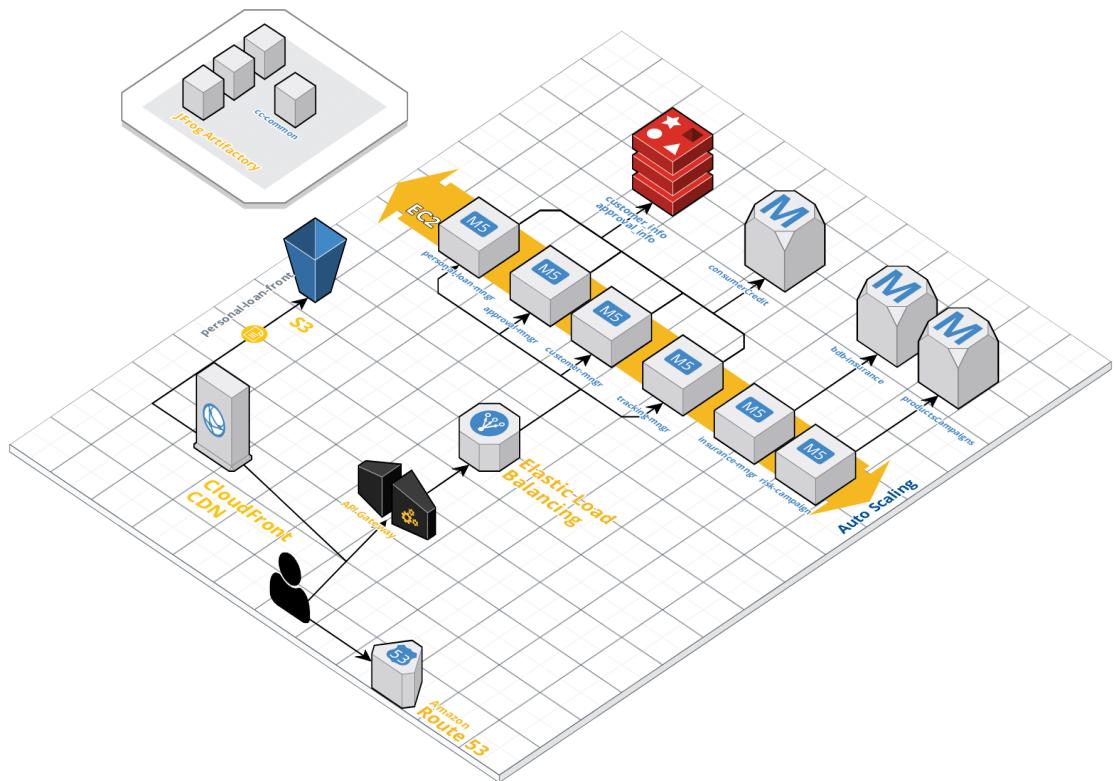


Imagen: Arquitectura de referencia de los productos del laboratorio digital del Banco de Bogotá
Fuente: Autores

El cliente accede gracias al servicio Route 53 de AWS, a un cloudFront que apunta a un bucket de S3 que contiene la aplicación en su interior, esta aplicación (frontEnd), realiza peticiones (usando el estilo arquitectónico REST) a un API gateway de AWS el cual enruta según la petición a un balanceador de carga y posteriormente a una instancia de AWS EC2 la cual contiene la aplicación backEnd requerida para responder a un servicio, ya sea para obtener información o para realizar alguna operación.

PARTE II. DESARROLLO DE LA INVESTIGACIÓN

CAPÍTULO 2. EVALUACIÓN DEL ESTADO ACTUAL DE LOS PROYECTOS FRONT-END (ARQUITECTURAS, TECNOLOGÍAS, LENGUAJES DE PROGRAMACIÓN) DE LOS PRODUCTOS DIGITALES DEL BANCO DE BOGOTÁ

En este capítulo se busca analizar la estructura actual de los proyectos front-end de los productos digitales del banco de Bogotá, con el fin de conocer a fondo su estructura, bajo qué arquitectura se creó, qué tecnologías utiliza; además de verificar el cumplimiento de atributos de calidad relacionados con la mantenibilidad de los sistemas.

Con respecto a la Arquitectura del FrontEnd de Fuerza Especializada de Vivienda, se cuenta con el esqueleto principal que ofrece una aplicación Angular, la cual cuenta con:

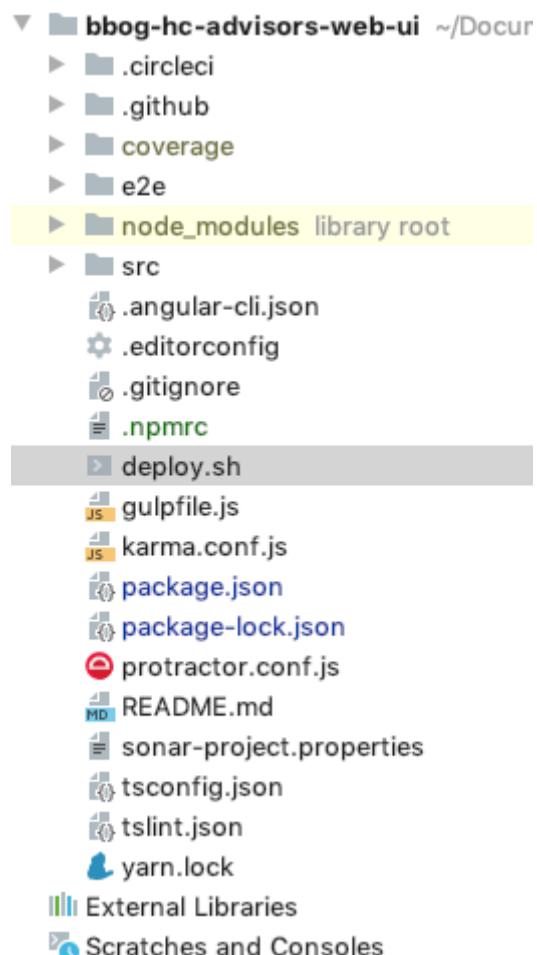


Imagen: Estructura aplicación Fuerza Especializada de Vivienda Fuente: Autores

A continuación se enumeran y describen los elementos más relevantes del esqueleto de la aplicación:

- Carpeta circleci: contiene los archivos necesarios que requiere CircleCI, para ejecutar los Jobs y realizar el despliegue necesario según el ambiente actual.
- github: Contiene archivos de configuración de GitHub.
- coverage: carpeta resultante después de ejecutar test unitarios, consta de reportes de cobertura, calidad de código, entre otros.
- node_modules: alberga todas los módulos y librerías que necesita la aplicación para trabajar correctamente.
- src: código fuente de la aplicación
- .angular-cli.json: archivo de configuración de la aplicación.
- .editorconfig: archivo de configuración del editor.
- deploy.sh: archivo ejecutable que realiza tareas sobre AWS durante el despliegue.
- karma.conf.js: archivo de configuración para el servidor de pruebas.
- package.json: archivo que especifica cada librería necesaria con su respectiva versión.
- protractor.conf.js: Archivo para configuración de reportes generados desde el servidor de pruebas.
- tsconfig.json: configuración para archivos TypeScript.
- tslint.json: configuración de estilos de código.

En cuanto al código fuente de la aplicación, se maneja la siguiente estructura.

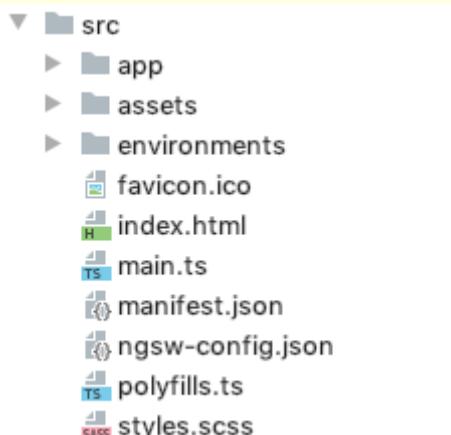


Imagen: estructura de la carpeta src del producto de Fuerza Especializada de Vivienda Fuente: Autores

A continuación se enumeran y describen los elementos más relevantes de la carpeta de fuentes de la aplicación:

- app: contiene carpetas y archivos propios de lógica y presentación de la aplicación.
- assets: contiene la multimedia necesaria para la presentación de la aplicación.
- environments: contiene la configuración necesaria para cada ambiente de la aplicación (desarrollo, pruebas y producción).
- favicon: ícono central de la aplicación.
- index.html: página de acceso a la aplicación.
- manifest.json: configuración básica de presentación.
- styles.scss: estilos css generales de la aplicación

El producto front-end de Fuerza Especializada de Vivienda del Banco de Bogotá, se desarrolla sobre un esqueleto de aplicación monolítico, el cual contiene todas las vistas y funcionalidades dentro del mismo proyecto, pero separado por componentes y páginas de la siguiente forma:

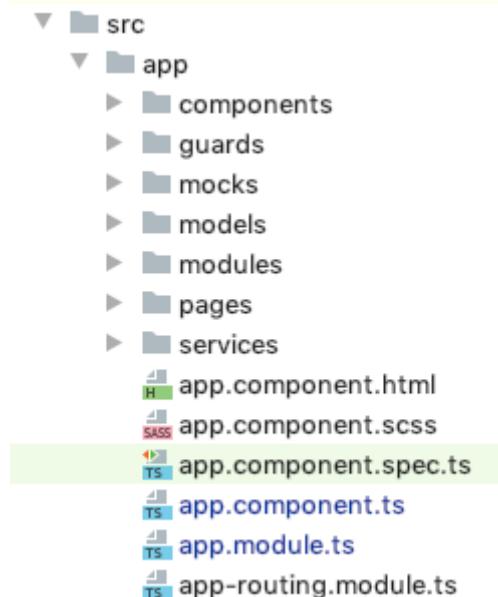


Imagen: estructura de aplicación del Producto de Fuerza Especializada de Vivienda. Fuente: Autores

A continuación se enumeran y describen los elementos más relevantes de la estructura de aplicación:

- components: contiene aquellos componentes (componentes angular: contienen archivo typeScript, de pruebas, de presentación HTML, y de estilos CSS), que hacen parte de una página (se pueden reutilizar en distintas páginas).
- guards: archivos typeScript de configuración de seguridad.
- mocks: archivos para simular datos reales en las pruebas unitarias.
- models: interfaces de angular que componen el modelo de negocio (clases propias).
- modules: módulos de Angular (contiene las importaciones necesarias para funcionar, lo que declara y lo que exporta para su utilización).
- pages: contiene componentes de angular (archivo typeScript, de pruebas, de presentación HTML, y de estilos CSS), pero contrario a los components, estos ocupan una página, esto quiere decir que deben ser accedidas mediante un enrutador y no estando embebidas en otras páginas.
- Services: contiene los servicios consumidos por las pages y los components.

Como se puede observar, es una aplicación que hace uso de componentes reutilizables pero dentro de la misma aplicación, ninguna otra aplicación puede hacer uso de sus componentes ni sus servicios. Otro aspecto clave a resaltar es que por tener una estructura tan amplia, cuando el producto crece en funcionalidad, decrece en mantenibilidad, debido a que se vuelve una tarea compleja encontrar un error y corregirlo sin afectar otros posibles elementos de la aplicación.

En la actualidad, los productos incorporan herramientas como ReactJs, Angular 8, entre otras. Pero para el producto en cuestión: Portal de Fuerza especializada de Vivienda, se hace uso de: Angular 5 como framework de presentación, para el manejo de estilos se hace uso de SASS versión 4.x; debido a que se brinda especial atención a la escritura de pruebas unitarias con el fin de tener controlado el código

y brindar mayor correctitud funcional, se hace uso de la herramienta Jasmine 3.X. Al igual que todos los demás productos del Laboratorio Digital, se hace uso de una herramienta vital para brindar a los clientes un producto de alta calidad: SonarCloud, herramienta encargada de realizar reportes de: cobertura de código (porcentaje de líneas cubiertas por las pruebas unitarias), código duplicado, fragmentos de código sin utilizar, importaciones que nunca se emplean, entre otros.

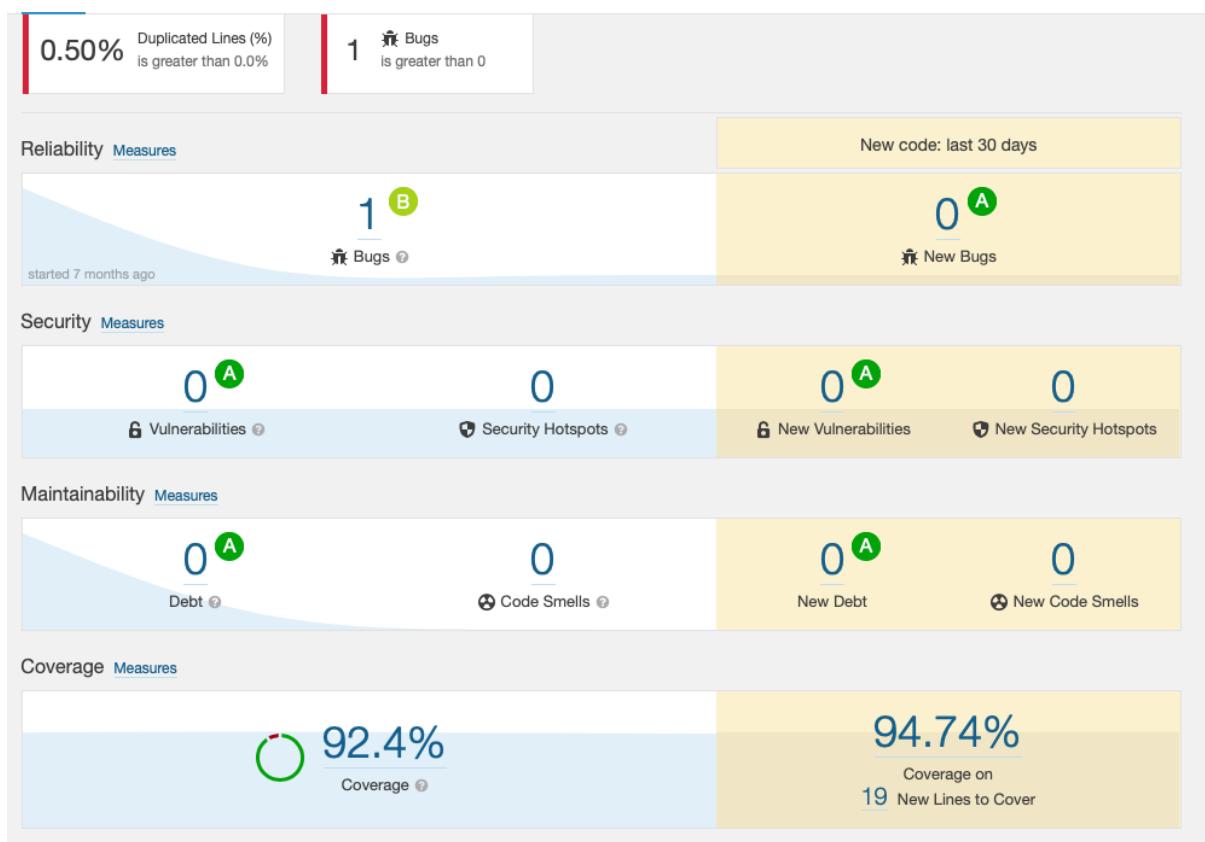


Imagen: Reporte generado por SonarCloud al producto de Fuerza Especializada de Vivienda
Fuente: Autores

Con respecto a la integración continua que se busca en el laboratorio Digital, el producto de Fuerza Especializada de Vivienda, al igual que todos los proyectos de FrontEnd, cumple con los tres Jobs necesarios para su despliegue, como se ve a continuación:

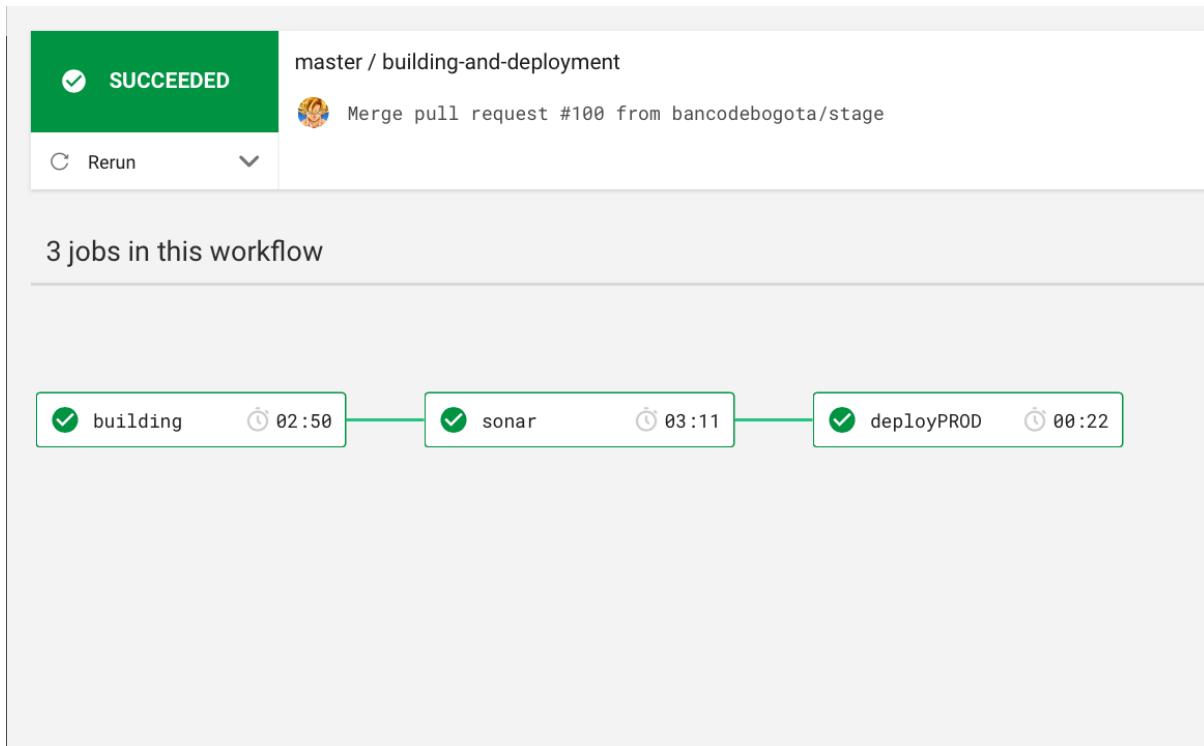


Imagen: Despliegue aplicación Fuerza Especializada de Vivienda Fuente: Autores

Como se ve en la imagen, existe el job `building`, encargado de hacer el montaje de la aplicación en una máquina de CircleCI. El siguiente job es `"Sonar"`, encargado de realizar las pruebas de la aplicación para realizar un posterior reporte e indicar si la aplicación cuenta con los estándares de calidad requeridos para su despliegue. El último paso que realiza es el `"deploy"`, encargado de copiar la aplicación a un "bucket" de S3 (AWS), donde se albergará la aplicación para poder ser consumida desde internet. Cabe aclarar que los pasos realizados son secuenciales (uno comienza cuando el anterior termina) y en caso de ocurrir algún fallo se detiene la ejecución y por tanto el despliegue. Esto da un parte de calma al saber que no se enviará código a producción que no cuente con los estándares de calidad establecidos con el banco y así brindarle un producto probado y confiable a los clientes.

CAPÍTULO 3. ESTUDIO Y ANÁLISIS DE LA CREACIÓN Y COMUNICACIÓN ENTRE COMPONENTES FRONT-END DESARROLLADOS EN ANGULAR

En este capítulo se explicará la forma de crear una biblioteca que proporcione un servicio, un componente y algunas interfaces que puedan ser utilizadas por otros proyectos.

Desde la versión 6 de Angular la posibilidad de crear bibliotecas es mucho más sencilla que versiones anteriores. Para empezar, se ejecuta el comando `ng new` para crear el proyecto base:

```
jorgeatuesta:~ jorgeatuesta$ ng new lib-demo --prefix ld
```

Comando de creación de nuevo proyecto Fuente: Autores

3.1. Creación de la biblioteca

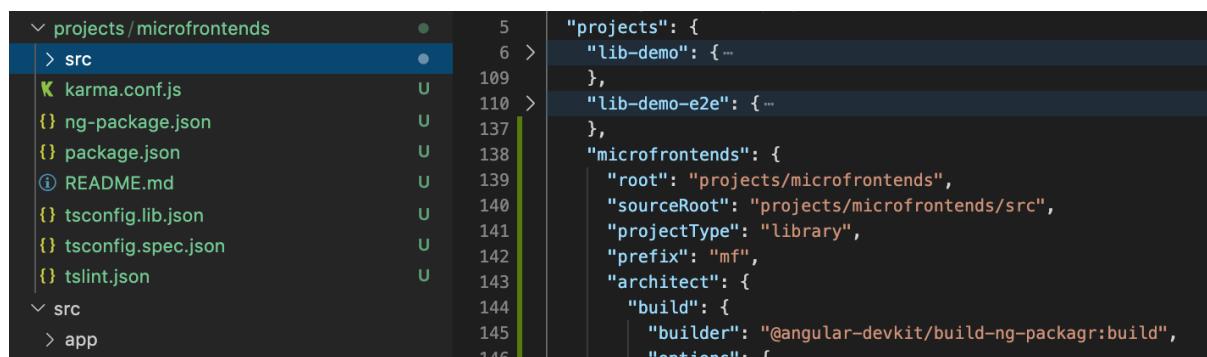
Con la versión 6 de Angular CLI, el formato del archivo de configuración cambió de una manera bastante sustancial. `angular.json` representa ahora lo que se llama un espacio de trabajo, posiblemente con múltiples proyectos. De forma predeterminada, obtenemos proyectos `lib-demo` y `lib-demo-e2e` generados, pero también se puede agregar más con el comando "`ng generate application [mi-app]`".

Lo que también es posible generar en lugar de la aplicación, es una biblioteca:

```
jorgeatuesta:lib-demo jorgeatuesta$ ng generate library microfrontends --prefix mf
```

Comando de creación de una biblioteca dentro del proyecto Fuente: Autores

El comando hace una modificación en el `angular.json` adicionando un esquema nuevo:



```
  "projects": {  
    "lib-demo": { ... },  
    "lib-demo-e2e": { ... },  
    "microfrontends": {  
      "root": "projects/microfrontends",  
      "sourceRoot": "projects/microfrontends/src",  
      "projectType": "library",  
      "prefix": "mf",  
      "architect": {  
        "build": {  
          "builder": "@angular-devkit/build-ng-packagr:build",  
          "options": { ... }  
        }  
      }  
    }  
  }  
}
```

Imagen del archivo angular.json resultante Fuente: Autores

En particular, se creó un nuevo directorio "projects", con la carpeta "micro frontends" dentro. Este nuevo proyecto también está referenciado en la configuración angular.json.

Para permitir que los desarrolladores que usan esta biblioteca tengan acceso a las interfaces, estas deben ser proporcionadas como una API pública de la biblioteca micro frontends. Esto se hace en el archivo `/microfrontends/src/public_api.ts` el cual tiene la siguiente información:

```
/*
 * Public API Surface of microfrontends
 */

export * from './lib/microfrontends.service';
export * from './lib/microfrontends.module';
export * from './lib/models/response.model';
```

Estructura del archivo public_api.ts Fuente: Autores

3.2. Creación de servicio

Como es posible apreciar, en el archivo se encuentran exportado tanto el módulo como un servicio y un modelo que va a contener la información que responde un endpoint REST que se consume dentro de dicho servicio de la siguiente manera:

```
import { Injectable } from '@angular/core';
import { Show } from './models/response.model';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class MicrofrontendsService {

  private readonly apiRoot = 'https://api.tvmaze.com';

  constructor(private httpClient: HttpClient) { }

  getShow(id: number): Observable<Show> {
    const url = `${this.apiRoot}/shows/${id}`;
    return this.httpClient.get<Show>(url);
  }
}
```

Imagen del servicio para consumir un API Fuente: Autores

No hay nada inusual en este servicio aparte de la configuración raíz de `provideIn: root` en el decorador `@Injectable`. En realidad, es una nueva configuración agregada en Angular 6.

"`provideIn: root`" permite proporcionar un servicio sin registrarlo explícitamente en ningún NgModule.

3.3. Creacion de componente

Esta biblioteca permite mostrar un póster de la respuesta que ofrece el api REST según un código específico [19]. Al igual que con un proyecto angular típico, no es necesario crear un componente a mano, se crea con el siguiente comando:

```
jorgeatuesta:lib-demo jorgeatuesta$ ng generate component poster --project=microfrontends
```

Comando de creación de un componente dentro de la biblioteca Fuente: Autores

Lo cual genera un componente dentro de la biblioteca que se está creando, el cual adicionando la estructura correspondiente para que se muestre la información que retorna el api, quedaría de la siguiente manera:

```
import { Component, OnInit, Input } from '@angular/core';
import { MicrofrontendsService } from '../microfrontends.service';
import { Observable } from 'rxjs';
import { map } from 'rxjs/operators';

@Component({
  selector: 'mf-poster',
  template: `<img *ngIf="posterUrl$ | async as src" [src]="src" />`,
  styles: [
    `
      :host {
        display: inline-block;
        overflow: hidden;
        border-radius: 4px;
        line-height: 0;
      }
    `
  ]
})
export class PosterComponent implements OnInit {

  @Input() showId: number;
  posterUrl$: Observable<string>;

  constructor(private microfrontendsService: MicrofrontendsService) {}

  ngOnInit() {
    this.posterUrl$ = this.microfrontendsService
      .getShow(this.showId)
      .pipe(map(show => show.image.medium));
  }
}
```

Imagen del componente creado Fuente: Autores

Para hacer que el componente esté disponible por fuera del entorno de la biblioteca, es necesario adicionarlo en la sección de `exports` del módulo de la biblioteca. El archivo `projects/tvmaze/src/lib/tvmaze.module.ts` debe verse de la siguiente manera:

```
import { NgModule } from '@angular/core';
import { MicrofrontendsComponent } from './microfrontends.component';
import { PosterComponent } from './poster/poster.component';
import { CommonModule } from '@angular/common';
import { HttpClientModule } from '@angular/common/http';

@NgModule([
  declarations: [MicrofrontendsComponent, PosterComponent],
  imports: [
    CommonModule, HttpClientModule
  ],
  exports: [MicrofrontendsComponent, PosterComponent]
])
export class MicrofrontendsModule { }
```

Estructura del módulo de la biblioteca Fuente: Autores

3.4. Compilación de la biblioteca

Antes de usar la biblioteca, es necesario construirla. De nuevo, Angular CLI proporciona un comando para realizar esto en un proyecto específico:

```
jorgeatuesta:lib-demo jorgeatuesta$ ng build microfrontends
```

Comando para compilar la biblioteca Fuente: Autores

3.5. Uso

Para usar la biblioteca, en primer lugar se debe importar el módulo de la misma en el módulo principal de la aplicación donde se quiere usar:

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { MicrofrontendsModule } from 'microfrontends';

You, a few seconds ago | 1 author (You)
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    MicrofrontendsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Impartición del módulo de la biblioteca Fuente: Autores

Y para utilizar el componente se hará como sigue:

```
import { Component } from '@angular/core';
import { Observable } from 'rxjs';
import { Show } from 'microfrontends';
import { MicrofrontendsService } from 'microfrontends';

You, a few seconds ago | 1 author (You)
@Component({
  selector: 'ld-root',
  template: `
    <mf-poster [showId]="336"></mf-poster>
    <pre>{{ show$ | async | json }}</pre>
  `
})
export class AppComponent {
  show$: Observable<Show>;
  constructor(private microFrontends: MicrofrontendsService) {
    this.show$ = this.microFrontends.getShow(336);
  }
}
```

Uso del componente de la biblioteca Fuente: Autores

Y al subir la aplicación se muestra el componente de la biblioteca:



```
{  
  "id": 336,  
  "url": "http://www.tvmaze.com/shows/336/marvels-agent-carter",  
  "name": "Marvel's Agent Carter",  
  "type": "Scripted",  
  "language": "English",  
  "genres": [  
    "Action",  
    "Adventure",  
    "Science-Fiction"  
  ],  
  "status": "Ended",  
  "runtime": 60,  
  "premiered": "2015-01-06",  
  "officialsite": "http://abc.go.com/shows/marvels-agent-carter",  
  "schedule": {  
    "time": "21:00",  
    "days": [  
      "Tuesday"  
    ]  
  },  
  "rating": {  
    "average": 8  
  },  
  "weight": 91,  
  "network": {  
    "id": 3,  
    "name": "ABC",  
    "country": {  
      "name": "United States",  
      "code": "US",  
      "timezone": "America/New_York"  
    }  
  },  
  "webchannel": null,  
  "externals": {}  
}
```

Imagen del componente creado desde el navegador Fuente: Autores

3.6. Publicación

Para que la biblioteca esté disponible en npm, todo lo que necesita hacer es crear una compilación de producción y ejecutar el comando `npm publish` desde el directorio dist del proyecto:

```
jorgeatuesta:lib-demo jorgeatuesta$ ng build microfrontends
Building Angular Package
Building entry point 'microfrontends'
Compiling TypeScript sources through ngc
Bundling to FESM2015
Bundling to FESM5
Bundling to UMD
Minifying UMD bundle
Copying declaration files
Writing package metadata
Removing scripts section in package.json as it's considered a potential security vulnerability.
Built microfrontends
Built Angular Package!
- from: /Users/jorgeatuesta/lib-demo/projects/microfrontends
- to:   /Users/jorgeatuesta/lib-demo/dist/microfrontends
```

Comando de construcción de biblioteca *Fuente:Autores*

```
jorgeatuesta:lib-demo jorgeatuesta$ cd dist/microfrontends/
```

Comando para cambiar al directorio que crea la construcción *Fuente:Autores*

```
jorgeatuesta:microfrontends jorgeatuesta$ npm publish
```

Comando de publicación de biblioteca *Fuente:Autores*

Para realizar este último paso es necesario crear una cuenta en npm primero y después realizar el acceso en la máquina con el comando `npm login`.

CAPÍTULO 4. ARQUITECTURA

El objetivo de este capítulo es diseñar la arquitectura que contemple las actuales problemáticas halladas en los componentes front-end de los productos digitales del banco de Bogotá.

Para comenzar, el presente caso de estudio está basado en una parte del proceso completo de créditos hipotecarios que se ofrecen en el Banco de Bogotá, que procede como sigue:

1. El líder de los asesores de vivienda asigna un caso(solicitud) a uno de los asesores de la fuerza especializada de vivienda.
2. El asesor se contacta vía telefónica o correo electrónico con el cliente solicitando los documentos necesarios para su estudio y posterior aprobación.
3. Una vez el asesor obtiene estos documentos procede a enviarles a un área la cual realiza una validación documental y toma la decisión de aprobar o no la solicitud de crédito hipotecario.
4. En caso de que la aprobación se lleve a cabo satisfactoriamente, se notificará al cliente de dicha decisión.

Para abordar la construcción de la arquitectura debe contemplar la misión de la empresa, la cual busca ser un Banco líder en Colombia para el mercado de empresas, personas y el sector social, oficial. Un Banco siempre a la vanguardia para brindar a sus clientes soluciones anticipadas, que les permitan vivir una experiencia bancaria satisfactoria.

El mejor apoyo para el crecimiento y progreso de sus clientes, porque los valora y está dispuesto y disponible para asesorarlos, prestandoles un servicio ágil, oportuno, amable y de calidad. Un Banco que cumple con los objetivos de liderazgo en eficiencia, rentabilidad, utilidad y generación de valor que esperan los accionistas.

Un Banco que cuenta con un grupo de talentosos colaboradores que trabajan con ingenio, dedicación, eficiencia, agilidad, compromiso, lealtad, siempre orientados al logro y motivados por el orgullo de pertenecer a la institución que les genera bienestar y crecimiento. El Banco a través de su ejemplo y apoyo ratifica su compromiso con la construcción e interiorización de sus valores, generando crecimiento, convivencia y bienestar a la comunidad.

A continuación se presenta un diagrama de procesos internos del Banco, el cual ilustra los subprocesos concernientes a la preaprobación y captura de documentos para su estudio por parte de análisis documental.

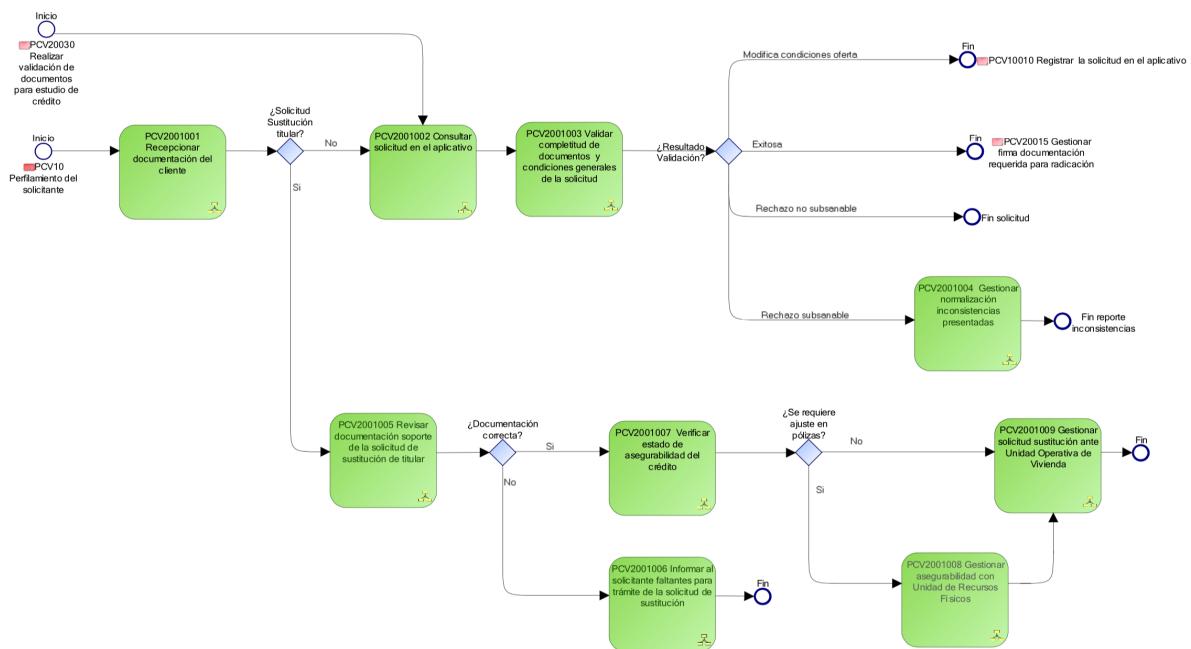


Figura: PCV 20010 Verificar consistencia de la documentación para el producto solicitado. Fuente: Banco de Bogotá.

Los procesos descritos son realizados por distintos roles, los cuales se listan a continuación:

- Cliente : Solicitar el crédito hipotecario en una oficina, si es canal tradicional debe brindar la información necesaria para la solicitud a un asesor en una oficina, si es canal digital debe diligenciar sus datos a través de un portal o brindar la información a un asesor quien los diligencia en un portal.
Debe brindar la documentación necesaria para obtener la aprobación de su crédito.
Debe brindar la documentación e información necesaria para iniciar el proceso de legalización del inmueble.
- Asesor : Recopilar los datos del solicitante, además de la documentación necesaria para realizar el estudio del crédito, brindar una guía sobre el diligenciamiento de los datos pertinentes para realizar la solicitud correctamente.
- Ejecutivo : Contactar vía telefónica al cliente para conocer su interés acerca de continuar con su proceso de crédito. De ser afirmativa la respuesta del cliente se procede a solicitar la documentación requerida para iniciar el proceso de legalización.
- Coordinador: Asignar solicitudes de crédito preaprobadas a los diferentes asesores de la fuerza especializada de ventas.
- Analista de crédito : Recibir los documentos y proceder a validarlos junto con la información del cliente para aprobar o negar una solicitud.

Para lograr ofrecer una solución tecnológica que vaya alienada a la misión de la empresa y sus objetivos organizacionales, se realiza el diseño de puntos de vista de negocio, aplicación, tecnología, motivación y proyecto.

4.1. Capa de negocio.

Esta capa incluye una cantidad de conceptos informativos adicionales que son relevantes en el dominio comercial: un producto y un contrato asociado, el significado de los objetos comerciales y el valor de los productos y servicios comerciales.

El aspecto de la estructura en la capa empresarial se refiere a la estructura estática de una organización, en términos de las entidades que componen la organización y sus relaciones.

Se distinguen dos tipos de entidades: entidades activas y entidades pasivas.

- La entidades activas realizan algunos comportamientos, como procesos o funciones de negocios, sean individuales, grupales o de recursos a largo plazo o más permanentes.
 - Actores de negocios
 - Roles comerciales
 - Personas individuales (por ejemplo, clientes o empleados)
 - Grupos de personas (unidades organizativas)
- Las entidades pasivas (también llamadas objetos comerciales) son manipuladas por comportamientos tales como procesos o funciones comerciales.

4.1.1. Punto de Vista de Organización.

En el caso de estudio que se presenta en este libro se analizará el proceso que se realiza en las solicitudes de crédito de vivienda del Banco de Bogotá, en particular la sección en la que se verifica la consistencia de la documentación para el crédito solicitado. En este proceso un solicitante se dirige a un acceso del Banco de Bogotá, ya sea físico o virtual, realiza una solicitud de crédito y dicha solicitud es analizada por una Fuerza Especializada de Vivienda, la cual se encarga de revisar los documentos solicitados para el crédito.

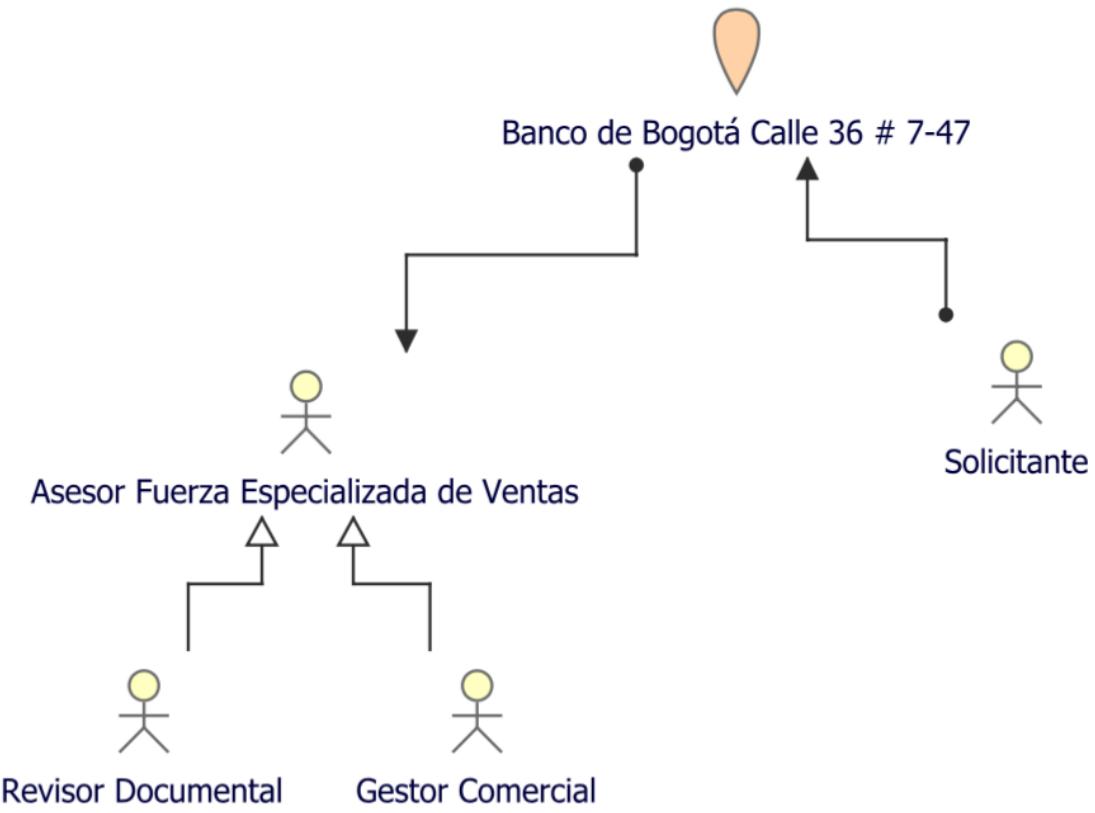


Figura: Punto de Vista de Organización. Fuente: Autores

4.1.2. Punto de Vista de Cooperación de Actor.

En este punto de vista se hace una especificación mejor del proceso que se quiere estudiar, un rol de solicitante, que puede ser aplicado a un cliente actual, cliente nuevo o un cliente prospecto(un cliente que no tiene productos con el Banco de Bogotá pero que se tiene alguna información dentro de las bases de datos del Banco), hace una solicitud de crédito de vivienda a través del portal digital, y en colaboración con el actor de asesor FEV, pueden hacer la carga y revisión de los documentos necesarios para llevar a buen término la solicitud.

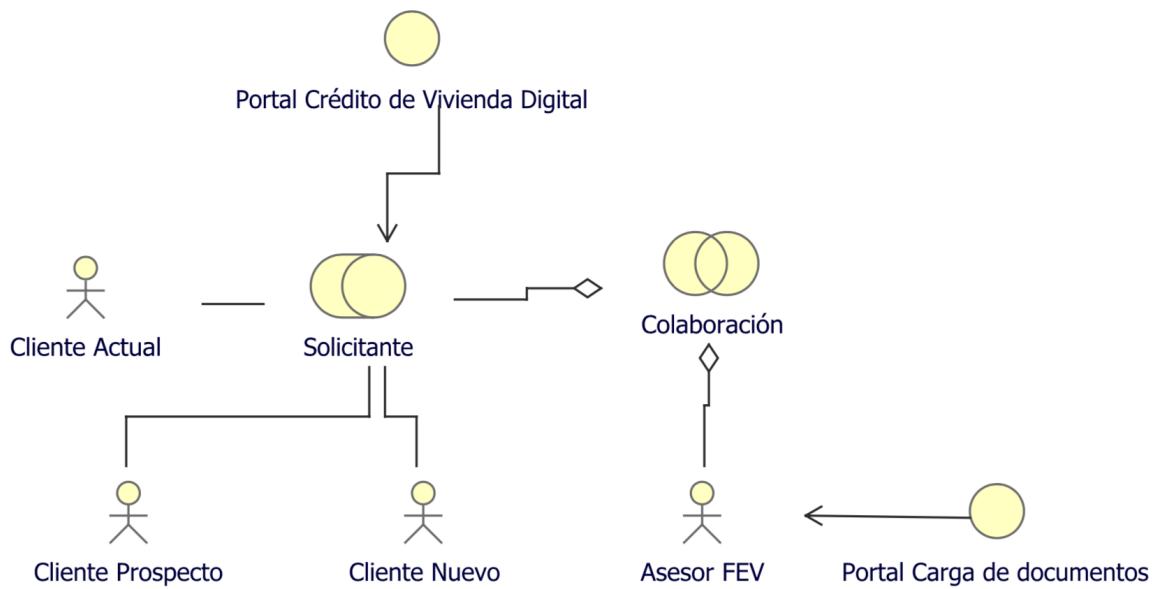


Figura: Punto de Vista de Cooperación de Actor. Fuente: Autores

4.1.3. Punto de Vista de Función de Negocio.

Dentro de las funciones del rol de radicador comercial está la de verificar la consistencia de la documentación según el perfil o el producto, reportar las inconsistencias de las verificaciones realizadas, y dentro de las funciones del solicitante son la de diligenciar los datos para el perfilamiento y radicar la documentación pertinente. Una función puede disparar otra como se muestra en el siguiente punto de vista.

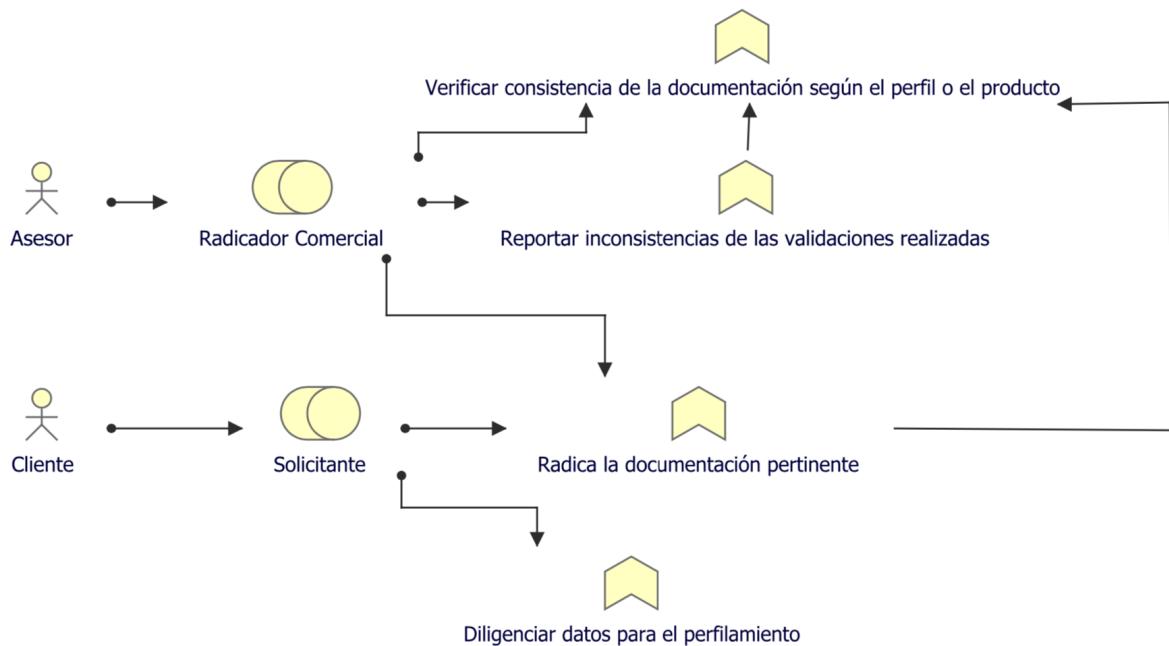


Figura: Punto de Vista de Función de negocio. Fuente: Autores

4.1.4. Punto de Vista de Proceso de Negocio.

En primer lugar se genera un evento que es el perfilamiento del cliente para luego generar una oferta comercial del crédito que ha pedido el solicitante, una vez que la oferta comercial se acepta de inicia el proceso de verificación documental que generará el servicio de obtener un crédito de vivienda. Dicho proceso central está especializado por varios subprocesos que se pueden ver a continuación.

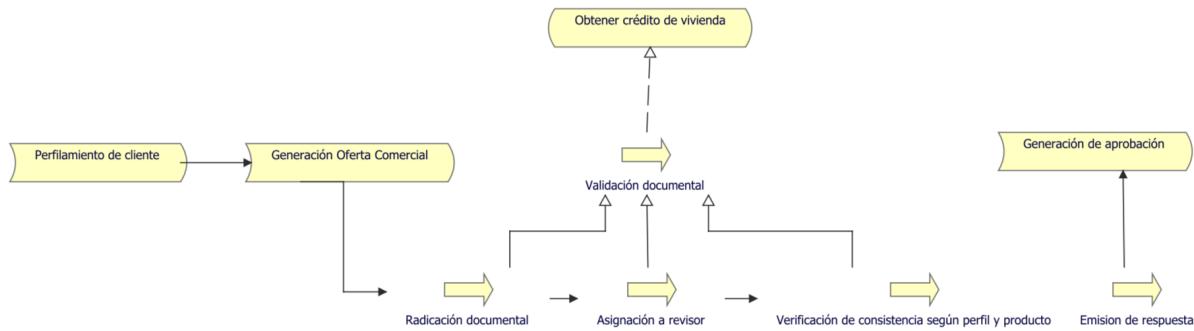


Figura: Punto de Vista de Proceso de Negocio. Fuente: Autores

4.1.4. Punto de Vista de Cooperación de Proceso de Negocio.

El portal de asesores de Fuerza Especializada de Vivienda es utilizado por el rol de Radicador Documental, que tiene como una de sus funciones revisar la validez de los documentos de una solicitud específica, siendo un subprocesso de dicha validación es la de emitir una respuesta positiva o negativa para poder o continuar o denegar la aprobación del crédito de vivienda.

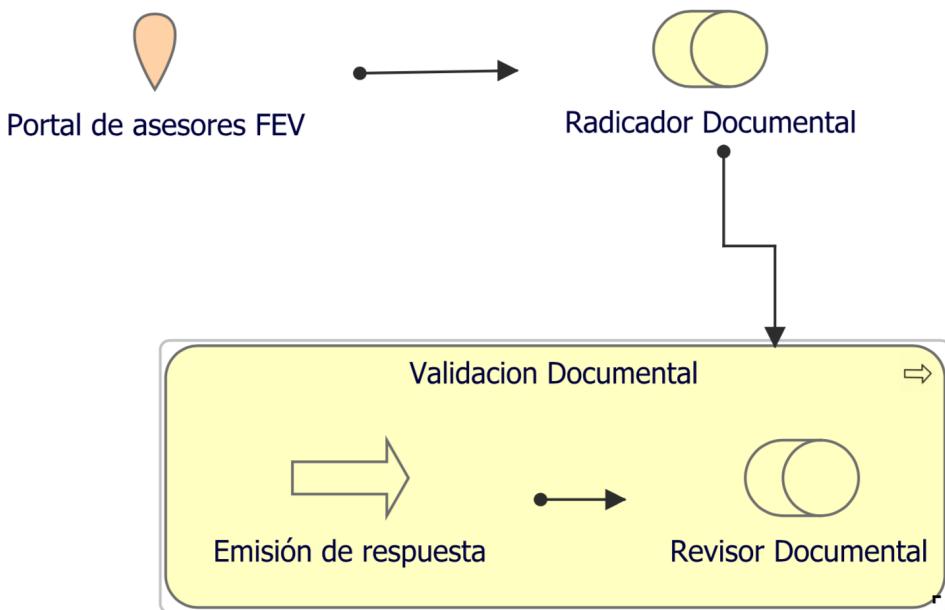


Figura: Punto de Vista de Cooperación de proceso de Negocio. Fuente: Autores

4.1.5. Punto de vista de Producto.

El producto final del caso de estudio es el portal de asesores de la Fuerza Especializada de Vivienda, vinculado a las políticas internas del Banco, el cual estudia la capacidad de endeudamiento, el cumplimiento, la cantidad de productos, información financiera externa del rol solicitante y apoyado con los servicios de cargar los documentos de la solicitud y la de enviar la solicitud a revisión documental, pueden generar los valores de Obtener la aprobación en menos de 48 horas y poder solicitar el crédito sin ir a una oficina física.

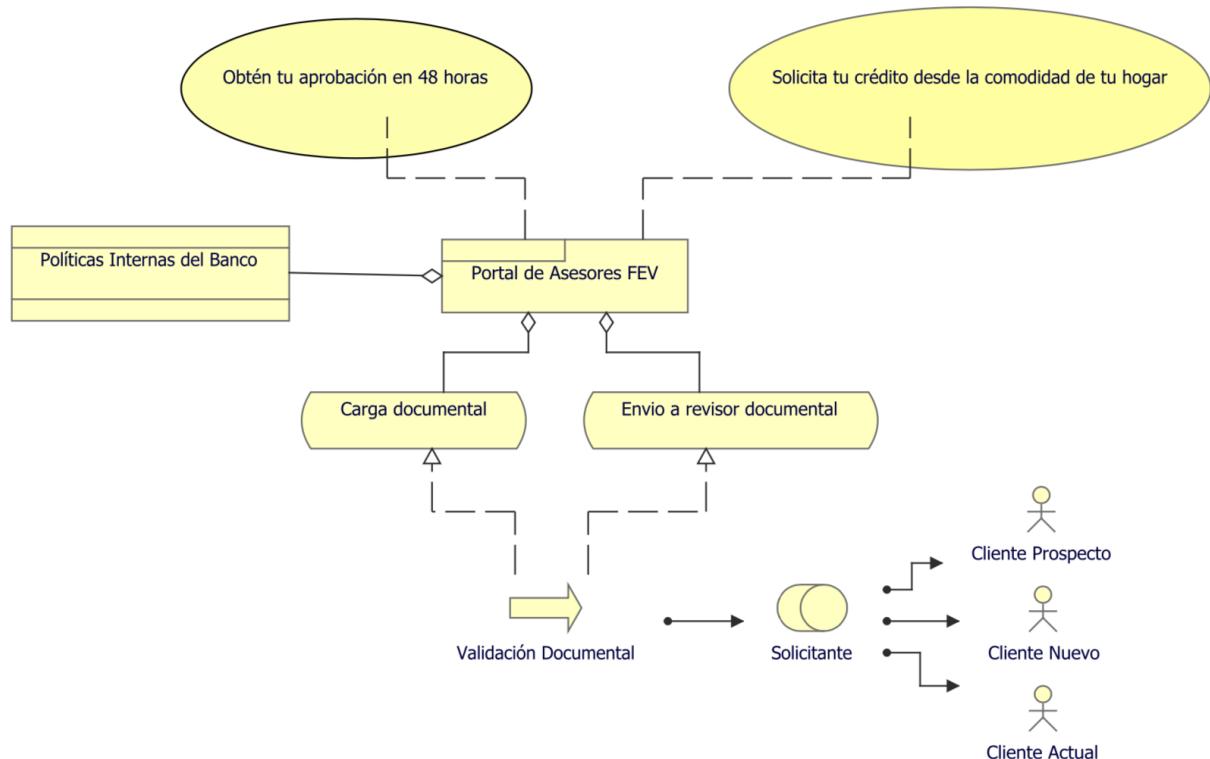


Figura: Punto de Vista de Cooperación de Producto. Fuente: Autores

4.2. Capa de Aplicación.

La capa de aplicación se usa típicamente para modelar las arquitecturas de los sistemas de información de la empresa, incluida la arquitectura de la aplicación que, tal como se define en el marco TOGAF, describe la estructura y la interacción de las aplicaciones.

Esta capa se compone de tres tipos de elementos estructurales los cuales son: activos, de comportamiento y pasivos.

En los elementos estructurales activos se encuentran los componentes, las colaboraciones y las interfaces.

En los elementos estructurales de comportamiento están las funciones, las interacciones, los procesos, los eventos y los servicios

En los elementos estructurales pasivos únicamente se encuentra los objetos de datos.

4.2.1. Punto de Vista de Comportamiento de Aplicación.

Para el caso de estudio del portal de asesores de la Fuerza Especializada de Vivienda del Banco de Bogotá, se cuenta con diversos componentes: Gestión de asesores, cuya función se limita a brindar la posibilidad a un coordinador de asignar solicitudes de vivienda a un asesor de su centro especializado. El componente de Carga documental, el cual permite a un asesor cargar los documentos requeridos al cliente y el componente de envío a análisis documental, el cual se encarga de hacer llegar los documentos al departamento de análisis de crédito.

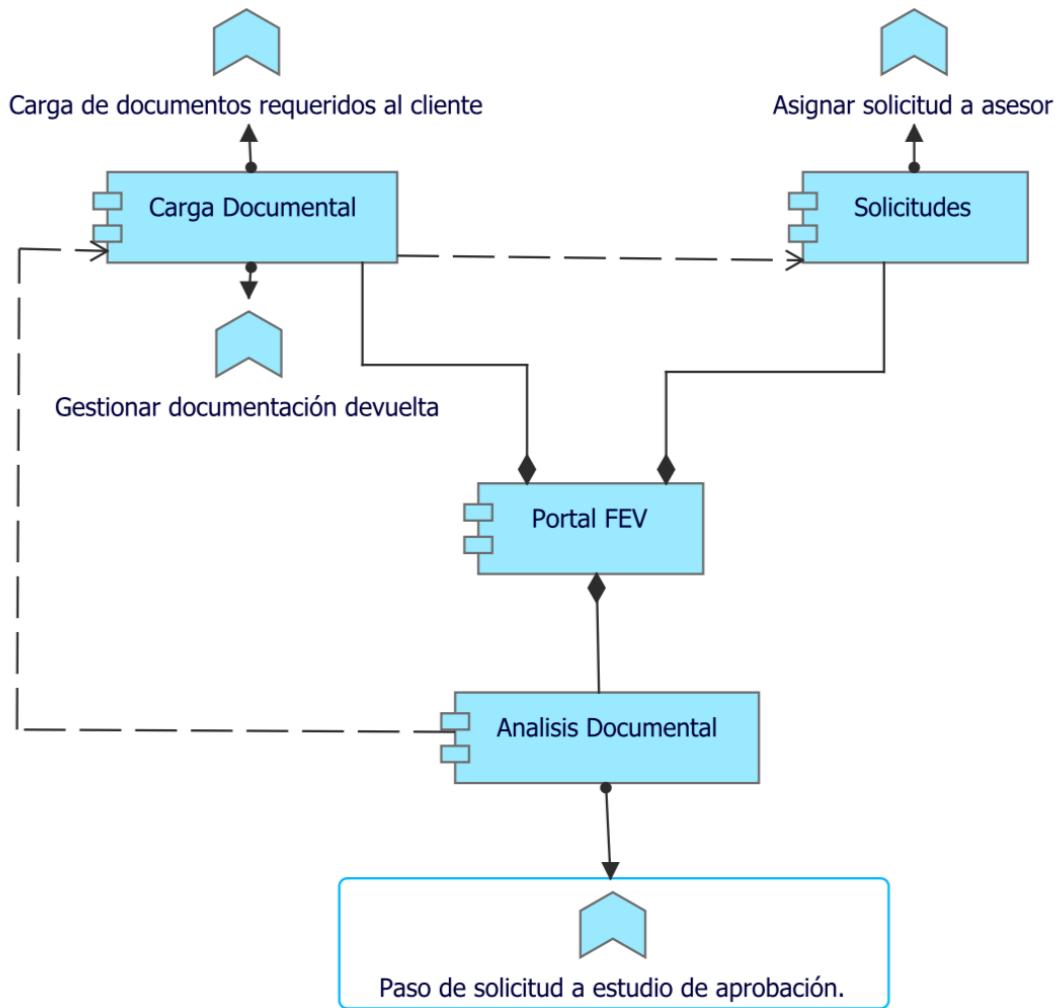


Figura: Punto de Vista de Comportamiento de Aplicación. Fuente: Autores

4.2.2. Punto de Vista de Cooperación de Aplicación.

Para el producto de la fuerza especializada de Vivienda del Banco de Bogotá se dividen en grupos de servicios y back end, de tal forma que se exponen componentes que prestan servicios específicos que a su vez hacen uso de servicios back end del banco.

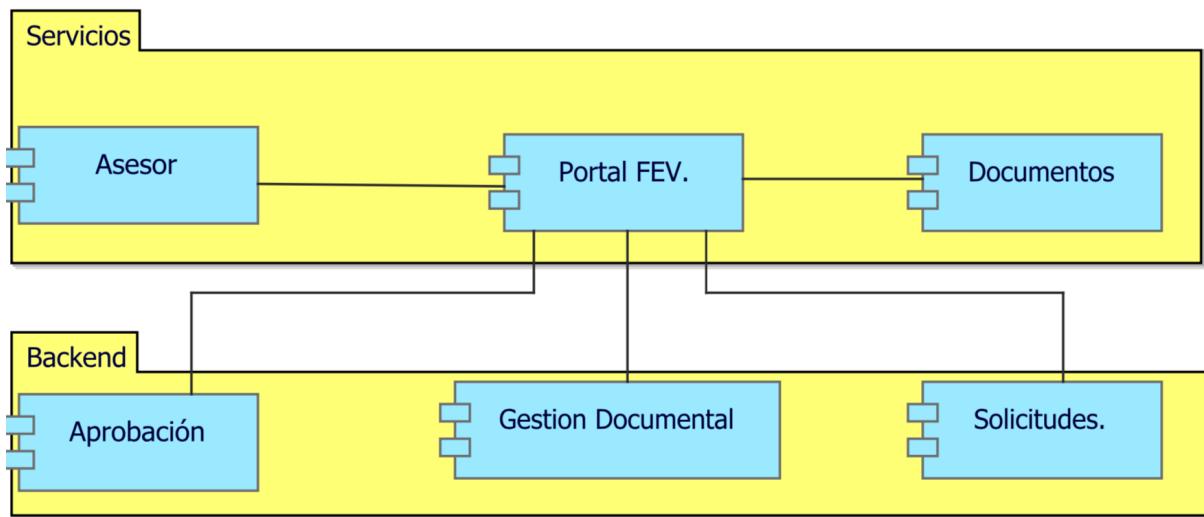


Figura: Punto de Vista de Cooperación de Aplicación. Fuente: Autores

4.2.3. Punto de Vista de Estructura de Aplicación.

El producto de Fuerza Especializada de Vivienda hace uso de diversos componentes que proveen distintos servicios que en conjunto logran la funcionalidad deseada para el producto: Aprobar solicitudes, cargar documentación del cliente, Asignar solicitudes a asesores y Almacenar la documentación del cliente para su posterior estudio.

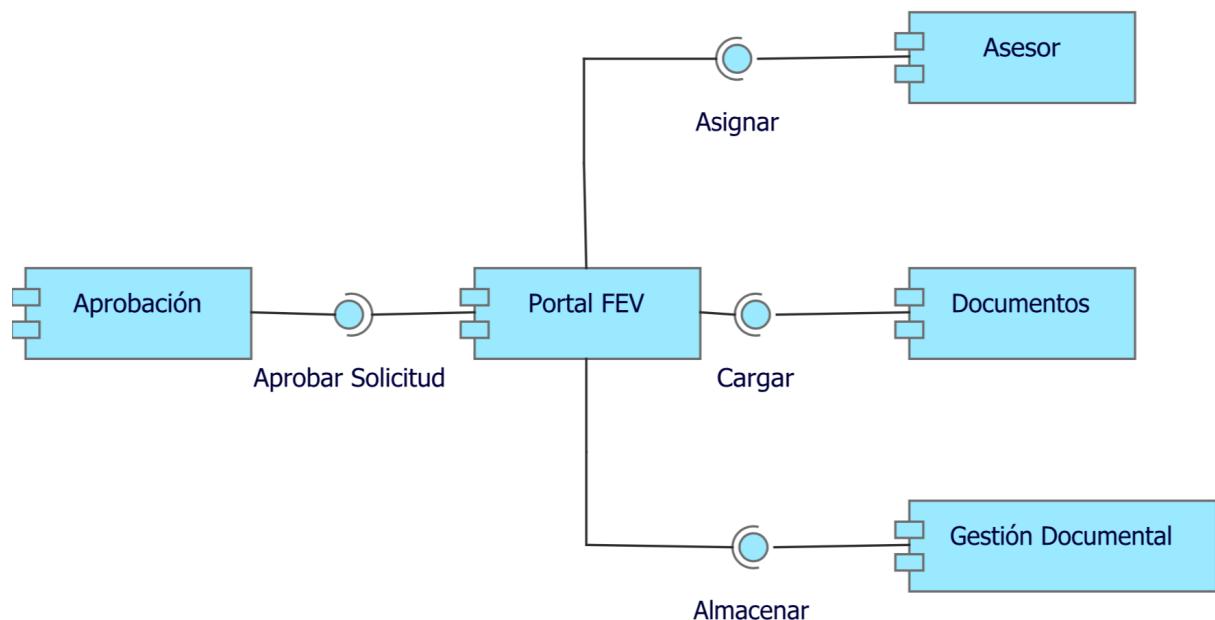


Figura: Punto de Vista de Estructura de Aplicación. Fuente: Autores

4.2.4. Punto de Vista de Uso de Aplicación.

Para el producto de Fuerza Especializada de Vivienda el evento de "pre-aprobación" dispara el proceso de recepción de documentos, el cual es realizado por el rol "Asesor", el cual hace uso de la interfaz portal asesores, el cual hace uso del componente de asesores encargado de todo el proceso de carga documental y posterior envío a estudio documental para su posterior aprobación de crédito.

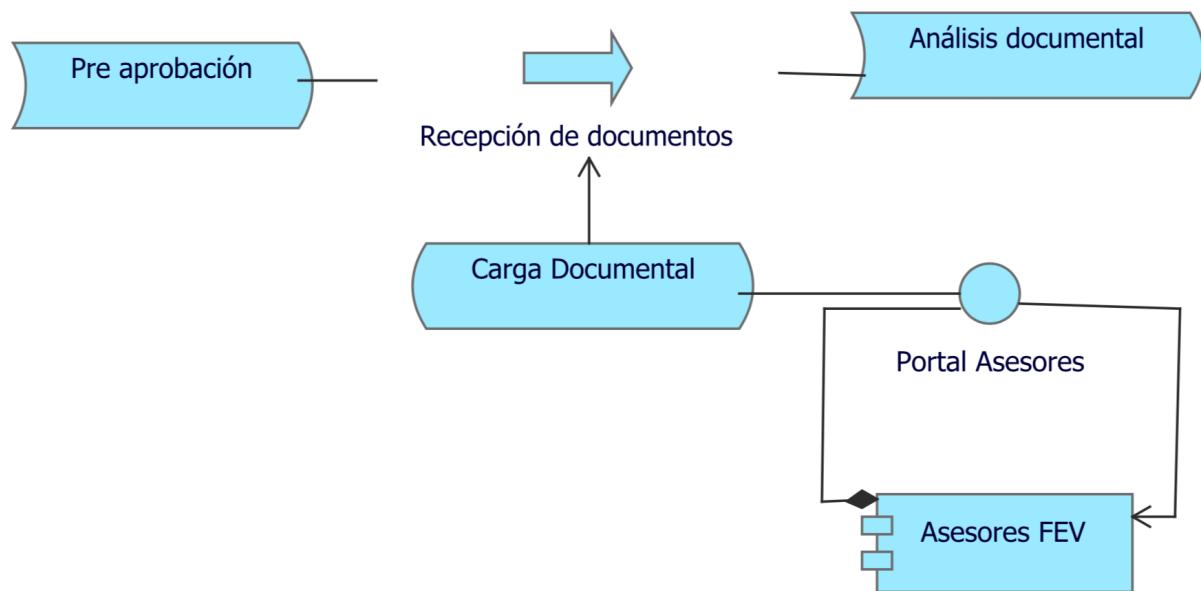


Figura: Punto de Vista de Uso de Aplicación. Fuente: Autores

4.3. Capa de Tecnología.

La capa de tecnología muestra los servicios tecnológicos, tales como los servicios de procesamiento, almacenamiento y comunicación, necesarios para ejecutar las aplicaciones, además del hardware, software necesarias para realizar dichos servicios. En esta capa se adicionan elementos físicos para modelar equipos físicos, materiales y redes de distribución.

En los elementos estructurales activos de esta capa se encuentra como concepto principal el nodo, también se encuentran las interfaces, las rutas y redes de comunicación, los dispositivos, sistemas de software y las colaboraciones.

Dentro de los elementos estructurales pasivos están los artefactos y los objetos de tecnología.

En los elementos estructurales de comportamiento se hallan las funciones, los procesos, las interacciones, los eventos y los servicios.

4.3.1. Punto de Vista de Tecnología.

Para poner al aire el aplicativo de Fuerza es necesario contemplar los elementos tecnológicos empleados para ello, en este caso mediante el uso de tecnologías propias de la computación en la nube, tecnologías que facilitan la publicación de contenido estático, el almacenamiento, bases de datos,平衡adores de carga, entre otros.

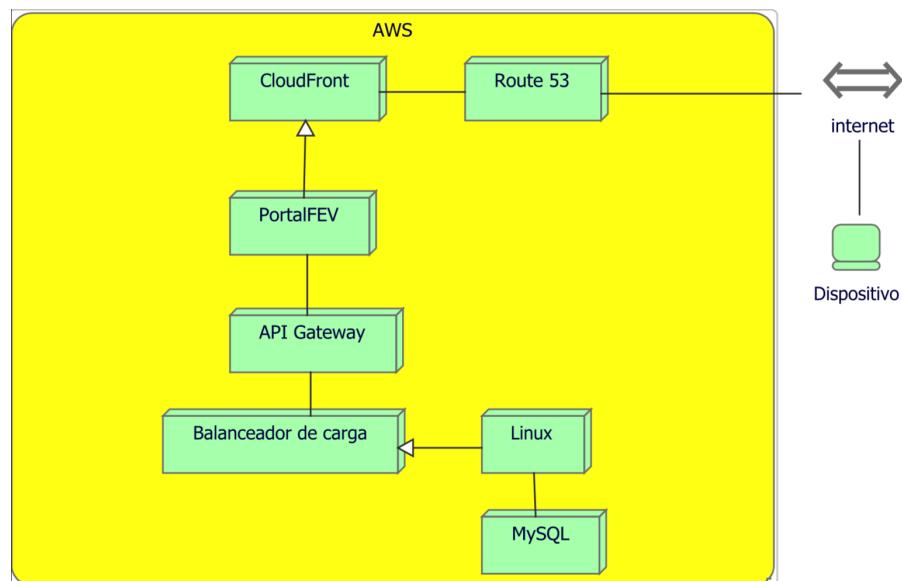


Figura: Punto de Vista de Tecnología. Fuente: Autores

4.3.2. Punto de Vista de Uso de Tecnología.

Para el aplicativo de Fuerza especializada de vivienda, mediante un componente (Portal FEV), se orquesta la interacción con los demás componentes. Componente de aprobación y preaprobación (encargados de llamar motores y almacenar resultados de los mismos), Asesor, encargado de gestionar solicitudes de vivienda asignadas a un asesor. Notificación, encargado de envío de correos y sms tanto para clientes como para asesores. Cola de mensajería para gestionar envíos de solicitudes a análisis documental y su correspondiente respuesta.

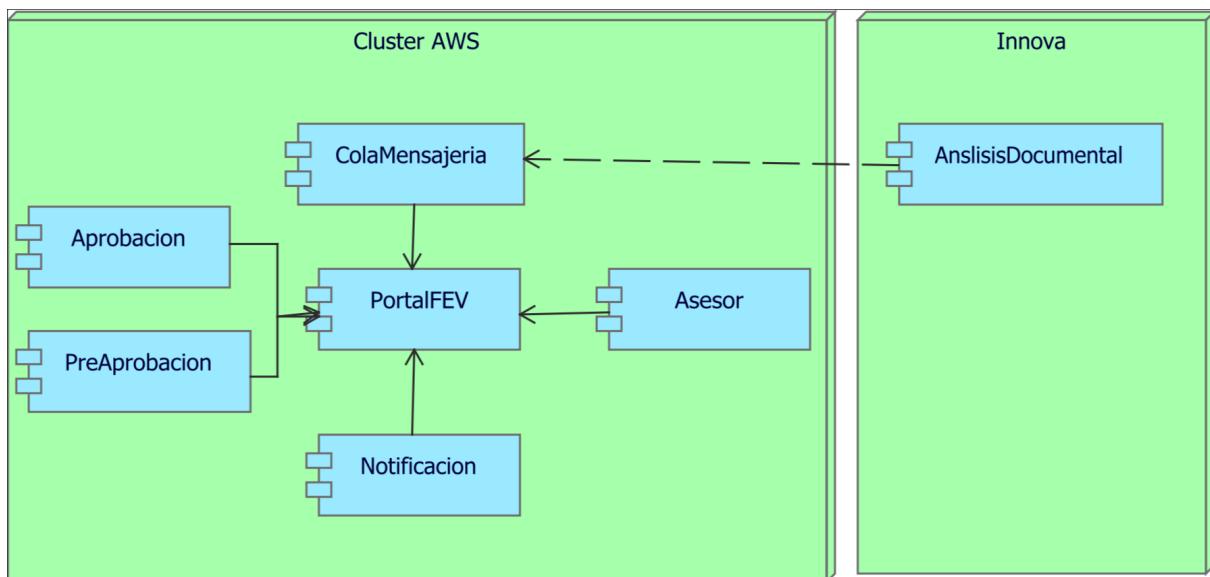


Figura: Punto de Vista de Uso de Tecnología. Fuente: Autores

4.3.3. Punto de Vista de Organización e Implementación.

Una vez que el asesor carga todos los documentos del cliente, los envía y pasan a estudio documental y una vez se revisen y sean válidos, se retorna una respuesta dando lugar a una aprobación de crédito de vivienda.

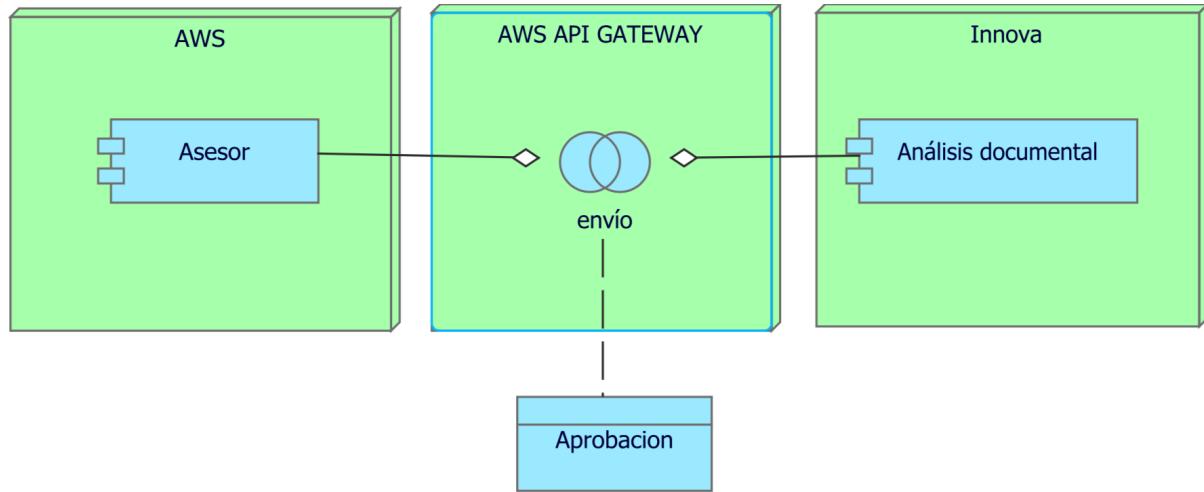


Figura: Punto de Vista de Organización e Implementación. Fuente: Autores

4.3.4. Punto de Vista de Estructura de Información.

El objeto Solicitud es el eje central de la presente estructura, debido a que la aprobación y pre aprobación dependen de que exista una solicitud. Los asesores son asignados a solicitudes y los clientes las crean, la solicitud está embebida en cada documento que compone el paquete de documentos el cual es enviado a análisis documental.

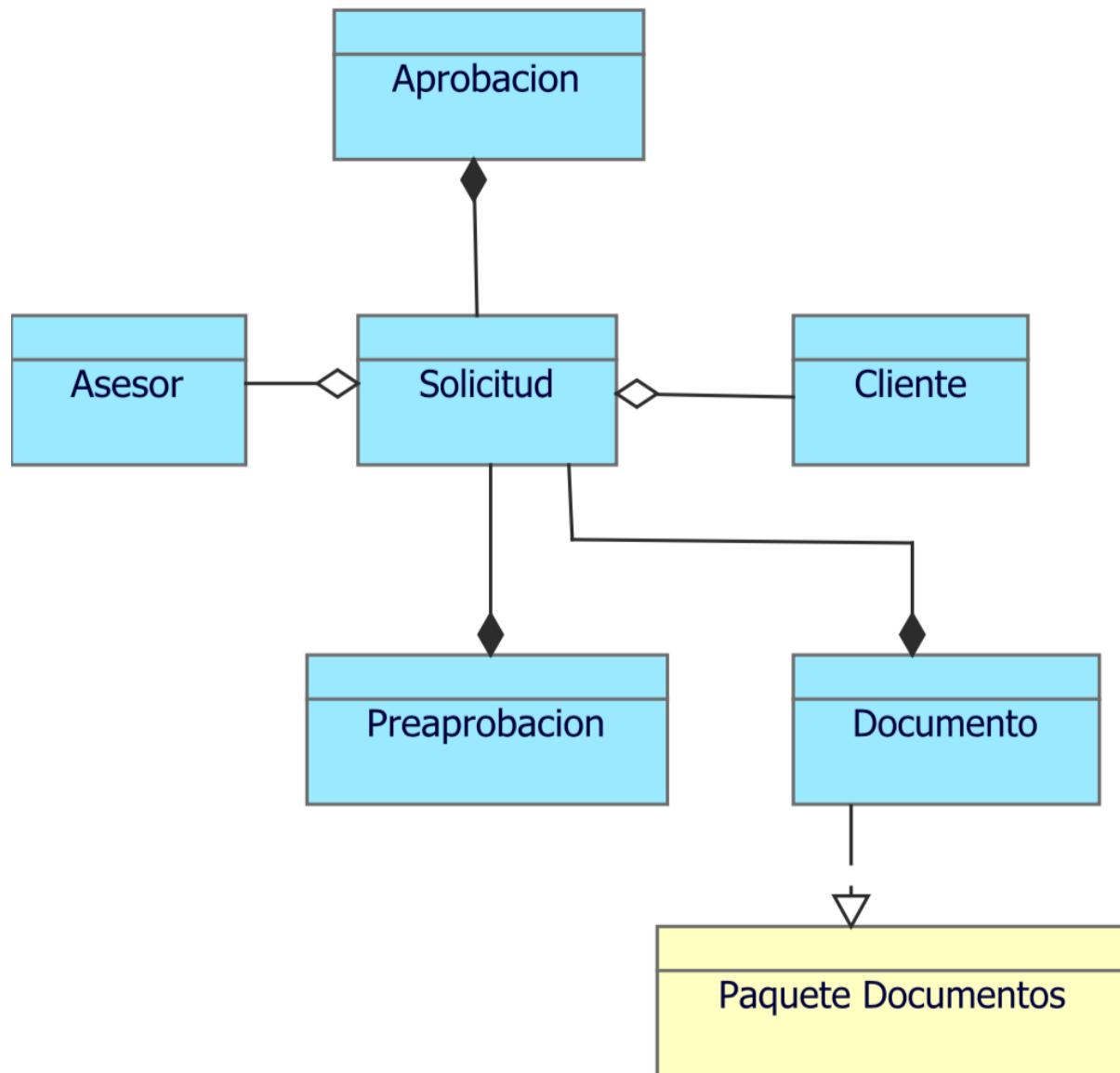


Figura: Punto de Vista de Estructura de Información. Fuente: Autores

4.3.5. Punto de Vista de Realización de Servicio.

El Asesor carga un paquete de documentos y los envía al departamento de análisis documental en el que se realiza su estudio, esto con el fin de lograr aprobar una solicitud realizada por el cliente para su posterior desembolso de crédito de vivienda.

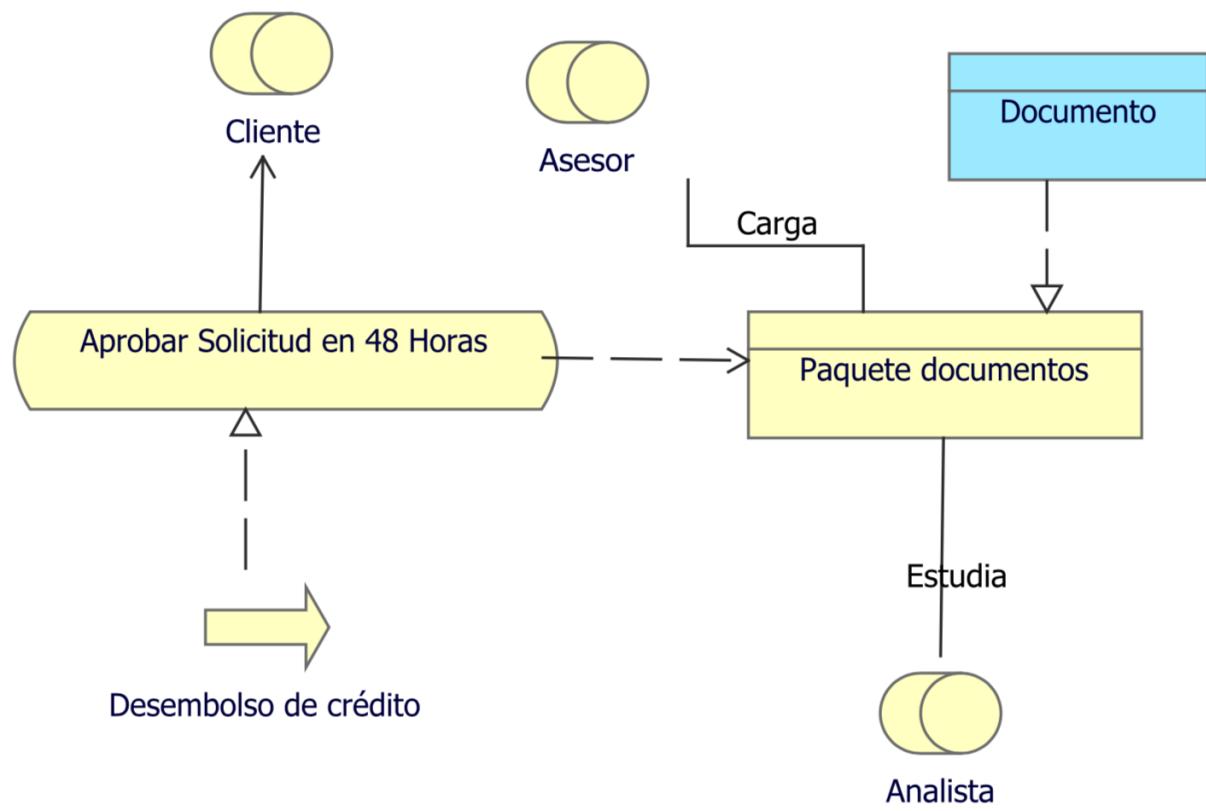


Figura: Punto de Vista de Realización de Servicio. Fuente: Autores

4.3.6. Punto de Vista de Capas.

Con el fin de que el cliente alcance su aprobación de crédito de vivienda digital, se presentan tres capas: negocio, la cual contiene el proceso de aprobación de crédito; aplicación, la cual contiene el componente de aplicación “PortalFEV”, en cuanto a la capa de infraestructura, en su interior lleva el nodo AWS, el cual contiene todos los elementos tecnológicos necesarios para el despliegue de la aplicación en la nube.

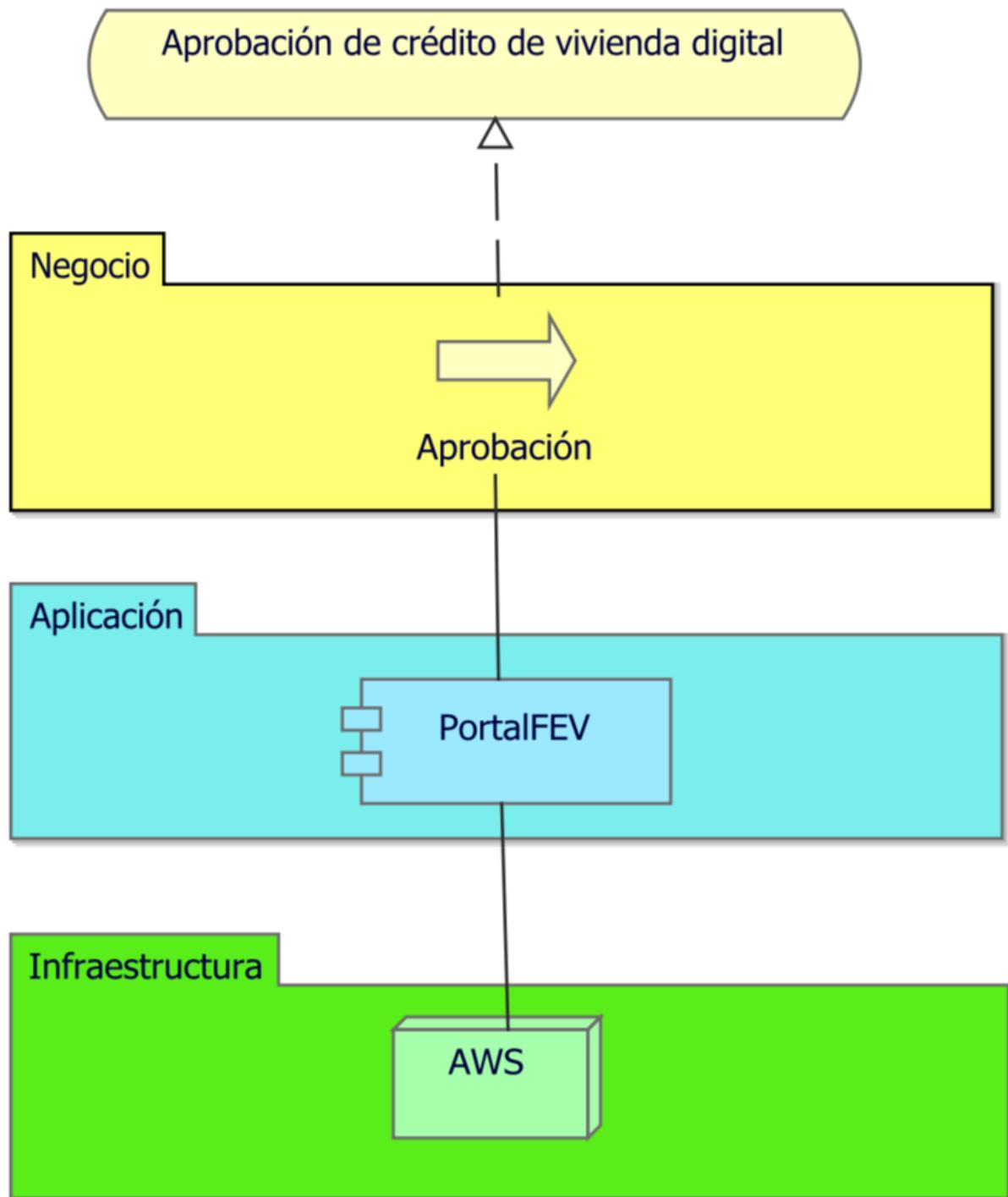


Figura: Punto de Vista de Capas. Fuente: Autores

4.4. Capa de Motivación.

Los conceptos motivacionales se utilizan para modelar las razones, que subyacen al diseño o cambio de alguna arquitectura empresarial. Estas motivaciones influyen, guían y limitan el diseño.

El metamodelo de motivación es una extensión en ArchiMate e incluye dos tipos de conceptos motivacionales, la fuente de intención y las motivaciones o intenciones.

La fuente de la intención se origina dentro o fuera de la empresa, lo que influye en los elementos motivacionales. Entre ellas se encuentran los stakeholders, conductores y evaluaciones.

Las intenciones o motivaciones están representadas por objetivos, principios, requisitos y limitaciones.

4.4.1. Punto de Vista de Stakeholder.

El asesor tiene el objetivo de asesorar al cliente en la carga de documentos para su posterior estudio (especialmente valioso para no perder clientes que carecen de conocimiento informático), con el fin de obtener una aprobación de crédito de vivienda sin necesidad de salir de casa.

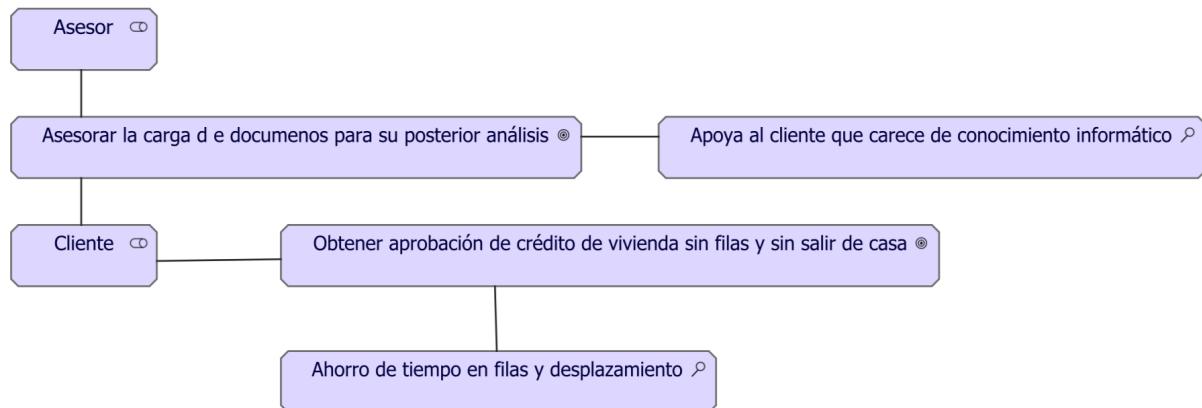


Figura: Punto de Vista de Stakeholder. Fuente: Autores

4.4.2. Punto de Vista de Realización de Objetivos

Para poder realizar la carga de documentos por parte del asesor FEV, el asesor debe tener asociada la solicitud del cliente (realizado por un coordinador), además, el asesor debe tener a la mano todos los documentos requeridos para el cliente, por último el cliente debe tener ya una pre aprobación de vivienda digital.

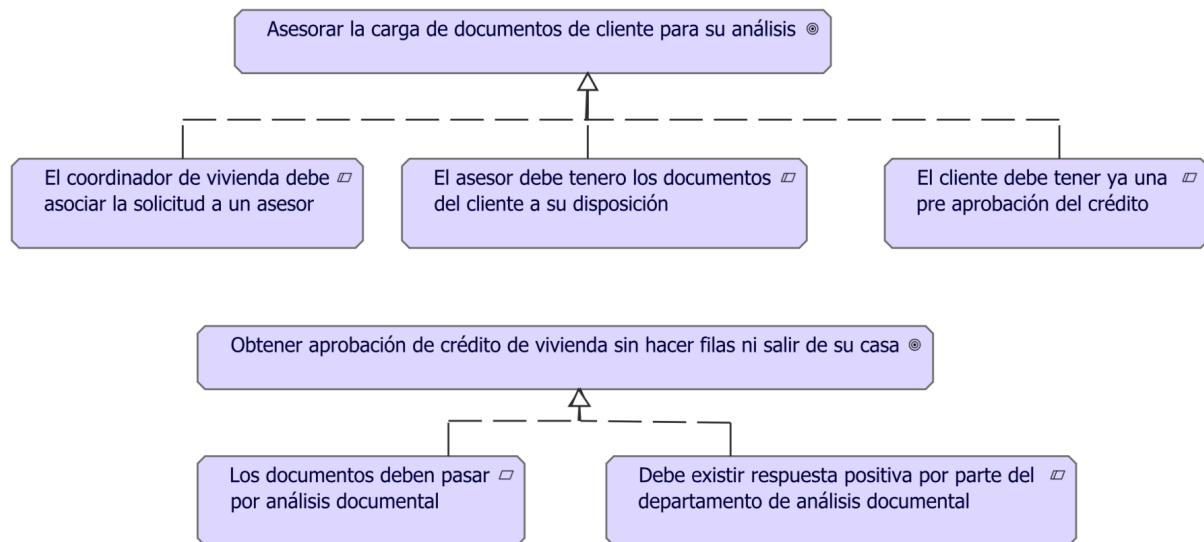


Figura: Punto de Vista de Realización de Objetivos. Fuente: Autores

4.4.3. Punto de Vista de Contribución de Objetivos

Para cumplir el objetivo de asesoramiento al cliente en la carga de documentos , el cliente debe tener previamente una pre aprobación digital de vivienda, lo cual aporta en gran medida a lograr una aprobación de vivienda y en menor medida al desembolso de manera ágil, esto debido a que cuando un cliente está presente en bases de datos de motores que aprueban inmediatamente, se evita al cliente este proceso.

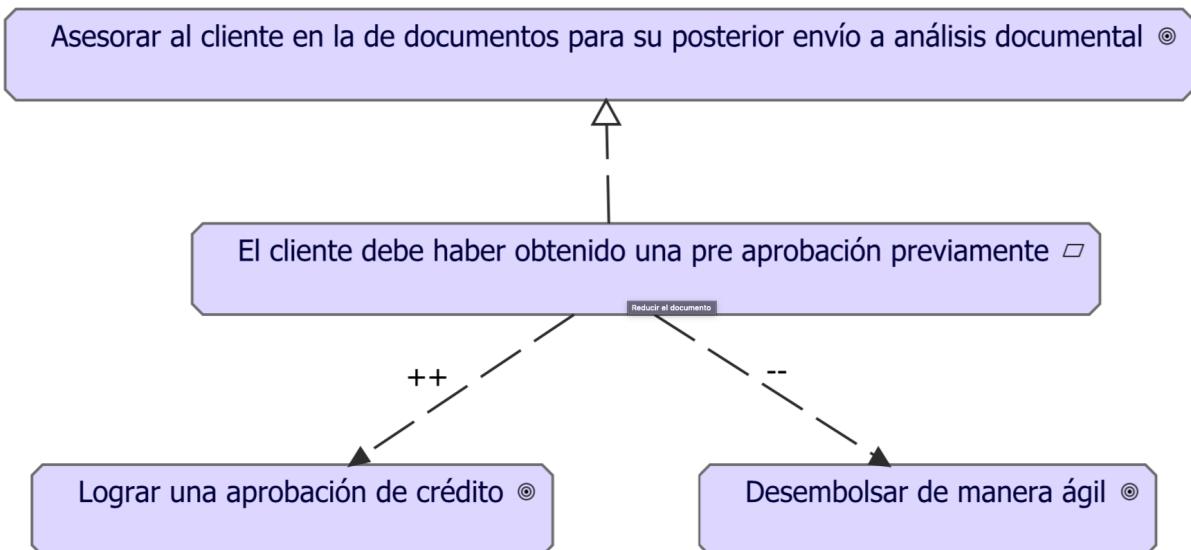


Figura: Punto de Vista de Contribución de Objetivos. Fuente: Autores

4.4.4. Punto de Vista de Principios

La aprobación de crédito de vivienda busca ofrecer al cliente: Facilidad, ofreciendo flujos cortos y desde casa, Agilidad, se ofrece al cliente aprobacion de credito de vivienda en 48 horas, por último Comodidad, lograr aprobación de crédito de vivienda desde la comodidad de casa y sin filas.

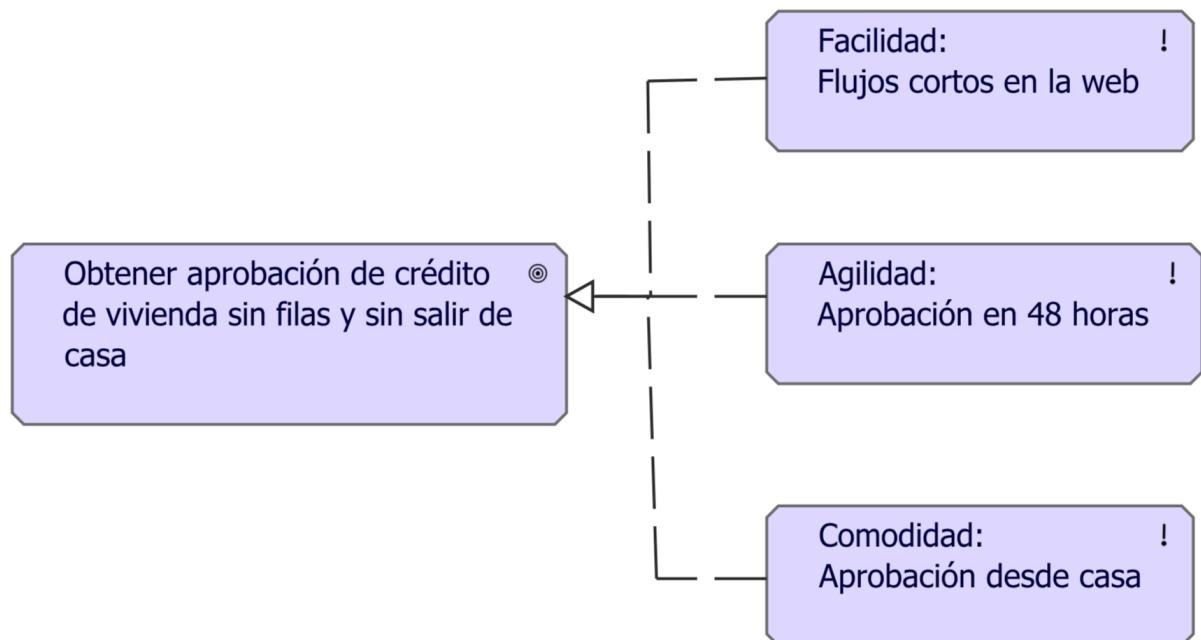


Figura: Punto de Vista de Contribución de Principios. Fuente: Autores

4.4.5. Punto de Vista de Realización de Requerimientos.

Para obtener una aprobación de vivienda digital sin salir de casa, se hace necesario tener: un asesor atado a la solicitud del cliente, luego este asesor debe tener a la mano todos los documentos, los cuales deben estar completos para su envío al departamento de análisis documental. También, debe tener una pre aprobación previa.

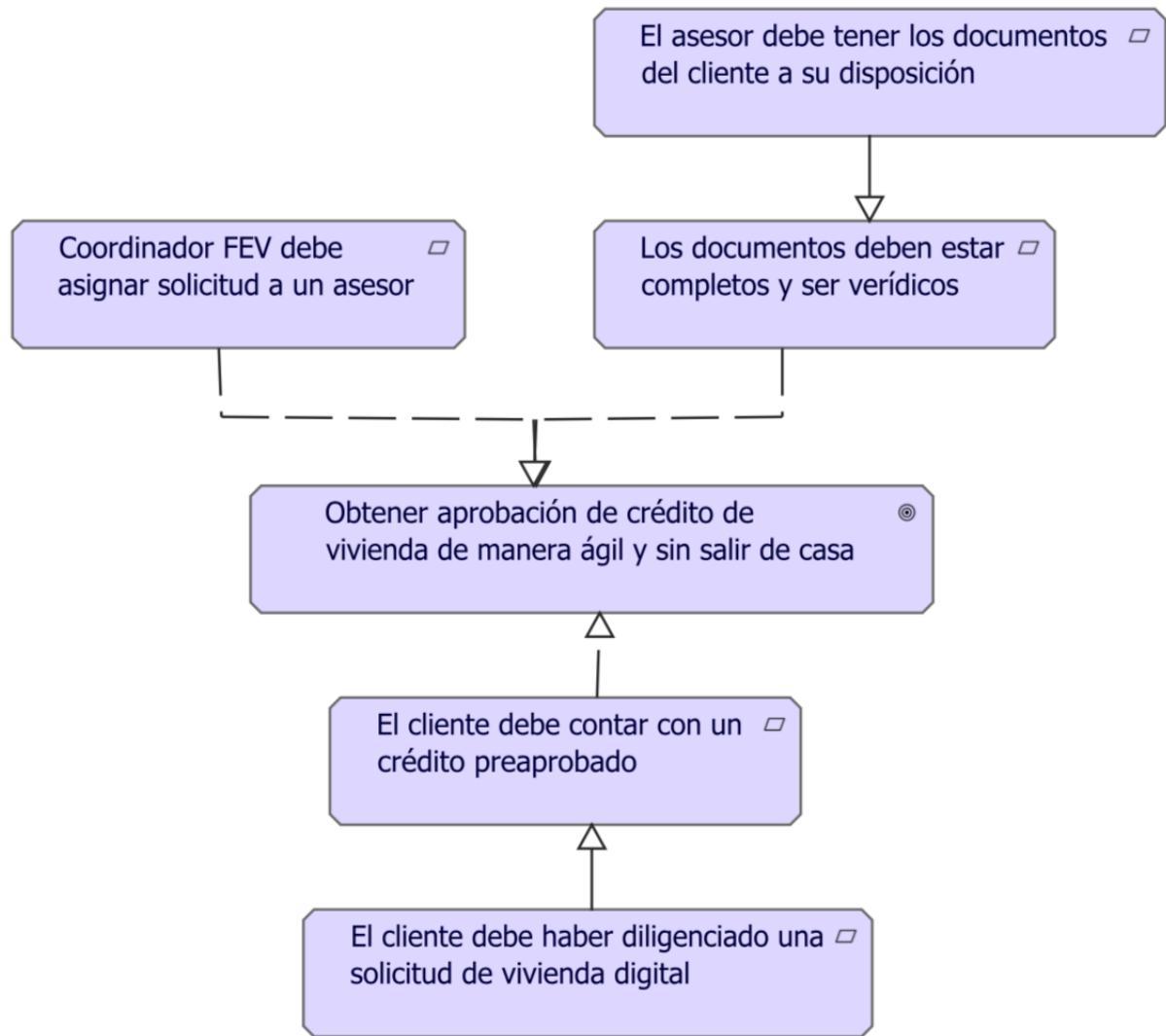


Figura: Punto de Vista de Realización de Requerimientos. Fuente: Autores

4.4.6. Punto de Vista de Motivación

La agilidad del proceso de aprobación de vivienda digital depende en gran parte de la óptima gestión del asesor FEV, el cual acompaña al cliente durante la carga de documentos después de la pre aprobación de su crédito.

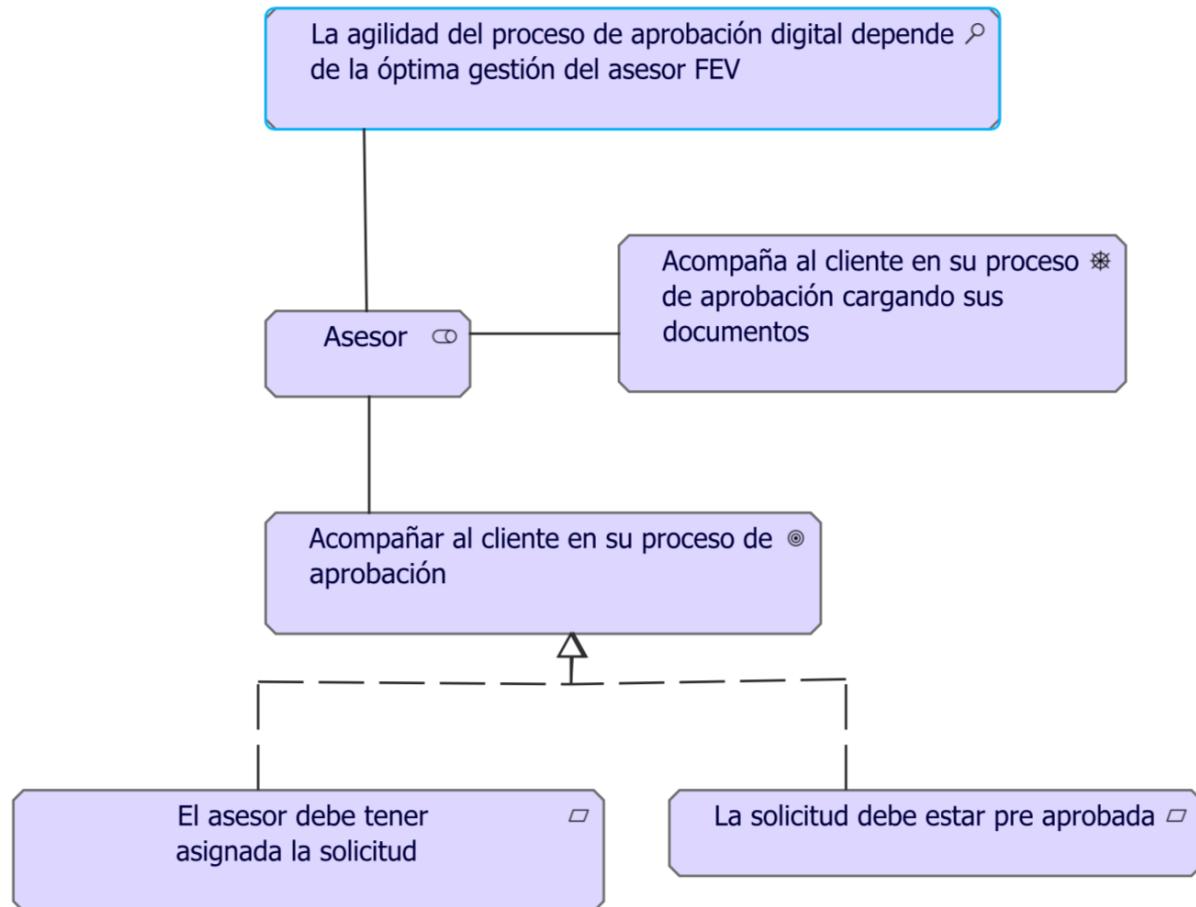


Figura: Punto de Vista de Motivación. Fuente: Autores

4.4. Diagramas de componentes arquitectura Micro Front-end.

A continuación se muestra el diagrama de componentes que describen la arquitectura micro front end propuesta:

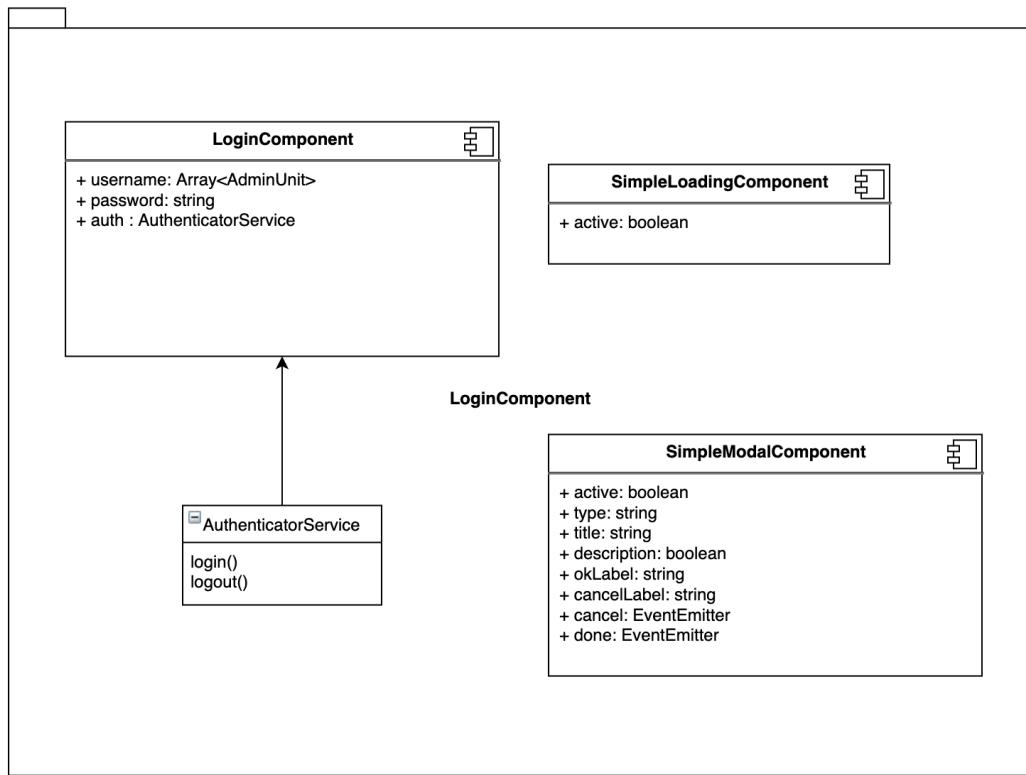


Figura: Diagrama de componentes de la página login del proyecto FEV. Fuente: Autores

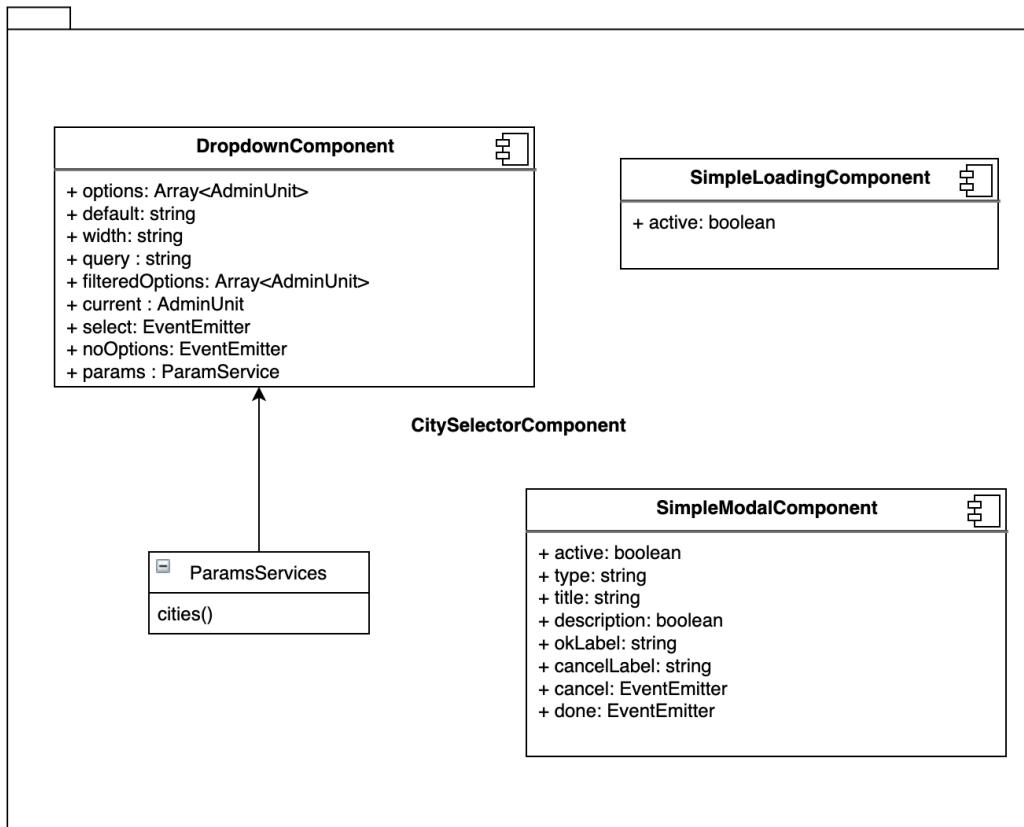


Figura: Diagrama de componentes de la pagina city-selector del proyecto FEV. Fuente: Autores

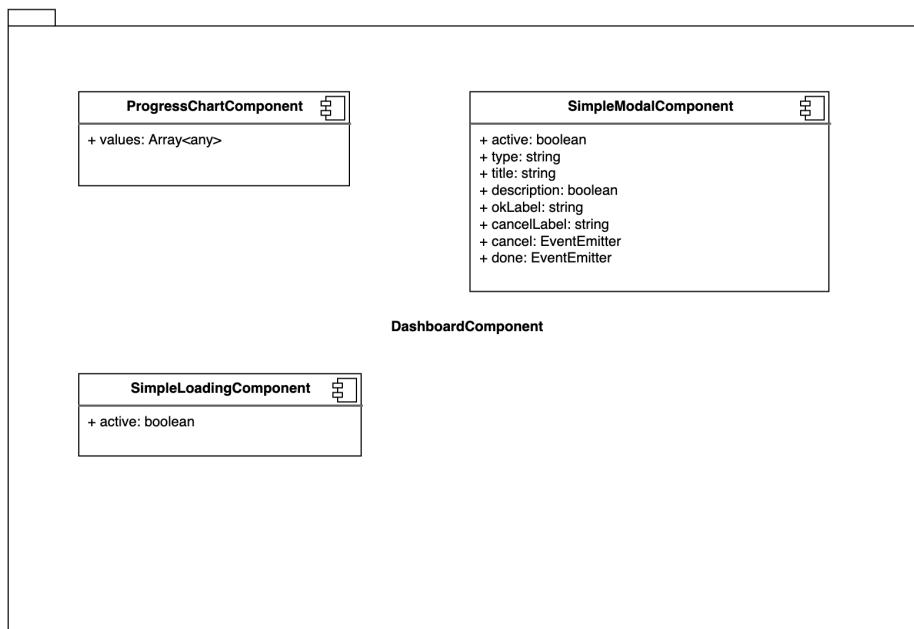


Figura: Diagrama de componentes de la página dashboard del proyecto FEV. Fuente: Autores

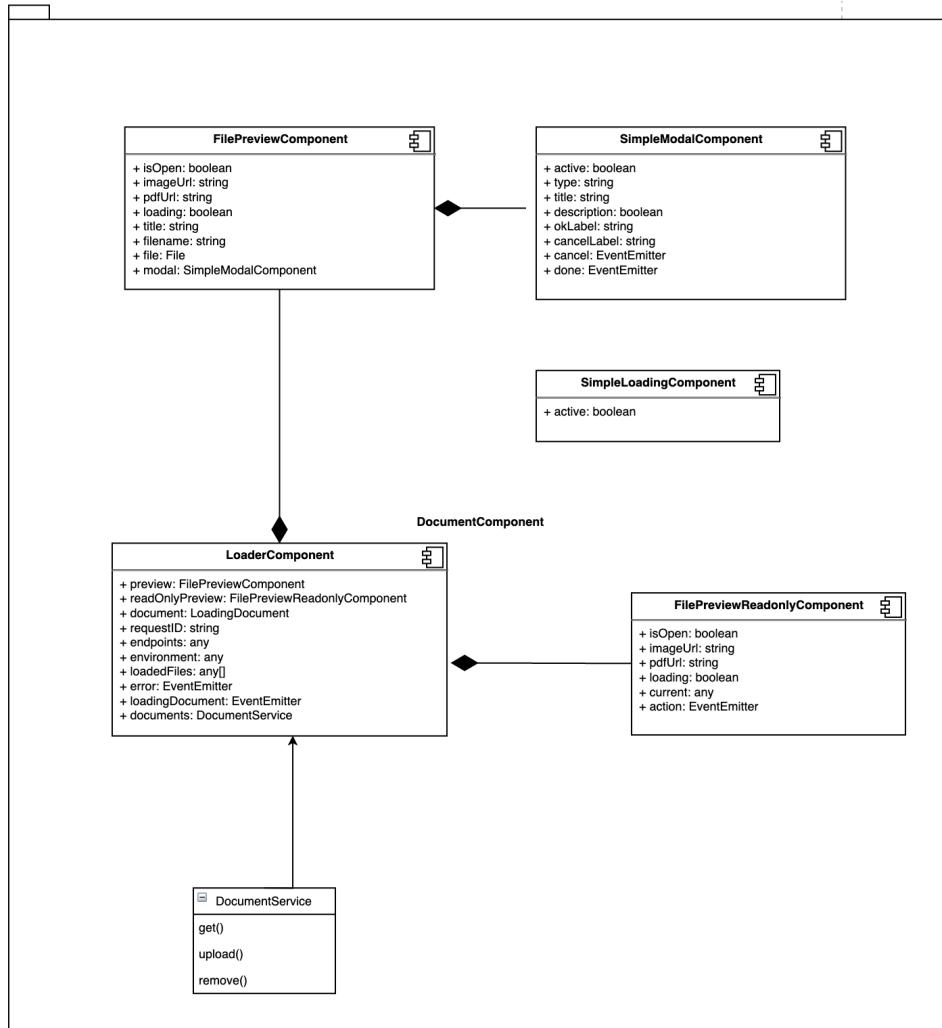


Figura: Diagrama de componentes de la página document del proyecto FEV. Fuente: Autores

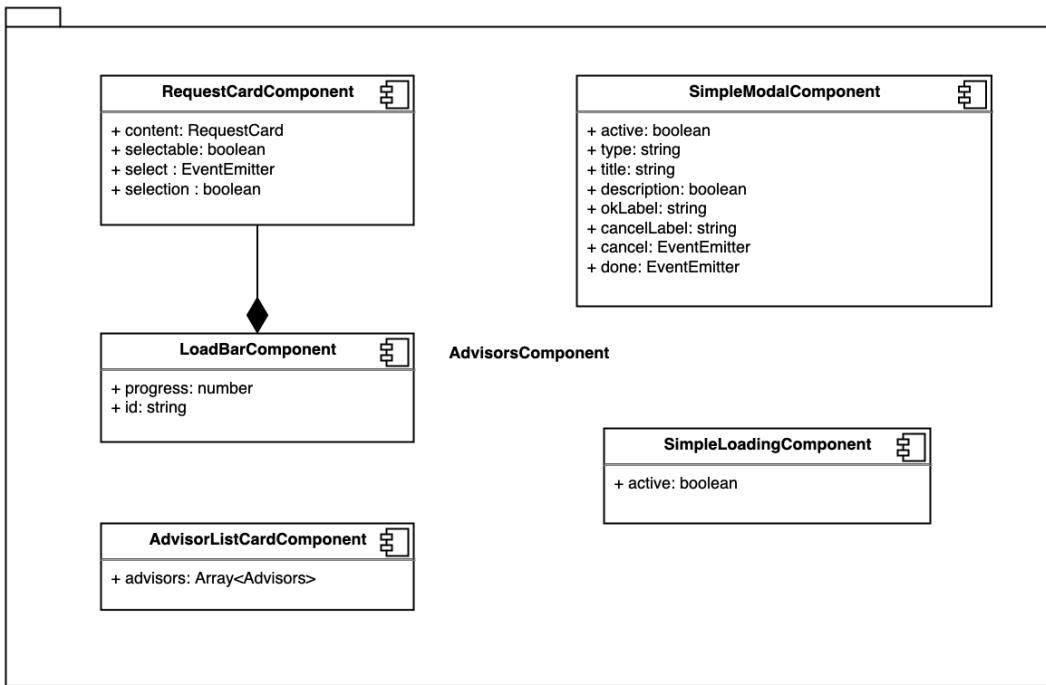


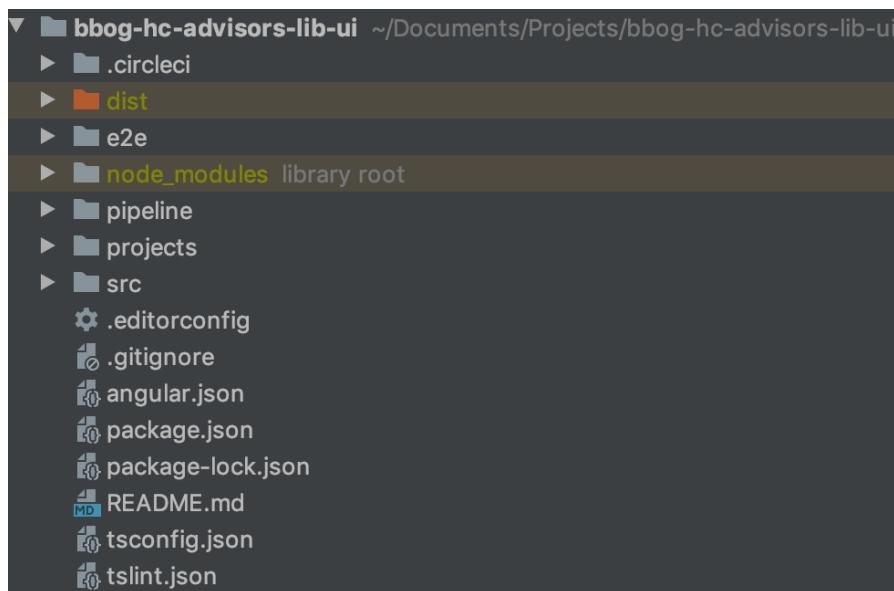
Figura: Diagrama de componentes de la página `advisors` del proyecto FEV. Fuente: Autores

CAPÍTULO 5. APLICACIÓN DE LA ARQUITECTURA LOGRADA AL PROYECTO FRONT DE LA FUERZA ESPECIALIZADA DE VIVIENDA DEL BANCO DE BOGOTÁ

Con base en la arquitectura planteada en el capítulo anterior se empieza a crear el proyecto que contiene a todos los componentes micro frontends que son utilizados posteriormente por el proyecto de Fuerza especializada de vivienda y por los demás productos que lo requieran dentro del laboratorio digital.

5.1. Descripción de la biblioteca

Este proyecto tiene como nombre ***bbog-hc-advisors-lib-ui***, siguiendo los lineamientos que la Dirección Digital del Banco para el nombramiento de los respectivos repositorios que se creen.



```
▼ └─ bbog-hc-advisors-lib-ui ~/Documents/Projects/bbog-hc-advisors-lib-ui
    ├ .circleci
    ├ dist
    ├ e2e
    └ node_modules library root
        ├ pipeline
        ├ projects
        ├ src
        │   .editorconfig
        │   .gitignore
        │   angular.json
        │   package.json
        │   package-lock.json
        │   README.md
        │   tsconfig.json
        │   tslint.json
```

Estructura del proyecto bbog-hc-advisors-li-ui. Fuente:Autores

5.2. Descripción de los componentes internos

La biblioteca se encuentra conformada por los siguientes componentes:

COMPONENTE: Loader	
Función desempeñada: Se encarga de subir los documentos al servidor S3 a través del consumo de un recurso expuesto.	
Selector: hc-loader	
ENTRADAS	SALIDAS
document: Objeto con información del documento	error: llamado a modal de error.
requestId: identificador de la solicitud de crédito	loadingDocument: ventana de carga en proceso y llamado a modal.
endpoints: ubicación de servicio backend encargado de la lógica de guardado.	
environment: variables de entorno	
loadedFiles: archivos previamente cargados.	

Imagen de funcionamiento:

The screenshot shows a user interface for uploading documents. At the top, there is a section for the 'Cédula de ciudadanía' (ID card), which includes a camera icon, a note saying 'Toma una foto o adjunta una fotocopia del documento por ambas caras.' (Take a photo or attach a copy of the document on both sides), and a green checkmark icon. Below this, there is a preview area labeled 'Documento Cargado' (File uploaded) with 'VER' (View) and 'Borrar' (Delete) buttons. To the right, there is a button to 'Agregar otro archivo' (Add another file) with a plus sign. Below this section, there is another for the 'Declaración de renta último año gravable' (Last year's tax declaration), which includes a file icon, a note saying 'La del último año con el sello del banco o descargada de la página web de la DIAN.' (The one from the last year with the bank seal or downloaded from the DIAN website), and an orange upload icon.

COMPONENTE: Simple Modal

Función desempeñada: Mostrar un modal con una información específica

Selector: hc-simple-modal

Imagen de funcionamiento:



ENTRADAS	SALIDAS
title: título del modal	close: cierre de modal
type: advertencia, éxito o error.	
description: mensaje de cuerpo de modal	
okLabel: mensaje que se muestra en el botón.	

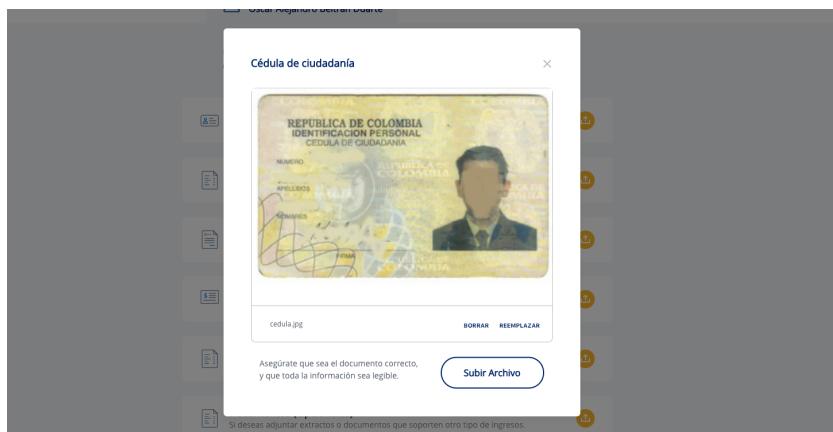
COMPONENTE: File Preview

Función desempeñada: Permite visualizar el documento para eliminarlo o cargar uno nuevo

Selector: hc-file-preview

ENTRADAS	SALIDAS
document: archivo a mostrar codificado en base64.	close: cierre de modal de visualización
requestId: identificador de la solicitud.	
endpoints: ubicación de servicio backend encargado de la lógica de guardado.	
environment: variables de entorno	

Imagen de funcionamiento:



COMPONENTE: File Preview Read Only

Función desempeñada: Permite visualizar el documento cargado sin poderlo modificar

Selector: hc-file-preview_READONLY

ENTRADAS	SALIDAS
document: archivo a mostrar codificado en base64.	close: cierre de modal de visualización

Imagen de funcionamiento:



COMPONENTE: City Selector

Función desempeñada: Permite seleccionar una ciudad para enviarla a un componente específico

Selector: hc-city-selector

Imagen de funcionamiento:

ENTRADAS	SALIDAS
endpoints: dirección de servicio que retorna listado de ciudades disponibles	ciudades disponibles para su selección.



Ingresá una ciudad para consultar solicitudes

BOGOTÁ, D.C. - BOGOTÁ D.C.- CUND.

COMPONENTE: Document Card

Función desempeñada: Muestra la información de los documentos que se han cargado y los que son necesarios para la solicitud

Selector: hc-request-card

ENTRADAS	SALIDAS
----------	---------

request: objeto que contiene la información a mostrar de la solicitud.	select: evento generado al dar clic sobre el panel.
selectable: booleano que indica si se puede cliquear o no el panel.	

Imagen de funcionamiento:

Oscar Alejandro Beltran Duarte CC. 1026577876	Monto solicitado \$ 25,000,000	Tipo de producto Crédito Directo	1
Email owen942009@gmail.com	Celular 3115450896	ID Mantiz 109744	
1/6 Documentos	13 días en el proceso		
			

COMPONENTE: Progress Chart

Función desempeñada: Muestra el progreso del cargue de los documentos de una solicitud

Selector: hc-progress-chart

ENTRADAS	SALIDAS

values: valores segmentados por días para realizar los cálculos de renderización del gráfico.

Imagen de funcionamiento:

Días en el proceso



COMPONENTE: Load-Bar

Función desempeñada: Muestra una barra de carga de la cantidad de documentos subidos en la solicitud.

Selector: hc-load-bar

ENTRADAS	SALIDAS
progress: cantidad del numerador y cantidad del denominador para el cálculo de renderización del gráfico.	

Imagen de funcionamiento:



5.1. Despliegue continuo

Adicionalmente, se define un pipeline de despliegue automático dirigido por la plataforma de integración continua llamada Circle CI, para ello se crean los archivos de .circle.yml, publish.sh y pipeline.sh, que además de ejecutar las pruebas unitarias, revisar la cobertura de código y demás atributos de calidad que son

reportados en sonar, realiza la publicación en el repositorio de artefactos dispuesto para el Banco de Bogotá llamado JFrog.

```

1 > CURRENT_VERSION=$(npm run version --silent)
2 echo $CURRENT_VERSION
3 SUFFIX=$(echo ${CIRCLE_BRANCH} | cut -f 1 -d "/" | cut -f 1 -d "_" | cut -f 1 -d "-" | cut -f 1 -d " ")
4 echo "SUFFIX: $SUFFIX"
5 if [ "${SUFFIX}" != "" ] && [ "${SUFFIX}" != "master" ]
6 then
7     git config --global user.email circleci@circleci
8     git config --global user.name CircleCI
9     git reset --hard
10    npm version "${CURRENT_VERSION}-${SUFFIX}"
11    npm run version --silent
12    echo "Aqui termina prerelease"
13 fi

```

Imagen de archivo de alistamiento de publicación de biblioteca. Fuente:Autores



Imagen de estructura de archivo de despliegue continuo. Fuente:Autores

```

1 >#!/usr/bin/env bash
2 cd $(dirname $0)...
3 set -ex
4 CURRENT_VERSION=$(npm run version --silent)
5 rm -rf ~/.npmrc | true
6 set +x
7 curl -u "${ARTIFACTORY_DEPLOY_USER}:${ARTIFACTORY_DEPLOY_PASSWORD}" "https://bbogdigital.jfrog.io/bbogdigital/api/npm/auth" >> ~/.npmrc
8 set -x
9 npm config set registry https://bbogdigital.jfrog.io/bbogdigital/api/npm/npm-bbta/
10 PUBLISHED_VERSION=$(npm info "@npm-bbta/bbog-hc-advisors-lib@$CURRENT_VERSION") || true
11 echo $PUBLISHED_VERSION
12 if [ "$PUBLISHED_VERSION" = "" ]; then
13     echo "Publishing the new release in JFROG..."
14     cd dist/
15     cp ./package.json .
16     npm version
17     npm publish
18 else
19     echo "Already exists the version $CURRENT_VERSION in JFROG"
20 fi
21 npm config set registry https://registry.npmjs.org/

```

Imagen de archivo de publicación de biblioteca. Fuente: Autores



Imagen de despliegue continuo de la biblioteca. Fuente: Autores

```
notice 72.0KB bbog-hc-advisors-lib/fesm5/npm-bbta-bbog-hc-advisors-lib.js
notice 51.3kB bbog-hc-advisors-lib/fesm5/npm-bbta-bbog-hc-advisors-lib.map
notice 151B bbog-hc-advisors-lib/lib/bbog-hc-advisors-lib.component.d.ts
notice 49B bbog-hc-advisors-lib/lib/bbog-hc-advisors-lib.module.d.ts
notice 436B bbog-hc-advisors-lib/lib/file-preview/readonly/file-preview-readonly.component.d.ts
notice 641B bbog-hc-advisors-lib/lib/file-preview/file-preview.component.d.ts
notice 1.2kB bbog-hc-advisors-lib/lib/loader/loader.component.d.ts
notice 1.3kB bbog-hc-advisors-lib/lib/models/documents.d.ts
notice 283B bbog-hc-advisors-lib/lib/models/encryption.d.ts
notice 234B bbog-hc-advisors-lib/lib/models/request.d.ts
notice 195B bbog-hc-advisors-lib/lib/services/crypto.service.d.ts
notice 3.3kB bbog-hc-advisors-lib/lib/services/document.service.d.ts
notice 2.8kB bbog-hc-advisors-lib/lib/services/request.service.d.ts
notice 1.2kB bbog-hc-advisors-lib/lib/services/storage.service.d.ts
notice 572B bbog-hc-advisors-lib/lib/simple-modal/simple-modal.component.d.ts
notice 366B bbog-hc-advisors-lib/npm-bbta-bbog-hc-advisors-lib.d.ts
notice 26.1kB bbog-hc-advisors-lib/npm-bbta-bbog-hc-advisors-lib.metadata.json
notice 733B bbog-hc-advisors-lib/package.json
notice 347B bbog-hc-advisors-lib/public-api.d.ts
notice 1.1kB bbog-hc-advisors-lib/README.md
notice === Tarball Details ===
notice name: bbog-hc-advisors-lib
notice version: 0.1.17
notice package size: 175.8 kB
notice unpacked size: 826.6 kB
notice shasum: 2b4c801242dde85ca276a78893830bfd201ba8bc
notice integrity: sha512-14xnF1Fg/WoCA[...]bmDv3jj+QPoMQ==
notice total files: 57
notice
+ bbog-hc-advisors-lib@0.1.17
+ npm config set registry https://registry.npmjs.org/
```

Imagen de publicación de la versión 0.1.17 de la biblioteca en JFrog. Fuente: Autores

5.1. Uso

Una vez se ha publicado la versión en el repositorio de artefactos, la biblioteca de micro frontends ya está disponible para su uso. Para tal fin es necesario ejecutar la siguiente línea de comandos en el proyecto al cual se quiere importar la biblioteca:

```
jorgeatuesta:bbog-hc-advisors-lib-ui jorgeatuesta$ npm install bbog-hc-advisors-lib@Y.Y.Y --registry https://bbogdigital.ifrog.io/bbogdigital/api/npm/npm-bbta/
```

Comando para la instalación de la biblioteca. Fuente: Autores

donde Y.Y.Y es la versión que se quiere instalar

Cuando la ejecución de la instrucción es hecha correctamente, se puede apreciar en los paquetes de node_modules que la biblioteca efectivamente esta importada. Realizada ya la importación correspondiente, el paso siguiente es la utilización de los componentes que allí se encuentran.

PARTE III. CIERRE DE LA INVESTIGACIÓN

CAPÍTULO 6. RESULTADOS Y DISCUSIÓN

Desde el inicio de esta investigación, uno de los ideales principales era optimizar el proceso de despliegue en integración continua de los componentes Front end desarrollados en el laboratorio digital del Banco de Bogotá. Esto debido a las cantidades de tiempo que estos demandan en la actualidad causado por los procesos intermedios que se realizan a lo largo de cada despliegue automático. Como ya se conoce por capítulos anteriores, el laboratorio hace uso de la herramienta CircleCI para automatizar los despliegues a los diferentes ambientes que se manejan (QA, Stage y producción), el cual realiza tres (en algunos casos cuatro) pasos para completar satisfactoriamente un despliegue: Build, donde se hace la descarga del código y se verifica que su compilación y construcción es correcta y no presenta ningún error. El segundo paso es Sonar, en el que se realizan las pruebas unitarias del proyecto front end y el tercer paso es Deploy, paso en el que se envía el proyecto a la infraestructura en la nube, el cual es el encargado de su despliegue y publicación en la Web.

En el momento en que una historia de usuario es asignada a un desarrollador, este comienza a trabajarla y una vez tiene la solución (que para él es definitiva), procede a enviarla de su rama local de Git a una nueva rama remota que lleva el mismo nombre de la rama local creada para dar solución a la historia de usuario propuesta, una vez dicho envío (Push) se hace efectivo, inicia la labor de Circle y para estos casos en los que la trama enviada no lleva como nombre “qa”, ni “stage”, ni “master”, los únicos procesos que realiza son “Build” y “Sonar”; en el mejor de los casos este proceso tarda cuatro minutos como se puede apreciar en la siguiente imagen, en la que se hace despliegue de una rama de un desarrollo de historia de usuario para un componente Front end de complejidad baja del laboratorio Digital.

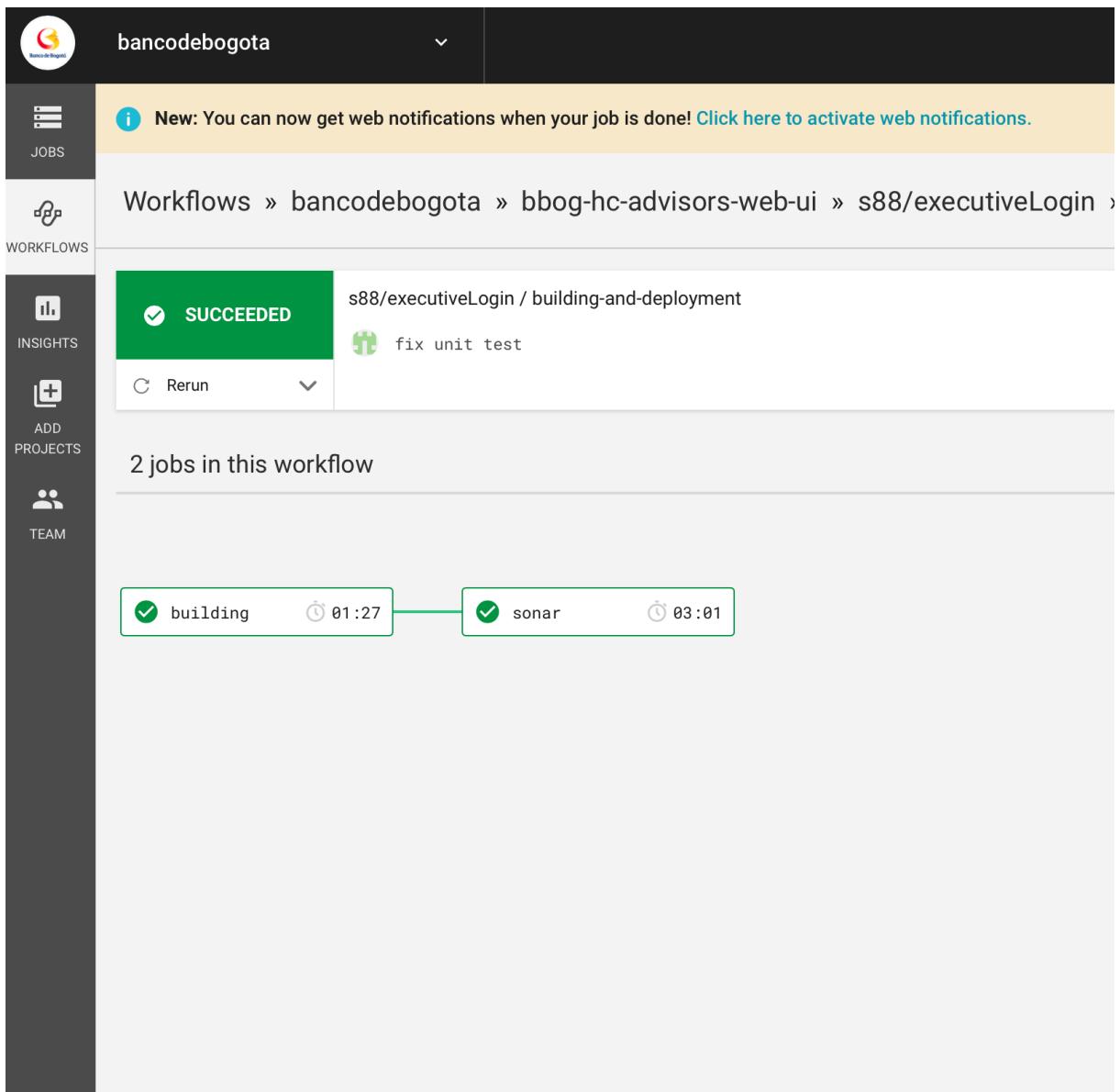


Imagen: Despliegue de una rama remota en CircleCI para un producto de complejidad baja,

Fuente: Autores

Al hacer la misma revisión pero sobre un componente de mayor complejidad y tamaño, se puede observar que el tiempo se ve drásticamente afectado:

The screenshot shows the CircleCI web interface. On the left, there's a sidebar with icons for JOBS, WORKFLOWS, INSIGHTS, ADD PROJECTS, and TEAM. The main area displays a workflow named "revertMorning" under the project "bancodebogota". A banner at the top says "New: You can now get web notifications when your job is done! Click here to activate web notifications." Below the banner, the workflow name is shown. The first job, "building", is listed as "SUCCEEDED" with a duration of "01:22". The second job, "sonar", is also listed as "SUCCEEDED" with a duration of "12:12". A button labeled "Rerun" is available for each job.

Imagen: Despliegue de una rama remota en CircleCI para un producto de complejidad alta,
Fuente: Autores

Como se puede observar el tiempo se triplica, casi catorce minutos nada más enviando los cambios de una rama local a una remota. El siguiente paso que realiza el desarrollador es crear un Pull Request (solicitud de envío de cambios a otras ramas), el cual requiere de revisión de pares para poder pasar los cambios de código realizados a la rama de QA: Una vez que los cambios son aprobados y se ha hecho efectiva la mezcla, vuelve a entrar en acción la herramienta de integración y despliegue continuo, pero esta vez si realiza un despliegue, es decir que para este escenario si realiza un tercer paso en el que envía el código a la cuenta encargada de la gestión de la infraestructura cloud del ambiente de QA.

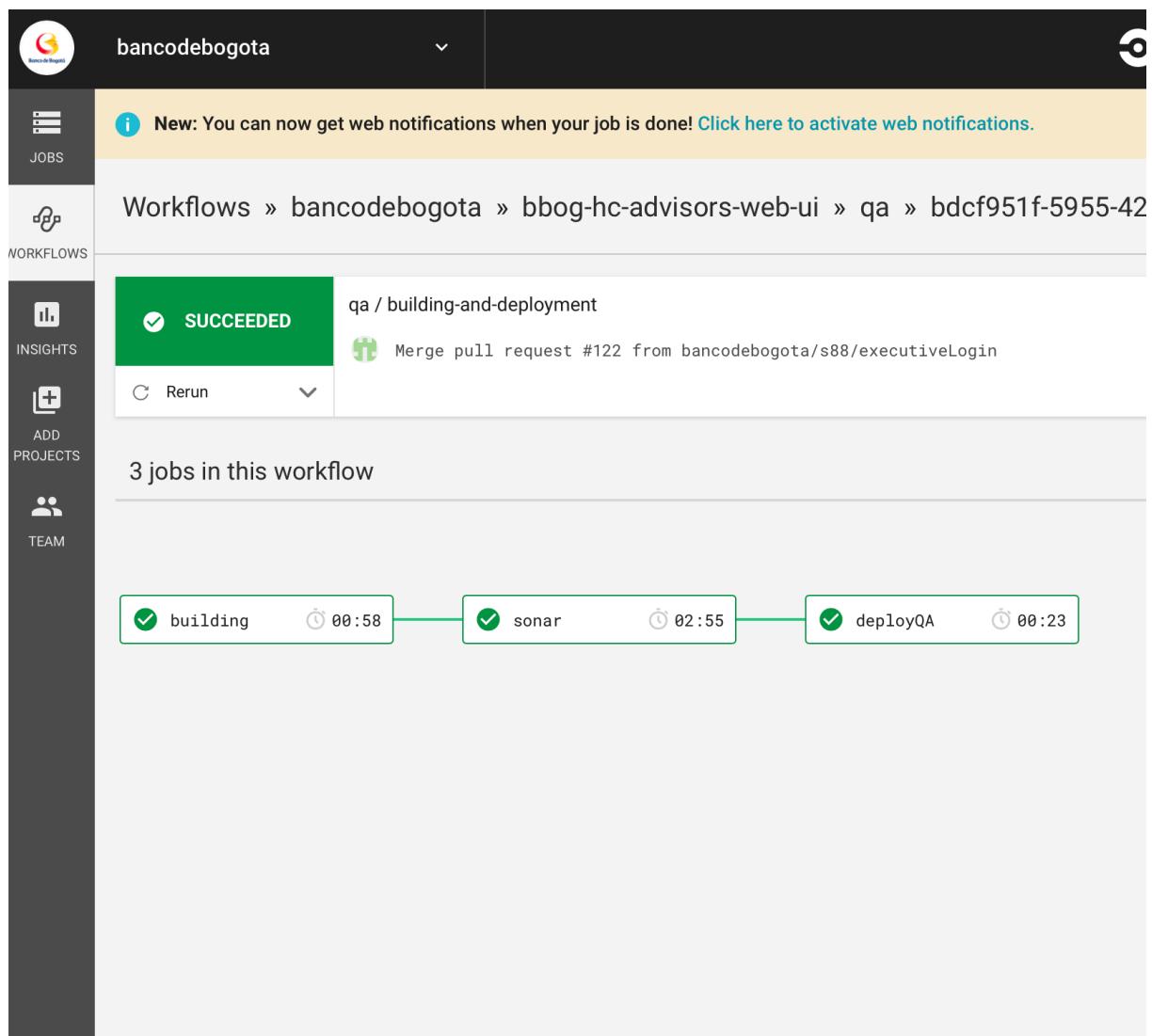


Imagen: Despliegue de la rama QA en CircleCI para un producto de complejidad baja, Fuente: Autores

Como se puede apreciar en la anterior imagen, en total el despliegue hacia el ambiente de QA, tarda aproximadamente 5 minutos (el paso de Build tarda poco debido a el uso de caché), es decir que a este punto, el desarrollador ha tenido que esperar 9 minutos en total (cuatro minutos de la rama local + cinco minutos de la rama QA). Ahora, se ilustra el despliegue de un componente de complejidad alta con el fin de observar la afectación en tiempos de despliegue.

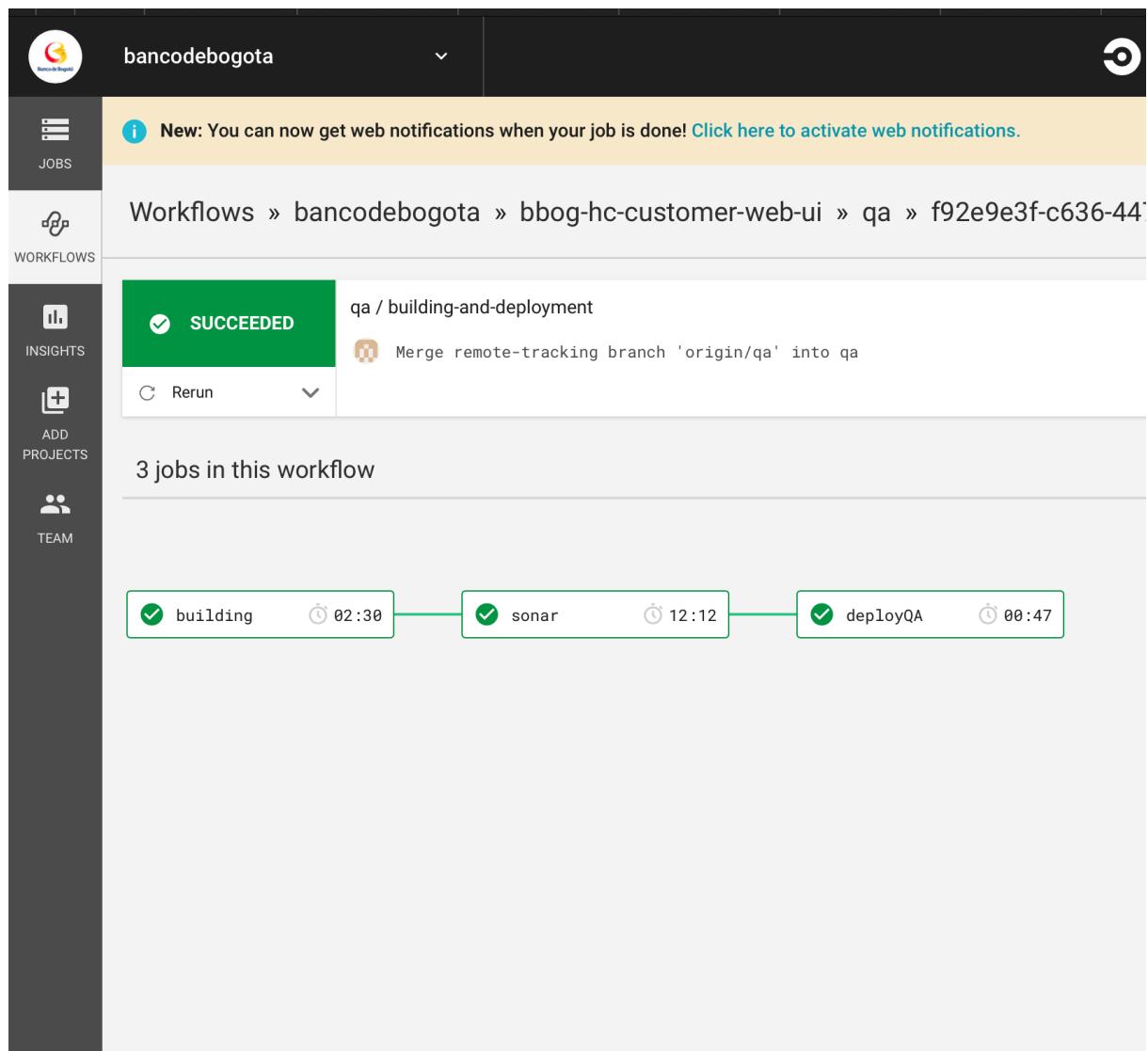


Imagen: Despliegue de la rama QA en CircleCI para un producto de complejidad alta, Fuente: Autores.

Una vez el desarrollador logra posicionar sus cambios en QA, procede a realizar pruebas funcionales y de no afectación. Una vez ve que el desarrollo funciona según lo esperado, procede a repetir el paso anterior (creación y aprobación de pull request) pero esta vez hacia el ambiente de Stage, para este escenario también se ejecutan los mismos pasos del despliegue a QA (build, sonar y deploy), pero esta vez los pasos están configurados para hacer el despliegue sobre la infraestructura usada para stage.

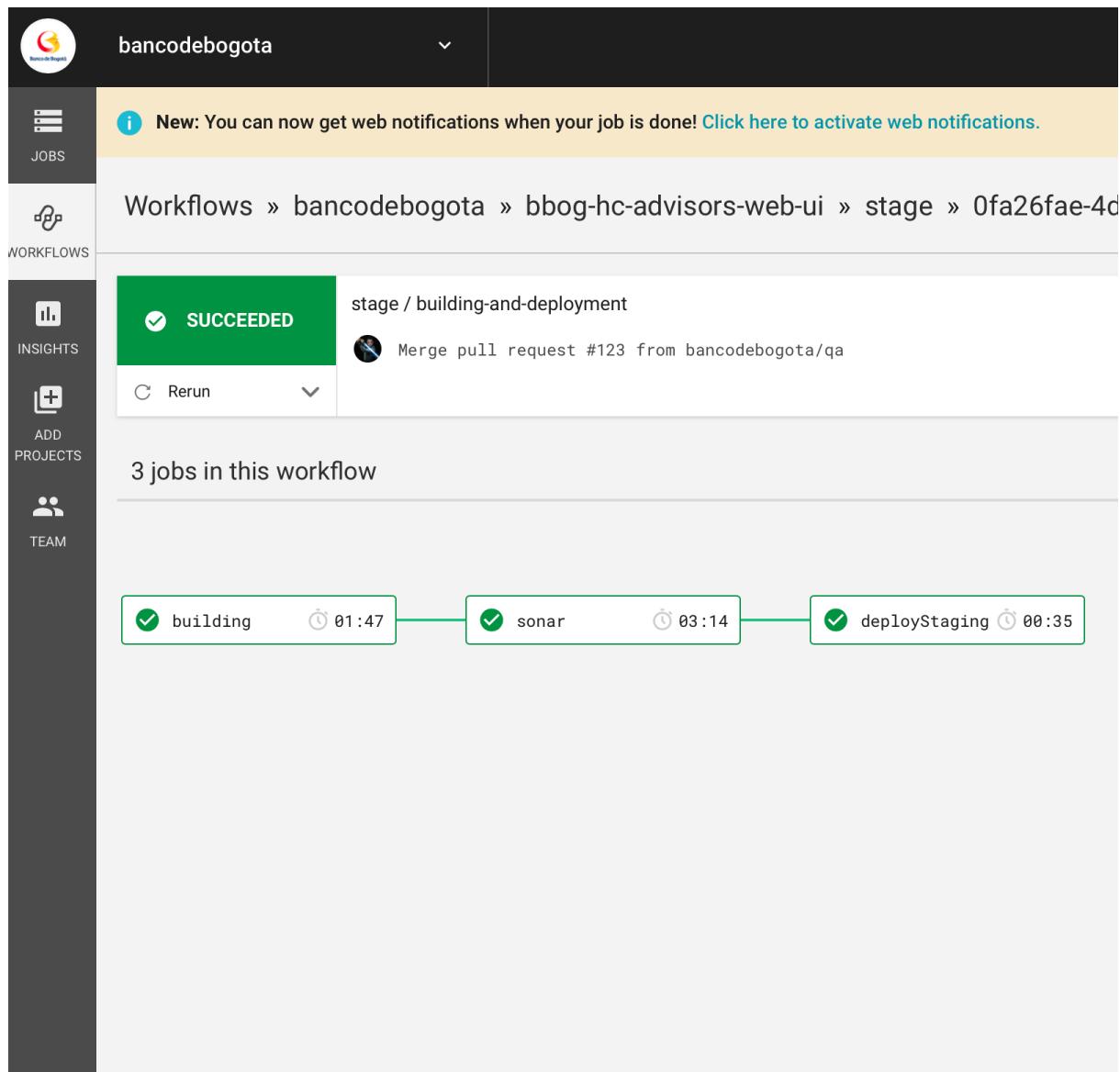


Imagen: Despliegue de la rama STAGE en CircleCI para un producto de complejidad baja,
Fuente: Autores.

Para el escenario de despliegue hacia Stage en un componente de baja complejidad del laboratorio se encuentra un tiempo de aproximadamente seis minutos, para un total de 15 minutos (cuatro de la rama local + cinco de la rama QA + seis de la rama stage). Ahora se realiza el mismo ejercicio de comparación de tiempos con respecto a un componente de complejidad alta.

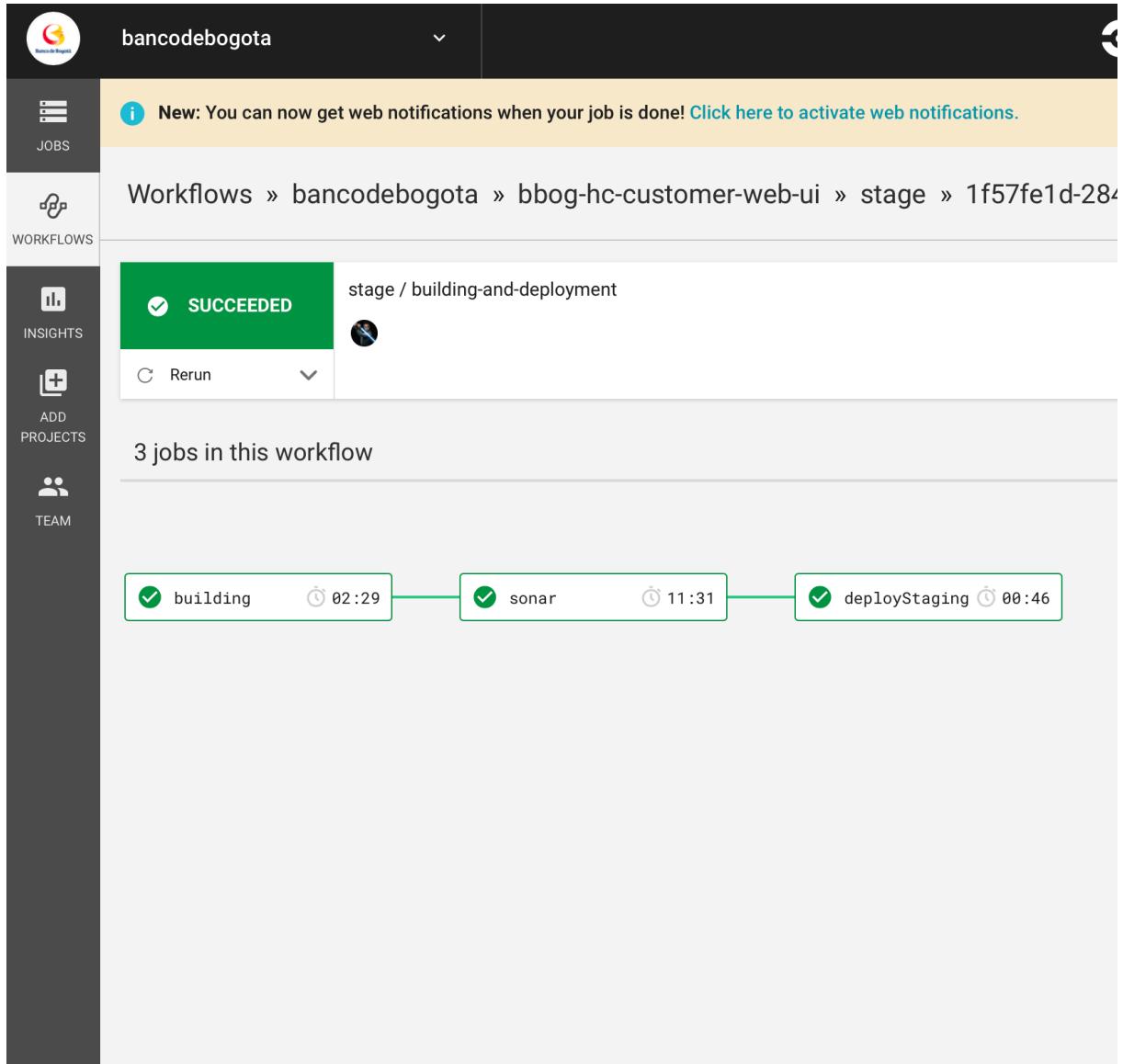


Imagen: Despliegue de la rama STAGE en CircleCI para un producto de complejidad alta,
Fuente: Autores.

Si se suman los tiempos a esta altura del despliegue de un componente de alta complejidad, lleva un total aproximado de 44 minutos. El último paso de la cadena de despliegues continuos hasta producción comienza con la creación del pull request hacia la rama MASTER. A continuación se muestran los pasos realizados para un componente de baja complejidad.

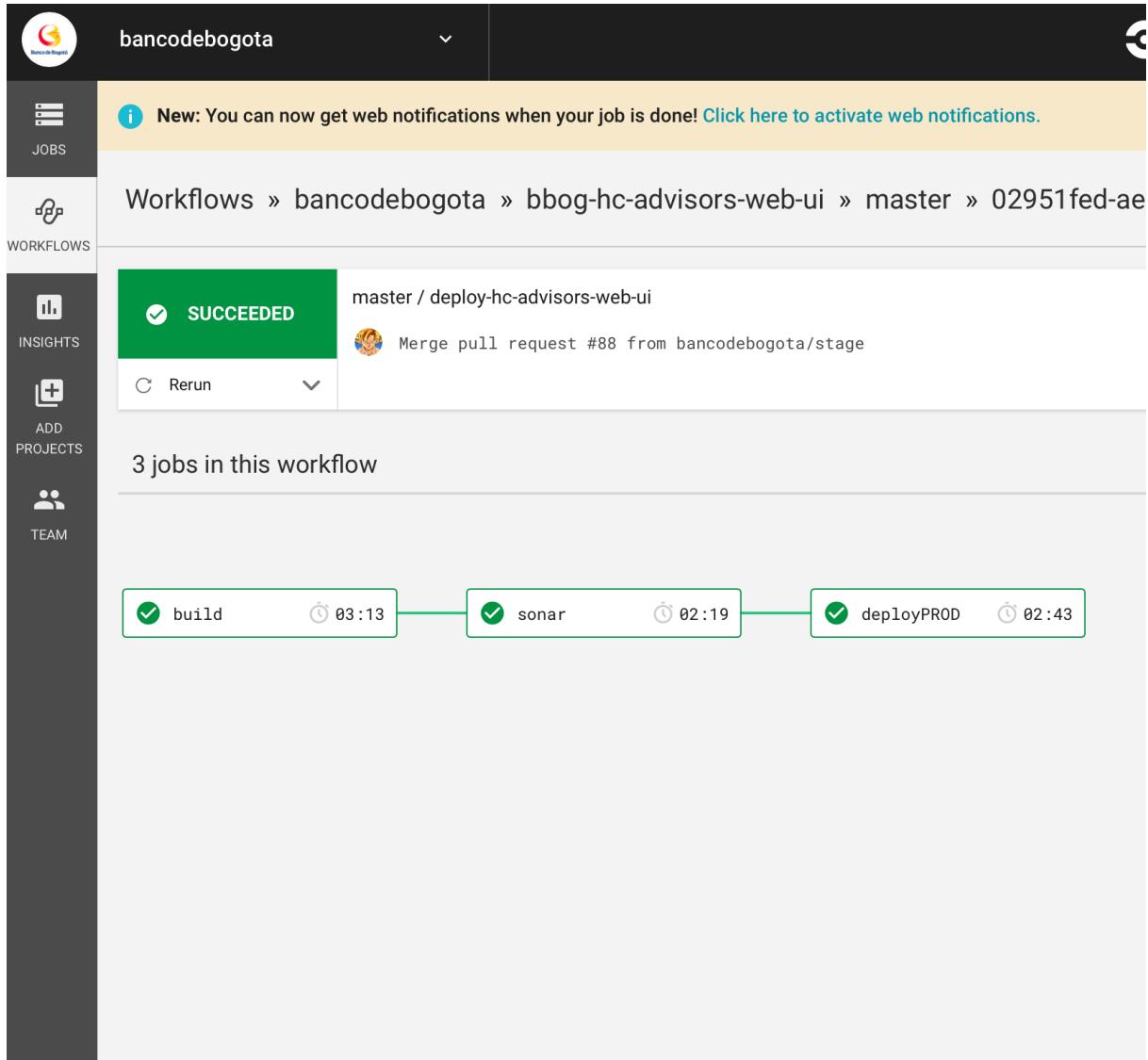


Imagen: Despliegue de la rama MASTER en CircleCI para un producto de complejidad baja,
Fuente: Autores.

Al final de la cadena de despliegue de un componente de baja complejidad del laboratorio el tiempo empleado es de aproximadamente 23 minutos sumando todos los despliegues desde la rama remota hasta Master. Ahora se realiza la comparación con respecto a un componente de alta complejidad.

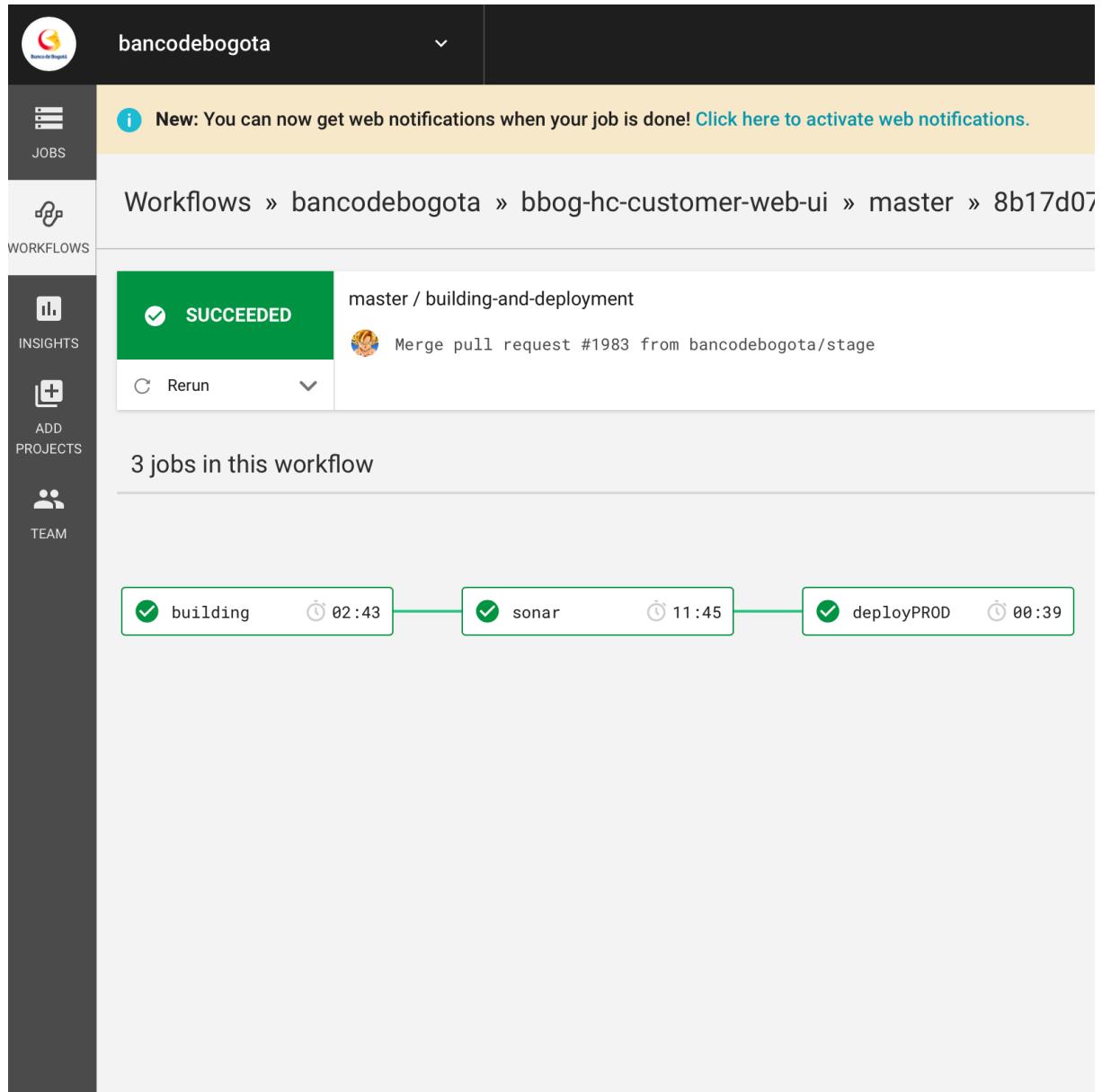


Imagen: Despliegue de la rama MASTER en CircleCI para un producto de complejidad alta,
Fuente: Autores.

El total de tiempo empleado para un componente de complejidad alta del laboratorio es aproximadamente de una hora (sumando los tiempos desde la rama remota hasta la rama master). Entonces, en conclusión en la actualidad, los productos front end tardan entre 23 minutos y una hora, y esto en el mejor de los casos, suponiendo que el desarrollo no tiene ningún tipo de error funcional y que no se ven afectadas las pruebas de regresión, además que todas las pruebas unitarias ejecutadas desde CircleCI se ejecutan de manera exitosa, debido a que si alguno de estos factores se ve afectado, los tiempos podrían llegar a triplicarse.

Para desarrollar esta investigación se decide tomar como estudio el proyecto front end encargado de la gestión de Asesores de la Fuerza Especializada de Vivienda, el cual funciona como producto auxiliar al producto de crédito de Vivienda digital del Banco de Bogotá, este proyecto front end se considera de complejidad baja debido a la baja cantidad de páginas y componentes con los que cuenta. La razón para elegir este proyecto como caso de estudio radica en que su despliegue y pruebas unitarias tardan menos tiempo, además debido a su menor tamaño comparado con otros proyectos hace que se reduzca la posibilidad de afectación de componentes al momento de intervenir algún fragmento de código.

Después de todo el proceso de investigación y desarrollo de la misma, se logra migrar diversos componentes (expuestos en el capítulo 5) como librería (publicados en JFrog), los cuales pueden ser utilizados por otros proyectos debido a su ligereza y bajo acoplamiento. Se obtiene una respuesta favorable en cuanto a la funcionalidad de los componentes migrados, ya que con el uso correcto de variables de entrada y emisión de eventos a los elementos padres, la aplicación no se ve afectada en lo más mínimo, pero a su vez aporta de manera importante al objetivo de centralizar todos aquellos elementos reutilizables con el fin de evitar doble trabajo y reprocesos, es decir que dos equipos no se preocupen por desarrollar la misma funcionalidad sino que por el contrario si una funcionalidad ya existe pueda ser fácilmente utilizada por cualquier proyecto sin tener problemas de compatibilidad y desempeño.

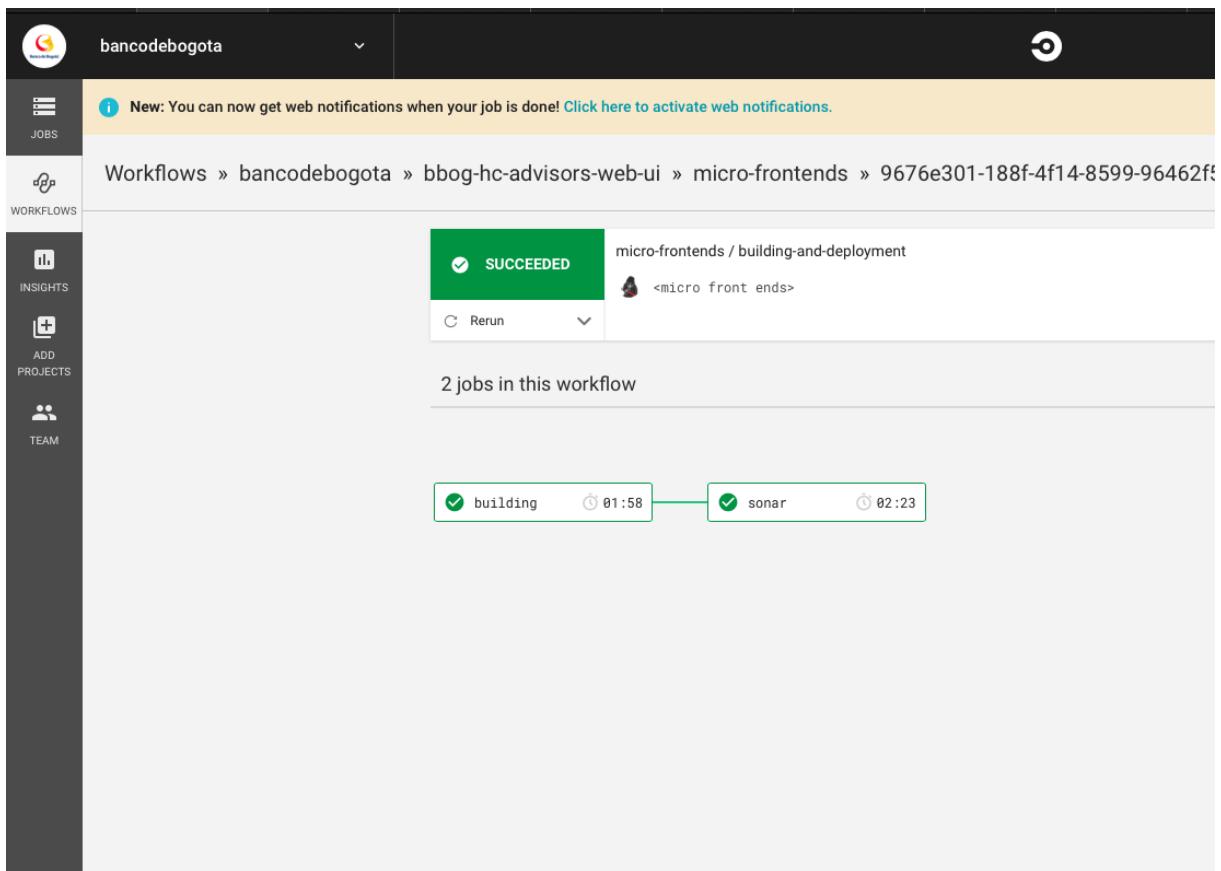


Imagen: Despliegue de la rama REMOTA en CircleCI para un producto de complejidad baja,
Fuente: Autores.

Una vez desplegada la aplicación a una rama remota, se puede ver que el tiempo reduce, quizás no a la mitad, pero esto se debe a que la complejidad de la aplicación no es la mayor y por tanto el cambio no es drástico, pero esto da la prueba que si reduce tiempos y que al aplicarse en proyectos de complejidad mayor, el tiempo de despliegue si se ve drásticamente afectado.

Cuando se logra centralizar componentes reutilizables, su funcionamiento correcto se certifica rápidamente debido que es implementado por distintos productos de software los cuales lo ponen a prueba desde su desarrollo (verificando que tiene las variables de entrada necesarias para que su funcionamiento esté desacoplado a cualquier lógica de negocio) hasta su puesta en producción donde sus pruebas funcionales deben ser ejecutadas satisfactoriamente.

CAPÍTULO 7. CONCLUSIONES

7.1 Verificación, contraste y evaluación de los objetivos

El objetivo principal que se presentó al inicio de este trabajo de investigación era el de desarrollar un modelo de arquitectura el cual utiliza los principios planteados en los microservicios aplicándolos en el front-end, utilizando como línea base la aplicación de la fuerza especializada de vivienda del Banco de Bogota. Al lograr dicho modelo el cual se expone en el capítulo 4, se puede demostrar que es posible elaborar un proyecto que es una composición de pequeñas partes de otro y que puede mantenerse por equipos diferentes que se encargaran de crear nuevos micro front-ends o dar soporte a los actuales.

Lo que se busca con la última afirmación es la reducción de la complejidad del proyecto, esto es, que la aplicación de fuerza especializada de vivienda tenga muchos menos componentes dentro de su estructura base, haciendo que sea mucho más específica en sus funcionalidades.

Dentro del desarrollo del proyecto se realizó la prueba de concepto del modelo arquitectural planteado, realizando un prototipo implementando alguna de las funcionalidades que se encontraban descritas dentro de él, esto con el fin de verificar el impacto que genera el uso de los micro front-ends tanto en la velocidad de despliegue como en los atributos de calidad que se proponen dentro del trabajo de la Dirección Digital del Banco de Bogotá, además de reducir los tiempos y el esfuerzo que se realiza al ejecutar las pruebas de regresión.

Al tener desplegados los micro front-ends que componen la fuerza especializada de vivienda en un ambiente en el cual todos los equipos de desarrollo de la Dirección Digital pueden acceder, no únicamente al equipo de vivienda, es posible realizar ampliar el espectro de la mantenibilidad y la escalabilidad de los mismos, ya que muchas más personas ven las funcionalidades que el componente le ofrece y pueden realizar mejoras e incluso aumentar la calidad de las pruebas unitarias que se realizan.

7.2 Síntesis del modelo propuesto

El modelo de arquitectura front-end basado en micro front-end para la fuerza especializada de vivienda del Banco de Bogotá está compuesto de 5 paquetes que

hacen parte de la base del proyecto. Dichos paquetes integran en su interior los diferentes componentes micro front-end que están desplegados en un ambiente diferente en el que se encuentra el mismo. Estos componentes internos se comunican tanto con el paquete en el que se están utilizando como con ellos mismos a través de variables de entrada y de salida. A su vez se pueden utilizar los componentes en otros proyectos siempre y cuando se haga el correcto uso de la especificación de sus entradas y se utilicen adecuadamente las salidas que estos proporcionan.

7.3 Aportes originales

Dentro de los aportes que se hacen al realizar el modelo de arquitectura y el prototipo inicial de implementación se pueden encontrar el uso de interfaces estandarizadas para el funcionamiento de componentes que únicamente muestran contenido HTML y también para componentes que consumen servicios REST que están desplegados en diferentes ambientes.

7.4 Trabajos o publicaciones derivadas

Del desarrollo del modelo de arquitectura y de la construcción del prototipo se deriva el proyecto inicial en el cual ya se encuentran algunos de los micro front-ends que utiliza el proyecto de fuerza especializada de vivienda. Este proyecto cumple con los estándares de desarrollo y de despliegue continuo que se proponen dentro de la Dirección Digital , además de estar desplegados en un ambiente de uso exclusivo de los integrantes del laboratorio digital, los cuales también pueden acceder al repositorio donde está alojado el código fuente.

CAPÍTULO 8. PROSPECTIVA DEL TRABAJO DE GRADO

8.1 Líneas de investigación futuras

El desarrollo que se propone este trabajo fue el de realizar los micro front-ends haciendo uso del framework de Angular en su versión 6.0. Una línea de investigación en la que se puede trabajar es la de el desarrollo de micro front-ends

en otros frameworks como son LitElement , Stencil o Vue.js y ver su complejidad tanto al momento de desarrollarlos, en su publicación y en su acoplamiento en proyectos que no están hechos en el mismo framework.

8.2 Trabajos de investigación futuros

En este trabajo de investigación es posible continuar en la implementación de los demás micro front-ends que se proponen en el modelo arquitectural tomando como base el proyecto creado en el prototipo.

Otro trabajo que se propone realizar en el futuro, teniendo en cuenta que ya se está realizando la ejecución del pipeline de despliegue el cual contiene los jobs que corren las pruebas unitarias , evalúan el código a través de sonarCloud y finalmente realizan el despliegue de la biblioteca en JFrog, todo de manera automática, es evaluar la posibilidad de realizar los componentes micro front-end también de manera automática, esto basado en unos estándares específicos y definiendo un alcance en particular. Además también es posible plantear dentro del pipeline de despliegue un modelo de automatización que ejecute las pruebas de aceptación, pruebas de regresión y pruebas de carga que se planteen para cada uno de los micro front-ends.

REFERENCIAS

- [1] M. Trigas, *Metodología SCRUM*. Cataluña, 2018, p. 36.
- [2] P. Clements et al., *Documenting Software Architectures*.
- [3] Kavourgias, C. (2015) What's the Difference Between the Front-End and Back-End? [Mensaje en un blog]. Recuperado de <http://blog.digitaltutors.com/whats-difference-front-end-backend/>.
- [4] E. Gamma, *Design patterns*. 1994.
- [5] R. Martin, *Clean code*. Upper Saddle River [etc.]: Prentice Hall, 2009.
- [6] K. Lange, "THE LITTLE BOOK ON REST SERVICES", <https://www.kennethlange.com>, 2016. [Online]. Available: <https://www.kennethlange.com/books/The-Little-Book-on-REST-Services.pdf>. [Consultada: 18- Oct- 2019].
- [7] C. Richardson, "Microservices From Design to Deployment", <https://www.nginx.com>, 2016. [Online]. Available: <https://www.nginx.com/blog/microservices-from-design-to-deployment-ebook-nginx/>. [Consultado: 18- Oct- 2019].
https://ungrid.unal.edu.co:8888/img/Microservices_Designing_Deploying.pdf
- [8] ISO. (2015). ISO 25000. Recuperado el 17 de 10 de 2019, de <http://iso25000.com/index.php/normas-iso-25000/iso-25010>
- [9] I. Arachchi, Continuous Integration and Continuous Delivery Pipeline Automation for Agile Software Project Management. Moratuwa, 2018.
- [10] I. Jovanovic, "Introduction to micro-frontends", oreilly.com, 2018. [Online]. Available: <https://conferences.oreilly.com/fluent/fl-ca/public/schedule/detail/64834>. [Consultada: 24- Sep- 2019].
- [11] G. Steinacker, "Scaling with Microservices and Vertical Decomposition", otto, 2019. [Online]. Available: https://www.otto.de/jobs/technology/techblog/artikel/scaling-with-microservices-and-vertical-decomposition_2014-07-29.php. [Consultada: 26- Sep- 2019].
- [12] L. mezzalira, "Adopting a Micro-frontends architecture", medium.com, 2019. [Online]. Available: <https://medium.com/dazn-tech/adopting-a-micro-frontends-architecture-e283e6a3c4f3>. [Consultada: 25- Sep- 2019].

- [13] Joel, "Things You Should Never Do, Part I", *jelonsoftware.com*, 2000. [Online]. Available: <https://www.jelonsoftware.com/2000/04/06/things-you-should-never-do-part-i/>. [Consultado: 26- Sep- 2019].
- [14] A. Basalo, "Manual de AngularJS", *https://programacion.net*, 2016. [Online]. Available: https://programacion.net/files/code/20161021121055_manualdeangularjs-manualcompleto.pdf. [Consultada: 16- Oct- 2019].
- [15] Página web de reactjs. <https://reactjs.org/>. [ReactJS home page].
- [16] Página web de vueJs. <https://vuejs.org> [vuejs home page].
- [17] R. Saquete, "Qué son y en qué consisten los Web Components", *https://www.humanlevel.com*, 2019. [Online]. Available: <https://www.humanlevel.com/articulos/desarrollo-web/que-son-y-en-que-consisten-los-web-components.html>. [Consultada: 20- Oct- 2019].
- [18] M. Geers, Micro Front Ends in Action. Manning Publications. 2019.
- [19] Sułkowski, T., 2018. *How To Build A Library For Angular Apps?*. [online] Available at: <https://medium.com/@tomsu/how-to-build-a-library-for-angular-apps-4f9b38b0ed11> [Consultada 14 April 2020].

