ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Universidad Politécnica de Cartagena

Trabajo Fin de Estudios

Desarrollo, mejora e implementación de plataforma de análisis de datos para la automatización de servicios de planificación financiera.

AUTOR: Ginés Molina Abril
DIRECTOR: Alejandro Martínez Sala

Octubre / 2019

Autor	Ginés Molina Abril
E-mail del autor	gines.moli@gmail.com
Director	Alejando Martínez Sala
E-mail del director	alejandros.martinez@upct.es
Título del TFE	Desarrollo, mejora e implementación de plataforma de análisis de datos para la automatización de servicios de planificación financiera.
Resumen	Existe un problema importante en el mundo actual en el que se une la mala situación financiera producto de fórmulas tradicionales de financiamiento y de gestión de fondos y la escasa alfabetización en términos financieros con millones de personas afectadas. Este problema tiene que ver con eventos inesperados como el pago de una factura importante o la pérdida de trabajo del sustentador principal de la familia, el poco ahorro para la jubilación y las insuficientes pensiones estatales y por último los clientes inversores que no tienen acceso a asesoramiento financiero. En este contexto nace y toma fuerza una nueva industria financiera a través de empresas FinTech o Financial Technology que aplican tecnología para crear productos financieros innovadores y ofrecen respuestas a los usuarios en multitud de productos y servicios financieros. El objetivo prioritario de estas compañías es la creación de herramientas y servicios financieros fáciles de contratar, entender y con un precio estandarizado que permite el acceso a un mayor número de personas y empresas. La digitalización del sector financiero llega para quedarse con estos nuevos modelos y con un contexto favorable con la utilización de tecnologías disruptivas como Big Data. Se propone el desarrollo y mejora del producto de una empresa FinTech ubicada en Edimburgo que plantea una plataforma de análisis de datos automatizando el proceso de planificación financiera a través de los datos bancarios que ofrecen las empresas bancarias que actúan como clientes y requieren un sistema holístico, realista y optimizado basado en datos. En concreto, la plataforma evalúa la situación financiera de los usuarios, proyecta los objetivos financieros, garantiza una óptima planificación fiscal y proporciona una API de nivel empresarial para permitir crear soluciones de planificación financiera personalizadas. Además, dota de acceso instantáneo a Open Banking, plataformas de inversión y sistemas de back-office de asesores.
Titulación	Grado en Ingeniería Telemática.
Departamento	Tecnología de la Información y las Comunicaciones.
Fecha de presentación	Octubre - 2019

Índice:

Índice de figuras:

Capítulo 1 Introducción, contexto y motivación

1.1 Entorno de realización

Para mí es un orgullo poder finalizar esta etapa en mi aprendizaje ofreciendo una solución a todo lo que se plantea como banca tradicional y ofreciendo una visión renovada que, gracias a la tecnología, nos lleva hacia un sector clave mucho más controlado, seguro, transparente y ofreciendo servicios financieros con el menor impacto negativo posible que afecta tan directamente a la vida de las personas.

El entorno de realización del proyecto se engloba a través del programa "Erasmus+ Prácticas" en la compañía Inbest. La defensa de las soluciones planteadas y su posterior justificación me ha llevado a ganar experiencia de debate, análisis y desarrollo desde un proyecto empresarial internacional y con tecnologías a la vanguardia en un sector en plena evolución.

El reto planteado es el de realizar una aplicación que contribuya a la plataforma integral de la compañía realizando un análisis de los datos financieros de los clientes que aporte valor a entidades bancarias para detectar tendencias y características según cada tipo de cliente.

1.2 Problema asociado - Nacimiento de las Fintech

El estallido de la crisis bancaria es el detonante principal en la reducción del número de entidades de crédito que operan en España y ha supuesto para muchas familias un problema importante al que hacer frente. Las medidas tomadas por el gobierno español para paliar los efectos de la crisis bancaria reflejan la notable reducción de estas redes de sucursales bancarias tradicionales [1] y sus empleados [2], haciéndonos cuestionar el marco regulador del sistema financiero que propició cambios desfavorables para así intentar evitar efectos tan negativos en el futuro.

Gracias al esfuerzo de saneamiento del sector en España cercano a los 300.000 millones de euros y a la presión de los organismos reguladores se ha conseguido que los bancos españoles hayan incrementado de forma considerable tanto el volumen como la calidad de sus recursos propios y aunque los niveles actuales son inferiores a los del año 2008, se trata de un camino optimista. El principal objetivo en la actualidad para las entidades bancarias es el de reducir costes para hacer frente al entorno actual de baja rentabilidad.

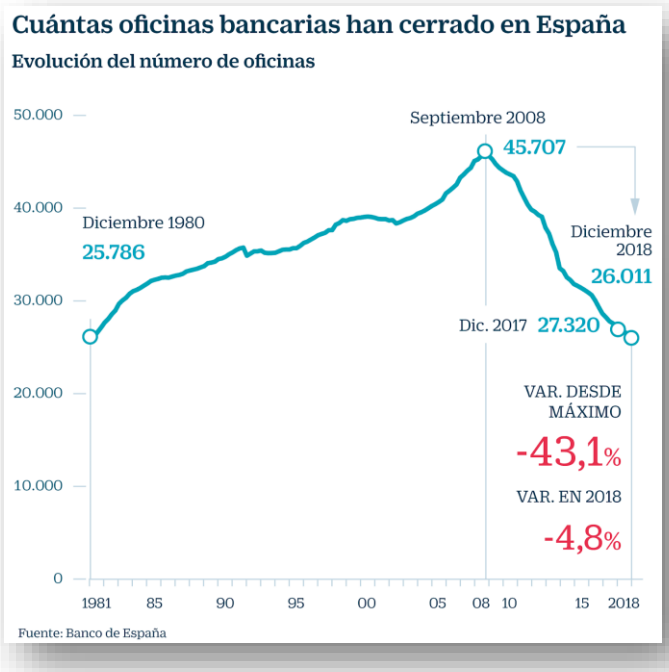


Figura 1 - Oficinas bancarias cerradas en España

Figura 2 - Empleados de la banca en España

La tecnología en este contexto aparece para generar nuevos competidores más ágiles y con una excelente respuesta a la gran mayoría de necesidades financieras con un coste inferior a los bancos tradicionales ya que no tienen que rentabilizar costosas estructuras de personal y de red como con estas nuevas propuestas. Se trata de un reto importante de adaptación al que se enfrentan grandes bancos europeos del sector ya que, según el Fondo Monetario Internacional, un tercio podría enfrentarse a problemas estructurales graves en los próximos años [3]. Invertir en tecnología y aliarse con empresas FinTech ha sido la estrategia por llevar a cabo asumiendo que los avances tecnológicos actuales están cambiando el modelo de hacer banca e incluso el marco monetario.

Según un informe sobre las perspectivas del sector a nivel global, los principales problemas de negocio Big data en el sector bancario son: con el 77% la mejora de la experiencia de usuario, con un 66% el marketing dirigido a una clientela determinada, con un 52% la mejora en gestión de riesgos, y con un 41% la eficiencia de procesos y la reducción de costes [4]. Otro de los informes realizado a banqueros de todo el mundo refleja en un 76% que el motor comercial para la adopción de técnicas Big data en el sector se debe centrar en la mejora del compromiso, retención y lealtad de los clientes. Un 71% afirma que, para aumentar sus ingresos, necesitan entender mejor a sus clientes y que las técnicas Big data les ayudarán a ello. Y, por último, un 55% afirma que tener una visión en tiempo real de los datos proporciona una ventaja competitiva significativa y creen que los procedimientos de procesamiento de datos por lotes son ineficaces [5].

1.3 Análisis, ecosistema e impacto en el sector

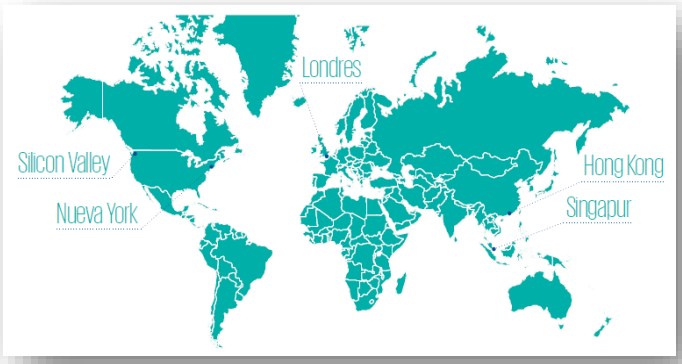


Figura 3 - Principales centros Fintech mundiales

En la actualidad existen más de 15.000 empresas tecnológicas-financieras en todo el mundo, con EE. UU. y Reino Unido a la cabeza de este universo Fintech. Los principales centros Fintech mundiales están representados en la figura a continuación.

En España se afronta la oportunidad generada por el Brexit y se ha convertido en un mercado interesante para la industria Fintech gracias a que la mayor parte de la población está bancarizada y el índice de penetración del móvil es elevado. En la actualidad es el 6º país del mundo por número de Fintech, con más de 100 millones de euros en facturación y dando trabajo a más de 3.500 trabajadores [6]. El 55% de las empresas encuestadas lo forman menos de 15 empleados y el 52% adoptan un modelo B2B orientadas a proveer servicios que satisfagan necesidades de otras compañías. Las principales verticales Fintech están reflejadas en el diagrama de al lado.

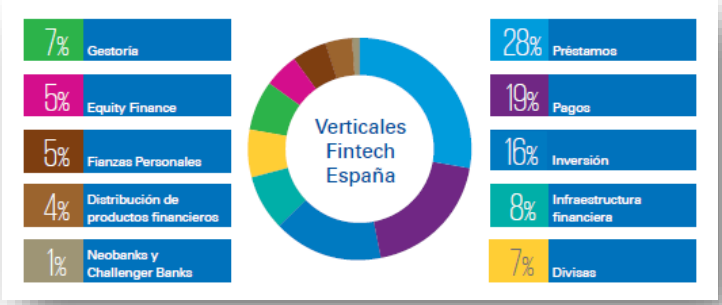


Figura 4 - Principales verticales Fintech en España

El modelo de negocio de las Fintech cuenta con las siguientes características:

- Sus productos financieros son online. Recurren a la comodidad y ahorra al cliente desplazamientos a la sucursal tradicional. Se utilizan eficientemente los canales digitales y aprovechan el grado de penetración de los dispositivos inteligentes en nuestras vidas para tener de forma rápida y eficiente cualquier información.
- Las soluciones planteadas son habilitadas de manera rápida usando tecnologías disruptivas, estructuras flexibles y metodologías ágiles.
- Persiguen un enfoque orientado a la personalización e inmediatez de las soluciones añadiendo valor a los servicios financieros actuales.
- Fomentan el compromiso y fidelidad del cliente favoreciendo la inclusión financiera de grupos de población no bancarizados y democratizan el acceso a una mayor cantidad de servicios financieros transparentes.

- Por su estructura, reducen de modo eficiente el nivel de costes de los servicios actuales, actuando a lo largo de toda la cadena de valor.

Los principales retos a los que se enfrentan las compañías Fintech son:

- Conseguir el apoyo de los inversores, aumentar la confianza y reputación.
- Establecer acuerdos con la banca cuando se percibe un valor añadido recíproco y fomentar el conocimiento sobre las Fintech.
- Potenciar un entorno jurídico común y lograr apoyo institucional.
- Y, por último, atraer el talento y ejercer una labor pedagógica sobre sus modelos para potenciar la cultura financiera en la sociedad.

1.4 Modelo de negocio de Inbest - Evolución

Figura 5. Inbest Logo

Inbest fue fundada en 2014, con la misión de desarrollar una plataforma digital para ayudar a los consumidores a entender sus propias finanzas para que puedan tomar mejores decisiones financieras.

Desde su inicio, han colaborado con investigadores académicos interesados en el análisis de datos y diseño de interfaz de usuario analizando modelos que incorporan indicadores socioeconómicos para analizar el bienestar económico de los hogares. Más tarde se desarrolla un sistema para ofrecer asesoramiento financiero automático capaz de adaptarse a los cambios de las condiciones del mercado o del usuario y así ayudar a la toma de decisiones conforme a las tendencias relevantes.

Las plataformas existentes estaban especializadas en un solo aspecto de la planificación financiera y no proporcionaban un enfoque integrado hacia la gestión financiera personal. La compañía lleva a cabo un modelo B2B y es encargada de ayudar a las instituciones financieras a crear soluciones digitales para distribuir seguros de protección, inversiones y productos de pensiones en los canales de venta directos y recomendados. Los proveedores financieros también usan Inbest para llegar a los hogares en momentos clave e hitos de la vida en los hogares, lo que permite aumentar las ventas de productos a los clientes existentes además de ver mejorada su productividad y así poder dedicar más tiempo a servicios de valor agregado que requieren empatía.

1.5 Servicios en la actualidad – Módulos & API

Inbest cuenta con un módulo de situación financiera en la que se ofrece la oportunidad al cliente de saber si debe hacer algún cambio en sus finanzas personales y cuánto se pueden permitir ahorrar para alcanzar sus objetivos. También cuenta con un módulo de planificación financiera que calcula las métricas de riesgo, rentabilidad y probabilidad de alcanzar los objetivos definidos. Otro de los módulos ofrece soporte sobre la planificación fiscal para realizar aportaciones y retiradas de capital con sentido y de acuerdo con el contexto financiero del cliente. Y, por último, cuenta con un conjunto de módulos de cálculo impulsados a través de una API con la que permiten a los clientes construir sus propias soluciones de ahorro,
inversión y planificación financiera.

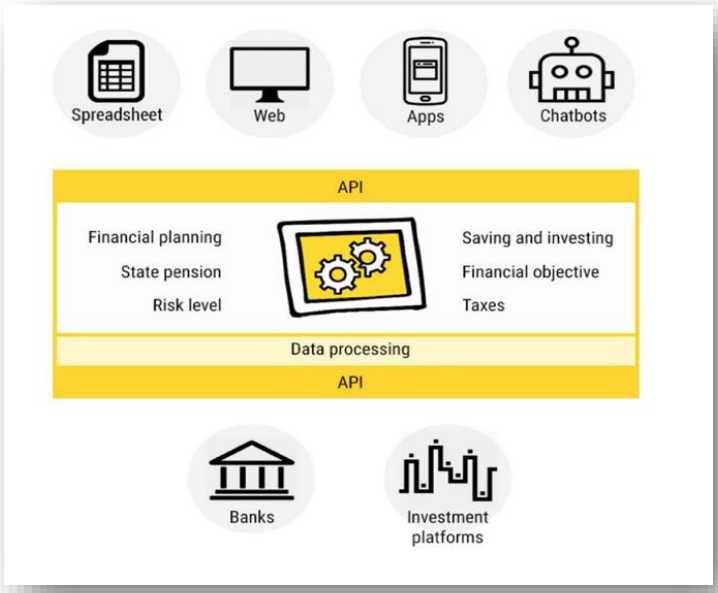


Figura 6 - Estructura API Inbest

1.6 Análisis de requerimientos - Especificaciones generales

Teniendo en cuenta los retos de implementación propuestos por la compañía, en este capítulo se analiza, determina y establece un marco tecnológico global que perseguir durante toda la duración y realización del proyecto.

1.6.1 DEFINICIÓN GENERAL DEL PROYECTO

El proyecto consiste en el análisis y desarrollo de una plataforma optimizada que utiliza grandes volúmenes de datos bancarios para el análisis, filtrado y visualización de los datos de los clientes que pueda dar respuesta a los servicios que las compañías bancarias demandan para extraer un mayor número de conclusiones acertadas sobre los productos bancarios de sus clientes.

De una manera sencilla, cualquier analista financiero podrá segmentar por tipos de cliente y sus parámetros y así podrá visualizar de una manera rápida cualquier aspecto importante en sus productos bancarios. Además, podrá descubrir tendencias según el sesgo o tipo de cliente y proteger sus capacidades de ahorro creando productos personalizados para aquellos en los que se descubra cierta tendencia negativa.

1.6.2 REQUERIMIENTOS

Se requiere el desarrollo de la aplicación en un entorno web multiplataforma y responsive en el que sea fácilmente accesible desde cualquier dispositivo para explotación de grandes volúmenes de datos alojados en bases de datos no relacionales NoSQL. La plataforma debe ser rápida con el tratamiento y visualización de los datos ofreciendo una buena experiencia de usuario: sin cortes. Además, deberá ser flexible en cuanto al número de usuarios que lo estén utilizando simultáneamente y escalable para facilitar el desarrollo de nuevas gráficas y servicios de automatización y monitorización en el futuro.

La plataforma será encargada de visualizar estadísticas obteniendo los datos desde una fuente, limpiarlos para su óptimo tratamiento, realizar los cálculos oportunos, almacenarlos y construir los objetos gráficos que arroja este análisis en conjuntos determinados. Además, contará con un módulo de filtrado que actualizará el conjunto de datos en tiempo real y ofrecerá sus gráficas asociadas a distintos conceptos financieros que se pueden determinar a partir de un modelo de datos a gran escala.

La empresa cuenta con una plataforma de análisis de datos en el lenguaje R y está desarrollada con ayuda de la librería Shiny. Este proyecto trata de captar la idea de esta plataforma y mejorarla con tecnologías centradas en la utilización de Python siendo acorde a la línea de migración que ha marcado la compañía en sus nuevos productos. Por lo que se deberá contar con un modelo de datos que deberá ser analizado y tratado contando con partes reutilizables de la infraestructura que también deberá ser analizada para lograr su máximo desempeño.

Con todo ello se deberá conceptualizar una pila tecnológica unificada e integral que trabaje bien con grandes cantidades de datos sin hacer uso de los mejores servidores de procesamiento de datos, con una infraestructura web de costes asumibles. Asumiendo el

coste de la utilización de tecnologías tan novedosas y en continuo desarrollo, se deberá atender a la documentación de estas tecnologías para lograr su mayor aprovechamiento y alejar el desarrollo de posibles estancamientos productivos en la utilización de procesos, funciones o servicios previsiblemente obsoletos. Los pilares esenciales en los que se va a sostener el proyecto será la versatilidad, la flexibilidad y el máximo desempeño, por lo que toda mejora deberá ir encaminada en satisfacer estos requerimientos.

1.7 Retos de implementación – Desarrollo por etapas

A continuación, se pasa a detallar la planificación inicial y los retos de implementación a desarrollar organizado en una escala temporal por etapas.

Etapa 1 - Análisis y plan de acción:

La compañía debe definir, guiar y poner en conocimiento su marco de productos y servicios actual. Teniendo en cuenta el stack tecnológico, se deberá investigar las ventajas y desventajas de la utilización de estas tecnologías asociadas y encontrar alternativas tecnológicas viables. A continuación, se debatirá qué soluciones implementar según se acerquen más a los objetivos en términos de tiempos de carga y experiencia de usuario. El aprendizaje y evaluación de estas tecnologías proporcionará un análisis exhaustivo de las opciones viables de integración para este proyecto y su alcance, por lo que en este momento queda fijado un plan de acción para llevar a cabo las mejoras.

Etapa 2 - Desarrollo del entorno tecnológico básico:

Ejecución del plan de acción establecido por la compañía a través de los cambios y mejoras en las diferentes capas de producto: entorno virtual, librerías adicionales de análisis de datos, librerías de extensión de componentes o middlewares que realizan balanceo de carga, infraestructura web y componentes adicionales de integración continua. En este punto se deberá ser capaz de ejecutar un entorno tecnológico escalable y básico con estas herramientas para mejorar en etapas posteriores.

Etapa 3 - Diseño de la Interfaz de Usuario y filtrado de datos:

Una vez obtenida la primera versión de la plataforma de análisis de datos, se deberá centrar los esfuerzos en el diseño de una interfaz de usuario (UI) que ofrezca los servicios de la empresa de acuerdo con el análisis de datos realizado y así entender como obtener el máximo valor de los datos evaluados que se convierten en servicios y productos. En esta etapa toma especialmente importancia el filtrado de datos a realizar y el primer diseño del dashboard.

Etapa 4 - Mejora de la Experiencia de usuario y visualización de datos:

En el momento en el que queda definida la interfaz de usuario (UI) y el diseño inicial del dashboard se deberá ejecutar las tareas asociadas a la visualización de las gráficas y a la mejora de la experiencia de usuario (UX) mediante el uso de frameworks o librerías que ofrecen simplicidad y practicidad en la muestra de datos. En este punto tomará especialmente importancia la mejora visual de los datos a través de las gráficas.

Etapa 5 - Mejora de tiempos de carga:

A continuación, se deberá centrar los esfuerzos en la mejora de los tiempos de carga mediante el análisis y prueba de los algoritmos y funciones que intervienen en el análisis de los datos a escala masiva. La compañía en este momento establece las funciones y variables con prioridades y debe asesorar sobre lo que afecta a cada una de ellas al negocio para lograr un conjunto funcional y práctico. Se deberá evaluar el modelo de datos nuevamente en profundidad y analizar cómo afecta en los tiempos de carga. Se asume un nuevo análisis y desarrollo de algunas partes afectadas por este nuevo modelo.

Etapa 6 - Pruebas definitivas, resultados y documentación:

En último lugar, se deberá realizar las pruebas de testeo definitivas y llevar a cabo las labores de documentación de las funciones y servicios desarrollados que serán utilizadas en la presentación del producto ante clientes bancarios.

Capítulo 2 Arquitectura, metodología y modelo de datos

2.1 Arquitectura del sistema - Interconexión de módulos

En este apartado se pasa a describir y analizar la composición de los módulos, su interconexión, su disposición jerárquica, una breve descripción de su funcionamiento y por último la justificación de uso de cada una de estas herramientas o tecnologías.

2.1.1 PROCEDIMIENTOS DE DESARROLLO Y EJECUCIÓN

Al tratarse de una aplicación web multiplataforma y responsive, su ejecución es posible desde cualquier dispositivo electrónico con conexión a internet y con un navegador actualizado a su última versión. La aplicación en su entorno natural no será accesible por personal no autorizado, por lo que cuenta con control de acceso con contraseña.

La ejecución es básica: abrir el navegador, acceder a la dirección web, ingresar las credenciales y una vez ingresado en la plataforma, seleccionar los filtros y visualizar la información según el tipo de características del producto/clientes que se desea analizar.

El lenguaje principal de desarrollo es Python y se utilizan librerías para simplificar el trabajo como Dash, Pandas o Numpy. Se utiliza Docker para la generación de contenedores, Pipenv como herramienta de gestión de entornos virtuales y se utiliza Github para el control de versiones. Por otro lado, también se utiliza Jenkins y Kubernetes para aplicar Integración Continua.

Por cada entorno de desarrollo la aplicación cuenta con dos dominios propios para cada uno de los dos entornos de desarrollo diferenciados: sitanalytics.inbest.ai y analytics.inbest.ai que apuntan a las ramas de desarrollo y producción respectivamente. Por un lado, el entorno de desarrollo sirve para controlar y probar los cambios que posteriormente serán actualizados en el entorno de producción. El entorno SIT (System Integration Test) ejecuta los procesos de prueba de errores al más alto nivel para comprobar que cada módulo es integrado con éxito para conseguir los resultados esperados.

Por otro lado, en cada rama o entorno de desarrollo se pueden diferenciar dos etapas o sub-entornos: un entorno de aplicación de metodologías ágiles de desarrollo y control de versiones (GIT) y otro en el que se llevan a cabo tareas de Integración Continua o tareas más asociada al comúnmente denominado rol de DevOps. En cada uno de estos sub-entornos Docker cobra un papel fundamental para conseguir flexibilidad en cada contexto

independientemente de los equipos en los que el desarrollador ha trabajado y en los servidores independientemente de las características y particularidades de cada uno.

El procedimiento habitual de desarrollo se trata de la implementación de las mejoras con realimentación en el propio desarrollo, testeo y depuración de errores, la generación de una nueva versión estable que contiene nuevas características y por último la actualización de la rama de producción. Así, se ha podido configurar y probar la adaptación de las herramientas al entorno, así como su rendimiento y desempeño.

El proceso es continuo y se trata de desarrollar tareas y mejoras por orden de prioridad. En este caso, resulta clave evaluar continuamente el rendimiento global del entorno en el que a veces se ha determinado dar otro tipo de soluciones para encontrar mejoras que vayan en el camino requerido, aunque suponga volver a rediseñar parte del entorno o el modelo de datos, como se comentará en capítulos posteriores.

2.1.2 INTERCONEXIÓN DE MÓDULOS

La disposición jerárquica del diagrama a continuación ofrece la diferenciación entre tecnologías involucradas en metodologías ágiles de soluciones de desarrollo y las tecnologías asociadas en el proceso de testeo, depuración y puesta en marcha en los servidores.

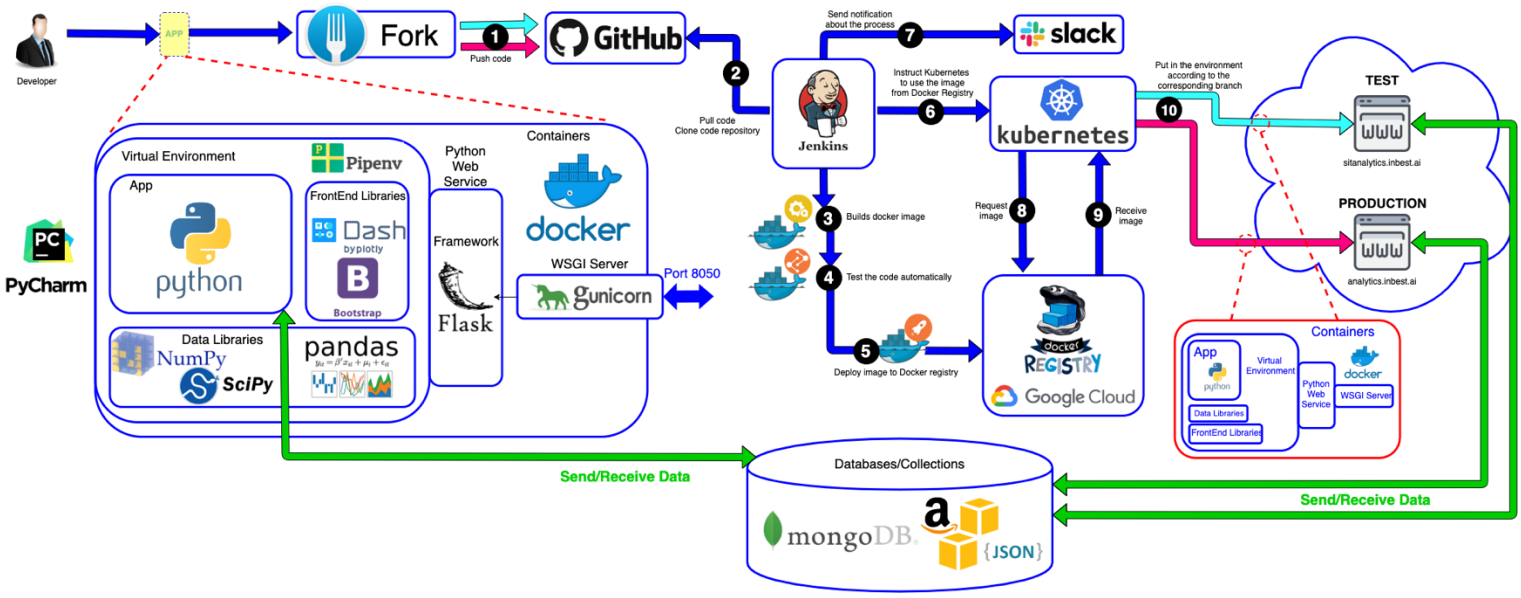


Figura 7. Disposición jerárquica de tecnologías

El modelo de datos y BBDD – Las bases de datos deberán ser alojadas en un sistema auto controlado y podríamos contar con AWS, Google Cloud o Microsoft Azure. Además, serán almacenadas en formato JSON a través de MongoDB. Por lo que lo ideal sería encontrar la infraestructura que facilite la gestión y uso de este tipo de bases de datos.

Control de versiones – Se utiliza Github como control de versiones que será el agente que conecte ambos escenarios de desarrollo y testeo-depuración. El desarrollo se llevará a cabo en uno de los repositorios privados de la compañía.

Python como lenguaje principal – La aplicación inicial estaba desarrollada en R pero optamos por utilizar Python por su versatilidad y por la variedad de componentes que podrían ayudar a mejorar tiempos de carga y eficiencia.

Docker aporta flexibilidad – Resulta imprescindible "dockerizar" las aplicaciones y llevar un control de contenedores o imágenes para posteriormente facilitar el deploy a los servidores y tener así facilidad a la hora de cargar dependencias que harán que la aplicación funcione con las mismas características con las que se ha desarrollado.

Entorno Virtual - Docker en Python establecerá el entorno a través del fichero Dockerfile que hará funcionar a través del framework Flask y del servidor WSGI Gunicorn para instalar las dependencias necesarias en el entorno. Se cuenta además con pipenv como entorno virtual de desarrollo que será encargado de contener todo ese contexto de librerías y variables de entorno.

Librerías de Python - Podemos optar por librerías para análisis y tratamiento de datos como NumPy, SciPy y pandas. Por otro lado, en la parte del frontend será necesario una librería como Dash que sea encargada de facilitar la visualización e interacción con los elementos visuales de la interfaz de usuario. En cuanto al estilo visual y al diseño se ha optado por utilizar Bootstrap por su facilidad de uso e integración con los elementos Dash.
IDE y software de desarrollo - El IDE utilizado es PyCharm por su potencia para ejecutar pruebas y por su integración con Docker y GIT. Para un mejor control de versiones se utiliza Fork.

Integración Continua - Jenkins será el encargado de monitorizar los cambios que se realicen en las dos ramas para coordinar las acciones de testeo y despliegue. Kubernetes servirá como plataforma de gestión de contenedores y subida de nuevas versiones a los servidores de manera automatizada controlado por Jenkins. Los contenedores se almacenan en el Docker registry en la plataforma de la compañía en Google Cloud.

Comunicación y organización - Se utiliza Slack como plataforma de comunicación del equipo y notificaciones de nuevos eventos. Asana como plataforma de coordinación de tareas aplicando SCRUM como metodología ágil. OneNote como plataforma de documentación y toma de notas de las tareas. OneDrive como plataforma de creación de

contenido del proyecto, edición en tiempo real y comunicación con el director de TFG. Y Draw.io como plataforma de edición de diagramas.

Documentación y depuración – Se ha utilizado Doctest y Docstring para la documentación y se ha utilizado el modo depuración de Dash para controlar los errores.

2.2 Metodología de desarrollo de soluciones - GIT Control Version

El desarrollo de las soluciones se realiza a través de SCRUM siguiendo los principios de metodologías ágiles. Para esto es muy importante las reuniones diarias para actualizar el estado de las tareas y quincenales con el resto del equipo para controlar el estado global de la plataforma.

Se utiliza el software Asana, que sirve como un tablero en el que plantear la lluvia de ideas, los sprints y el estado de las implementaciones a llevar a cabo a través de etiquetas que reflejan las características de cada tarea:

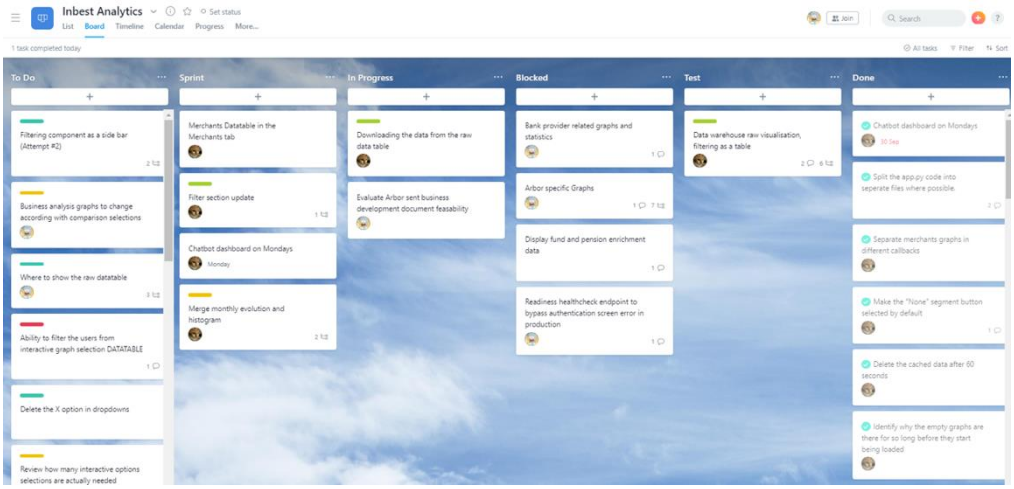
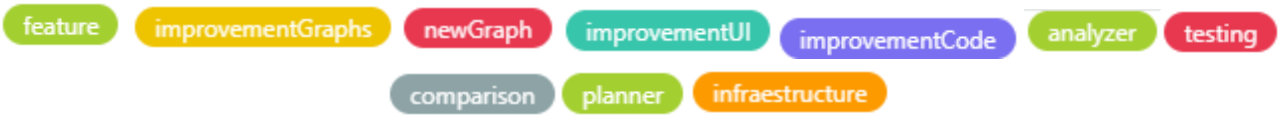


Figura 8. Panel principal de Asana



En esta reunión diaria se debate sobre las soluciones que han sido implementadas y están en proceso de testeo, se establecen el orden de los retos a implementar, se discute sobre los tiempos de ejecución y se plantea una visión global de posibles mejoras asociadas a estos cambios.

Una vez se tiene claro la instantánea del estado del proyecto y las tareas asociadas se comienza el desarrollo o solución de los retos planteados siguiendo el siguiente diagrama de procesos:

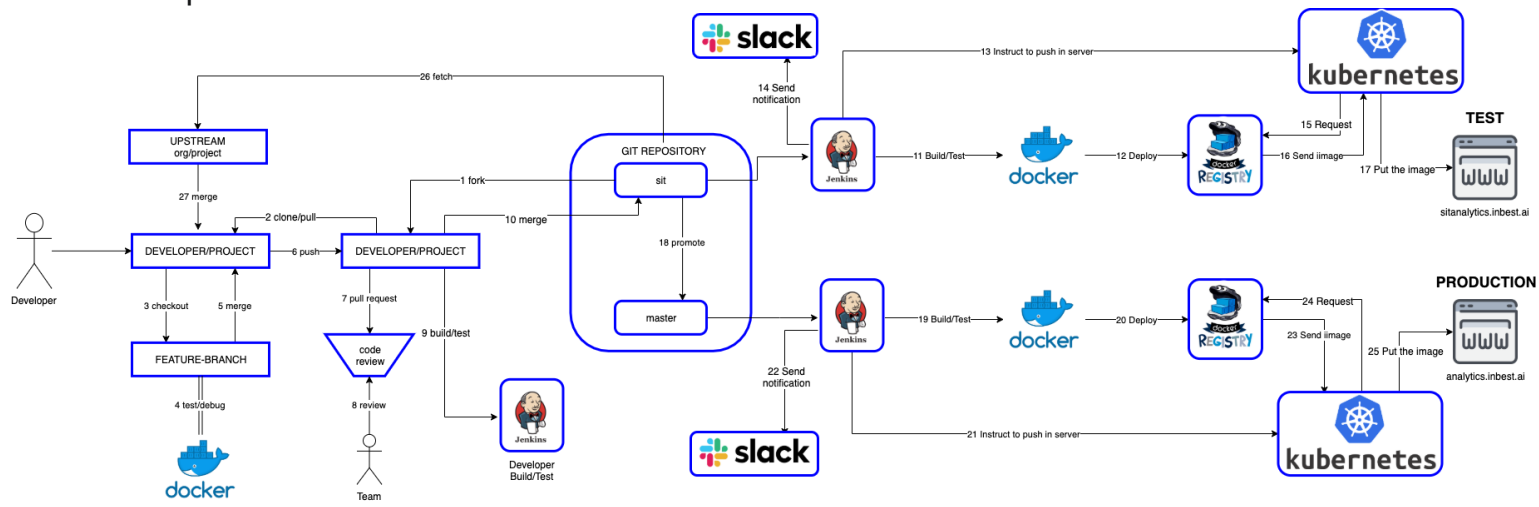


Figura 9. Entornos de desarrollo, test y despliegue

1. **Fork.** En primer lugar, es necesario guardar una copia del repositorio en el equipo para que los cambios efectuados no afecten al original. Su objetivo, por lo general, es utilizar esta nueva copia para proponer cambios en el proyecto de la organización. Directamente se puede pasar al punto 2 y clonar el proyecto como punto de partida: esto quiere decir entonces que los cambios se realizarían sobre el repositorio de la compañía.
2. **Clone/pull.** En este caso se realiza una clonación del repositorio en el sistema en el que se desea trabajar y así tener una copia que no afecte sobre el repositorio inicial, pero que si persiga actualizarlo directamente. Este punto será el de partida cuando se quiera actualizar los cambios que otras personas hayan hecho sobre el proyecto. Será importante realizar un pull para obtener una fotografía del momento actual del proyecto y comprobar el estado de las ramas o tareas en curso. Es una buena acción si se desea seguir buenas prácticas, ya que si no se tiene el estado actualizado por cambios ajenos puede afectar en el correcto funcionamiento del repositorio.
3. **Checkout.** Al realizar clone/pull, lo ideal es realizar el checkout sobre la rama desde la que se desea partir o comenzar una nueva.
4. **Commit/Test/Debug.** En este punto se pueden realizar las tareas sobre la rama apropiada siguiendo una serie de pautas establecidas por el Project Manager.
5. **Merge.** Una vez se han realizado los cambios sobre la rama y se desea incorporar al proyecto se deberá realizar un merge. Con esto se une la rama “en cambios” con la original creando un nuevo commit y fusionando con los cambios efectuados.
6. **Push.** Cuando se decida actualizar el estado del proyecto base en remoto independiente de la copia en el sistema se deberá realizar push. Si no se realiza el push con los cambios del proyecto de copia local, estos cambios jamás se verán reflejados en el repositorio remoto.

7. **Pull request.** Una vez se desea incorporar los cambios y el estado actual al repositorio se deberá realizar un pull request. Esta acción se ejecuta para preparar un review de todos los cambios que se han efectuado desde la última actualización y comprobar si existen discrepancias y solucionarlas.
8. **Review.** En este punto se obtiene el estado de "Code Review". El equipo, un miembro asignado por el Project Manager o el mismo Project Manager coordina y revisa el estado de la actualización de los cambios.
9. **Build/test.** Cuando se ha realizado la revisión de código se pueden solicitar cambios en busca de la mejora o simplemente en busca de las mejores prácticas de documentación y escritura de código que pueden ser resueltas por cada miembro del equipo. Esto llevará a un estado en el que se comprueben los cambios para poder preparar el entorno para la posterior actualización del repositorio de la compañía, por lo que es importante saber si los cambios podrán ser ejecutados en otro entorno, como un servidor. En este punto resulta interesante la utilización de Docker en local para comprobar si la imagen puede generar alguna inconsistencia en el servidor.
10. **Merge.** Una vez se tienen los cambios efectuados en la rama remota y se aprueba el pull request, se tiene opción de actualizar la rama base en el repositorio de la compañía a través de un merge. Se ejecutará correctamente si no existen discrepancias con el repositorio de la compañía.

Si se requiere actualizar directamente desde el repositorio de la compañía:

26. **Fetch.** Con esta acción se actualizan los cambios en el repositorio local, pero se actualizan los cambios del repositorio remoto a la rama origin/master que funciona como espejo.
27. **Merge.** Una vez se tienen los cambios en origin/master, esto nos ofrece flexibilidad para realizar merge a una rama local y poder continuar con el desarrollo.

A continuación, se detallan una serie de consideraciones que se llevan a cabo por orden del Project Manager para un correcto desarrollo del proyecto siguiendo buenas pautas, automatizar y facilitar los cambios:

- El desarrollador en primer lugar debe comprobar el estado de las ramas en Fork y comprobar si ha realizado el pull en la rama adecuada para obtener todos los cambios, el checkout a la rama correcta (si ya existe) o si necesita la creación de una nueva rama porque se entiende que se trata de una solución independiente y paralela al trabajo anterior.

- Para la preparación de las reuniones diarias se debe realizar una actualización con el último punto que se ha estado trabajando. Se requiere un último commit diario con los cambios y el punto exacto en el que se encuentra el trabajo cada día. Así el

supervisor puede preparar la reunión con información previa y organizar las posibles o tareas prioritarias.

- Es importante mencionar que se deberá estar seguro de que en la tarea o rama no se realiza pull request a alguna de las ramas sit o master si aún quedan mejoras por implementar en esa tarea y se entiende que esa tarea no ha sido aún finalizada (aunque posteriormente se pueda realizar una mejora). El pull request marca la finalización parcial de una tarea.

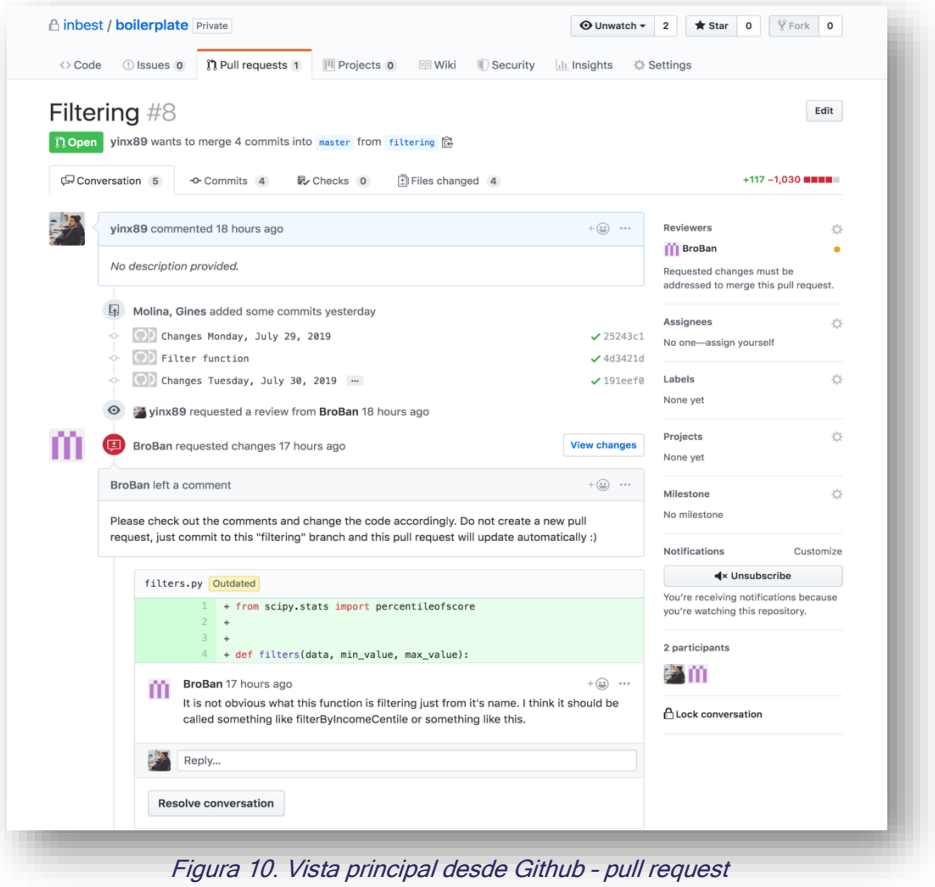


Figura 10. Vista principal desde Github - pull request

- El desarrollador siempre ha de realizar los pull request a la rama “sit”, nunca a master. Esto es para prevenir posibles errores e inconsistencias que podría generar el desarrollo de nuevos componentes y librerías.

- Siempre que se realiza un pull request, el supervisor realizará anotaciones que deberán ser atendidas por el desarrollador para conseguir las exigencias que suponen los nuevos cambios. No se deberá cerrar el pull request si se exigen nuevos commits. Se deberá atender cada uno de los comentarios de mejora, responder en el caso de duda e implementarlos en un o varios nuevos commits. Una vez sean atendidos, se deberá requerir un Request al supervisor para que pueda confirmar los cambios y si fuera necesario, se volvería a requerir cambios.

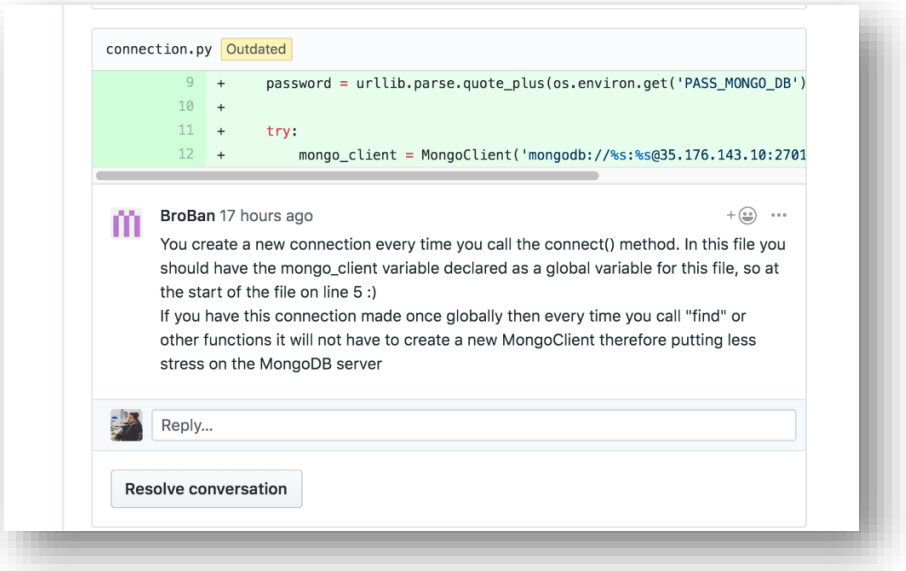


Figura 11. Revisiones en pull request

- Tampoco se deberá resolver la conversación porque en este caso el supervisor no va a poder gestionar ni evaluar los cambios que ha requerido.

- Cada una de las ramas atiende a una o varias soluciones llevadas a cabo para conseguir finalmente un estado funcional de la plataforma, con uno o varios cambios efectuados. Es importante marcar un título clarificador y aportar toda la documentación que sea necesaria para entender qué supone ese commit y los cambios que han sido afectados. Aunque siempre se pueden ver los cambios de cada uno de los archivos en Fork.

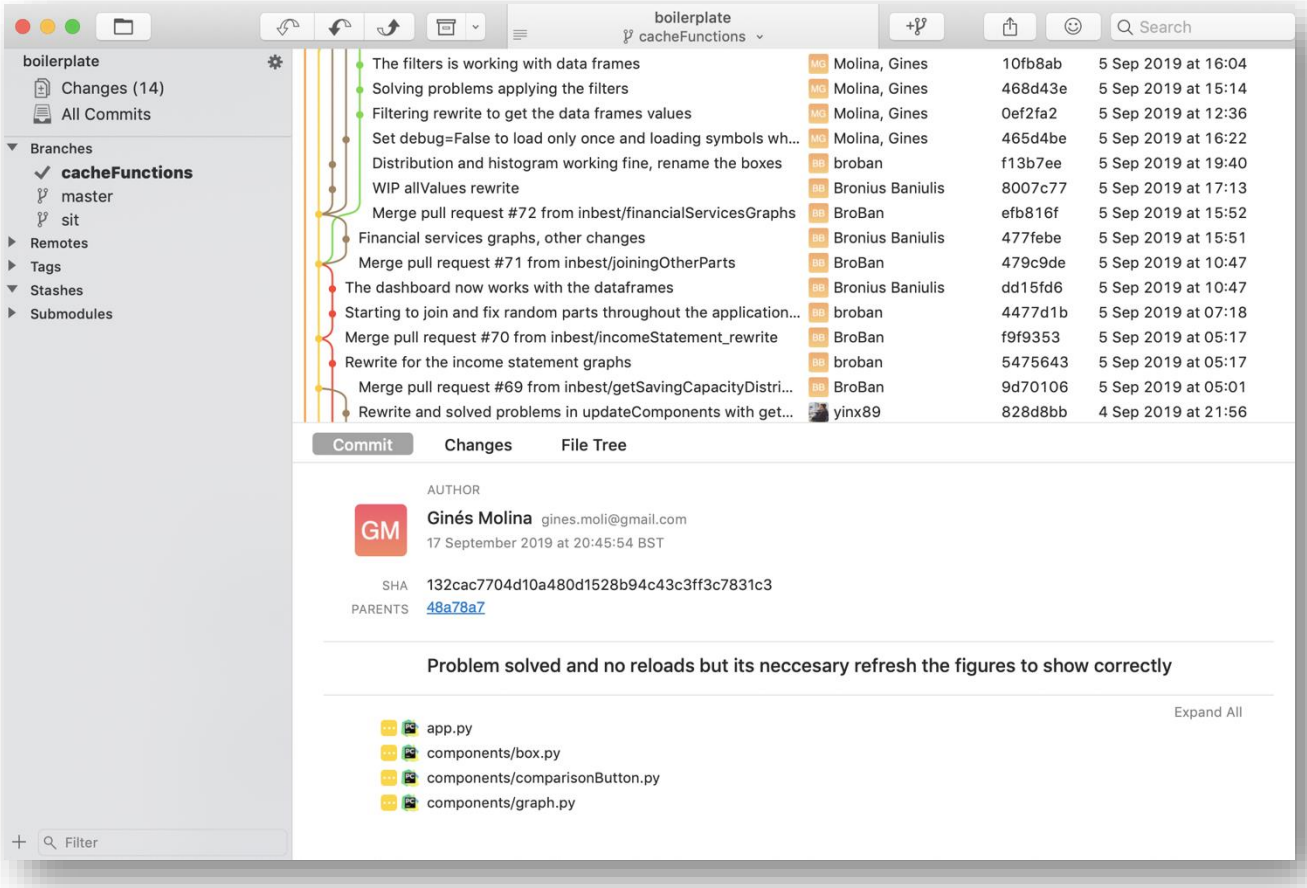


Figura 12. Vista principal de Fork - Control de versiones

- La rama máster apuntará al dominio analytics.inbest.ai y la rama sit apuntará al dominio sitanalytics.inbest.ai para tener bien diferenciados los dos entornos y las dos ramas. La rama máster sólo implementará los cambios con versiones estables y con actualizaciones más duraderas.

- Cuando se requiere una actualización global para entender mejor el estado actual del desarrollo se deberá realizar merge de todas las ramas a la rama sit y resolver posibles conflictos que puedan surgir entre las inconsistencias que se pueden generar entre las distintas versiones de los archivos.

2.3 Testeo y despliegue - Integración Continua

En este apartado cobra especial importancia Jenkins y Kubernetes, encargados de construir una imagen que testar y desplegar en cada uno de los servidores siguiendo un proceso detallado a continuación:

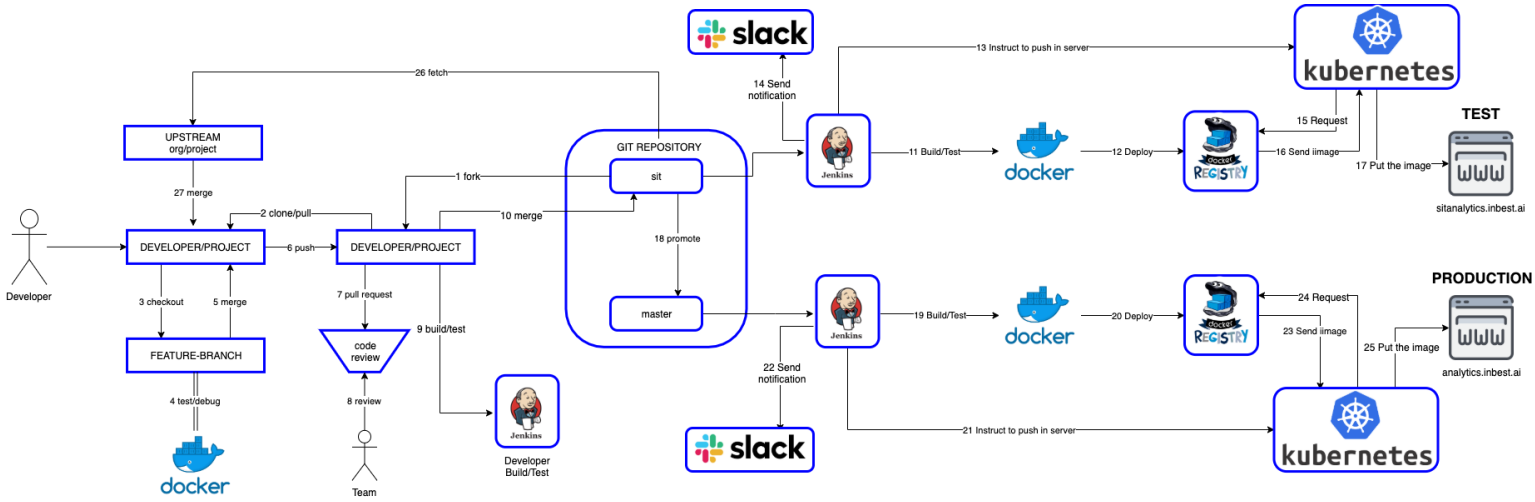


Figura 13. Entornos de desarrollo, test y despliegue

11. **Build/test.** Una vez se ha realizado merge y se actualiza el repositorio de la compañía, Jenkins actúa de listener ejecutando tres acciones. La primera es construir la imagen conforme al código del repositorio de la compañía y pasar las pruebas oportunas.
12. **Deploy.** A continuación, realiza el despliegue de la imagen a Docker registry y así da por finalizada la primera de las acciones.
13. **Instruct to push in server.** La segunda acción es informar a Kubernetes de que se ha desplegado una nueva versión que posteriormente deberá subir o actualizar en el servidor. Se le pasarán los argumentos apropiados que apuntan al directorio donde está la imagen en Docker Registry.

14. **Send notification**. Por último, Jenkins envía la notificación a Slack en la que queda reflejado el cambio que se acaba de efectuar.
15. **Request.** Kubernetes recibe la orden de Jenkins y solicita a Docker registry la imagen correspondiente.
16. **Send image.** Se realiza el envío de la imagen a Kubernetes.
17. **Put the image.** Por último, se sube o actualizan los cambios en el servidor con la imagen construida.

Si se quieren actualizar los cambios en la rama master:

18. **Promote.** Se realiza la promoción de los cambios si se comprueba que la rama de testeo y depuración indica un estado preparado para publicar y ha pasado las pruebas. Actualiza el repositorio con un nuevo merge a través de un commit teniendo en cuenta los cambios efectuados desde la última actualización.
19. **Build/test.** Una vez se ha realizado merge y se actualiza el repositorio de la compañía, Jenkins actúa de listener nuevamente. Construye la imagen conforme al código del repositorio de la compañía y pasa las pruebas oportunas.
20. **Deploy.** Realiza el despliegue de la imagen a Docker registry.
21. **Instruct to push in server.** Informa a Kubernetes de que se ha desplegado una nueva versión que posteriormente deberá subir o actualizar en el servidor.
22. **Send notification.** Por último, Jenkins envía la notificación a Slack en la que queda reflejado el cambio que se acaba de efectuar.
23. **Request.** Kubernetes recibe la orden de Jenkins y solicita a Docker registry la imagen correspondiente.
24. **Send image.** Se realiza el envío de la imagen a Kubernetes.
25. **Put the image.** Por último, se sube o actualizan los cambios en el servidor de producción con la imagen construida.

Las pruebas que realiza Jenkins están detalladas en la sección A del Anexo y siguen los siguientes 'stages' o etapas: 'Build & Test', 'Checkout', 'Setup GCloud Authentication', 'Build Custom Environment', 'Push Docker Image To Container Registry' y 'Deploy to Environment'.

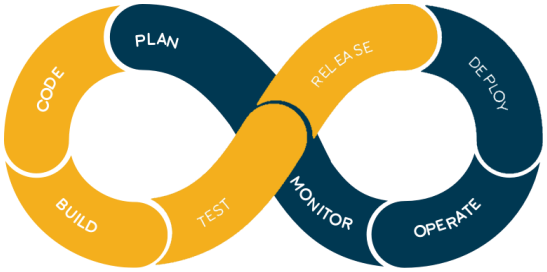
```
else if (env.BRANCH_NAME == "sit"){
  IMAGE = "gcr.io/inbestcloud/github-inbest-boilerplate:${GIT_VERSION}"
  node{
      stage ('Build Custom Environment') {
        sh "echo ${IMAGE}"
        buildEnv = docker.build("gcr.io/inbestcloud/github-inbest-boilerplate:${GIT_VERSION}")
      }
      stage ('Push Docker Image To Container Registry') {
        sh "/var/lib/jenkins/GoogleCloudSDK/google-cloud-sdk/bin/gcloud docker -- push gcr.io/inbestcloud/github-inbest-boilerplate:${GIT_VERSION}"
        sh "sed -i.bak 's#gcr.io/inbestcloud/github-inbest-boilerplate:#gcr.io/inbestcloud/github-inbest-boilerplate:${GIT_VERSION}#' deployment/deploymen
      }
      stage ('Deploy to SIT Environment') {
        sh "/var/lib/jenkins/GoogleCloudSDK/google-cloud-sdk/bin/kubectl --namespace=sit replace -f deployment/deployment.yaml"
      }
  }
```

Figura 14. Pruebas en Jenkins

Aplicar integración continua proporciona solidez al producto, fiabilidad, un nivel de desarrollo a prueba de errores exitoso y monitorización en tiempo real del estado y los fallos reportados por cada módulo. La idea se trata de compilar automáticamente con cada nueva integración de nuevas características o mejoras en el código. De esta manera, el proyecto permanece íntimamente ligado a la cultura DevOps en la que se va a poder automatizar procesos para comprobar como funcionaría en un entorno global de manera flexible y periódica, y sería algo así como si fuera una réplica de prueba del original.

Jenkins es una herramienta que permite aplicar Integración continua, gratuita y open-source. Su funcionamiento se basa en permanecer en 'modo escucha' ante el desarrollo del producto y ejecutando pruebas ante un posible escenario. Se pueden programar tareas, se puede notificar su estado y, además, cobra especial importancia cuando sirve como enlace entre los dos entornos del proceso de desarrollo. Si además es utilizado junto a Kubernetes como entorno de administración centrado en contenedores sería óptimo.

En definitiva, se trata de aprovechar lo mejor de la cultura Scrum como metodología ágil de desarrollo, lo mejor de la cultura DevOps para aplicar integración continua sobre la infraestructura y automatizar procesos y, por último, lo mejor de la utilización de los entornos flexibles a través de Docker y los contenedores.



2.4 Diseño del modelo de datos - Big Data

A continuación, se pasa a detallar las entidades involucradas en el sistema en un entorno inicial y cómo finalmente son optimizadas para su mejor tratamiento por la plataforma. En este apartado además se diferenciará en tres tipos de datos: de entrada, internos y de salida.

2.4.1 MODELO DE DATOS INICIAL

En un principio la compañía cuenta con el modelo de datos CustomerObject con la información de los productos y servicios bancarios de cada cliente actuando como datos de entrada que son almacenados en bases de datos no relacionales. En este modelo inicial

siempre se obtiene un objeto en formato JSON que contiene características por cada componente de este modelo y cliente. El diagrama de clases del modelo es el siguiente:

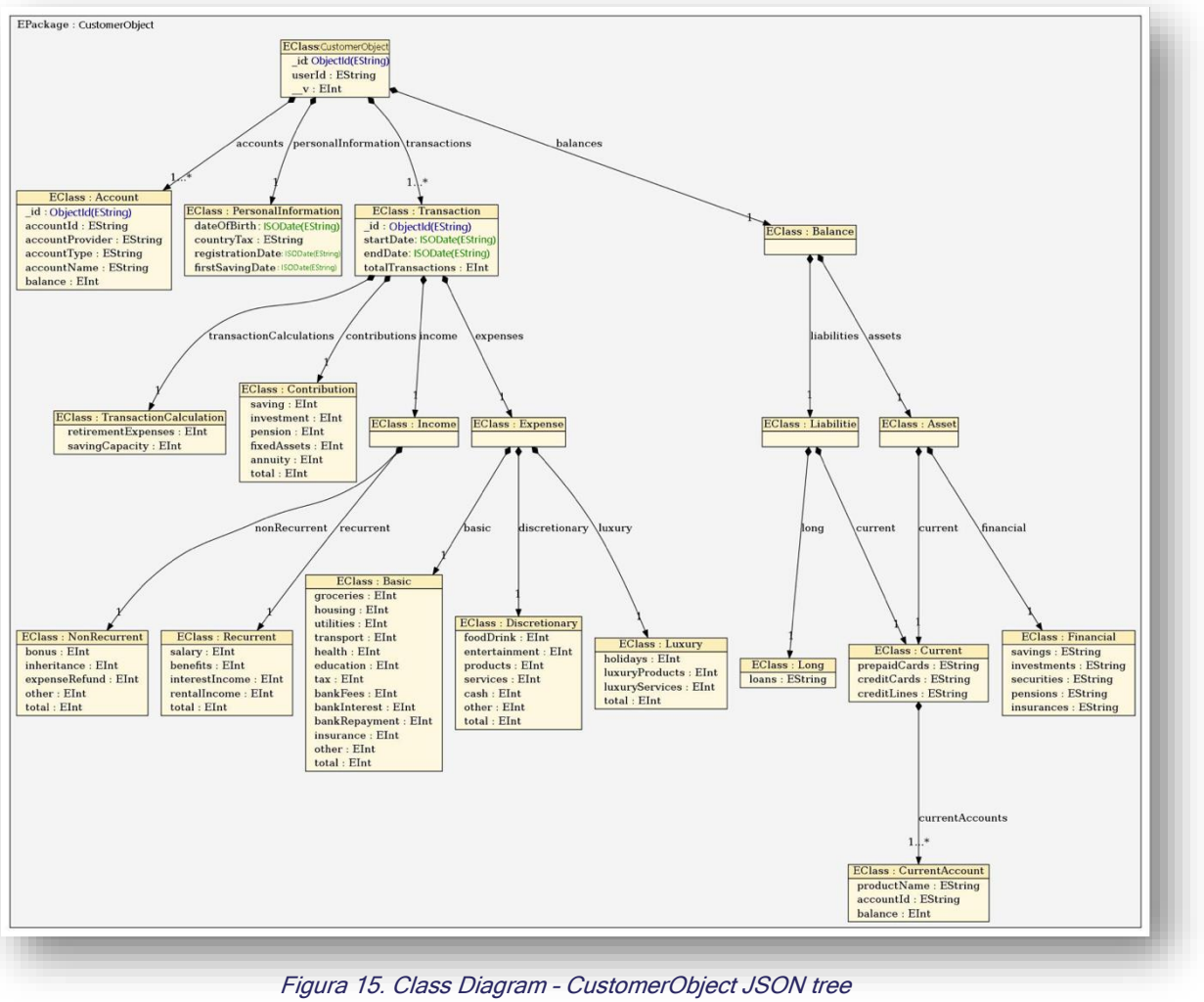


Figura 15. Class Diagram - CustomerObject JSON tree

Los datos internos son obtenidos en objetos JSON y almacenados y tratados en diccionarios, listas, enteros y strings.

Lo más importante con este primer modelo es la flexibilidad en el tratamiento de los datos para poder verificar la funcionalidad de las soluciones llevadas a cabo de manera rápida y sencilla de entender en contexto con el resto de las herramientas con tipos de objetos comunes y sencillos para facilitar su integración. Posteriormente se podrá depurar y pulir en mayor medida la utilización de uno u otro tipo de datos según cada propósito.

Los datos de salida son mostrados en formato HTML tras ser tratados y renderizados por Dash en cada uno de sus componentes para su utilización en las gráficos y componentes de la plataforma. Dash utiliza JSON como formato de configuración, por lo que, para la construcción de los componentes y gráficas, en la mayor parte de ellos, no es necesario en la mayoría de los casos un tratamiento específico del tipo de datos.

2.4.2 MODELO DE DATOS FINAL

Tras el análisis y desarrollo de las soluciones, se lleva a cabo una posterior adaptación del modelo de datos para poder optimizar tiempos de carga y realizar un análisis más orientado en tecnologías Big Data a través de la librería pandas utilizando dataframes como tipo de datos internos. Esto quiere decir que, aunque se realiza la consulta sobre el mismo modelo de datos CustomerObject de la base de datos, se tratará la información importante de manera intermedia a través de un tipo de datos que es conveniente para soluciones Big data. El diagrama a continuación refleja el nuevo modelo de dataframes:

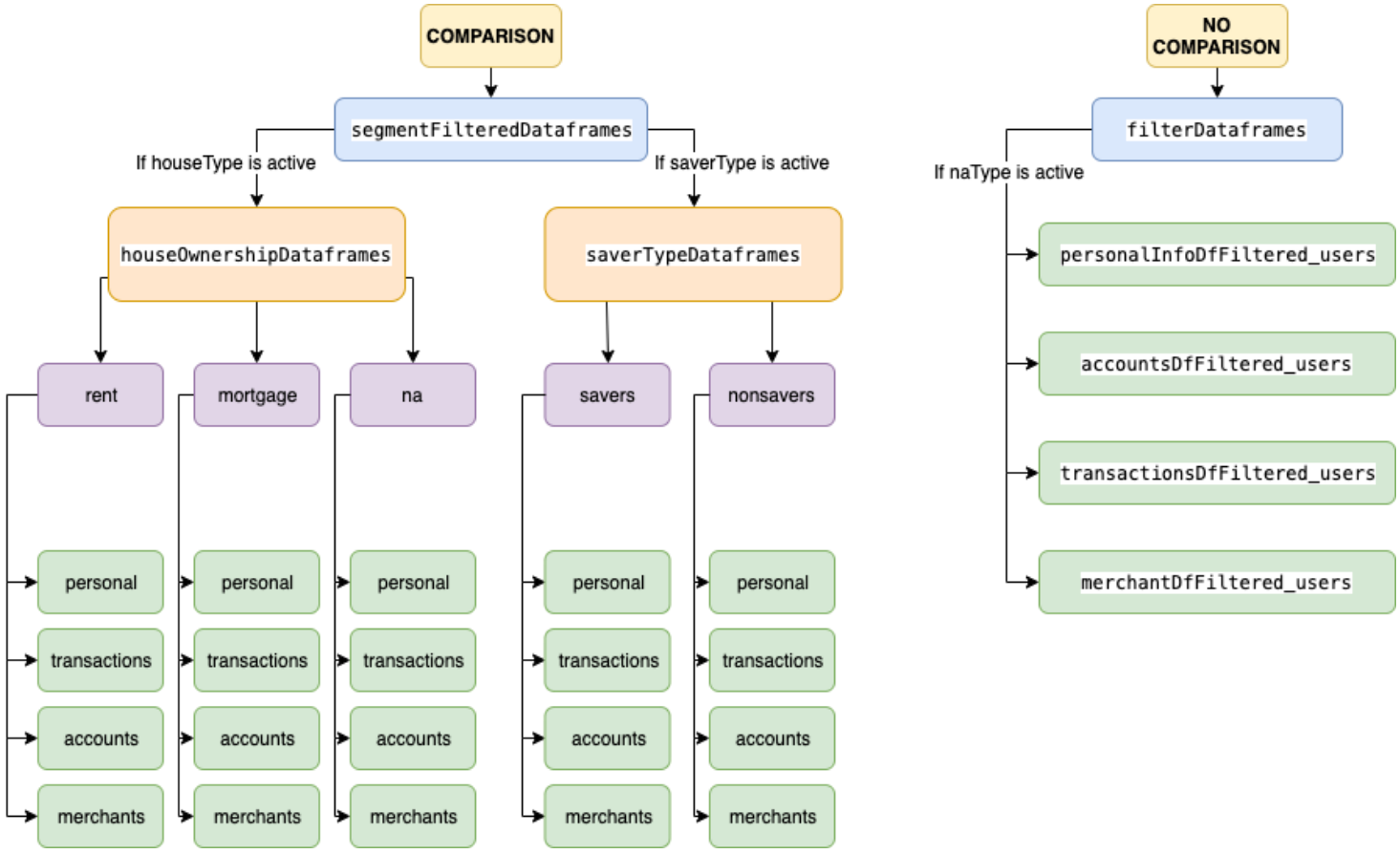
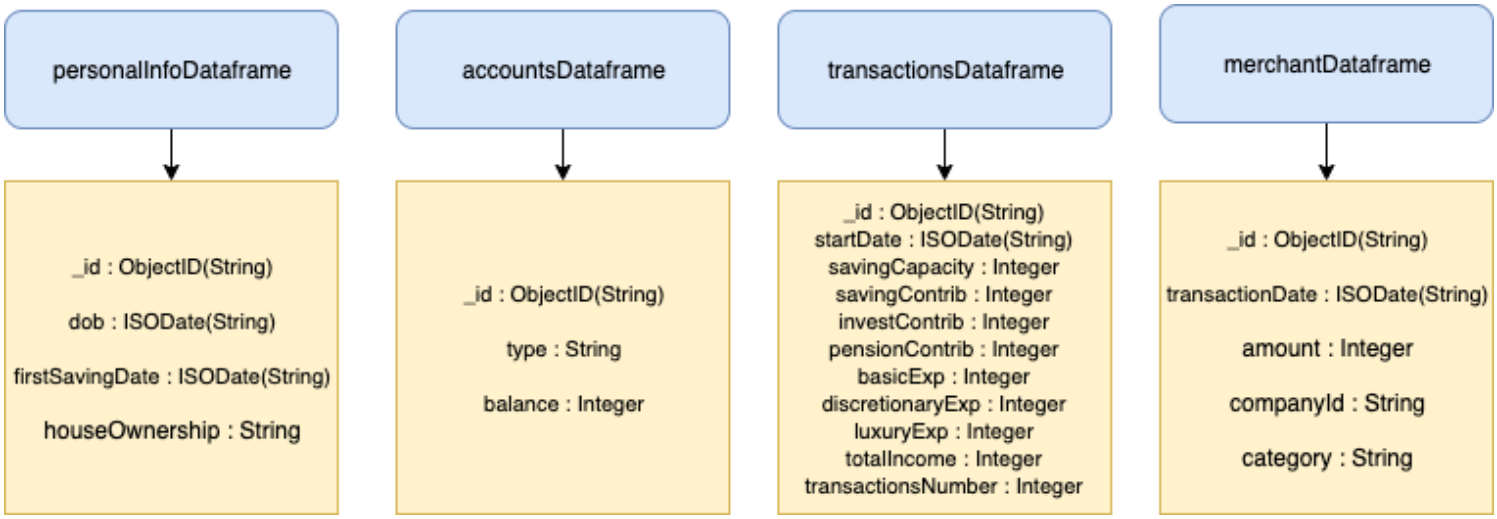


Figura 16. Nuevo modelo de datos - Dataframes

En capítulos posteriores se analizará en detalle la solución del almacenamiento en caché. Por ahora, cabe destacar que se obtiene un sistema basado en dataframes según el tipo de comparación seleccionado por el usuario. Este modelo de datos ha sido ideado con la necesidad de salvaguardar los datos entre procesos internos y con la idea de no saturar la base de datos con demasiadas y muy pesadas consultas. La composición interna de cada uno de los dataframes se presentan a continuación:



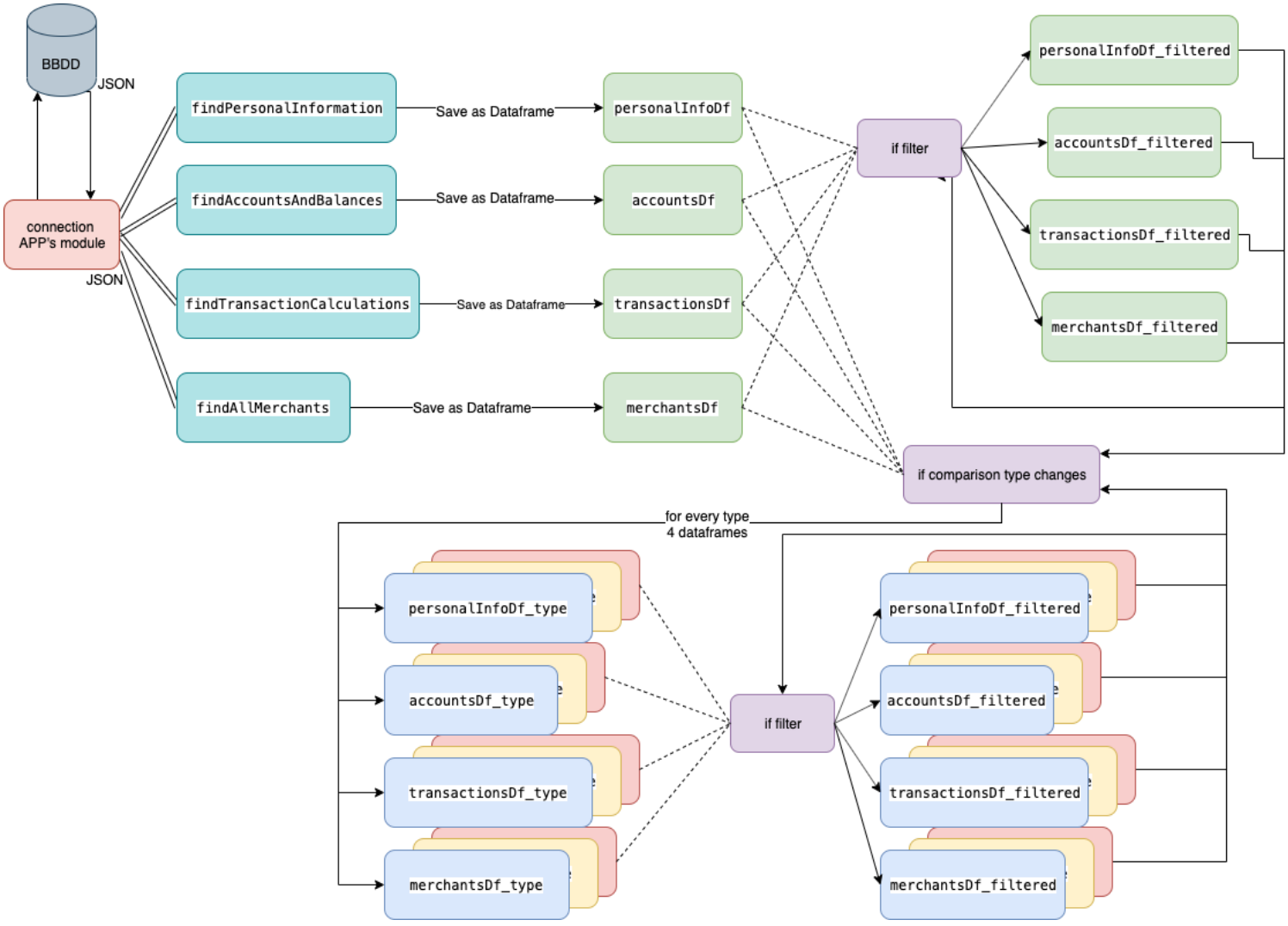
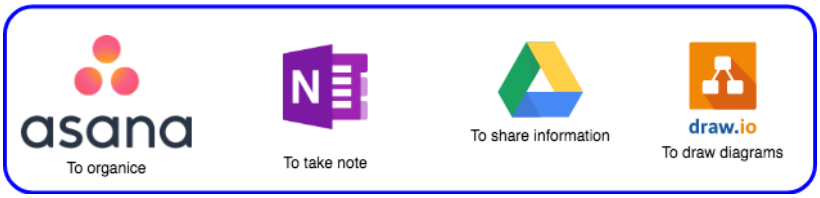


Figura 17. Composición de Dataframes y proceso

En el proceso general, se obtienen los datos de entrada de la base de datos y se almacenan en los distintos dataframes. Posteriormente pueden ser filtrados y tratados por los procesos internos para su evaluación, si se aplica filtrado, o segmentados si se aplica un tipo de comparación. Si se selecciona la comparación por tipo de propietario o ahorrador, existirán 4 dataframes con los datos de cada tipo de comparación. Si se ha seleccionado algún tipo de filtrado, estos dataframes atienden también al filtrado de cada dataframe. Finalmente, los dataframes son utilizados en la muestra de uno o varios valores específicos a través de su renderizado en distintos componentes de la plataforma a través de Dash.

2.5 Documentación y depuración



2.5.1 DOCUMENTACIÓN - DOCSTRING

En un proceso de desarrollo, y en este caso con tecnologías tan a la vanguardia, es muy importante el conocimiento del funcionamiento de las herramientas, módulos o funciones del producto a través de la documentación. Es el camino para ofrecer un buen producto ya que puede escalar en un futuro y la base sólida sobre todo el conocimiento que se ha aplicado en cada sección o módulo resulta imprescindible para continuar aplicando mejoras. Se van a utilizar ciertas herramientas que van a ayudar a documentar y preparar, a su vez, pruebas sencillas donde muestra el funcionamiento de ciertas funciones importantes y su resultado esperado.

El primer paso es la organización estructurada del código (a través de buenas prácticas) y ayudarse de comentarios importantes (en su justa medida) que ayude al entendimiento de la función o de cierto módulo o comando que aplica algo importante y que podría ser determinante conocer en el futuro, sobre todo tratando el porqué se hace de ese modo y no otro. Para comentarios se han utilizado las 3 dobles comillas “”” antes y después de cada comentario y # al inicio de cada línea de comentario.

Se ha utilizado DocTest para realizar pruebas sencillas en el primer modelo de datos estableciendo >>> al inicio de cada comando que se quiere incluir en el interior del bloque de comentarios siguiendo el patrón:

```
"""
COMENTARIO SOBRE LA FUNCIÓN
>>> inicializacionVariable = []
>>> nombrefunción(parámetros)
>>> bucle
... continuación bucle
RESULTADO
"""
```

Al final de cada función es importante importar la librería doctest y ejecutar el test con doctest.testmod(). Si el resultado es el correcto según los parámetros indicados en el bloque de comentarios no aparecerá nada. Si hay un error aparecerá así:

```
File “app.py”, line 433, in _main_.mainCallback
Failed example:
       getMarks(18,34,’age’)
Expected:
       {0:’0’,…,100:’100’}
Got:
       {10:’10,…,120:’120’}
```

Con la utilización del nuevo modelo de datos a través de Dataframes se opta por la utilización de la librería DocString especializada en pandas Dataframes y se ha seguido la guía de la documentación oficial [7] para seguir las mejores prácticas para llevarlo a cabo, sobre todo, en los archivos de figureHelpers y figureFunctions:

```
def getSavingCapacitiesEvolution(transactionsDf):
    """
    Calculate the average Saving Capacity per Date from all user's transactions.

    First extract only the startDate and its savingCapacity into a new dataframe.
    Then group by the startDate, sum all the savingCapacities and get the average by
    the total of users.

    Parameters
    ----------
    transactionsDf : DataFrame
        Contains all the transactions for the filtered or non filtered users.

    Returns
    -------
    datesList : list
        All the Dates in the field startDate in every transaction grouped together.
    savingCapacities : list
        All values savingCapacities summed by each data value.

    See also
    --------
    getSavingCapacityDistribution : SavingCapacity from all the type of contributions

    """
    totalUsers = len(transactionsDf.index.unique())
    savingCapacity = transactionsDf[["startDate", "savingCapacity"]]
    savingCapacity = savingCapacity.groupby(['startDate']).sum()
    savingCapacityAverage = savingCapacity.div(totalUsers)
    datesList = savingCapacity.index.tolist()
    savingCapacities = savingCapacityAverage['savingCapacity'].tolist()

    return datesList, savingCapacities
```

También ha utilizado el etiquetado en la documentación a través de TODO para mostrar secciones específicas que cambiar y como anotación para volver más adelante a cambiar algo en concreto.

Se ha planteado la utilización de Sphinx en el futuro para generar documentación en formato HTML de manera automática y más accesible para estos propósitos en los que no sólo es práctica para labores de desarrollo.

2.5.2 DEPURACIÓN CON DASH

Dash permite la "recarga en caliente" de manera predeterminada en modo depuración y actualiza automáticamente el navegador ante cualquier cambio en el código. Esto permite una depuración ágil y flexible. Pero esto es uno de los muchos aspectos que ofrece la configuración de Dash en modo de desarrollo.

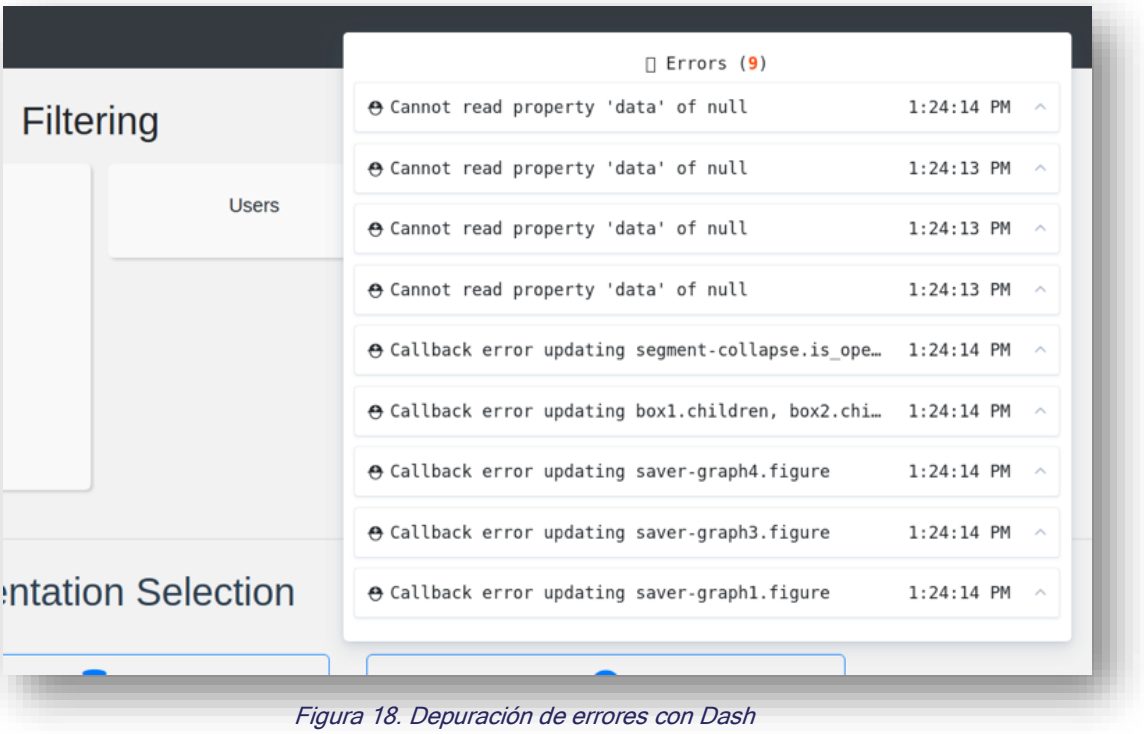


Figura 18. Depuración de errores con Dash

Dash ofrece la posibilidad de trabajar con esta configuración en modo de desarrollo (por defecto deshabilitada) para minimizar el riesgo de que los usuarios ejecuten estas características con ciertas implicaciones de seguridad en modo de producción. De esta manera, hay un flag simple para activar o desactivar todas estas funciones compatibles con la utilización del servidor WSGI. Las características que más ventajas aportan a la depuración en tiempo real son debug_display_javascript_errors y debug_display_python_debugger. Estas funciones activan o desactivan la visualización de los mensajes de error de Javascript y Python.

Además, se cuenta con PyCharm para establecer el modo debug sobre ciertos breakpoints en los que es necesario un entendimiento profundo sobre cómo se está comportando la aplicación y qué datos está tratando en cada momento. Para la utilización de los dataframes es indispensable saber en cierto momento las características de este dataframe.

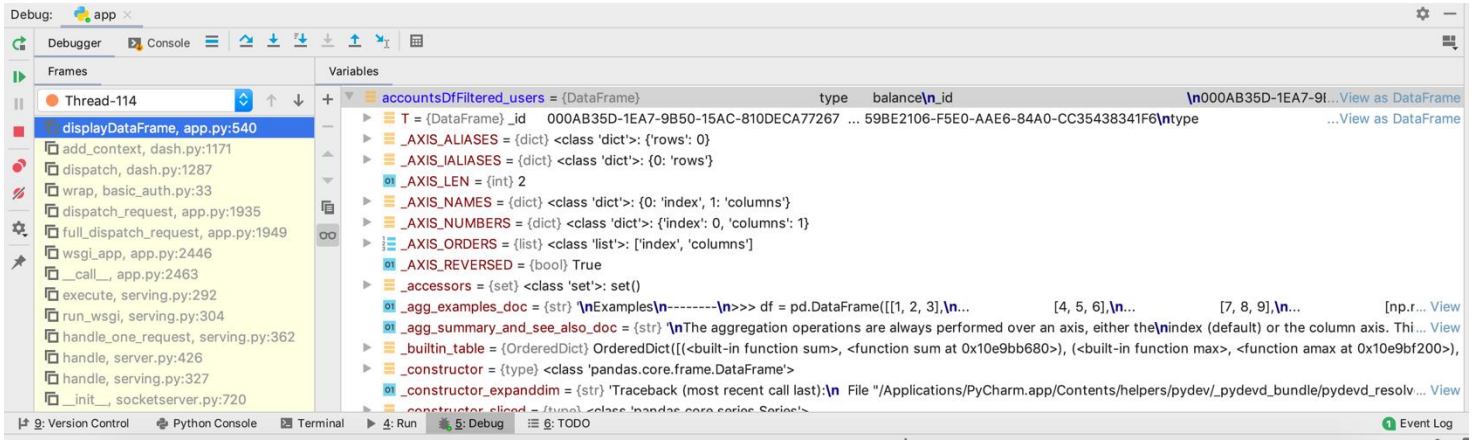


Figura 19. Depuración en PyCharm

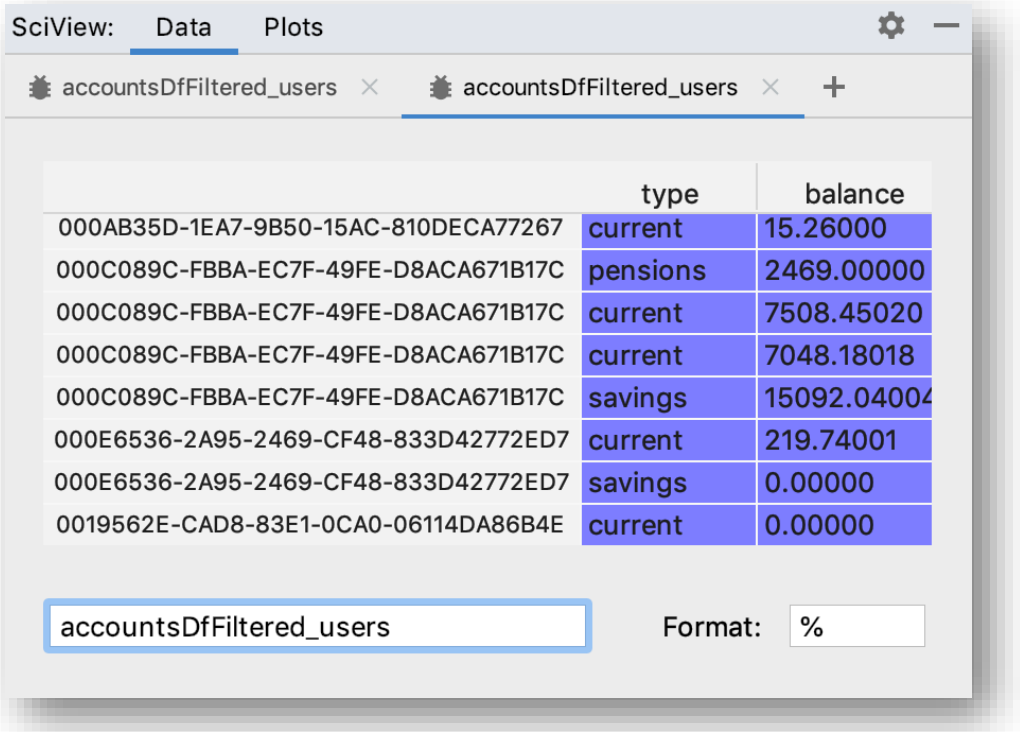


Figura 20. Vista de Dataframe en PyCharm

Capítulo 3 Diseño, desarrollo y testeo de soluciones

3.1 Configuración de entornos - Stack tecnológico

3.1.1 AWS CLOUD INFRASTRUCTURE + MONGODB BY BITNAMI

En primer lugar, se lleva a cabo la configuración de la instancia que trabajará en la infraestructura de AWS para brindarnos respuesta a la base de datos de manera ágil y rápida. A través del registro de la compañía en AWS, se nos proporciona el perfil de usuario para llevar a cabo la configuración de la instancia como primer paso.

A continuación, se lleva a cabo la selección del tipo de instancia que se va a necesitar y se opta por el modelo t3.large que cuenta con 2 vCPU, memoria de 8 GiB, ancho de banda de hasta 5 Gbps, 36 CPU créditos/hora y almacenamiento sólo a través de EBS [8](Elastic Block Store - Almacenamiento de bloque de alto rendimiento y con facilidad de uso a cualquier escala). A continuación, se ofrece un cuadro resumen del resto de modelos t3:

Name	vCPUs	Memory (GiB)	Baseline Performance/vCPU	CPU Credits earned/hr	Network burst bandwidth (Gbps)	EBS burst bandwidth (Gbps)	On-Demand Price/hr*	1-yr Reserved Instance Effective Hourly*	3-yr Reserved Instance Effective Hourly*
t3.nano	2	0.5	5%	6	5	1.50	$0.0052	$0.003	$0.002
t3.micro	2	1.0	10%	12	5	1.50	$0.0104	$0.006	$0.005
t3.small	2	2.0	20%	24	5	1.50	$0.0209	$0.012	$0.008
t3.medium	2	4.0	20%	24	5	1.50	$0.0418	$0.025	$0.017
t3.large	2	8.0	30%	36	5	2.05	$0.0835	$0.05	$0.036
t3.xlarge	4	16.0	40%	96	5	2.05	$0.1670	$0.099	$0.067
t3.2xlarge	8	32.0	40%	192	5	2.05	$0.3341	$0.199	$0.133

Figura 21. Cuadro resumen de modelos t3 de AWS

La elección del modelo de instancia [9] de uso general es porque resulta una combinación equilibrada de recursos, memoria y red, además de poder usarse para distintas cargas de trabajo. Este tipo de instancia T3 proporciona un nivel básico de rendimiento de la CPU con posibilidad de ampliar el uso de la CPU en cualquier momento durante el tiempo que sea necesario, así tenemos controlados los posibles picos de peticiones y no sufrimos cortes o demoras importantes. Es por este motivo que este modelo resulta ideal para ofrecer servicio a través de una base de datos que entendemos crítica para el servicio.

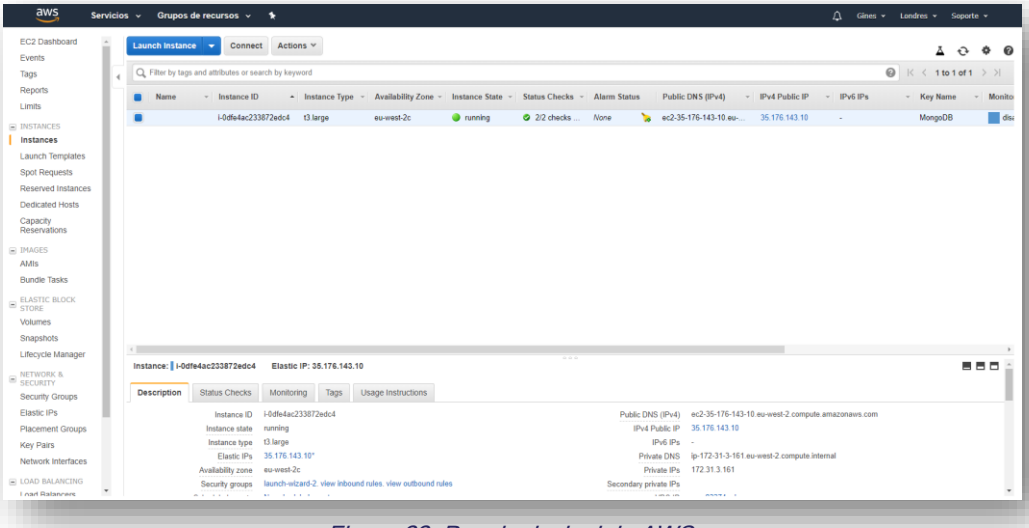


Figura 22. Panel principal de AWS

Desde el panel de control de usuario se puede acceder a Servicios -> EC2 y a continuación hacer click en “Launch Instance” para establecer la configuración estándar de la instancia.

Ahora es necesario llevar a cabo la configuración de MongoDB en la instancia de AWS a través de Bitnami [10].

En primer lugar, es necesario generar un SSH key pair a través del dashboard de Amazon que almacenamos para, posteriormente, establecer la conexión SSH con el archivo .pem almacenado y las credenciales de nuestra aplicación Bitnami de la siguiente manera:

```
ssh -i MongoDB.pem bitnami@USER.SERVER
bitnami@ip-IP:~$ sudo /opt/bitnami/ctlscript.sh status
mongodb already running
```

Una vez se comprueba que el servicio de mongodb está funcionando en nuestra instancia, se puede acceder al servicio y crear la primera base de datos de prueba y un usuario asociado a ella.

Por último, se debe establecer una dirección IP dedicada y estática para la instancia a través de “Network & Security -> Elastic IPs” en el panel de AWS y hacer click en “Allocate New Address”. Se elige la acción “Associate Address” y será establecida en nuestra instancia.

3.1.2 MONGODB EN LOCAL

Instalar MongoDB como servicio local en entornos Microsoft [11] es sencillo, tan sólo será necesario descargar el ejecutable e instalarlo [12].

En equipos MacOS además de descargarlo será necesario una serie de pasos descritos a continuación [13]:

- Se deberá instalar Homebrew [14] a través de una terminal y actualizarlo.
- Descargar mongodb a través de brew.
- Se puede ejecutar “mongod” para el ejecutar el servidor y “mongo” para ejecutar el cliente. Para salir del servidor ctrl+C y para salir del cliente “exit”. Es importante que el directorio db donde se establecerán los archivos de datos de mongodb tengan los permisos adecuados.

3.1.3 DOCKER IN PYTHON

Para realizar la configuración de Docker en la aplicación Python primero es necesario descárgalo e instalarlo.

Para enumerar las imágenes ejecutamos:

```
C:\Users\BlackSwan>docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
hello-world         latest              fce289e99eb9        6 months ago        1.84kB
```

Para enumerar los contenedores en ejecución:

```
C:\Users\BlackSwan>docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
PORTS               NAMES
```

Y para enumerar los contenedores creados:

```
C:\Users\BlackSwan>docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
PORTS               NAMES
294e1718b323        hello-world         "/hello"            8 minutes ago       Exited (0) 7 minutes
ago                 optimistic_khayyam
```

3.1.4 GUNICORN - EJECUTAR APLICACIONES EN PRODUCCIÓN USANDO DOCKER

Gunicorn nace de la necesidad de tener un servidor WSGI que proporcione solución a la siguiente advertencia:

```
* Serving Flask app "my_script" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
Running on http://0.0.0.0:8050/
Debugger PIN: 606-971-502
```

Dash utiliza la librería Flask cuando se ejecuta y, por defecto, usa el servidor de desarrollo werkzeug. Se trata de una buena solución para ejecutar rápidamente un servidor en desarrollo, pero sin garantías en cuanto a seguridad y rendimiento en entornos de producción. Por ello, es necesario la utilización de un servidor WSGI a través de Gunicorn.

Si se quiere ejecutar junto a Docker entonces se debe configurar la última línea en el archivo Dockerfile de la siguiente manera:

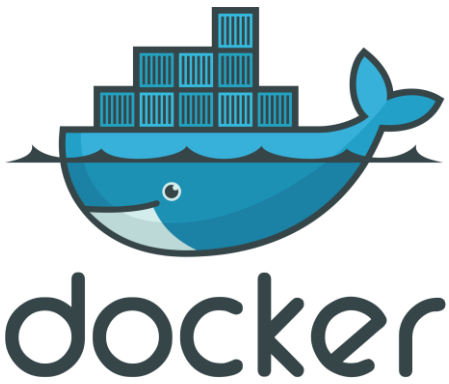
```
CMD ["gunicorn", "--workers", "1", "--threads", "2", "app:app.server", "-b", "0.0.0.0:8050", "--
timeout", "120"]
```

De este modo, ofrece la posibilidad de la configuración de hilos y del establecimiento de timeouts con la mirada puesta en la optimización del servidor en producción. Estos cambios serán aplicados tanto en el servidor de prueba como en el de producción gracias a Docker.

3.1.5 ENTORNO VIRTUAL - LIBRERÍAS Y VARIABLES DE ENTORNO

A) ¿Qué es Docker?

Docker es una tecnología o plataforma especializada en despliegues de aplicaciones. Encaja con modelos de microservicios pero más bien se trata de la unidad de despliegue para esos microservicios o aplicaciones.



La utilidad básica es incorporar dependencias de frameworks externos a la aplicación funcionando sólo a través de una única imagen. Su principal ventaja es que trata de incorporar un proceso en el que eliminamos características propias del sistema operativo, por lo que se trata de una imagen más ligera que posteriormente podrá ser ejecutada en cualquier máquina de desarrollo y en los servidores independientemente del tipo de servidor o entorno.

El funcionamiento básico de Docker queda reflejado en el siguiente diagrama:

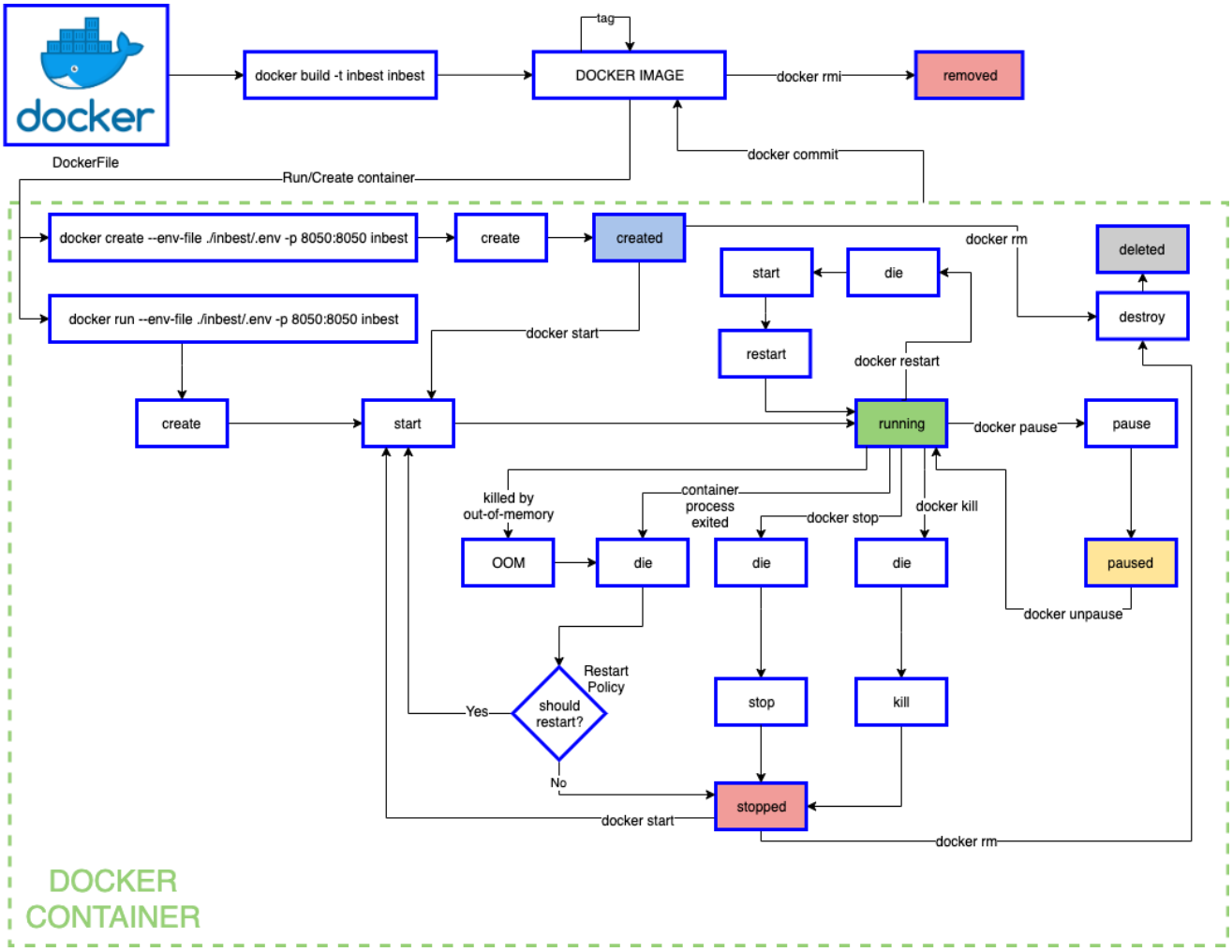


Figura 23. Funcionamiento de Docker

Se busca obtener compilaciones rápidas y reproducibles en cualquier entorno. En producción no hay problema con las variables de entorno gracias a Kubernetes. Se puede configurar una carpeta con los archivos deployment.yaml y service.yaml que contienen la información del despliegue en el servidor y de las variables de entorno, entre otras características:

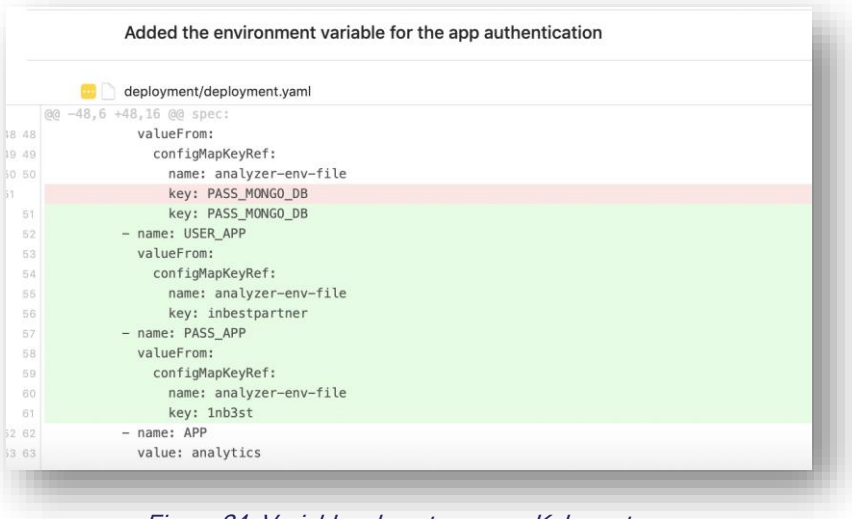


Figura 24. Variables de entorno con Kubernetes

Para el entorno de desarrollo local, donde se van a realizar los cambios se utiliza el plugin EnvFile [15] de PyCharm. Es la manera más sencilla de tener actualizadas las variables de entorno en un fichero que automáticamente carga PyCharm en cada nueva ejecución. El archivo será llamado dev.env y deberá ser incluido en .gitignore (para no subir a los repositorios). El fichero contiene la información de las variables de entorno necesarias con el formato:

```
PASS_MONGO_DB=passs888772778__00d
USER_MONGO_DB=serviceclient
USER_APP=client
PASS_APP=client_pass
```

También se puede incluir en la ejecución de Docker así:

```
docker run --env-file ./boilerplate/dev.env -p 8050:8050 boilerplate
```

B) Docker con pipenv

Se decide la utilización de Pipenv [16] para gestionar las librerías de manera ágil. Será configurado en PyCharm para que, al seleccionarlo como intérprete pueda crear automáticamente los ficheros Pipefile (dependencias lógicas - sección B del Anexo) y PipeFile.lock (dependencias ancladas) en lugar del fichero requirements.txt y así, añadir directamente las librerías. Para instalarlo se ejecuta: pip install pipenv. Una vez instalado se lleva a cabo la configuración en PyCharm [17].

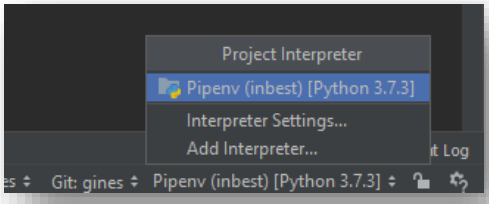


Figura 25. Selección de intérprete en PyCharm

Al configurar Pipenv como intérprete añadirá todos los paquetes disponibles desde el origen definido en Pipfile y todos los paquetes se instalan, eliminan y actualizan desde este archivo en lugar de a través de pip3. El archivo Dockerfile deberá contener la siguiente información para instalar y ejecutar pipenv con los valores correctos en cualquier entorno al que se vaya a desplegar:

```
RUN pip3 install pipenv
COPY Pipfile ./
COPY Pipfile.lock ./
RUN set -ex && pipenv install --deploy --system
```

Por último, si se realizan cambios en el fichero Pipfile será necesario actualizarlos en Pipfile.lock a través de: pipenv lock. Una forma de establecer una librería con una versión concreta conocida como compatible con el entorno sería así:

```
[packages]
dash="~=1.0.2"
dash-daq="~=0.1.0"
```

3.2 Bases de datos y arquitectura cloud - MongoDB & AWS

3.2.1 MIGRACIÓN BASES DE DATOS ACTUALES A AWS

Para la migración de las bases de datos a la arquitectura de AWS será necesario utilizar la herramienta Mongodump [18]. Para exportar las bases de datos a la carpeta /dump se utiliza el siguiente comando:

```
mongodump --host IP -d DATABASE --port PORT --username USER --password PASS
```

```
2019-07-11T17:53:53.440+0100    [........................]          moneydashboard.2017MDB   100MB/5.25GB   (1.9%)
2019-07-11T17:53:53.440+0100    [#.......................]          arbor.dev_transactions   104MB/2.32GB   (4.4%)
2019-07-11T17:53:53.441+0100    [###.....................]        afterbanks.transactions    104MB/719MB  (14.4%)
2019-07-11T17:53:53.443+0100    [........................] moneydashboard.dev_transactions  78.0MB/4.46GB   (1.7%)
```

Figura 26. Proceso de mongodump

Para restaurar una base de datos conforme a los ficheros exportados anteriormente se utiliza Mongorestore a través el siguiente comando:

```
mongorestore --host IP --port PORT --username USER --password PASS ./dump
```

Para crear usuarios y establecer roles se sigue el siguiente procedimiento:

```
mongo admin --username USER --password PASS
db = db.getSiblingDB('database')
> db.createUser( { user: "USER", pwd:"PASS", roles: ["readWrite", "dbAdmin" ]} )
[{"_id" : "bankia.root",
  "userId" : UUID("7eaf64f0-146a-43b7-a09a-d8c2cdd8641a"),
  "user" : "root",
  "db" : "bankia",
  "roles":[{"role":"readWrite","db":"bankia"},{"role":"dbAdmin","db":"bankia"}],
  "mechanisms" : ["SCRAM-SHA-1","SCRAM-SHA-256"]}]
```

3.2.2 ACTUALIZAR COLECCIONES DE GOOGLE CLOUD A AWS

Para actualizar las colecciones existentes en AWS conforme a las de Google Cloud se puede utilizar también mongoexport:

```
mongoexport --host IP --port PORT --username USER --collection COLL --db DB --out FILE.json
```

A continuación, se elimina la colección actual de AWS:

```
db.COLLECTION.remove({})
```

Y por ultimo se utiliza mongoimport:

```
mongoimport --host  IP --port PORT --username USER --password "PASS" --collection COLL --db DB --file FILE.json
```

3.3 Diseño del dashboard - Python Dash app layout & libraries

3.3.1 CONFIGURACIÓN DEL DASHBOARD CON DASH

Dash está construido sobre Flask, es de código abierto y lo que va a permitir es crear el panel con Python y ejecutar la disposición de componentes en el navegador web. Podríamos decir que se trata de una librería centrada en el FrontEnd con una potente configuración de componentes basadas en los gráficos interactivos de las librerías de Javascript Plotly.js, React y React JS. Esto va a permitir, en cualquier momento que sea necesario, un nivel de personalización óptima y nos permite la creación de los nuestros propios componentes a través de estas librerías de Javascript.

Dash diferencia sus aplicaciones en dos partes bien diferenciadas. La primera parte está orientada en el diseño de los componentes y la segunda parte está más orientada en la interactividad de la aplicación.

En primer lugar, será necesario instalar las librerías dash y dash-daq [19]. Dash proporciona las clases apropiadas para generar los componentes HTML a través de Python con sencillos bloques JSON de configuración.

Figura 27. Primera gráfica con Dash

Se pasa a configurar el dashboard [20] y realizar una primera versión sobre la que ir mejorando. Se utilizará la clase Graph de dash_core_components para generar las gráficas interactivas usando plotly.js. Esta clase dibujará un componente ´figure´ por medio de los datos establecidos en ´data´, con las características que describe su ´layout´ y un 'style' personalizado como se detalla en las siguientes estructuras de componentes:

Para inicializar la aplicación será necesario llamar a la clase Dash:

```
app = dash.Dash()
```

Para atribuirle un diseño:

```
app.layout = html.Div(style={},children=[])
```

Para configurar el componente gráfico:

```
dcc.Graph(id='Graph', figure = {'data':[{'x':list(),'y':list(),'type':type,'name':name}],
'layout':{}})
```

Existen multitud de vías para la carga de datos. En primer lugar, la carga de datos a través de un fichero JSON sería así:

```
with open('bankia_CustomerObject.json') as file:
    data = json.load(file)
savingCapacities = []
for transaction in data['transactions']:
    savingCapacities.append(
        transaction['transactionCalculations']
            ['savingCapacity'])
```

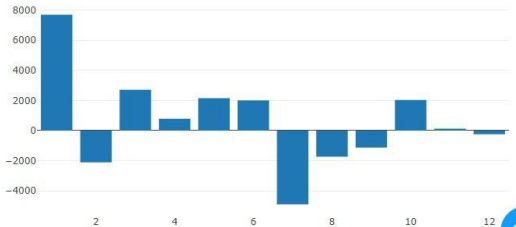


Figura 28. Gráfica a partir de fichero JSON

De esta manera se puede construir un gráfico estableciendo en el eje X los meses y en el eje Y, el cálculo de la capacidad de ahorro de uno de los clientes.

```
'data': [{'x': [1,2,3,4,5,6,7,8,9,10,11,12],'y': datos, 'type':'bar','name':'SF'}]
```

Sería bueno tener un componente en el tablero que mostrara el número total de registros en cualquier colección de la base de datos en tiempo real, y así poder mostrar como realizar esta conexión. Se utiliza la librería pymongo de la siguiente manera:

```
import urllib.parse
from pymongo import MongoClient

username = urllib.parse.quote_plus('username')
password = urllib.parse.quote_plus('pass')
client = MongoClient('mongodb://%s:%s@IP:PORT/database'
                % (username, password))
db = client.get_database()
var1 = db.dev_customers.count_documents({})
```

Y se muestra el valor de la siguiente manera:

```
html.Div(children='''%s No. of Wells''' % var1)
```

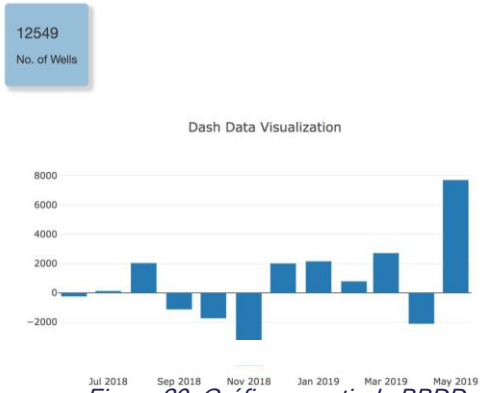


Figura 29. Gráfica a partir de BBDD

A continuación, se carga un estilo visual a través de un fichero style.css alojado en la carpeta /assets (que es cargado automáticamente en la inicialización del tablero) y se obtiene una primera visualización del tablero poniendo en práctica el método de obtención de datos a través de un fichero y a través de la conexión a la base de datos.

Por último, el aspecto de Dockerfile luce de la siguiente manera, destacando la línea "COPY . ." que tiene como objetivo la mejora del tiempo de construcción de la imagen, al no necesitar descargar continuamente las librerías una y otra vez:

```
FROM python:3
RUN pip install dash==1.0.2
RUN pip install dash-daq==0.1.0
EXPOSE 8050
COPY . .
CMD ["python", "./my_script.py"]
```

A) El papel de las funciones callbacks en Dash

Estas funciones proporcionan interactividad a la aplicación a través de componentes de entrada y salida de forma declarativa a través del decorador: `@app.callback`. En Dash, estas entradas y salidas son sencillamente las propiedades de los componentes. Cada vez que una propiedad de entrada cambia, la función callback será llamada automáticamente y proporciona un nuevo valor a la propiedad de salida. Más adelante se profundizará más en detalle en este aspecto para mejorar el desempeño de la aplicación.

3.3.2 BOOTSTRAP PARA MEJORA DE DISEÑO DEL DASHBOARD

Aunque Dash cuenta con una interfaz atractiva, se decide utilizar Bootstrap con Dash para ofrecer un mejor aspecto visual, una mejor disposición de componentes y un diseño responsive/adaptable. La instalación [21] requiere el paquete dash-bootstrap-components y a través del siguiente comando se añaden las stylesheets con un tema alternativo precompilado a la aplicación:

```
app = dash.Dash(__name__, external_stylesheets=[dbc.themes.BOOTSTRAP])
```

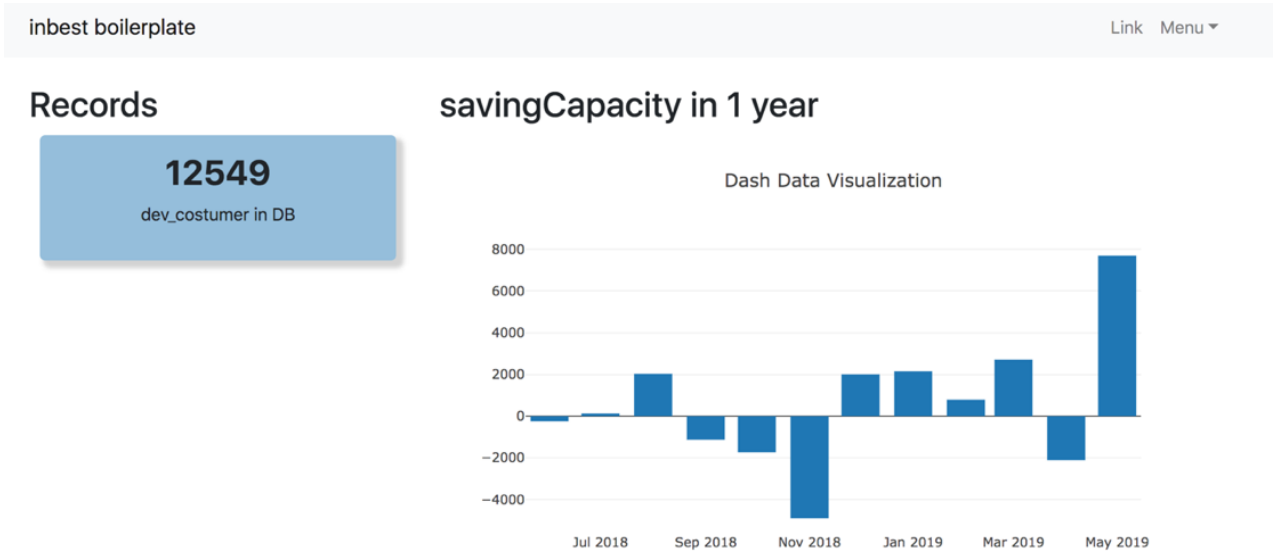


Figura 30. Aplicando Bootstrap

3.3.3 DISEÑO BASE

Para comenzar, se valora la utilización de algunos de los componentes prediseñados de Bootstrap para Dash. Además, se debe tener en cuenta algunos de los componentes actuales de la antigua plataforma accesible desde:

https://inbest.shinyapps.io/riskAnalytics_BKIA/

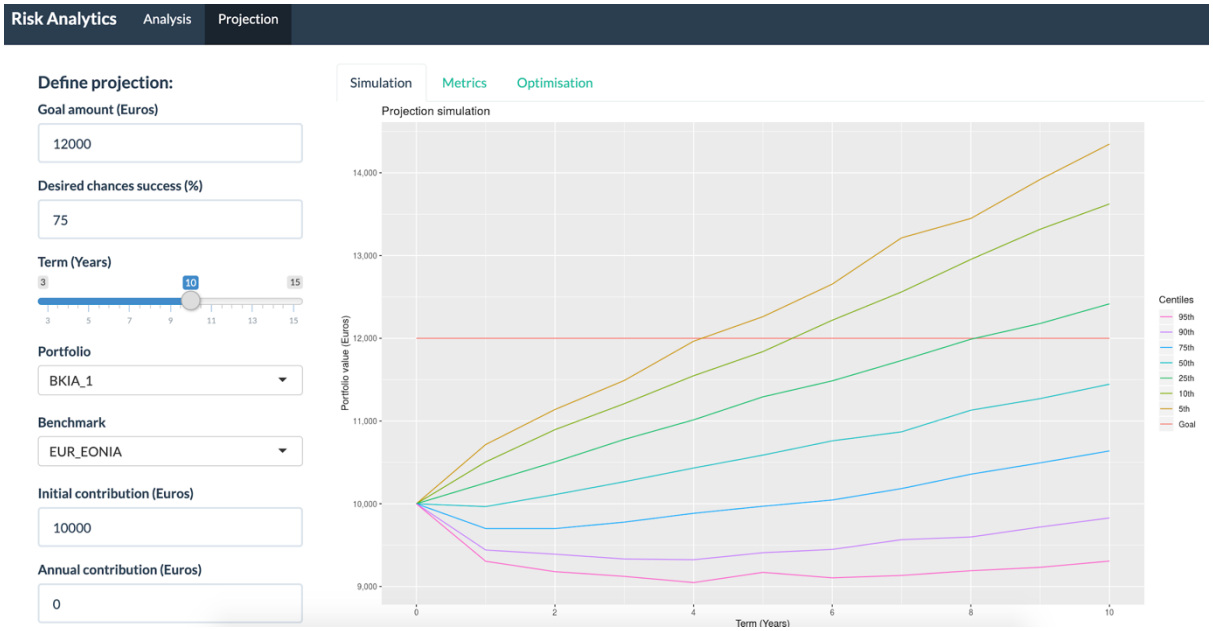


Figura 31. Vista de antigua plataforma

En la plataforma actual aparecen elementos de carga de componentes iniciales, filtros slider, filtros dropdown, pestañas, tablas de visualización de datos y gráficas. La vista es muy simple y aporta la información, pero requiere un elevado tiempo de carga de los datos y se trata de un modo poco intuitivo de entender a simple vista. La experiencia de usuario no está tan cuidada, y las gráficas, aunque son personalizables a través de los filtros, resultan difíciles de interpretar. Es prioritario un diseño simple que busque ofrecer una disposición de componentes con un mayor impacto de las comparativas y aporten valor desde el primer instante.

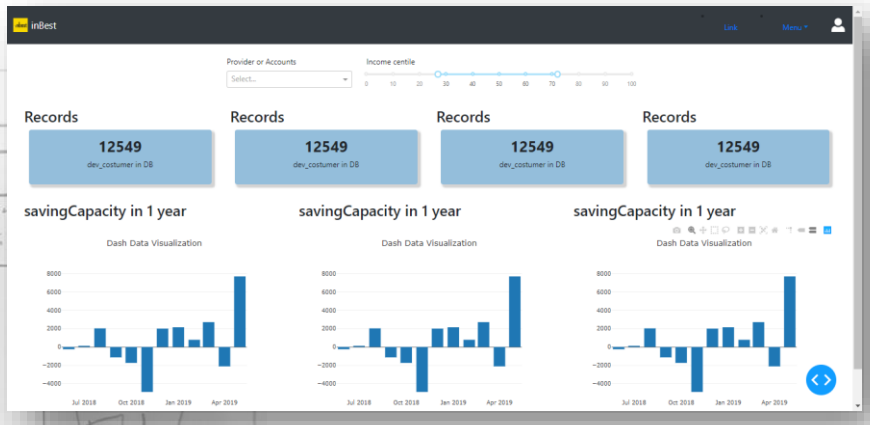


Figura 32. Diseño base puesto en marcha

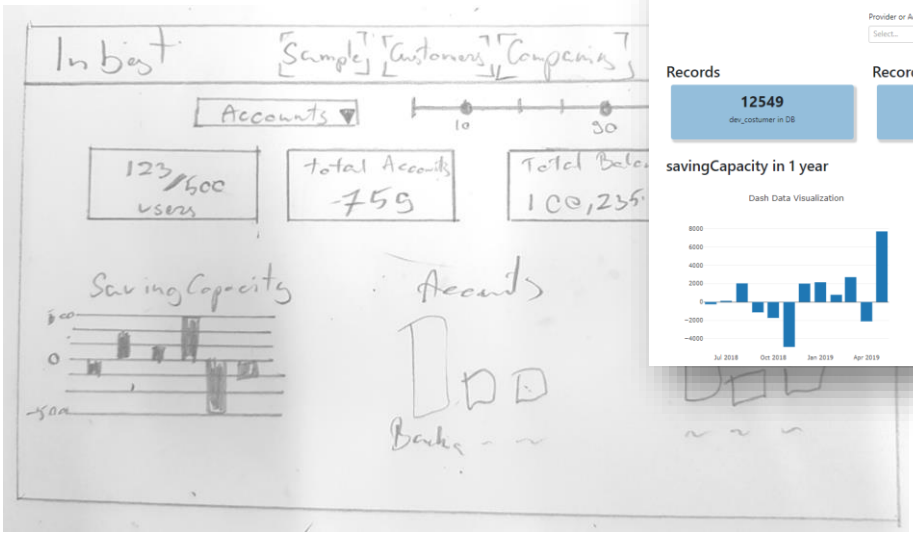


Figura 33. Boceto de diseño base

Las secciones serían: barra superior, módulo de filtrado, módulo de registros estadísticos y módulo de gráficos. En este caso resulta de gran importancia la simplicidad acompañada del valor que aporta esta disposición de módulos y así, de esta manera, podemos tener un primer layout base sobre el que trabajar en secciones y capítulos posteriores. La

organización a través de los componentes dbc.Col y dbc.Row ofrecen flexibilidad para ajustar correctamente los elementos.

3.3.4 MÓDULO DE REGISTROS ESTADÍSTICOS.

Una vez se tiene un primer diseño base inicial se pasa a desarrollar el módulo de registros estadísticos que muestran detalles de los registros o estadísticas sobre las colecciones que se están filtrando. Esto tiene importancia para saber en todo momento el número de usuarios filtrados y que serán analizados, el número de cuentas asociadas a esos usuarios y el número total de transacciones. De una manera simple y vistosa se va a poder tener constancia de las variables que se manejan en cada contexto en tiempo real.

All_Users	Users	Accounts	AccountsUser	Transactions	TransactionsUser
All_Users	22	263	12	17,766	808

Figura 34. Antiguos registros estadísticos

De esta manera, el valor que obtiene el analista es mucho más llamativo que la tabla de la plataforma inicial.

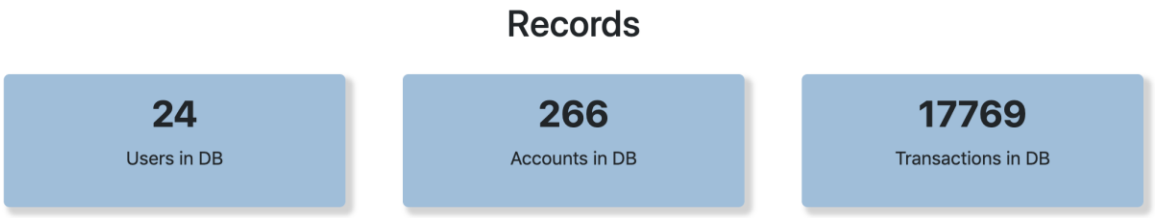


Figura 35. Nuevo módulo de registros estadísticos

3.4 Mejora de Interfaz de Usuario - Filtrado de datos en tiempo real

3.4.1 DISEÑO BASE DEL MÓDULO DE FILTRADO

Se pasa a mejorar la interfaz de usuario partiendo con el desarrollo del módulo de filtrado de datos en tiempo real. En un primer lugar, se hace uso de la librería de componentes de Bootstrap y, atendiendo al diseño propuesto, se decide aplicar en un primer lugar un filtrado simple con la configuración de un control deslizante que permite filtrar por el centil de ingresos de cada usuario y filtrar aquellos que se encuentren entre los valores mínimo y máximo seleccionados.

El componente de control deslizante en Bootstrap es un dcc.RangeSlider y tiene como características principales:

```
cc.RangeSlider(id="range-slider",
                min=0, max=100,
                step=1, value=[6, 81],
                marks={
                    0: '0',
                    10: '10',
                    …
                    100: '100'
                })
```

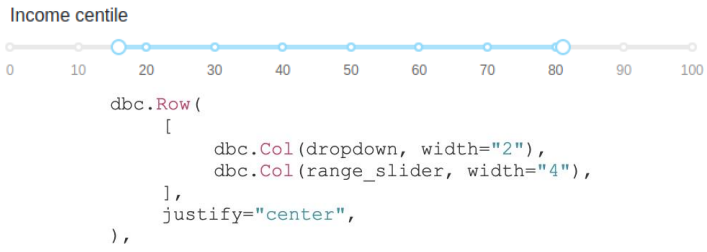


Figura 36. Slider del módulo de filtrado

Para obtener los valores del control deslizante en cada una de las funciones de la aplicación se deberá establecer el componente como entrada en la función de callback atendiendo al ID del componente y la propiedad que se desea utilizar como 'listener'. Cuando esta propiedad cambia, ejecuta la función con el valor de entrada del componente y devuelve un valor a la propiedad del componente de salida:

```
@app.callback(
Output(component_id='figura1', component_property ='figure'),
[Input(component_id='range-slider', component_propety ='value')])
def function(rangeSliderTupleValue):
    …
    valor1 = rangeSliderTupleValue[0]
    valor2 = rangeSliderTupleValue[1]
    …
    return figure
```

Las dependencias Input y Output se deberán importar desde la librería dash.dependencies. También se podrá incorporar la dependencia State para evaluar el estado o valor de una propiedad específica. No funciona como listener y trigger/lanzador, pero sí que se podrá utilizar como entrada en la función y podrá ser evaluado.

```
@app.callback(
Output(component_id='figura1', component_property ='figure'),
[Input(component_id='range - slider', component_propety ='value')],
State(component_id='range - slider', component_propety ='marks'))

def function(rangeSliderTupleValue, rangeSliderMarks):
    …
    valor1 = rangeSliderTupleValue[0]
    valor2 = rangeSliderTupleValue[1]
    marks = rangeSliderMarks
    …
    return figure
```

3.4.2 CONEXIÓN DEL COMPONENTE DE FILTRADO A LA FUNCIÓN DE FILTRADO

A continuación, se desarrolla una función callback multi-salida que actúe conforme a unos datos de entrada y devuelva cada uno de los objetos de salida a los componentes. La estructura de la función tras recibir los valores del componente de filtrado como entrada pasa por:

- Obtener los datos de los usuarios de la base de datos.
- Filtrar los datos a través de la función filterByIncomeCentile pasándole como argumentos los datos, el valor mínimo y el valor máximo.

- Obtener los registros estadísticos a través de las funciones getNumberOfUsers, getNumberOfAccounts y getNumberOfTransactions.
- Obtener los valores de capacidad de ahorro de cada usuario filtrado y su listado de fechas asociadas.
- Configurar el objeto figure que devolver, estableciendo en el eje x la lista de fechas y en el eje y la lista con los valores de capacidad de ahorro.
- Por último, devolver los valores para el registro estadístico y la gráfica.

La función filterByIncomeCentile tendría la siguiente estructura:

```
from scipy.stats import percentileofscore
def filterByIncomeCentile(data, min, max):
    usersIncomes = []
    for user in data:
        userTotalIncome = calculateUserTotalIncome(user["transactions"])
        user['totalIncomeTransactions'] = userTotalIncome
        usersIncomes.append(userTotalIncome)

    filtered_users = []
    for user in data:
        value = percentileofscore(usersIncomes, user['total_income_transactions'])
        if (value >= min_value and value <= max_value): filtered_users.append(user)
    return filtered_user

def calculateUserTotalIncome(transactions):
    userTotalIncome = 0
    for transaction in transactions:
         userTotalIncome += transaction['income']['recurrent']['total'] +
                                    transaction['income']['nonRecurrent']['total']
return userTotalIncome
```

Primero es necesario calcular el valor de los ingresos totales (recurrentes y no recurrentes) de cada usuario a través de la función calculateUserTotalIncome. Una vez son almacenados en una lista, se va iterando por cada usuario y se calcula cada valor centil de cada usuario determinado por la función percentileofscore a través de la clase scipy.stats. Esta función utiliza la lista completa de todos los usuarios y establece un ranking de cada usuario conforme al valor de sus ingresos totales para que posteriormente se pueda aplicar el filtro sobre este valor y si lo cumple, incluir al usuario en la lista de usuarios filtrados. El percentil del 80% de un valor X significa que el 80% de todos los valores dados están por debajo del valor X.

La función getSavingCapacity a la que se le pasa, por el momento, las transacciones de un único usuario de prueba, reúne la lista con los valores de capacidad de ahorro respecto al tiempo del usuario y una lista con las fechas asociadas a estos valores para posteriormente utilizarlos en los distintos ejes de la gráfica.

3.4.3 ACTUALIZAR Y MOSTRAR MARCAS SELECCIONADAS DEL FILTRO DESLIZANTE

Para que aparezcan los valores seleccionados exactamente en la parte inferior del control deslizante y así tener que incluirlos en la propiedad de marcas del componente, se deberá configurar la función getMarks(min,max) y devolver el diccionario de marcas generado en cada actualización del componente incluyendo los valores máximo y mínimo de la siguiente manera:

```
def getMarks(min, max):
    marks = {
        0: '0',
        …
        100: '100'}
    marks.update({min: str(min)})
    marks.update({max: str(max)})
    return marks

@app.callback(
    [...Output(component_id='range-slider', component_property='marks')],
    [Input(component_id='range-slider', component_property='value')])
def update_slider(rangeSliderTupleValue):
    min = rangeSliderTupleValue[0]
    max = rangeSliderTupleValue[1]
    marks = getMarks(min, max)
    ...

return ...marks
```

3.4.4 IMPLEMENTACIÓN DE LOS CAMBIOS Y MEJORAS ASOCIADAS AL FILTRADO

Durante la realización del pull request con los cambios para realizar el filtrado sobre los registros de la base de datos se requieren varios cambios a efectuar para completar el código de manera adecuada:

- Mover todo el material de conexión a la base de datos a un nuevo archivo llamado connection.py. Este archivo es responsable de la conexión a la base de datos y de la recuperación de datos siguiendo el modelo “Data Access Object (DAO)” [22]. La función findByQuery(variableName, variableValue) buscará en el campo de la base de datos el valor específico y devolverá una serie de objetos asociados.
- Tener en consideración la utilización de nombres de variables más largos pero que expliquen de qué tipo de valor se trata con solo mirar su nombre. Como, por ejemplo, si se requiere una variable para el valor de las transacciones, no sería adecuado una variable con el nombre “trans”. O si se trata de la suma total de transacciones, podría llamarse totalTransactions.
- Cuando se establece una variable que actúa de contador, lo más adecuado sería llamarle por el tipo de contador que se refiere, no llamarlo tan sólo “contador” o “counter”.

- Normalmente se debería dividir el código en funciones para que sólo tengan una responsabilidad, esto ayuda a la hora de modular.
- Se deberá establecer una única conexión a la base de datos. En el fichero connection.py debería tener la variable de MongoClient que establece la conexión con MongoDB declarada como variable global para ese fichero y así poder utilizarlo en el resto de las funciones. Si la conexión se realiza una vez globalmente, cada vez que sea necesaria una consulta, no se tendrá que volver a realizar una nueva conexión de nuevo con MongoClient y así evitar tensión en el servidor de MongoDB.
- Para los nombres de funciones no usar el carácter "_". Se deberá utilizar camelCase, en concreto utilizaremos lowerCamelCase [23], con la primera letra de cada palabra en mayúscula a excepción de la primera palabra que es minúscula. Se trata de un estilo de escritura aplicada a frases o palabras compuestas con la finalidad de facilitar la lectura de las palabras. La ventaja principal en desarrollo de software es que, al tener que utilizar palabras contiguas, se pueden crear nombres más significativos utilizando una secuencia descriptiva de palabras que reflejen claramente el propósito de la variable y sin violar la limitación de nombres.
- Para instalar las dependencias, como se ha comentado, se utilizará Pipenv. Por lo que, para cada nuevo uso de librerías se deberá utilizar la secuencia: pipenv install <package>. Para estar seguro de que se han instalado automáticamente en el PipeFile se puede usar Docker.

3.4.5 DESARROLLO DE NUEVA FUNCIÓN PARA FILTRAR POR INGRESOS MÍNIMOS Y MÁXIMOS

Se decide tener otra opción para filtrar por el total de ingresos y se decide la realización de un selector para seleccionar entre el filtro realizado anteriormente por centil y un nuevo filtro por ingresos totales mínimo y máximo. De esta manera se puede tener otro filtrado alternativo.

La disposición sería en dos columnas. La primera columna estaría dividida en dos filas, en la primera fila estará el selector para seleccionar entre los dos filtros que estarán en la segunda fila para filtrar por centil o filtro por ingresos mínimo y máximo a través de dos inputs numéricos, y en la segunda columna aparecerá el filtro por edad.

En primer lugar, será necesario recuperar e importar la librería que se cargó en la instalación de Dash: dash_daq. Dash DAQ se trata de una biblioteca de controles que tiene el objetivo de simplificar el desarrollo de estos componentes y las acciones necesarias para el correcto funcionamiento y control del dashboard. Se recuerda que todas las dependencias deberán ser incluidas en el fichero PipeFile y PipeFile.lock.

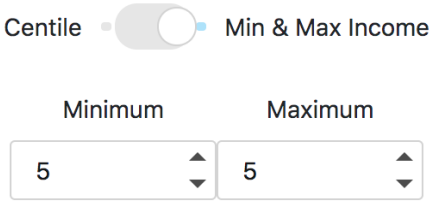


Figura 37. Filtro por ingresos mínimos y máximos

En primer lugar se hará uso de dos componentes daq.NumericInput para la entrada de los valores mínimo y máximo por los que se desea filtrar a los usuarios. El componente facilita la configuración del cuadro de entrada numérico ofreciendo la posibilidad de establecer un tamaño, una etiqueta superior y los valores máximo y mínimos permitidos. Para la realización del selector se utilizará el componente daq.ToggleSwitch que se encargará de seleccionar entre las dos posibilidades de filtrado teniendo en cuenta los ingresos totales del usuario. Ambos componentes irán compuestos cada uno en un componente dbc.Row y finalmente ambas filas compuestas en un componente dbc.Col.

Cabe destacar que es muy importante realizar la configuración de los componentes sin la utilización del componente Form. Esto podría refrescar continuamente la página conforme las entradas reciban un estímulo. El método para decidir qué cambios efectuar ante un estímulo sigue siendo, como se ha comentado en capítulos anteriores, a través de las llamadas a callbacks. En este caso será necesario una función sencilla que decide qué filtro mostrar cuando se acciona el selector:

```
app.callback(
    Output('typesOfFilter', 'children'),
    [Input('selector', 'value')])
def turn_dark(valor):
    if (valor):
        return numeric_component
    else:
        return range_slider
```

3.4.6 DISPOSICIÓN DEL MÓDULO DE FILTRADO

Se decide incluir finalmente el módulo de filtrado al mismo nivel que el módulo de registros estadísticos y así, de la siguiente manera, aprovechar mejor el espacio:

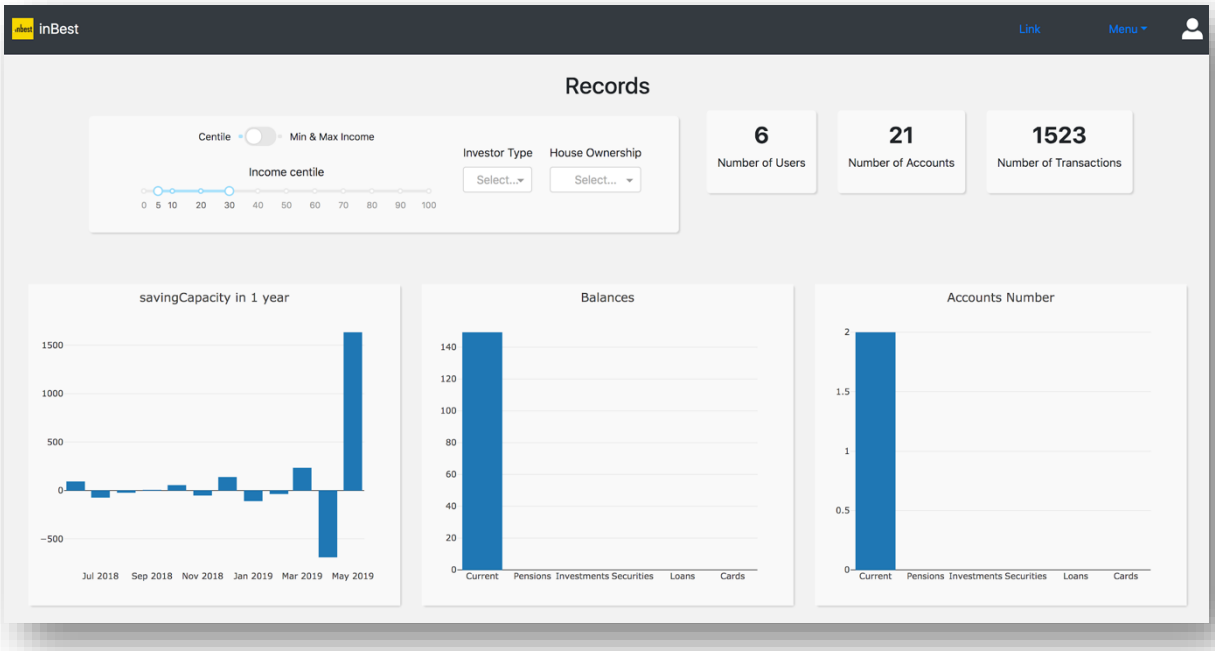


Figura 38. Disposición del módulo de filtrado

3.4.7 DESARROLLO DE NUEVA FUNCIÓN PARA FILTRAR POR EDAD MÍNIMA Y MÁXIMA

En un principio se pensó incluir dos componentes NumericInput, pero resulta mucho más práctico un control deslizante con los valores de edad como escala. La estructura del componente control deslizante es la misma que en apartados anteriores.

Figura 39. Módulo de filtrado al completo

Se define la función filterByAge que, a través de los datos de los usuarios, el valor mínimo y el valor máximo, devolverá una serie de usuarios que cumplen las especificaciones de manera similar a la anterior función de filtrado. Por cada usuario, y sólo si existe un campo "dateOfBirth" en el apartado "personalInformation" (si no existe este parámetro devuelve un Trackerror), calcula su edad y si este valor está dentro del límite establecido, añade el usuario al objeto de retorno con el resto de los usuarios que cumplen el filtro.

Para el cálculo de la edad se hace uso de la función calculateUserAge:

```
def calculateUserAge(dateOfBirth):
    today = date.today()
    age = today.year - dateOfBirth.year - ((today.month, today.day) < (dateOfBirth.month, dateOfBirth.day))
    return age
```

Usando la librería "date" primero se resta el año de nacimiento al año actual. Se evalúa si el mes y el día actual es menor que el mes y el día de nacimiento. En caso afirmativo resta 1, de lo contrario resta 0.

3.4.8 CREACIÓN DE LA FUNCIONALIDAD DE VISUALIZACIÓN POR COMPARATIVA

Se considera importante un módulo que ofrezca la posibilidad de segmentación por tipo de propietario de vivienda y segmentación por tipo de usuarios ahorradores y no ahorradores. Este módulo debería tener un impacto visual y funcional más allá del simple filtrado de usuarios. En realidad, no se trata de filtrado de usuarios como tal, el número de usuarios es el mismo, en este caso se pretende distribuir los usuarios por tipo de segmentación para valorarlos independientemente en cada gráfica como dos orígenes de datos distintos y así tener al instante la diferenciación entre cada tipo de usuario. Su diseño sería así:

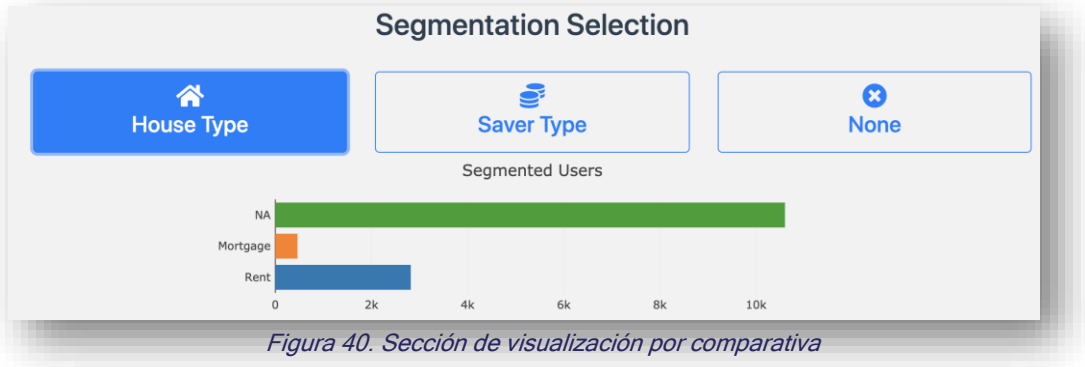


Figura 40. Sección de visualización por comparativa

Por defecto será inicializado a “None” para no aplicar segmentación. Cuando se aplica alguno de los dos tipos de segmentación, se visualizará justo debajo un resumen de usuarios según el grupo de segmentación que refleja la proporción de cada tipo de usuario en cada grupo. Se utiliza un componente gráfico similar a los anteriores y se devuelve la propiedad “figure” que pintar en la gráfica segmentation-stats-graph:
La función en un principio filtra los usuarios y conforme el tipo de selección que sea pulsado, obtiene el objeto graphXandXDate con cada uno de los objetos graphData y los usuarios que cumplen las especificaciones. La función que realiza las acciones se encuentra en el fichero figureHelpers, que actúa como módulo en el que desarrollar todas las funciones que ayudan a dibujar las gráficas:

```
def updateHorizontalBarFigure(graphXandYData, title):
    figure = {
        'data': [],
        'layout': {
            'title': title,
            ...}}
    for graphData in graphXandYData:
        dataObject = {
            "type": graphData.getType(),
            "name": graphData.getName(),
            "x": graphData.getX(),
            "y": graphData.getY(),
            "orientation": "h",
            "showlegend": False,
        }
    figure["data"].append(dataObject)
return figure
```

Lo que se consigue es que a través de este objeto graphXandYData que se tiene como entrada, se puedan insertar en la propiedad “figure” cada uno de los dataObject o tipo de usuario atendiendo a la clase creada GraphData.

A) Clase GraphData

La clase GraphData viene a solucionar el problema de la superposición de orígenes o tipo de datos en una misma gráfica para diferenciar entre uno u otro. Tiene los atributos name, graphType, x e y. Se establece el nombre del tipo u origen de datos, el tipo de gráfica que dibujar y los datos en el eje X respecto a los del eje Y.

Cada objeto GraphData puede obtener características como nombre, variables X, variables Y y tipo de gráfica. En cada uno de estos objetos GraphData estarán contenidos cada tipo

de usuarios que cumplen los requisitos. En secciones posteriores se comentará el modo de separación por tipo de usuarios.

B) Desarrollo de nueva función de filtrado por tipo de propietario de vivienda

Se pueden diferenciar tres tipos de usuarios según tipo de propietario de vivienda. Los que viven de alquiler, los que tienen hipoteca y los que no aplican a ninguno de los dos tipos. Por cada usuario, se evalúa el CustomerObject y si su información personal cuenta con característica houseOwnership con valor rent, se incluye en la lista rentList con el resto de los usuarios que viven de alquiler. Si, por el contrario, su información personal cuenta con característica houseOwnership con valor mortgage, se incluye en la lista mortgageList con el resto de los usuarios que viven con hipoteca. Y si no aplica a ninguno de los dos, se incluye en la lista naList:

```
def filterByHouseOwnership(users):
    rentList = []
    mortgageList = []
    naList = []
    for user in users:
        if 'houseOwnership' in user["personalInformation"]:
            if user["personalInformation"]["houseOwnership"] == 'rent':
                rentList.append(user)
            elif user["personalInformation"]["houseOwnership"] == 'mortgage':
                mortgageList.append(user)
        else:
            naList.append(user)
    return rentList, mortgageList, naList
```

C) Desarrollo de nueva función de filtrado de usuarios ahorradores/no ahorradores.

Se pueden diferenciar dos tipos de usuarios atendiendo a si son ahorradores o no ahorradores. Por cada usuario, se evalúa el CustomerObject y si su información personal cuenta con característica firstSavingDate se incluye en la lista savers con el resto de los usuarios ahorradores. Y si, por el contrario, su información personal no cuenta con característica firstSavingDate se asume un usuario no ahorrador y se incluye a la lista nonsavers con el resto de los usuarios no ahorradores:

```
def filterBySaver(users):
    savers = []
    nonsavers = []
    for user in users:
        if 'firstSavingDate' in user["personalInformation"]:
            savers.append(user)
        else:
            nonsavers.append(user)
    return savers, nonsavers
```

3.5 Mejora de Experiencia de Usuario - Visualización de datos

3.5.1 DISEÑO DE PESTAÑAS PARA MOSTRAR NUEVOS MÓDULOS

Una forma sencilla de visualizar distintas secciones en una única página es a través de las pestañas. Dash ofrece la posibilidad de configurar las pestañas [24] de manera sencilla.

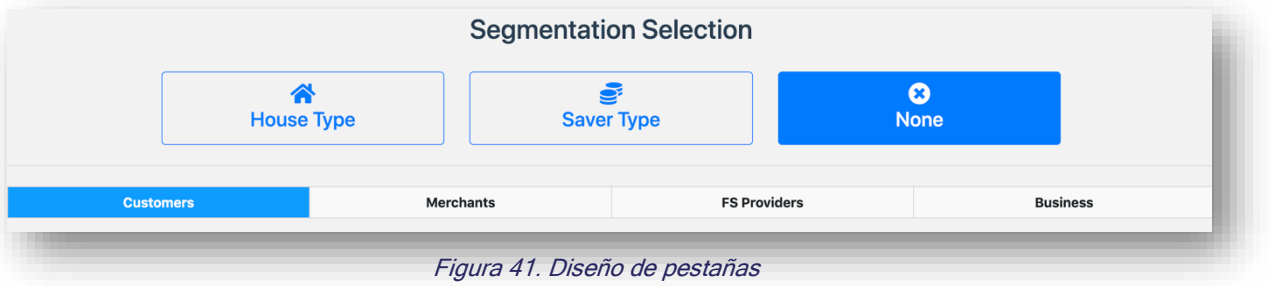


Figura 41. Diseño de pestañas

Se establecen los valores de configuración de estilo de las pestañas en el archivo style.css que será cargado automáticamente desde la carpeta /assets. Las clases son .tab_styles, .tab_style y .tab_selected_style. En cada una de las características de estilo es importante marcarlas como '!importart', para que sean tomadas como valor final, en vez del valor por defecto tras el renderizado y así evitar que se sobre-escriban estos valores.

Por último, se añade el componente dcc.Tabs como grupo y los componentes dcc.Tab como componentes individuales de cada sección con un label, un valor y un estilo configurado. Para establecer estilos según las pestañas seleccionadas se realiza a través de una callback que toma como valor de entrada el valor seleccionado y devuelve una figura del contenido a mostrar.

A) Otra forma de mostrar las pestañas - Sin callback

Otra forma de mostrar las pestañas sin la utilización de la función callback sería configurando los componentes dcc.Tab según el Método 2 de la ayuda de Dash a través de configurar los componentes HTML en la propiedad 'children' de cada pestaña. El componente grupo dcc.Tabs es el que almacena el valor de la pestaña que está seleccionada. Cada una de las pestañas aplica uno u otro estilo conforme ha sido seleccionada o no.

3.5.2 DISEÑO DEL MÓDULO DE GRÁFICOS SEGÚN PESTAÑAS

Hasta ahora, la analítica de saldos y cuentas de los usuarios reflejan datos interesantes, pero se debe seguir apostando por buscar servicios diferenciales. En esta sección será objeto de análisis el concepto en el que cada mes los usuarios invierten su dinero, para así valorar y hacer un estudio de las situaciones con un mayor grado de certeza y exactitud.

A continuación, se detalla el diseño del módulo de gráficos atendiendo a la información que se va a querer ver reflejada en las gráficas. Para empezar, se utilizará el diseño de las pestañas de la sección anterior para disponer de un tablero single-page completo en el que ir interactuando entre ellas y se dividen entre:

1. Customers tab - Información referente a diferentes aspectos de clientes

    - Promedio de ingresos, respecto a gastos y capacidad de ahorro.
    - Distribución de la capacidad de ahorro según contribuciones.
    - Importe total de saldos según cada tipo.
    - Figura interactiva de evolución mensual.
    - Gráficas de distribución e histograma.

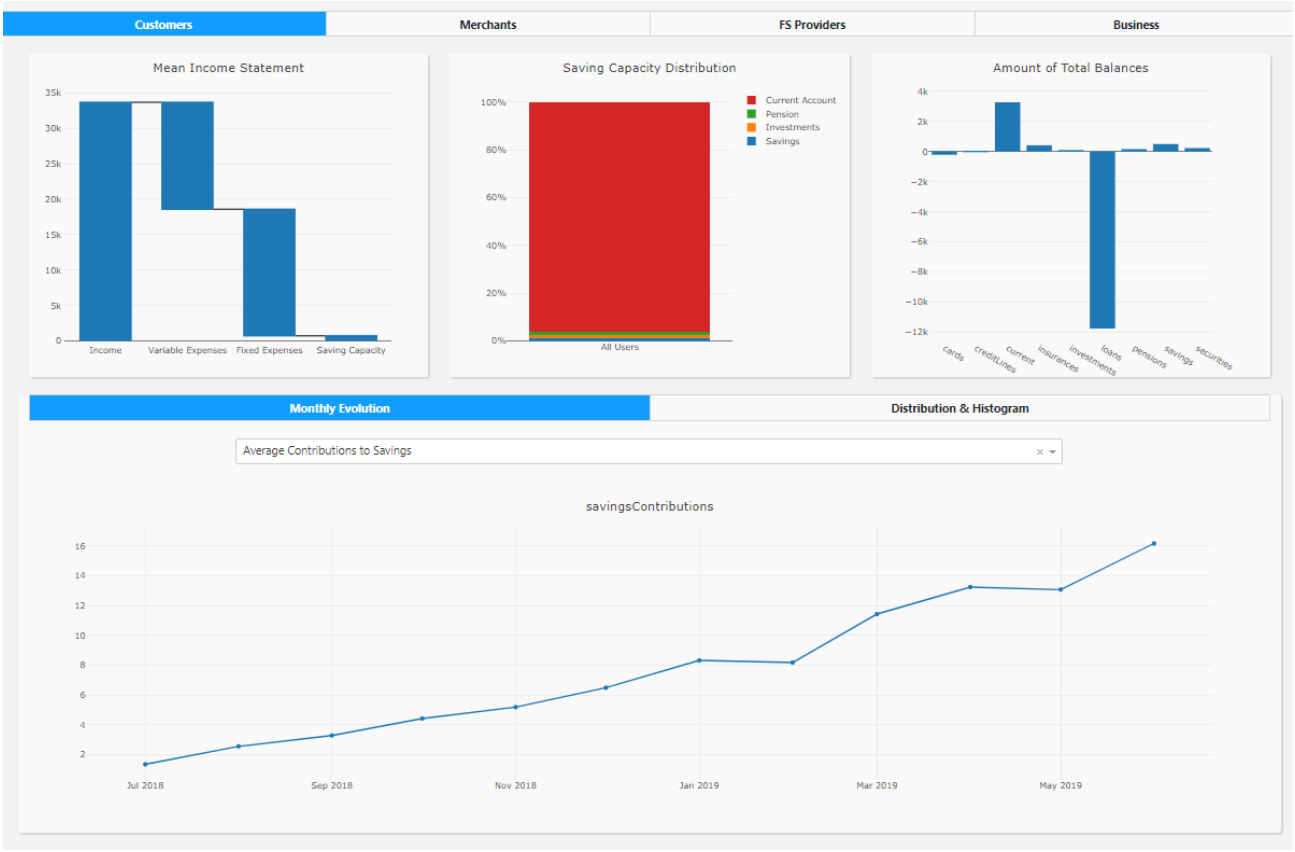


Figura 42. Vista de la pestaña Customers

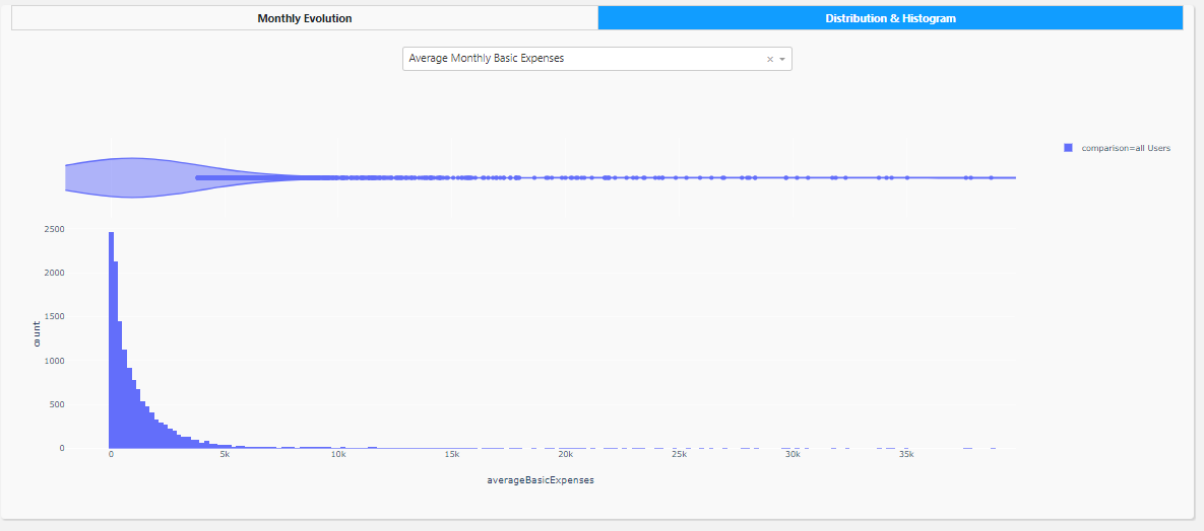


Figura 43. Histograma de la sección Customers

2. Merchants - Información analizada respecto a comercios/compañías

   Según tipo de comercio:
   - Gasto total por usuario
   - Transacciones totales por usuario
   - Importe medio por transacción
   - Histograma con importes de transacciones respecto al total.

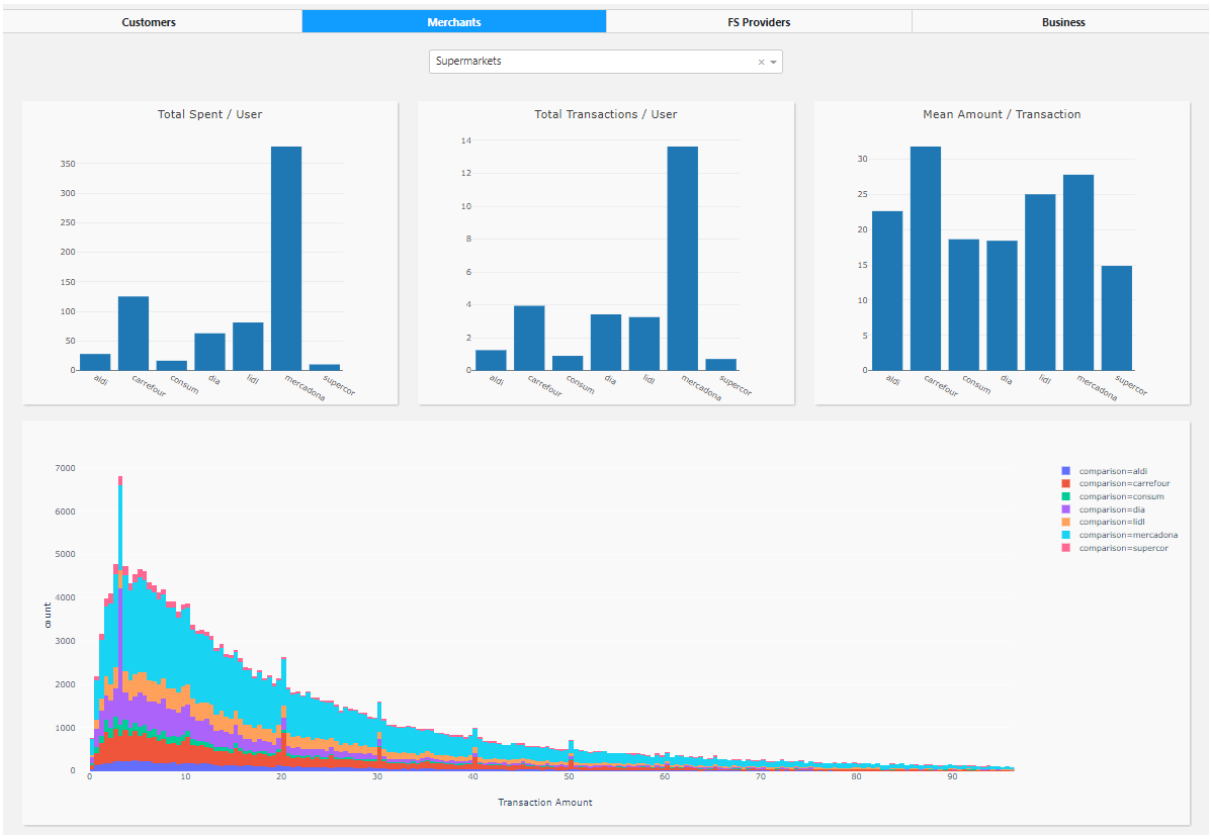


Figura 44. Vista de la pestaña Merchants

3. FS Providers - Referente a proveedores de servicios financieros

    - Saldos totales según cada tipo.
    - Número de cuentas respecto a cada tipo.

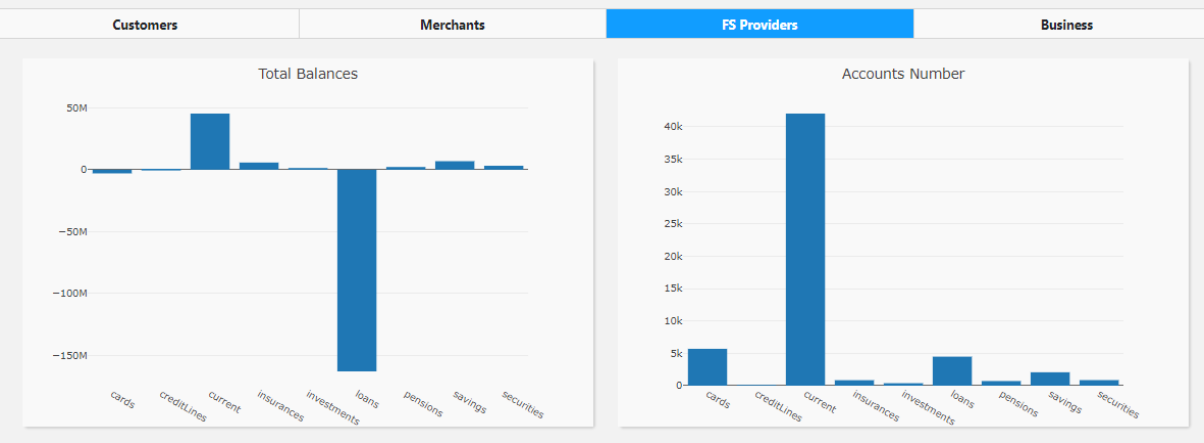


Figura 45. Vista de la pestaña FS Providers

4. Business - Aspectos relacionados con negocio

    - Número de personas ahorradoras del último año.
    - Importe medio de ahorro del último año
    - Balance acumulado del último año

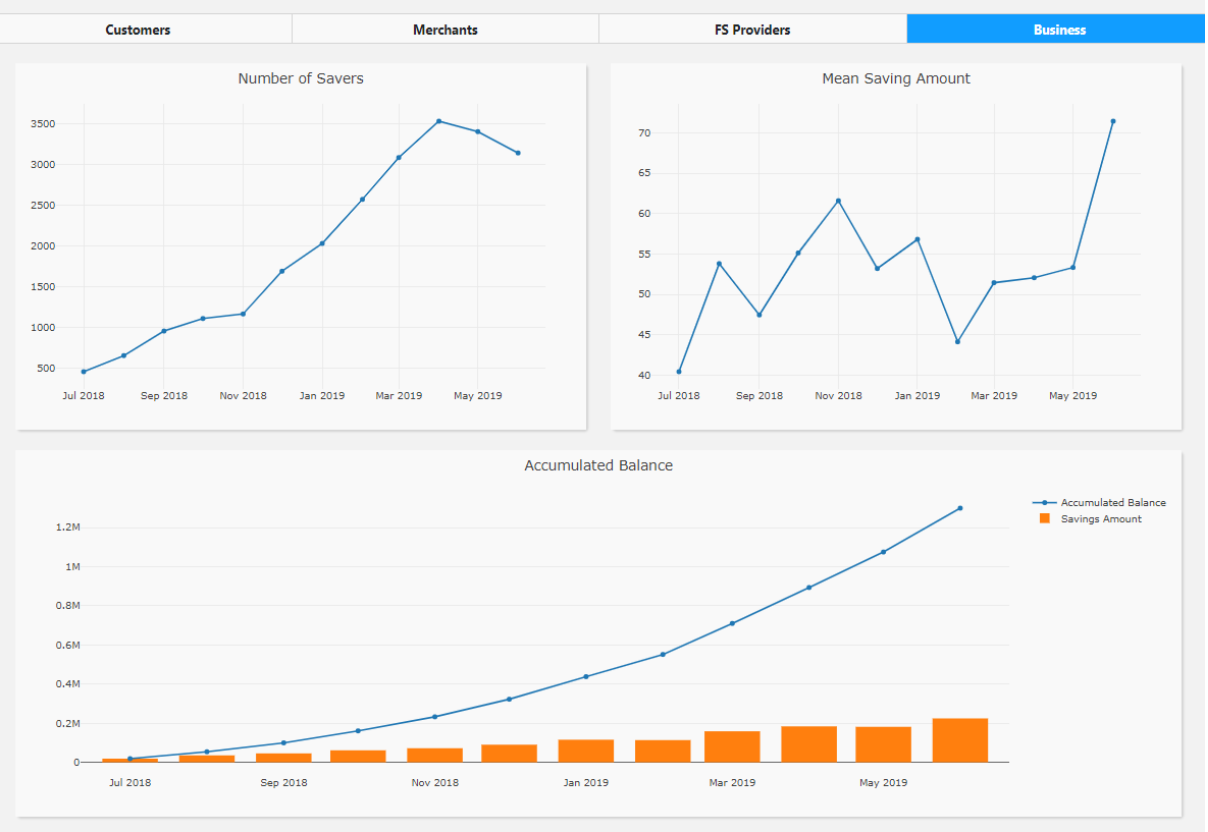


Figura 46. Vista de la pestaña Business

3.5.3 ANÁLISIS Y DESARROLLO DE TAREAS - CUSTOMER TAB

A) Promedio de ingresos, respecto a gastos y capacidad de ahorro.

Primero, se obtienen los valores medios de: ingresos totales (recurrentes y no recurrentes), gastos totales variables (discrecionales y de lujo), gastos totales fijos (básicos) y capacidades de ahorro, y se dividen entre el número total de clientes. A continuación, se crea la gráfica a través de la clase GraphData con los valores resultantes. Por último, se llama a la función updateWaterfallFigure con las clases GraphData creadas que devuelve la propiedad 'figure' lista para ser dibujada.

```
allIncomeStatement = figureFunctions.calculateIncomeStatement(data)
allIncomeStatementGraph = figureFunctions.GraphData("All Users",
                                                    "waterfall",
                                                    allIncomeStatement.keys(),
                                                    allIncomeStatement.values())
incomeStatementGraphs = [allIncomeStatementGraph]
...
incomeStatement = figureHelpers.updateWaterfallFigure(incomeStatementGraphs,
                                                    "Average Income Statement",
                                                    incomeStatementMeasure,
                                                    incomeStatementXValues)
```

B) Distribución de la capacidad de ahorro según contribuciones.

Para calcular los valores de capacidad de ahorro en el total de los usuarios primero es necesario obtener los valores de capacidad de ahorro de todas las transacciones de los últimos 12 meses por cada usuario. Si cada uno de estos valores de capacidad de ahorro es mayor que 0, se suma cada valor de cada categoría de contribución al total correspondiente. Sobre el valor total se divide entre el número de clientes y por último se obtiene en términos porcentuales.

A continuación, se obtienen los diferentes GraphData conforme a cada contribución que forman savingCapacityDistributionGraphs. Finalmente lo pasamos como dato de entrada a la función updateSavingDistributionFigure para configurar la propiedad 'figure' respecto a cada contribución.

```
savingCapacityDistributionData = figureFunctions.getSavingCapacityDistribution(filtered_users)
scSavings = figureFunctions.GraphData("Savings",
                                      "bar",
                                      ["All Users"],
                                      [list(savingCapacityDistributionData.values())[1]])
…
scCurrentAccount = ...
savingCapacityDistributionGraphs = [ scSavings...scCurrentAccount]
savingCapacity = figureHelpers.updateSavingDistributionFigure("Saving Capacity Distribution %",
                                                              savingCapacityDistributionGraphs)
```

C) Figura interactiva de evolución mensual durante el último año

Según el componente dropdown 'product_selected_y' se puede seleccionar entre los distintos elementos a analizar. La función calculateAxisValuesForSelection funciona como un conmutador de funciones y así, según el valor que se proporciona a través del dropdown, llama a la función encargada de realizar los cálculos pasándole como valor de entrada los CustomerObjects actuales.

```
interactiveGraph = figureFunctions.calculateAxisValuesForSelection(product_selected_y, filtered_users)
graphClass = figureFunctions.GraphData("All Users", "line",
                                       interactiveGraph.keys(),
                                       interactiveGraph.values())
interactiveGraphsData.append(graphClass)
interactiveFigure = figureHelpers.updateStackedFigure(product_selected_y, interactiveGraphsData)
```

El objeto product_selected_y contiene la siguiente selección de cálculos:

1.1. *Promedio de ingresos –> Ingresos recurrentes + no recurrentes*
1.2. *Promedio de gastos variables -> Gastos discrecionales + de lujo*
1.3. *Promedio de gastos básicos -> Gastos totales básicos*
1.4. *Promedio de gastos -> Gastos básicos + discrecionales + de lujo*
1.5. *Capacidad de ahorro promedio -> SavingCapacity Total*
1.6. *Promedio de contribuciones -> Por ahorro + Inversión + Pensión*
1.7. *Promedio de contribuciones de ahorro -> Sólo de tipo ahorro*

El procedimiento es el mismo para cada función de cálculo. Por ejemplo, para el primer caso, por cada usuario se suman los ingresos recurrentes y no recurrentes totales de sus correspondientes transacciones en un período de 12 meses. Por cada valor total de ingresos de cada usuario según la transacción, se incorpora al objeto averageIncomes si no existe la fecha correspondiente, y si existe, suma el importe correspondiente con clave la fecha estimada en el campo startDate de la transacción. Finalmente, una vez se tienen todos los valores totales de todos los usuarios respecto a una fecha, se divide cada uno de estos entre el total de usuarios. Sigue esta estructura:

```
def getAverageIncomeEvolution(customerObjects):
    totalCustomerObjects = len(customerObjects)
    averageIncomes = {}
    for user in customerObjects:
        for transactionObject in user["transactions"][:constants.NUMBER_OF_MONTHS]:
            startDate = transactionObject["startDate"]
            totalUserIncome = transactionObject["income"]["recurrent"]["total"] +
                              transactionObject["income"]["nonRecurrent"]["total"]
            if startDate in averageIncomes:
                averageIncomes[startDate] += totalUserIncome
            else:
                averageIncomes[startDate] = totalUserIncome
    for obj in averageIncomes:
              averageIncomes[obj] = round(averageIncomes[obj] / totalCustomerObjects)
       return averageIncomes
```

D) Clase InteractiveGraphData

La clase InteractiveGraphData viene a solucionar el mismo problema de superposición de orígenes o tipo de datos en una misma gráfica, pero con la diferencia de que, en este caso, se realiza sobre las gráficas interactivas a las que ya no importa un tipo de gráfica.

En este caso interesan los atributos name, x, y, xOrigin e yOrigin. xOrigin e yOrigin sirven de ayuda para establecer los datos de los marcadores que deberán establecerse como nuevo dataObject dentro del objeto de respuesta en la función que los evalúa. Por un lado, estarán las líneas que establecen los datos según las fuentes x e y, y sus marcadores asociados, con el mismo color, pero estableciendo otro nuevo

E) Gráfica de distribución e histograma

Igual que en la gráfica interactiva, se seleccionará el valor que mostrará la gráfica a través de del componente dropdown 'product_selected_histogram'.

A continuación, se construye cada objeto GraphData y finalmente se les pasa a través de histogramGraphData a la función updateHistogramFigure que es la función encargada de dibujar la gráfica respecto a los orígenes de datos correspondientes. En este punto es importante la primera prueba trabajando con Dataframes de la librería pandas.

```
histogramGraphAxisData = figureFunctions.calculateAxisValuesForSelection(product_selected_histogram,
                                                                         filtered_users)
histogramGraph = figureFunctions.GraphData("All Users",
                                           "histogram",
                                           histogramGraphAxisData.values(),
                                           [])
histogramGraphData.append(histogramGraph)
histogram = figureHelpers.updateHistogramFigure(histogramGraphData,
                                                product_selected_histogram,
                                                "violin")
```

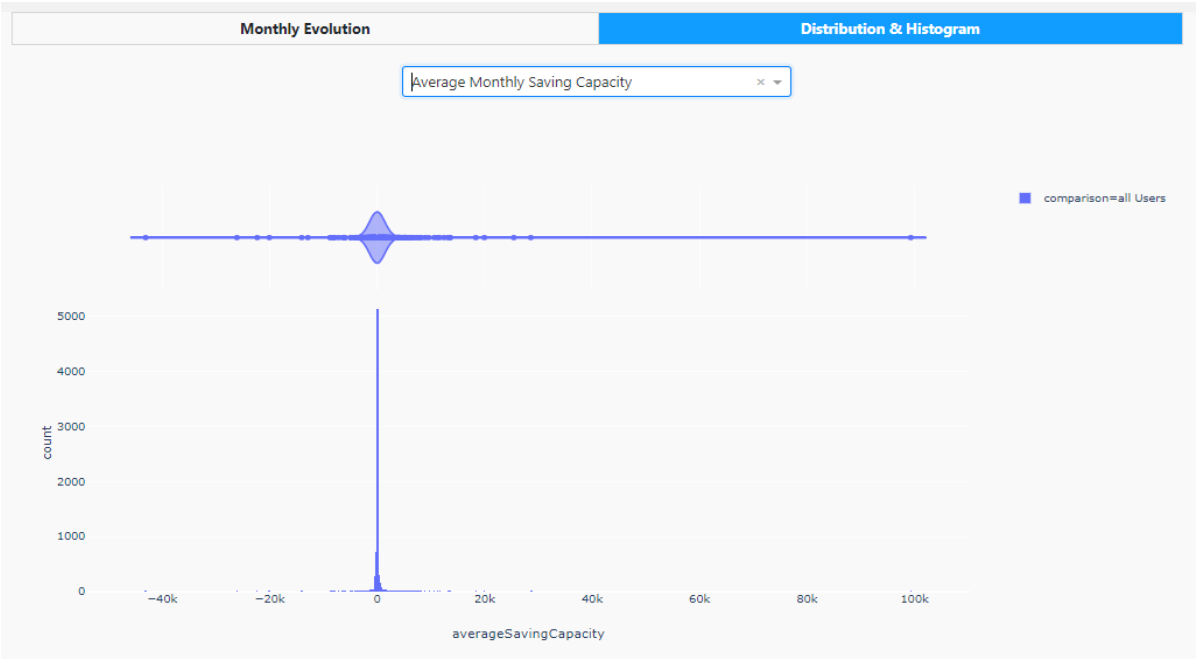


Figura 47. Histograma en sección Customers

Si se selecciona una sección con el ratón se puede tener un zoom sobre una zona concreta:

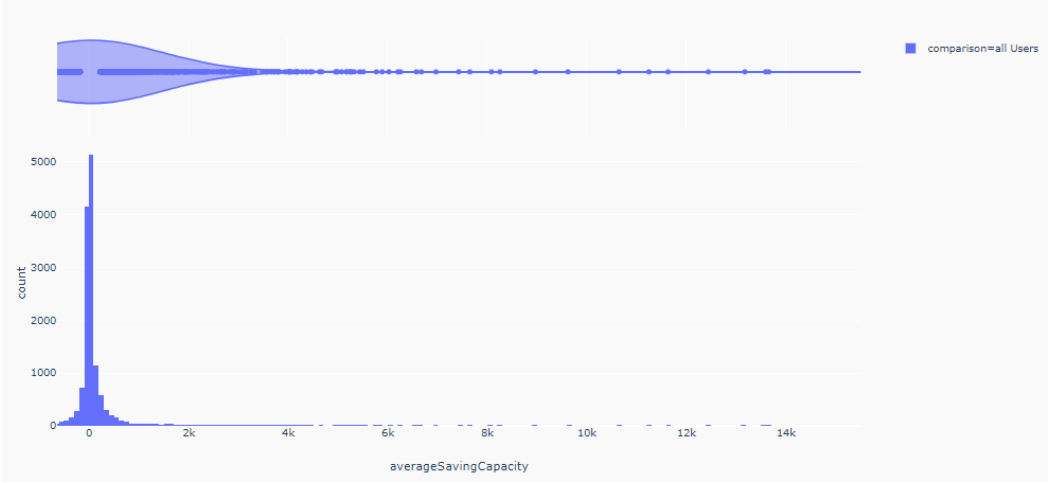


Figura 48. Zoom sobre histograma en sección Customers

3.5.4 ANÁLISIS Y DESARROLLO DE TAREAS - MERCHANTS TAB

La idea que persigue esta sección es el de organizar los distintos servicios financieros según el tipo de comercio (a través de la propiedad businessType y companyID en el CustomerObject) y mostrar unos gráficos comparativos entre distintas compañías. Por los análisis que se tienen que llevar a cabo, requiere una mejora respecto al modelo de datos.

A) Organización de categorías de comercios

Las gráficas atenderán al tipo de comercio que podrá ser seleccionado en un componente dropdown. Hasta el momento los tipos de comercios son:

- Supermercados
- Transporte
- Telecomunicaciones
- Seguros
- Entrega de comida
- Suscripciones a servicios y productos

Esta sería una muestra del Dataframe que sirve para analizar el flujo de dinero del objeto CustomerObject simplificando los datos en las siguientes columnas:

merchantDfFiltered_users

	transactionDate	amount	companyId	category
000E6536-2A95-2469-CF48-833D42772ED7	2018-05-01 00:00:00	16.99000	amazon	nan
000E6536-2A95-2469-CF48-833D42772ED7	2018-04-29 00:00:00	10.99000	amazon	nan
001048BC-869C-8269-89A2-0869B2274BFE	2019-07-10 00:00:00	157.33000	axa	insurance
001048BC-869C-8269-89A2-0869B2274BFE	2019-06-18 00:00:00	51.00000	axa	insurance
001048BC-869C-8269-89A2-0869B2274BFE	2019-06-14 00:00:00	71.61000	vodafone	telecom
001048BC-869C-8269-89A2-0869B2274BFE	2019-06-10 00:00:00	157.33000	axa	insurance
001048BC-869C-8269-89A2-0869B2274BFE	2019-05-15 00:00:00	50.00000	axa	insurance
001048BC-869C-8269-89A2-0869B2274BFE	2019-05-13 00:00:00	71.91000	vodafone	telecom
001048BC-869C-8269-89A2-0869B2274BFE	2019-05-09 00:00:00	143.00000	axa	insurance

Figura 49. Muestra de dataframe

Estas columnas y este modelo de datos resultan efectivo para segmentar por tipo de comercio y compañía de manera ágil aprovechando las ventajas de la utilización de los Dataframes.

Los cálculos se realizan a través de distintas funciones en el archivo figureFunctions y son muy similares a los anteriores. En el caso de la primera gráfica se calcula el gasto total por usuario respecto a cada compañía dentro de este tipo de comercio seleccionado en el dropdown. En la segunda gráfica se muestra el total de transacciones por usuario respecto a cada compañía. En la tercera gráfica se calcula el importe gastado medio por transacción también respecto a cada compañía.

Y, por último, la cuarta gráfica refleja un histograma superponiendo los datos de cada compañía según el número de transacciones que están en un límite de gasto. De esta manera se podrá ver la frecuencia de las transacciones por compañía y controlar el margen en el que se encuentran. Este histograma refleja bastante información sobre el margen en el que se realizan con más frecuencia transacciones según el tipo de comercio y compañía y, según el filtrado de usuarios, comprobar márgenes de importe de las transacciones según el tipo de usuario.

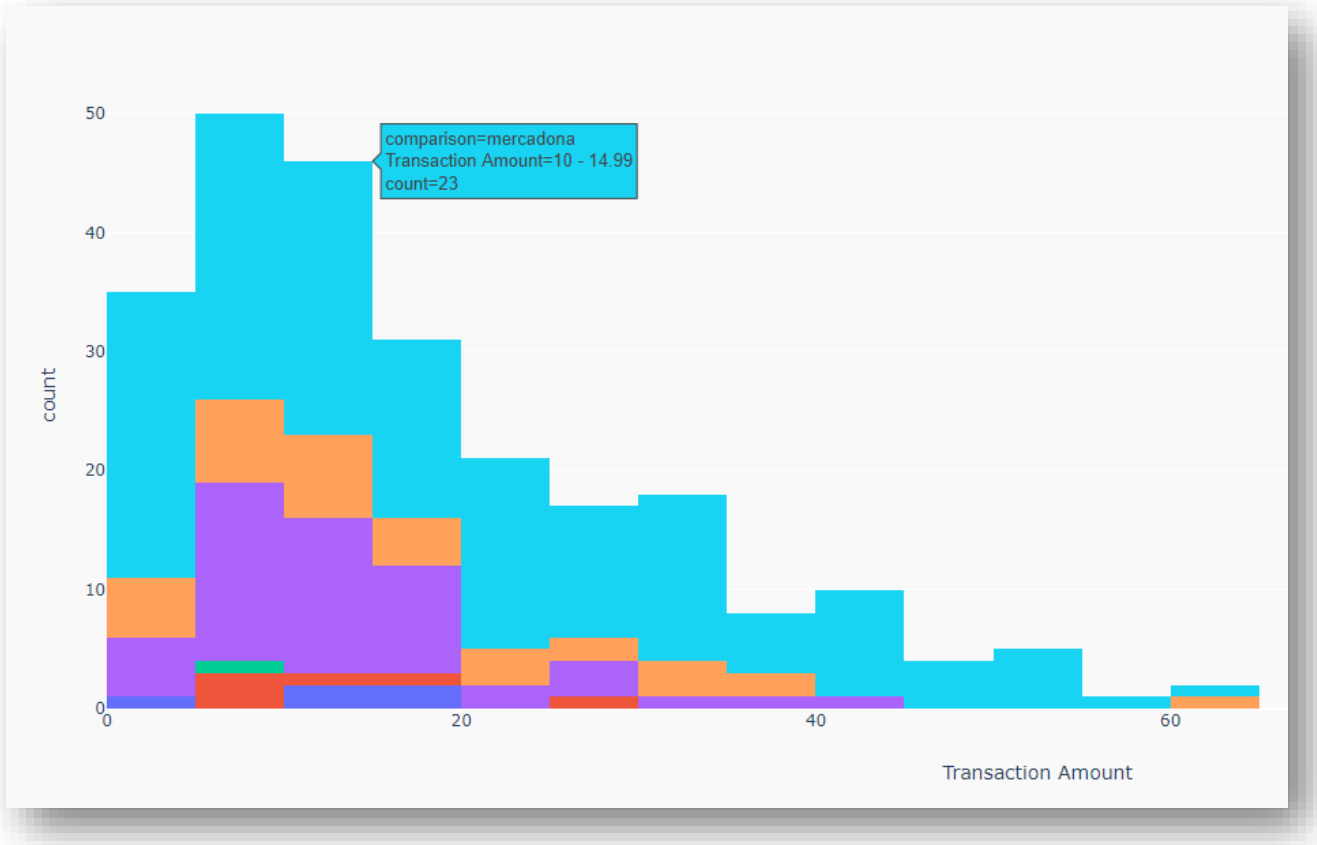


Figura 50. Histograma de la vista Merchants

3.5.5 ANÁLISIS Y DESARROLLO DE TAREAS - FS PROVIDERS TAB

El objetivo es crear dos gráficas que muestren información en tiempo real de la base de datos respecto al tipo de cuentas y saldos de los usuarios. Estas gráficas mostrarán los datos de los usuarios que cumplen los filtros y si se selecciona un tipo de comparación, deberán incorporar cada origen de datos según el tipo de datos de segmentación seleccionado.

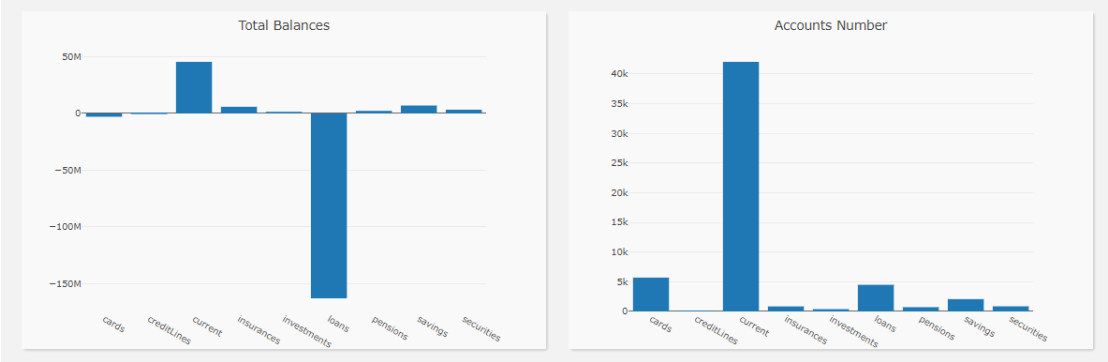


Figura 51. Gráficas de FS Providers

En el caso anterior y sin aplicar comparación, la estructura sería así para ambos casos (saldos y cuentas):

```
totalBalances = figureFunctions.getTotalBalances(filtered_users)
balances_x_filteredusers = 'Current',..., 'Cards'
balances_y_filteredusers = totalBalances['currentBalance'], ..., totalBalances['cardBalance']
filteredusersGraphBalances = figureFunctions.GraphData("Rent", "bar", balances_x_filteredusers,
                                                       balances_y_filteredusers)
balancesGraph = [filteredusersGraphBalances]
figureBalances = figureHelpers.updateStackedFigure("Amount of Total Balances",balancesGraph)
```

Primero sería necesario obtener el total de cada uno de los tipos de saldos y almacenarlos en un diccionario conforme al nombre del tipo de balance que irá representado en el eje X a través de la función getTotalBalances. Y, por último, obtener la figura gracias a la función de ayuda que dibuja la gráfica apilada con una estructura similar a como se ha visto en otras funciones similares, a las que se les pasa los valores de entrada de los usuarios y un título y acaban configurando la propiedad figure. La función getTotalBalances tendría la siguiente estructura:

```
def getTotalBalances(users):
    totalBalances = dict(currentBalance=0,
                         ...
                         cardBalance=0)
    for user in users:
        userBalances = getBalances(user['balances'])
        totalBalances['currentBalance'] += userBalances['currentBalance']
        ...
        totalBalances['cardBalance'] += userBalances['cardBalance']
    return totalBalances
```

La función getTotalBalances requiere obtener los saldos de cada usuario individualmente. Para esto, la función getBalances iría evaluando el array de saldos de cada usuario y sumando los valores a cada tipo de saldo para finalmente devolver un diccionario de la siguiente manera:

```
def getBalances(balancesArray):
    userBalances = {"currentBalance": 0,
                    ...
                    "cardBalance": 0}
    for currentAccount in balancesArray['assets']['current']['currentAccounts']:
        userBalances["currentBalance"] += currentAccount['balance']
    ...
    for cards in balancesArray['liabilities']['current']['creditCards']:
        userBalances["cardBalance"] += cards['balance']
       return userBalances
```

Aplicando un tipo de comparación, básicamente habría que calcular por cada tipo de usuarios el proceso de manera similar en paralelo y finalmente, pasar a las funciones ayudas un objeto con cada uno de los orígenes de datos que se quieren dibujar, como se explicó en el capítulo anterior y resultaría el gráfico con cada uno de los datos diferenciados según muestra la leyenda:

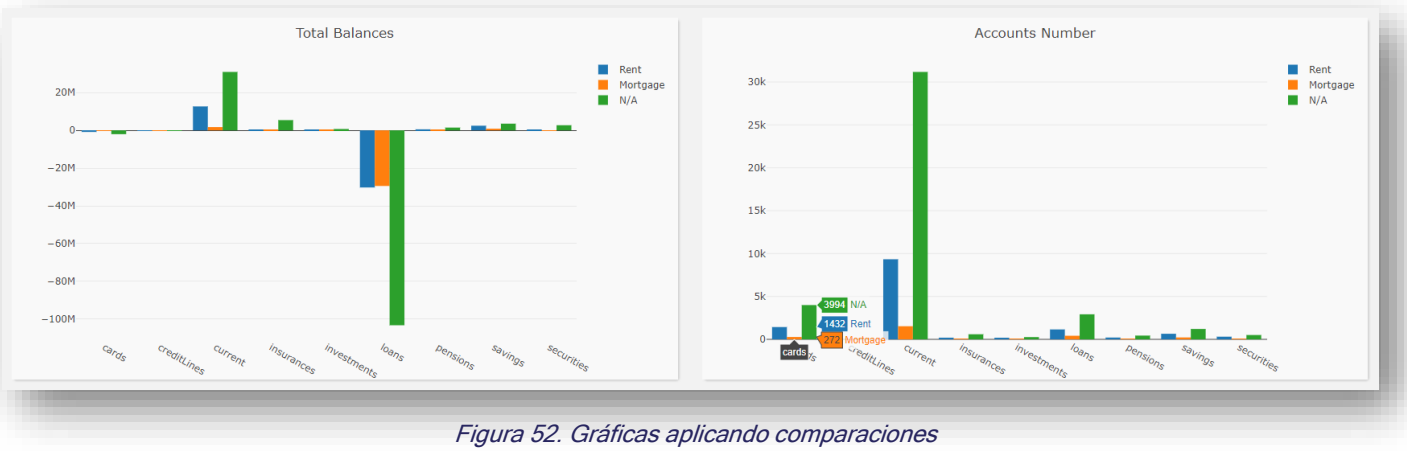


Figura 52. Gráficas aplicando comparaciones

A) Mejoras asociadas:

1. Disminuir margen entre gráficas y aumentar tamaño de la fuente

La primera solución que obtuve fue editando manualmente el fichero style.css y añadiendo sobre los elementos gráficos: transform:scale(1.2) para acercar el zoom. Esto no es buena solución [25] ya que las gráficas son generadas dinámicamente por Dash y construye elementos HTML svg, rect y g, que entre sí conforman la gráfica atendiendo a unas características predefinidas. Dash propone soluciones a través de sus documentos de ayuda [26] para configurar cualquier aspecto en relación con los elementos gráficos, por lo que nos quita el problema de tener que editar manualmente los elementos a través de CSS. Si además incluímos estas variables para establecer un margen entre gráficas [27] en la función updateFigure, problema resuelto.

1.1. Eliminar las opciones de edición sobre cada gráfico

El componente gráfico de dash, por defecto, incorpora una barra superior en el interior de cada gráfica para ofrecer distintas opciones. Para eliminar esta barra será necesario configurar en el elemento dcc.Graph el modo 'displayModeBar': False.

1.2. Cálculo de valor medio de saldos y Cuentas en vez de valor total

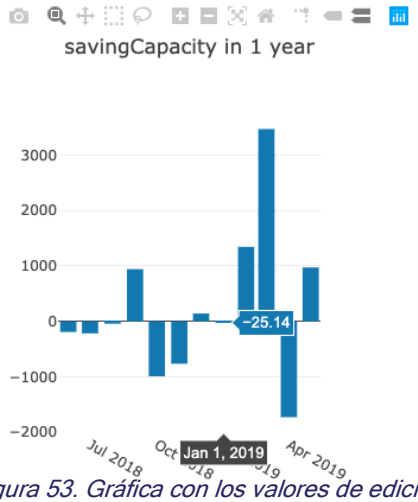


Figura 53. Gráfica con los valores de edición

Se divide cada uno de los valores obtenidos del diccionario entre el número de usuarios totales calculado a través de: totalCustomerObject=len(customerObject)

3.5.6 ANÁLISIS Y DESARROLLO DE TAREAS - BUSINESS TAB

En este apartado se pretende analizar el apartado relacionado con el negocio y mostrar información relevante como el número de ahorradores, el importe medio del ahorro por cliente y los saldos acumulados a lo largo del último año.

En primer lugar, será necesario obtener las cantidades promedio de ahorro, el total de contribuyentes, la cantidad total de ahorro de los clientes y el saldo acumulado total a través de la función getArborGraphData. Una vez obtenido, se construyen los objetos GraphData conforme a estos datos.

```
averageSavingAmounts, totalContributors, totalUsersSavingAmount, totalAccumulatedBalance =
                                                    figureFunctions.getArborGraphData(data_filter)
totalSaversGraph = figureFunctions.GraphData("Total Number of Savers",
                                             "lines",
                                             totalContributors.keys(),
                                             totalContributors.values())
averageSavingsGraph = figureFunctions.GraphData("Average Amount Saved",
                                             "lines",
                                             averageSavingAmounts.keys(),
                                             averageSavingAmounts.values())
accumulatedBalanceGraph = figureFunctions.GraphData("Accumulated Balance",
                                             "lines",
                                             totalAccumulatedBalance.keys(),
                                             totalAccumulatedBalance.values())
savingsAmountGraph = figureFunctions.GraphData("Savings Amount",
                                             "bar",
                                             totalUsersSavingAmount.keys(),
                                             totalUsersSavingAmount.values())
accumulatedSavingTotalGraphs = [accumulatedBalanceGraph, savingsAmountGraph]
```

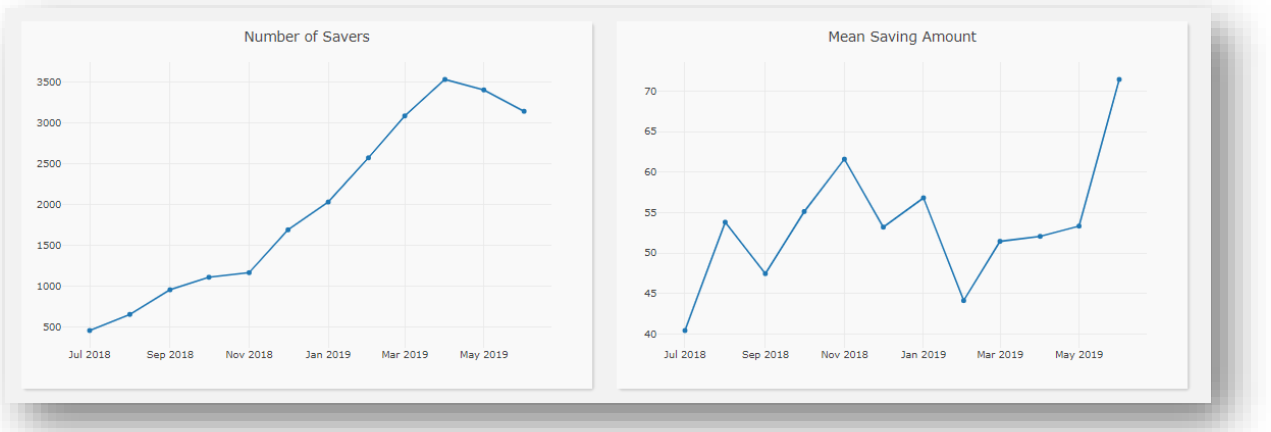


Figura 54. Gráficas superiores de negocio

Y, por último, se dibujan las figuras conforme las funciones updateLineFigure y updateStackedFigure.

```
saversGraph = figureHelpers.updateLineFigure(totalSaversGraph.getX(),
                                             totalSaversGraph.getY(),
                                             'Number of Savers')
meanSavingAmount = figureHelpers.updateLineFigure(averageSavingsGraph.getX(),
                                             averageSavingsGraph.getY(),
                                             'Average Saving Amount')
accumulatedBalance = figureHelpers.updateStackedFigure('Accumulated Balance',

                                                   accumulatedSavingTotalGraphs)
```

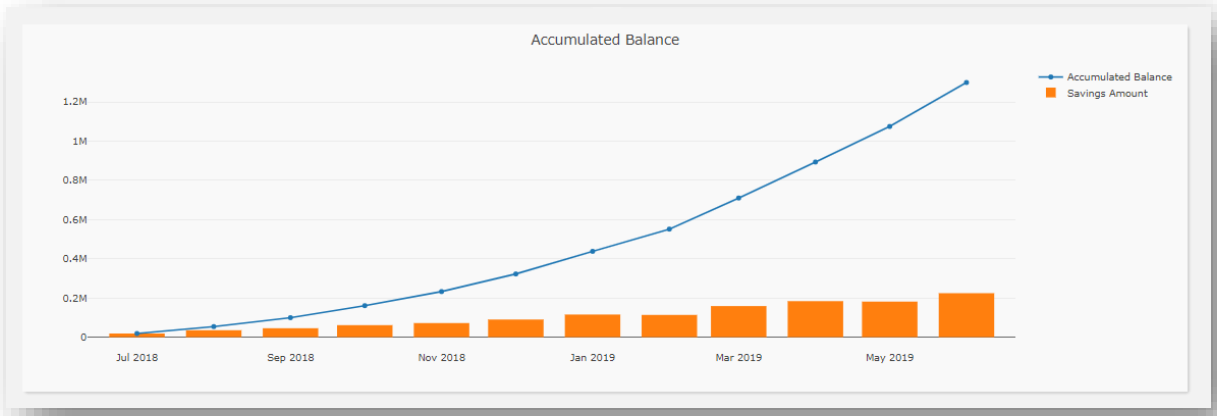


Figura 55. Gráfica balance de negocio

3.5.7 MEJORAS GENERALES EN LAS GRÁFICAS

A) Controlar cuando una lista es null

El problema es que cuando se establece un valor de filtro muy bajo o alguna de las listas es nula [], el programa muestra error. La solución deberá controlar estas limitaciones y comprobar que, si esa lista u origen de datos está vacía, que devuelva también un objeto vacío. De esta manera se puede obtener una gráfica vacía si no cuenta con ningún dato en su orígen de datos o en sus listas si se trata del procedimiento de comparación.

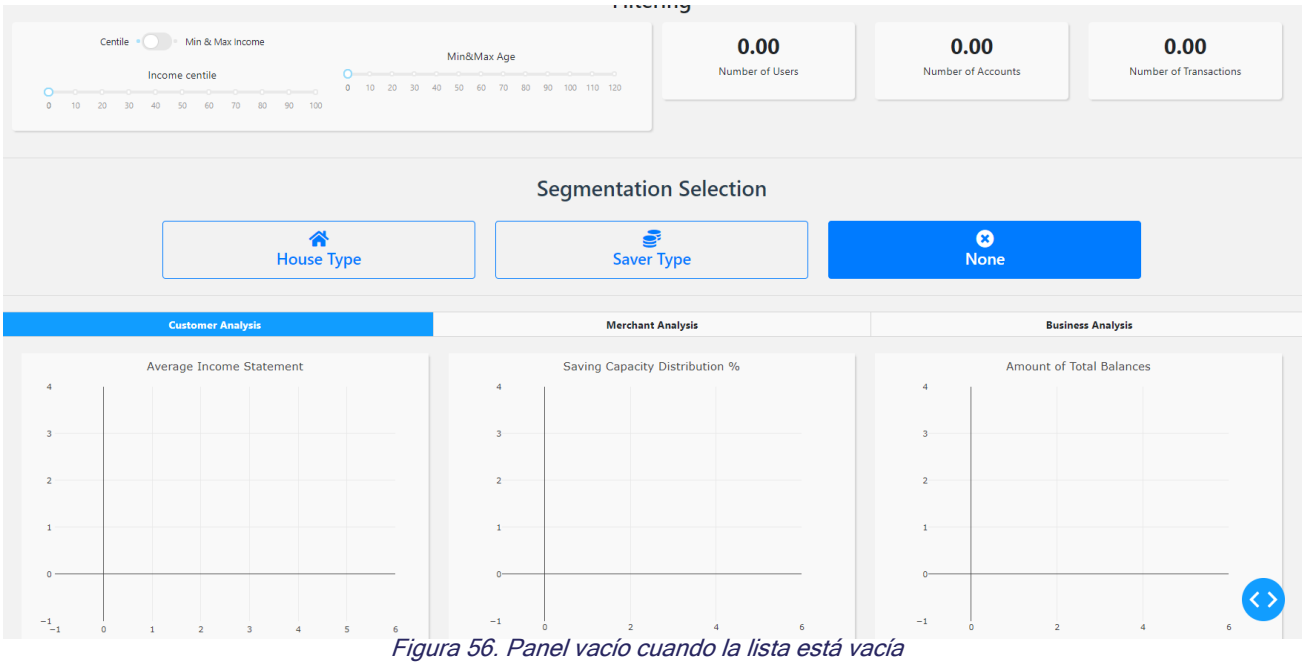


Figura 56. Panel vacío cuando la lista está vacía

B) Implementación del componente de carga de Dash

En el proceso de carga no todas las gráficas están listas a la vez. El usuario deberá ver este proceso a través de un icono de carga durante la carga de cada gráfica en su espacio correspondiente.

Para su desarrollo se utiliza la librería dash-core-components [28]. Sólo es necesario configurar el componente dcc.Loading sobre el componente dcc.Graph en todos los componentes gráficos del dashboard. La devolución de llamada funciona con este componente mientras cargan sus valores en la gráfica y no es necesario incluir ninguna devolución de llamada adicional para gestionar este componente.

```
dcc.Loading(children=[
   dcc.Graph(...)
], type='circle')
```

C) Mejora para no filtrar varias veces durante la carga inicial

Una vez se tiene el módulo de filtrado y módulo gráfico aparece el problema de la interacción entre filtros y gráficas. En la primera carga, y por cómo funciona Dash, se realizan multitud de cargas y filtrados que ralentizan la plataforma de manera notable. Por lo que es necesario entender más en profundidad cómo funciona el renderizado y las llamadas a funciones callbacks y aplicar las mejoras que permitan una carga rápida y eficiente de los componentes y filtrado de datos.

En primer lugar, se deberá filtrar por edad y por centil sólo si están seleccionados, por lo que se deberán eliminar los valores por defecto cuando se carga la página. Esto significa que todos los usuarios deben ser mostrados al principio y sólo serán filtrados cuando el usuario seleccione los límites de filtrado. De esta manera, se aplicarán los filtros conforme

el callback registra un nuevo evento en cada uno de los Inputs. Se eliminan entonces todas las llamadas a las bases de datos del tipo find(). Para que no llame continuamente a la base de datos, la clave es llamar una sola vez a la base de datos, almacenar esos datos en un objeto y, cuando sea necesario, cargar esos datos, nunca sobre-escribirlos.

Como cada callback será llamado al menos al inicio del dashboard, es muy importante controlar los valores Inputs y, en el caso de que sean nulos, se deberá controlar el tipo de objeto que se va a devolver. También será importante almacenar los datos para que todas las funciones callbacks puedan tener acceso siempre a los mismos datos (más adelante se comenta como se resuelve a través de las funciones y valores en Caché).

D) Solución de errores en el proceso de filtrado

En las funciones, cuando se intentan averiguar los datos del CustomerObject de cada usuario, si se intenta acceder a un valor concreto que no existe o se intenta hacer un cálculo no permitido sobre él, refleja el error siguiente:

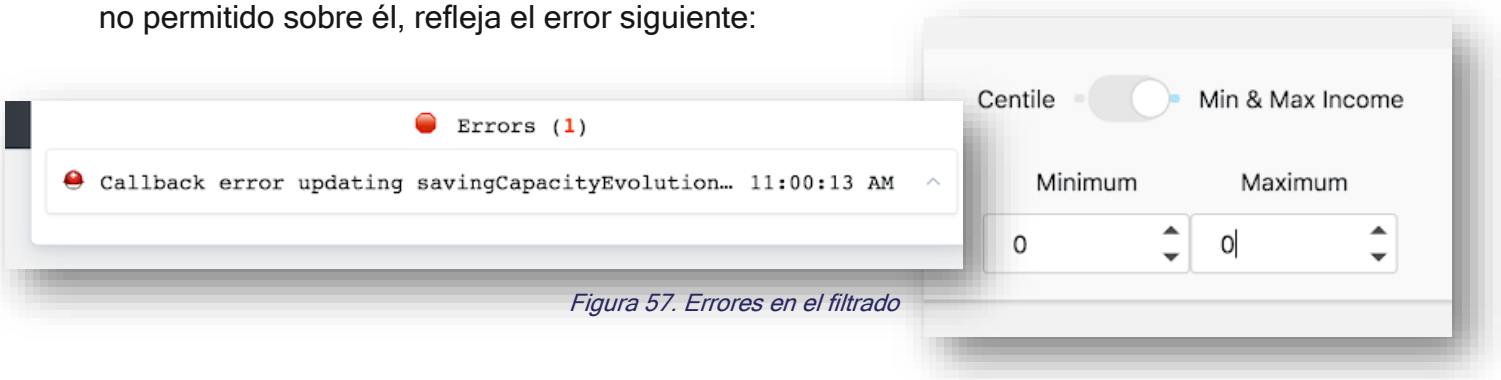


Figura 57. Errores en el filtrado

En este ejemplo, al filtrar un número concreto de usuarios muy bajo con ingresos 0, la función getSavingCapacityDistribution muestra el error anterior. Se intentaba dividir entre 0. Esto refleja que se deberá controlar cada vez que se intenta averiguar un valor o realizar un cálculo sobre él con un simple:

```
if savingCapacityDistribution[‘savingCapacity’] > 0:
        savingCapacityDistribution[value] /= savingCapacityDistribution[‘savingCapacity’]
```

3.5.8 GRAPHS COMPONENTS - DRAWGRAPH

El componente Graph ayuda a la construcción de las gráficas atendiendo a 3 funciones: drawGraph (para gráficas de ancho 4), drawGraph6 (para gráficas de ancho 6) y drawGraph12 (para gráficas de ancho toda la pantalla).

El tener esta separación de componentes es una buena práctica para tener una aplicación modular y no repetir código reutilizable en el archivo original. Además de que otra de las ventajas es el de asegurar soluciones de errores rápida.

3.6 Mejora en tiempos de carga – pandas Dataframes

3.6.1 MEJORA DE CÓDIGO GENERAL - RE-FACTORIZANDO

A) Importar todas las funciones del archivo en vez de una a una

Se importan las librerías al completo estableciendo en la parte superior de los archivos a través de import <librería>.

B) Archivo de constantes

Resulta práctico tener un archivo en el que se puedan establecer las principales constantes de la aplicación, para no tener que estar inicializándolas en varios puntos de ésta, como por ejemplo el número de meses a tener en cuenta en las transacciones, las rutas a la base de datos, los colores en los que se debe dibujar las gráficas, etc…

Las variables constantes deben aparecer siempre en mayúsculas:
IP_VALUE, PORT_VALUE, INTERACTIVE_GRAPH_OPTIONS…

1. Valores iniciales en la app

Al comienzo de la aplicación se conecta con la base de datos una única vez para obtener los datos en bruto que posteriormente serán utilizados en la aplicación.

1.1. Separación de componentes en archivos separados

La separación de componentes por módulos facilita el desarrollo ágil, la resolución de incidencias, la escalabilidad y el crecimiento del producto. Por ejemplo, se re-factorizan todos los componentes del módulo de registros estadísticos a través del archivo box.py con la función drawBox(number,title,id,id_container) encargada de dibujar el componente columna con el contenedor y los elementos que muestran cada registro estadístico.

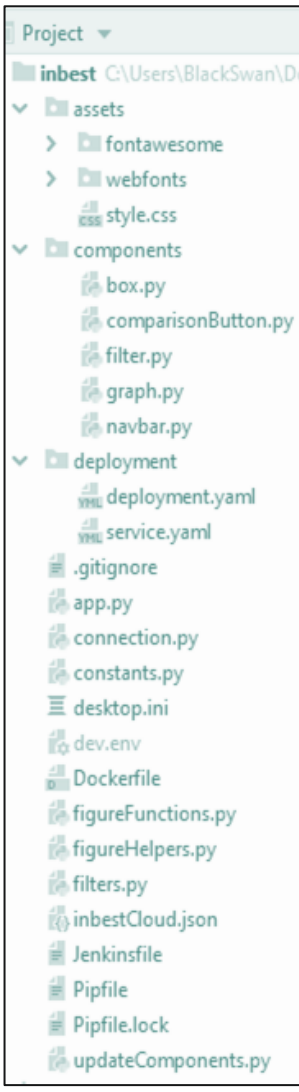


Figura 58. Disposición de archivos y componentes

1.2. Estilos en style.css

Las clases comunes que necesitan un estilo personalizado, como se ha comentado en anteriores secciones, deberán ir configuradas en el archivo style.css.

Por ejemplo, respecto a las pestañas, existen varias clases que representan cuando una pestaña ha sido seleccionada o no a través de las propiedades className y selected_className, y se encargan de cambiar el estilo de los componentes.

1.3. Separación del diseño de body en componentes

Se importan los archivos y funciones de la carpeta components y se utilizan en el código de manera:

```
components.comparisonButton.drawComparisonButton(...)
```

1.4. Separación de objetos gráficos a otros archivos

Aunque se perseguía modular todo el código contenido en body, hay algunos componentes que necesitan valores cargándose directamente en el body y no han podido ser modularizados. Algunos de los componentes del módulo gráfico si han podido ser separados en diferentes objetos como por ejemplo graphsObject.

3.6.2 DIVISIÓN DEL CALLBACK PRINCIPAL EN VARIOS CALLBACKS INDEPENDIENTES

Se desea dividir el callback principal en el que se ha ido desarrollando la funcionalidad principal en callbacks independientes. La limitación de Dash es que no pueden existir más de un callback para la misma Output, por lo que, se determina desarrollar una callback por cada componente Output que debe actualizarse conforme a los filtros o a la carga del tablero. En este punto será interesante también evaluar ciertos estados a través de State para controlar según los valores de varios elementos del dashboard y así realizar las labores de control sobre los objetos de retorno. Más adelante tomará más importancia este apartado que se sustentará en una estructura sólida a través de funciones en Caché y con las labores de señalización apropiadas para controlar los estados de carga de los elementos.

3.6.3 ARCHIVOS FIGUREHELPERS Y FIGUREFUNCTIONS

La función de figureHelpers es servir de ayuda para construir los componentes gráficos atendiendo a varias características.

La función de figureFunctions es de servir como soporte de cálculo para obtener valores o construir estructuras de datos a través de las clases GraphData o InteractiveGraphData.

En cada una de estas funciones se utilizarán Dataframes para aumentar el rendimiento de la aplicación y su mejor tratamiento sobre todo de manera interna entre los procesos. Se deberá evitar los bucles tanto como sea posible.

Como se ha documentado en el capítulo de modelo de datos, se utilizarán distintos Dataframes según cada propósito. Un ejemplo de uso sencillo podría ser por ejemplo en la función getBasicExpensesEvolution. Esta función recibe un objeto transactionsDf y se puede comprobar la diferencia de simplicidad entre uno y otro.

3.6.4 CAMBIO DE MODELO DE DATOS - VENTAJAS DEL USO DE DATAFRAMES

La utilización de la librería pandas es debido a que, en este contexto, necesitamos limpiar los datos en bruto y hacerlos aptos para el análisis a través de una estructura de datos tabular. Pandas fue diseñada originalmente para gestionar datos financieros como alternativa al uso de hojas de cálculo, pero hoy en día, se ha convertido en una librería muy utilizada para el análisis y procesamiento de datos a alto nivel en Python porque es capaz de representar y leer los datos de manera rápida y eficiente en apenas pocas líneas de código. Está preparado para estas tareas específicas.

Los Dataframes son estructuras de datos tabulares de 2 dimensiones con filas y columnas que ofrecen un alto rendimiento con grandes volúmenes de datos realizando segmentación, alineación y ordenación inteligente basado en etiquetas automáticas, que resulta similar al diccionario utilizado anteriormente desde la perspectiva de Python. Además, ofrecen flexibilidad para remodelar las características de los conjuntos de datos y permite un alto rendimiento en la lectura y escritura entre estas estructuras de datos.

Respecto a la revisión y refactorización del código para utilizar Dataframes en vez de estructuras de datos más simples se llevó a cabo el siguiente proceso:

- **Nueva recolección de datos** – Desde el archivo connection se deben reescribir las funciones que obtienen los distintos objetos desde la base de datos y el CustomerObject. Esta obtención de los datos simplifica el proceso al tener únicamente los valores apropiados para el análisis y guardarlos en el Dataframe correspondiente. La diferencia de memoria entre el antiguo modelo y el nuevo es notorio. Se adjunta estructura de fichero connection en la sección C del Anexo.

- **Reescritura de funciones de filtrado** – Desde el archivo filters se modifican las antiguas funciones para realizar el filtrado sobre la nueva estructura de datos. Lo importante en este apartado ha sido la facilidad de traducción de un modelo a otro, ya que anteriormente se trabajaba con diccionarios y en este nuevo caso, el funcionamiento es similar.

- **Funciones de ayuda en cálculos para gráficas** – Se ha traducido cada función desde el archivo figureFunctions y se ha simplificado en gran medida con esta nueva estructura gracias a las funciones que ahora si van a poder ser utilizadas para gestionar sencillamente los datos contenidos en los Dataframes. Lo que anteriormente se solucionaba con un bucle (y consumiendo demasiado tiempo para un gran volumen de datos) ahora es resuelto con un simple comando que ejecuta la acción al instante sobre el nuevo modelo tabular.

3.6.5 IMPLEMENTANDO CACHÉ Y SEÑALIZACIÓN

Con el nuevo modelo de datos y la nueva estructura que busca modular el proceso de actualización de los componentes, es necesario un sistema sólido de señalización y almacenamiento en caché de los datos internos que serán necesarios para actualizar los componentes atendiendo a unas reglas de filtrado y almacenamiento para evitar re-filtrados y procesos innecesarios que lo único que hace es aumentar el tiempo de carga de la página. El diagrama que describe el proceso es el siguiente:

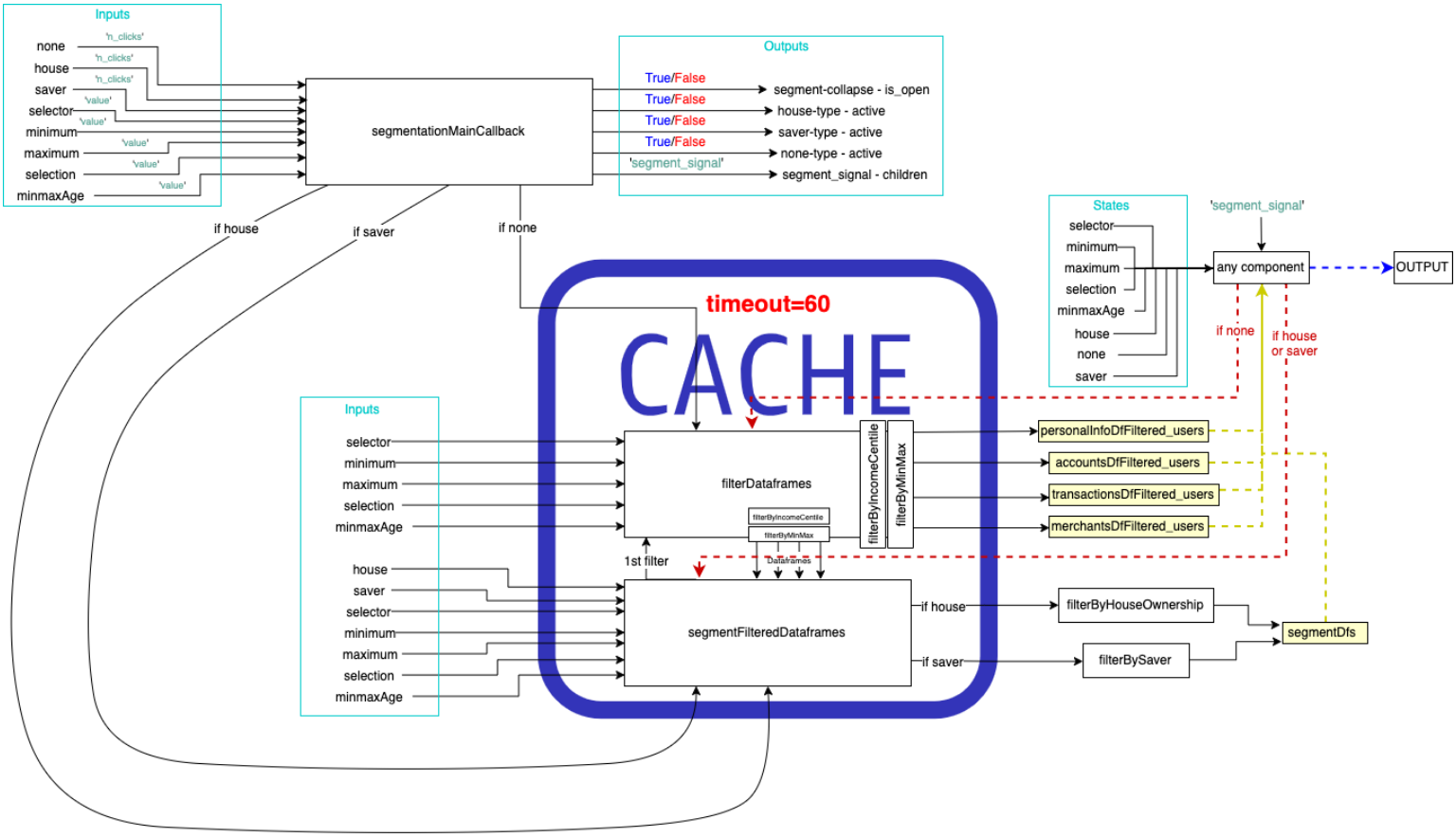


Figura 59. Diagrama del sistema de señalización y caché

Existen distintos componentes que aportan al proceso las variables de entrada que determinarán los parámetros de filtrado además de servir como localizadores de los datos en caché si previamente han sido utilizados. El proceso comienza con la carga de los datos de entrada en la función principal llamada segmentationMainCallback. Por defecto, dash inicializa las variables desde la llamada a cada una de las callbacks con los valores por defecto de las entradas, esto quiere decir que cada componente se debe inicializar desde su propia función callback y, en el caso de que no existan aún datos, devolver figuras u objetos vacíos. Por lo que, en un principio se aprovecha esta función por defecto para servir como inicializador.

3.6.6 IMPLEMENTACIÓN DEL FILTRADO

La función principal segmentationMainCallback establece el comienzo del primer filtrado conforme a sus datos de entrada y, lo más importante, marca el estado 'ready' a través de la señal 'segment_signal' cuando se ha realizado la función de filtrado para que cada componente pueda recuperar los datos a través de las funciones en caché filterDataframes o segmentFilteredDataframes han almacenado según los parámetros de filtrado.

Cada componente escucha si la variable almacenada en un div oculto llamada 'segment_signal' cambia su valor para que se pueda ejecutar la inicialización de componentes en paralelo conforme a estos primeros datos. La idea es que se parta de estos primeros datos, todos almacenados en caché, para que pueda configurar más rápido los componentes y que, conforme cambia un valor de entrada del módulo de filtros, poder ejecutar tan sólo una función de filtrado que dota en caché de los datos filtrados o segmentados al resto de componentes y los cargan en paralelo.

3.6.7 FILTROS DE EDAD E INGRESOS EN COMBINACIÓN

Se establece los filtros por edad e ingresos en combinación. Por lo que la función filterDataframes, según los parámetros de entrada, ejecuta la función filterByIncomeCentile o filterByMinMax incluyendo en los parámetros de entrada a las funciones la edad mínima y máxima seleccionadas o por defecto. Las funciones principales tendrían esta estructura:

```
@cache.memoize(timeout=60)
def filterDataframes(INPUTS):

    # Inicializar dataframes
    # Comprobaciones

    if…:
        filters.filterByIncomeCentile(…)
    else:
        filters.filterByMinMax(…)

    return Dataframes
```

```
@cache.memoize(timeout=60)
def segmentFilteredDataframes(INPUTS):

    # Obtener primero los dataframes de caché
    filterDataframes(INPUTS)

    # Comprobaciones

    if house:
        segmentDfs = filters.filterByHouseOwnership(DATAFRAMES)
    elif saver:
        segmentDfs = filters.filterBySaver(DATAFRAMES)

    return segmentDfs
```

Capítulo 4 Conclusiones y líneas futuras

4.1 Evaluación de soluciones y mejoras – HAR Analyzer

Para la evaluación de prestaciones se llevan a cabo varias pruebas y testeo de soluciones. Se optó en primer lugar en evaluar el tamaño de los objetos tras la carga de los objetos de la base de datos con scripts y librerías de soporte en Python tales como pympler y psutils antes de aplicar el cambio de modelo de datos y posteriormente, y se encuentra un cambio importante en tamaño, aunque no llega a ser determinante, pues sabemos que no se llegan a obtener todos los valores del CustomerObject y esto es lógico que no sea evidencia del cambio importante que supone, aunque se trata de optimización:

```
def memory_summary():
    from pympler import summary, muppy
    mem_summary = summary.summarize(muppy.get_objects())
    rows = summary.format_(mem_summary)
    return '\n'.join(rows)

print(memory_summary())
```

Versión pre-Dataframes:

types	# objects	total size
<class 'dict	679821	193.13 MB
<class 'str	327970	38.98 MB
<class 'datetime.datetime	333152	15.25 MB
<class 'list	169940	15.14 MB
<class 'bson.objectid.ObjectId	237569	12.69 MB
<class 'bytes	238030	10.21 MB
<class 'code	40330	5.56 MB
<class 'type	3404	3.47 MB
<class 'tuple	13654	1012.43 KB
<class 'property	11333	973.93 KB
<class 'float	30253	709.05 KB
<class 'set	1182	640.80 KB
<class 'collections.OrderedDict	278	419.38 KB
<class 'weakref	4848	416.62 KB
<class 'inspect.Parameter	4333	338.52 KB

Versión post-Dataframes:

types	# objects	total size
<class 'pandas.core.frame.DataFrame	4	35.18 MB
<class 'pandas.core.indexes.base.Index	8	21.12 MB
<class 'str	77737	19.88 MB
<class 'numpy.ndarray	84	9.25 MB
<class 'dict	10289	5.43 MB
<class 'code	36824	5.07 MB
<class 'type	2972	3.00 MB
<class 'pandas._libs.tslibs.timestamps.Timestamp	12534	1.63 MB
<class 'property	11126	956.14 KB
<class 'tuple	12226	909.62 KB
<class 'set	968	547.31 KB
<class 'list	3661	492.62 KB
<class 'weakref	5451	468.45 KB
<class 'collections.OrderedDict	279	420.77 KB
<class 'inspect.Parameter	4347	339.61 KB

En segundo lugar, se trata de evaluar desde un punto de vista cercano al renderizado y más a alto nivel, por la forma de funcionar de Python y la gestión de memoria, seguro que nos iba a dar más respuestas y evidencias de en lo que se traduce la mejoras llevada a cabo.

En este escenario se evalúa la versión pre-Dataframes y post-Dataframes desde el navegador Google Chrome y la sección Network desde las herramientas de desarrollador. Vamos a ser capaces entonces de evaluar conforme a la respuesta de la aplicación a alto nivel teniendo en cuenta el renderizado que aplica Dash. Cada versión ha sido almacenada como archivo.har y posteriormente analizada con la herramienta de Google HAR Analyzer [29]. HAR (HTTP Archive) se trata de un formato utilizado en la exportación de la sesión que nos permite analizar aspectos de la sesión HTTP en formato JSON.

Versión pre-Dataframes:

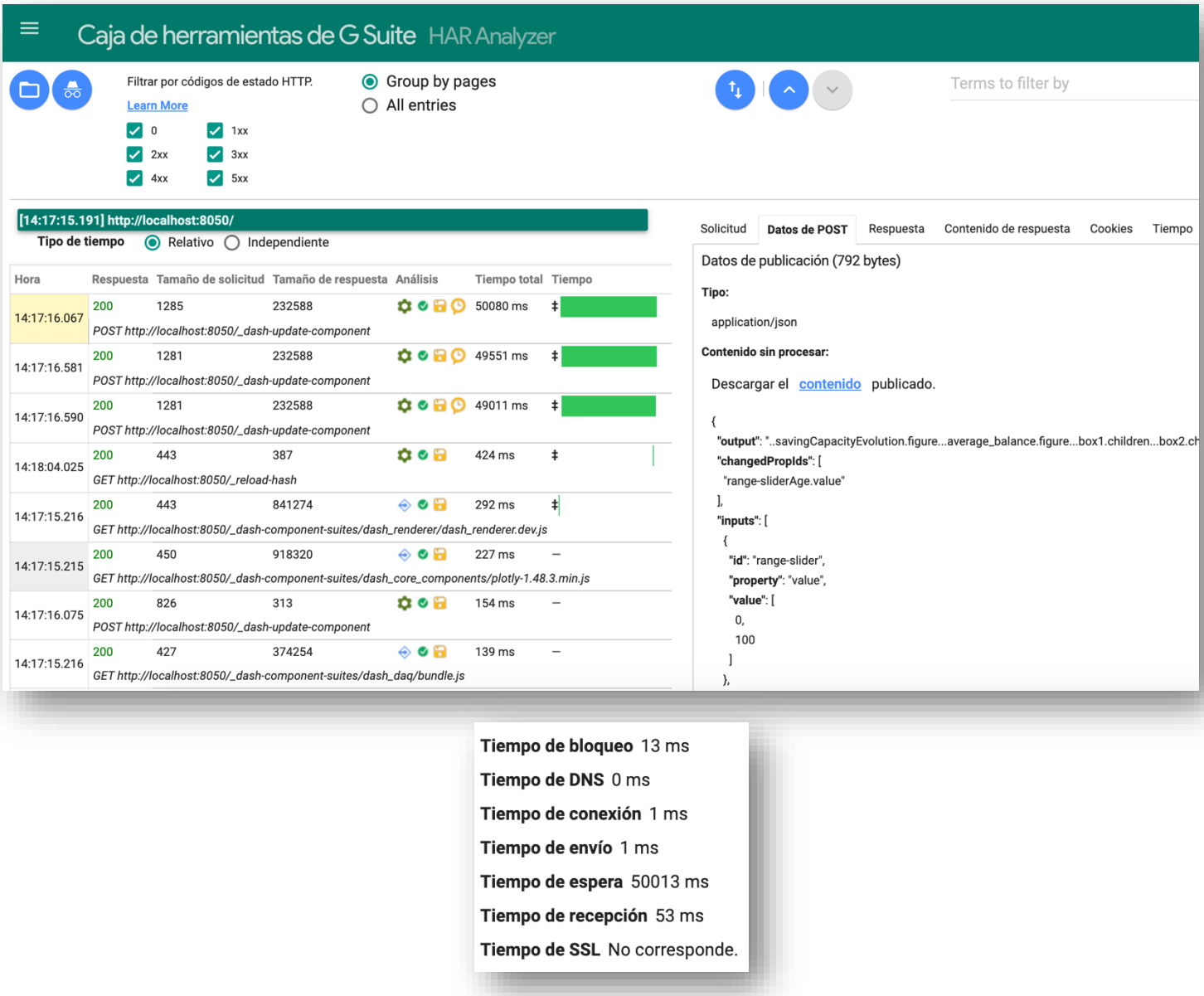


Figura 60. Resultados en la primera versión

Se observan varios elementos que atrasan la carga completa de la página a 50 segundos. Gracias a esta herramienta, se puede comprobar desde que función/funciones tienen lugar el update-component que ralentizan el proceso de carga, se pueden comprobar los valores de los distintos Inputs, y se puede comprobar el contenido de la respuesta y su tamaño:

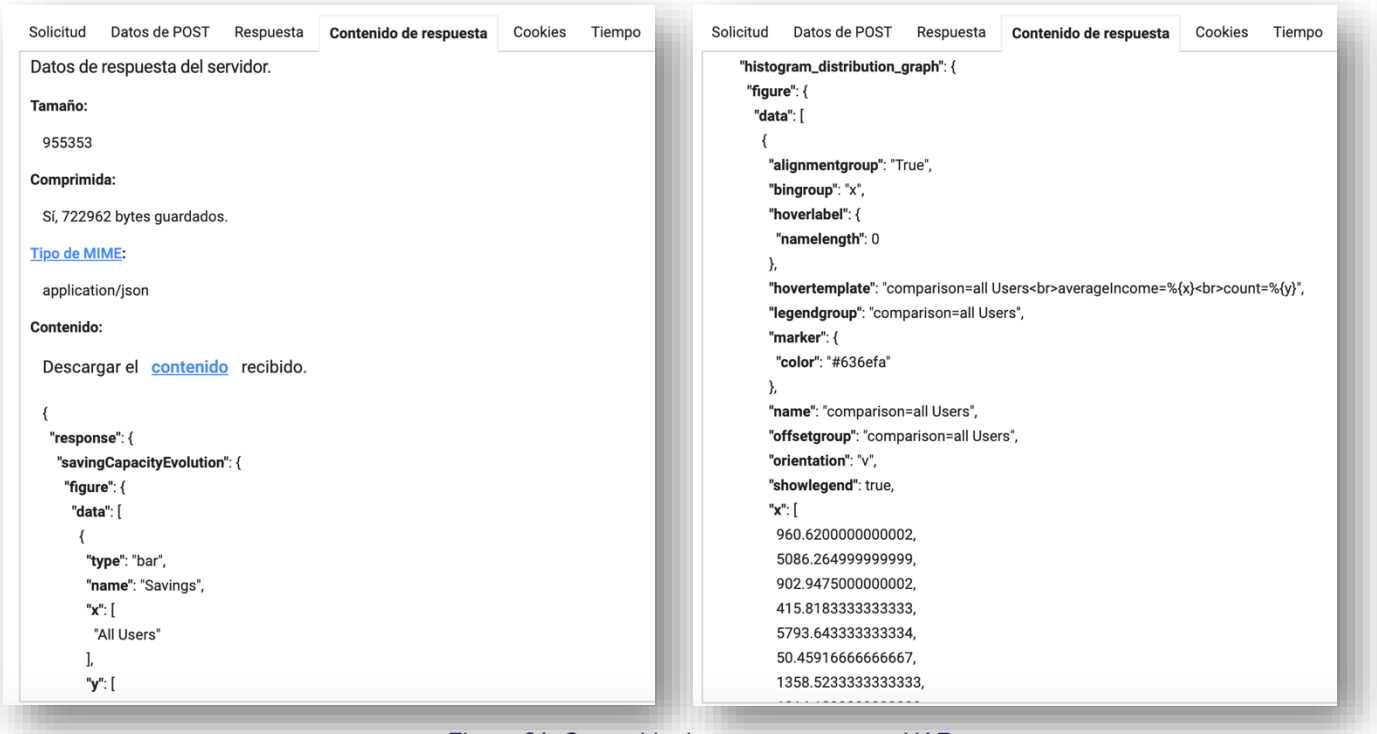
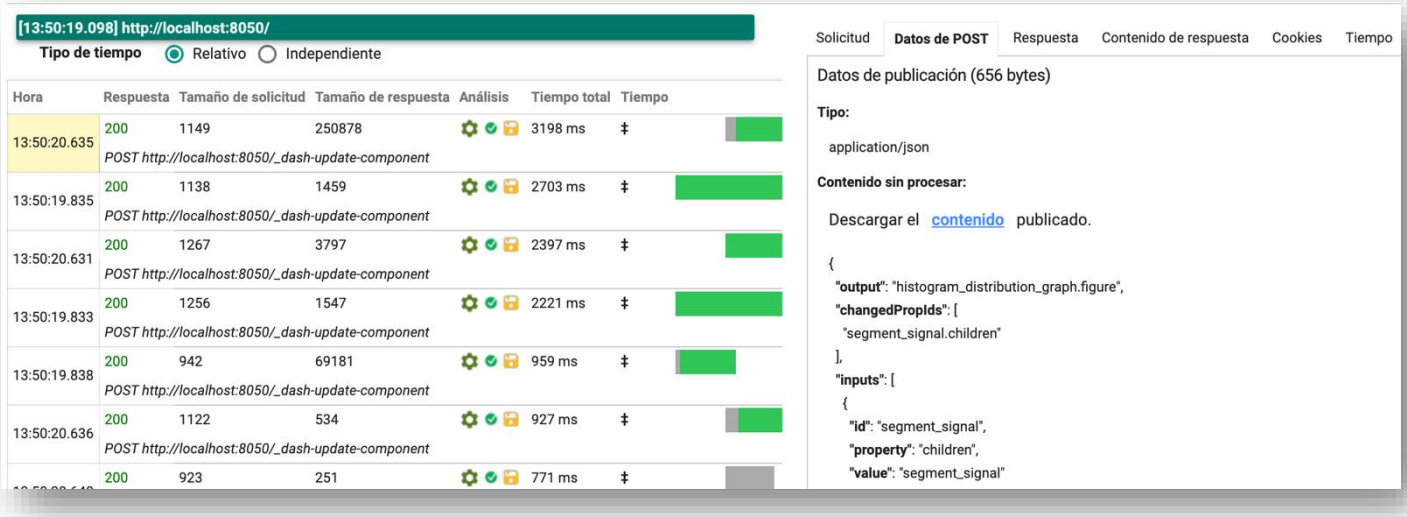


Figura 61. Contenido de una respuesta en HAR

Los tamaños están representados en bytes. Se puede encontrar la respuesta del servidor al completo en formato JSON sirviendo para labores de testeo, o incluso para conocer realmente como está siendo renderizado el componente a más alto nivel. En este caso, es evidente que hay un problema con la carga de 3 componentes gráficos, y siendo una versión con pocas gráficas respecto a la versión final, es un problema evidente que ha sido solventado con las distintas acciones que se han llevado a cabo, cada una de ellas ha sumado para encontrar otro escenario mucho más optimizado.

Versión post-Dataframes:



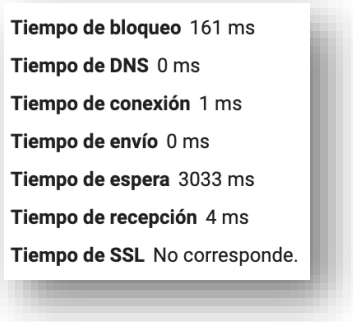


Figura 62. Resultados en la versión mejorada

En este caso, sobre la versión más reciente de la aplicación se comprueba que efectivamente el cambio es notable pasando de 50 segundos a 3,2 segundos para cargar todos los componentes teniendo en cuenta que en la versión anterior apenas había elementos gráficos y en la actualidad se trata de una aplicación mucho más completa y añadiendo gráficas que requieren un mayor análisis de contenido como las gráficas de diferenciación por tipo de comercio.

El tiempo de bloqueo superior que aparece en la carga de este componente se puede comprobar viendo justo en ese momento que componentes se están cargando y todos tienen en común la entrada de “segment_signal” utilizada para el sistema de señalización. Este es el motivo por el que este tiempo es ligeramente superior al anterior.

4.2 Oportunidades de crecimiento - Mejoras futuras

Tras conseguir las primeras mejoras en tiempos de carga se ha logrado obtener la oportunidad de presentar una prueba de concepto según las especificaciones del cliente como primer paso y a continuación se detallan las oportunidades que puede aprovechar la compañía y las mejoras que se están llevando a cabo en este momento en el dashboard con la mirada puesta a tener un servicio aún más completo, diferenciador y cumpliendo especificaciones que aporten valor al cliente.

4.2.1 ACTUALIZACIÓN DE LA SECCIÓN DE FILTRADO

Sería interesante aprovechar la rapidez del nuevo modelo de datos, del sistema de señalización y caché para filtrar de una manera mucho más diferenciadora respecto a los componentes anteriores.

La idea sería aprovechar varias propiedades al seleccionar un rango en el componente slider inferior o realizando zoom sobre el histograma que refleja claramente la distribución de usuarios.

La solución sería algo así como representa el siguiente módulo:

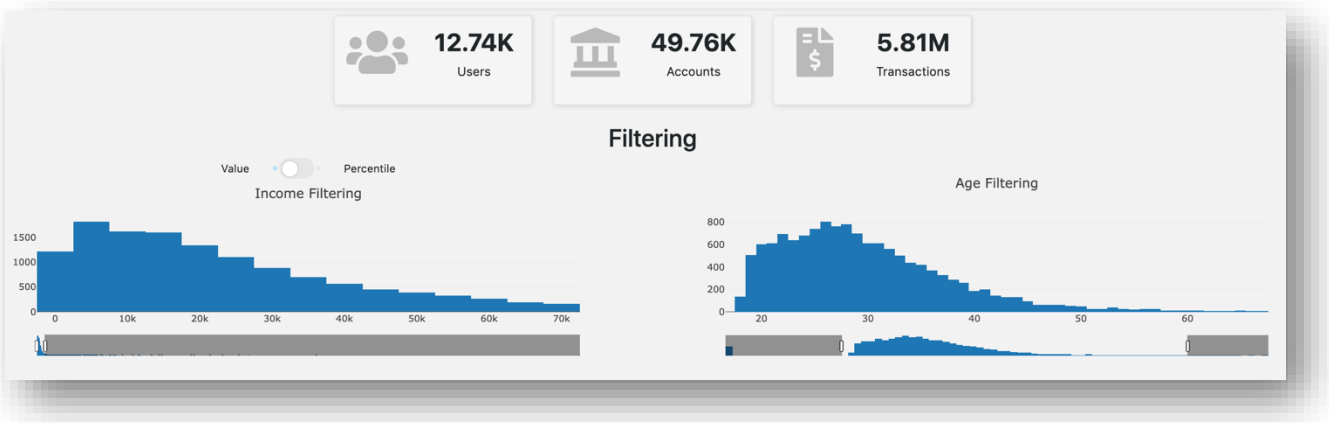


Figura 63. Nuevo módulo de filtrado

4.2.2 FILTRADO Y DESCARGA DE COMERCIOS - DATATABLE

Otra mejora sería ofrecer la oportunidad de filtrar, ordenar y descargar la información respecto a los comercios mostrándolos en un componente Datatable por si se quiere un mayor grado de profundidad en el filtrado y segmentación de tipo de transacción

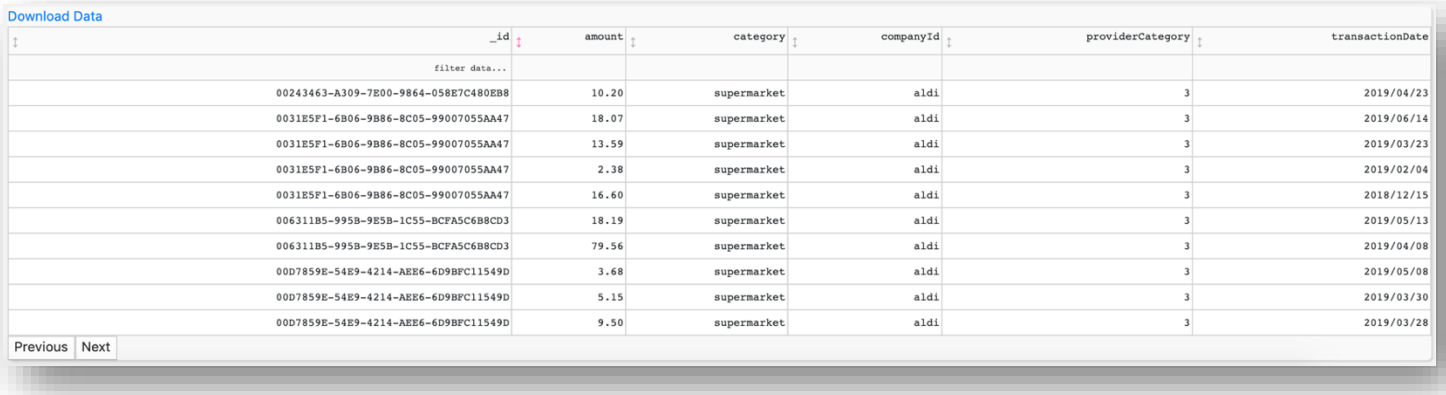


Figura 64. Filtrado y descarga de datos a través de DataTable

4.2.3 MENOS TIEMPO DE CARGA - ALREDEDOR DE 1 SEGUNDO

A través de la consecución de la reducción del tiempo de carga consideramos que es posible realizar una carga global en torno a 1 segundo si se optimiza el sistema de señalización, sobre todo, porque hemos comprobado que existe un tiempo en el que ciertos componentes no están cargando, pero podrían hacerlo. A través del análisis de las sesiones HTTP vamos a intentar modificar el sistema de señalización para permitir cargar todos los componentes en paralelo desde el inicio.

4.2.4 NUEVOS COMPONENTES MATERIAL-UI

Se puede permitir mayor interactividad entre componentes en el dashboard aplicando nuevos componentes material-ui [30]. Se busca que las interacciones entre componentes reflejen unas transiciones más cuidadas y así mejorar la experiencia de usuario y ofrecer un mayor número de servicios que pueden ser flotantes sin afectar al diseño.

4.2.5 CARACTERÍSTICAS DE COMPARACIÓN PERSONALIZABLES

El producto cuenta con dos valores de comparación según tipo de propietario de vivienda y según el tipo de ahorrador. Pero según el cliente, la aplicación debería poder ofrecer la oportunidad de comparar según otros parámetros, por lo que se propone la realización de un módulo estilo “Menú”, en el que ofrecer la oportunidad de configurar el tipo de comparación que el cliente necesite.

4.2.6 APROVECHAR NUEVAS TENDENCIAS

Podría ser interesante aplicar sobre la aplicación un servicio que incorpore tecnología Blockchain para análisis de ciertas operaciones según las necesidades de clientes bancarios u otras industrias.

Además, se podrían incluir modelos de inteligencia artificial que pudiera ofrecer otros nuevos servicios que tuviera en cuenta estos datos para realizar un ajuste personalizado o predecir patrones de forma según unas variables.

Capítulo 5 Bibliografía, referencias y documentación técnica

[1] https://cincodias.elpais.com/cincodias/2019/05/21/companias/1558465227_814702.html
[2] https://cincodias.elpais.com/cincodias/2019/03/22/companias/1553289119_647990.html
[3] https://www.europapress.es/economia/finanzas-00340/noticia-fmi-alerta-rentabilidades-insostenibles-tercio-gran-banca-mundial-20171011142206.html
[4] Gartner, <State of big data adoption>, 2015
[5] NGDATA IN CONJUNCTION WITH FINEXTRA AND CLEAR2PAY
[6] Informe del Observatorio de la Digitalización Financiera Funcas-KPMG, <Fintech, innovación al servicio del cliente>
[7] https://python-sprints.github.io/pandas/guide/pandas_docstring.html
[8] https://aws.amazon.com/es/ebs/
[9] https://aws.amazon.com/es/ec2/instance-types/
[10] https://docs.bitnami.com/aws/infrastructure/mongodb/
[11] https://www.guru99.com/installation-configuration-mongodb.html
[12] https://www.mongodb.com/download-center/community
[13] https://treehouse.github.io/installation-guides/mac/mongo-mac.html
[14] http://osxdaily.com/2018/03/07/how-install-homebrew-mac-os/
[15] https://plugins.jetbrains.com/plugin/7861-envfile
[16] https://realpython.com/pipenv-guide/
[17] https://www.jetbrains.com/help/pycharm/pipenv.html
[18] https://docs.mongodb.com/manual/reference/program/mongodump/
[19] https://dash.plot.ly/installation
[20] https://dash.plot.ly/getting-started
[21] https://dash-bootstrap-components.opensource.faculty.ai
[22] https://www.oscarblancarteblog.com/2018/12/10/data-access-object-dao-pattern/
[23] https://medium.com/@alonsus91/convenci%C3%B3n-de-nombres-desde-el-camelcase-hasta-el-kebab-case-787e56d6d023
[24] https://dash.plot.ly/dash-core-components/tabs
[25] https://medium.com/@maheshsenni/responsive-svg-charts-viewbox-may-not-be-the-answer-aaf9c9bc4ca2
[26] https://plot.ly/javascript/configuration-options/
[27] https://plot.ly/javascript/reference/
[28] https://dash.plot.ly/dash-core-components/loading_component
[29] https://toolbox.googleapps.com/apps/har_analyzer/
[30] https://github.com/StratoDem/sd-material-ui

Pandas for Everyone: Python Data Analysis: Python Data Analysis (Addison-Wesley Data & Analytics)
Publisher: Addison-Wesley Professional; 01 edition (26 Dec. 2017)
Language: English
ISBN-10: 0134546938
ISBN-13: 978-0134546933

Fluent Python
Publisher: O'Reilly Media; 1 edition (20 Aug. 2015)
Language: English
ISBN-10: 1491946008
ISBN-13: 978-1491946008

Python for Data Analysis, 2e
Publisher: O'Reilly; 2nd ed. edition (3 Nov. 2017)
Language: English
ISBN-10: 1491957662
ISBN-13: 978-1491957660

Capítulo 6 Anexos

A) Proceso de Jenkins contenido en el archivo Jenkinsfile:

```
#!groovy

String GIT_VERSION
String IMAGE

def notifySlack(String buildStatus = 'STARTED') {
    // Build status of null means success.
    buildStatus = buildStatus ?: 'SUCCESS'
    def color
    if (buildStatus == 'STARTED') {
        color = '#D4DADF'
    } else if (buildStatus == 'SUCCESS') {
        color = '#BDFFC3'
    } else if (buildStatus == 'UNSTABLE') {
        color = '#FFFE89'
    } else {
        color = '#FF9FA1'
    }
    def msg = "${buildStatus}: `${env.JOB_NAME}` #${env.BUILD_NUMBER}:\n${env.BUILD_URL}"
    slackSend(color: color, message: msg)
}

try {
  stage ('Build And Test'){
    node {
        def buildEnv
        def devAddress
          stage ('Checkout') {
            deleteDir()
            checkout scm
            GIT_VERSION = sh (
              script: 'git rev-parse --short HEAD',
              returnStdout: true
            ).trim()
          }
          if (env.BRANCH_NAME == "develop" || env.BRANCH_NAME == "sit" || env.BRANCH_NAME ==
"master"){
            if (env.BRANCH_NAME == "master"){
              notifySlack()
            }
            stage ('Setup GCloud Authentication') {
              sh '/var/lib/jenkins/GoogleCloudSDK/google-cloud-sdk/bin/gcloud auth activate-
service-account --key-file inbestCloud.json'
              sh '/var/lib/jenkins/GoogleCloudSDK/google-cloud-sdk/bin/gcloud --quiet config set
project inbestcloud'
              sh '/var/lib/jenkins/GoogleCloudSDK/google-cloud-sdk/bin/gcloud --quiet config set
compute/zone europe-west2-a'
              sh '/var/lib/jenkins/GoogleCloudSDK/google-cloud-sdk/bin/gcloud --quiet container
clusters get-credentials developer-cluster'
            }
          }
    }
  }
  // Develop branch create an image based on commit and release to dev env
  if (env.BRANCH_NAME == "develop") {
    IMAGE = "gcr.io/inbestcloud/github-inbest-boilerplate:${GIT_VERSION}"
    node{
        stage ('Build Custom Environment') {
          sh "echo ${IMAGE}"
          buildEnv = docker.build("gcr.io/inbestcloud/github-inbest-boilerplate:${GIT_VERSION}")
        }

        stage ('Push Docker Image To Container Registry') {
```

```
          sh "/var/lib/jenkins/GoogleCloudSDK/google-cloud-sdk/bin/gcloud docker -- push
gcr.io/inbestcloud/github-inbest-boilerplate:${GIT_VERSION}"
          sh "sed -i.bak 's#gcr.io/inbestcloud/github-inbest-
boilerplate:#gcr.io/inbestcloud/github-inbest-boilerplate:${GIT_VERSION}#'
deployment/deployment.yaml"
        }
        stage ('Deploy to DEV Environment') {
          sh "/var/lib/jenkins/GoogleCloudSDK/google-cloud-sdk/bin/kubectl --namespace=dev replace
-f deployment/deployment.yaml"
        }
    }
  }
  else if (env.BRANCH_NAME == "sit"){
    IMAGE = "gcr.io/inbestcloud/github-inbest-boilerplate:${GIT_VERSION}"
    node{
        stage ('Build Custom Environment') {
          sh "echo ${IMAGE}"
          buildEnv = docker.build("gcr.io/inbestcloud/github-inbest-boilerplate:${GIT_VERSION}")
        }
        stage ('Push Docker Image To Container Registry') {
          sh "/var/lib/jenkins/GoogleCloudSDK/google-cloud-sdk/bin/gcloud docker -- push
gcr.io/inbestcloud/github-inbest-boilerplate:${GIT_VERSION}"
          sh "sed -i.bak 's#gcr.io/inbestcloud/github-inbest-
boilerplate:#gcr.io/inbestcloud/github-inbest-boilerplate:${GIT_VERSION}#'
deployment/deployment.yaml"
        }
        stage ('Deploy to SIT Environment') {
          sh "/var/lib/jenkins/GoogleCloudSDK/google-cloud-sdk/bin/kubectl --namespace=sit replace
-f deployment/deployment.yaml"
        }
    }
  }
  else if (env.BRANCH_NAME == "master"){
    IMAGE = "gcr.io/inbestcloud/github-inbest-boilerplate:${GIT_VERSION}"
     node{
        stage ('Build Custom Environment') {
          sh "echo ${IMAGE}"
          buildEnv = docker.build("gcr.io/inbestcloud/github-inbest-boilerplate:${GIT_VERSION}")
        }

        stage ('Push Docker Image To Container Registry') {
          sh "/var/lib/jenkins/GoogleCloudSDK/google-cloud-sdk/bin/gcloud docker -- push
gcr.io/inbestcloud/github-inbest-boilerplate:${GIT_VERSION}"
          sh "sed -i.bak 's#gcr.io/inbestcloud/github-inbest-
boilerplate:#gcr.io/inbestcloud/github-inbest-boilerplate:${GIT_VERSION}#'
deployment/deployment.yaml"
        }
        stage ('Deploy to PRODUCTION Environment') {
          sh "/var/lib/jenkins/GoogleCloudSDK/google-cloud-sdk/bin/kubectl --namespace=production
replace -f deployment/deployment.yaml"
        }
    }
  }
} catch (e) {
    currentBuild.result = 'FAILURE'
    throw e
} finally {
    notifySlack(currentBuild.result)
}
// vim: set syntax=groovy :
```

B) Archivo PipeFile:

```
[[source]]
name = "pypi"
url = "https://pypi.org/simple"
verify_ssl = true

[dev-packages]

[packages]
dash="~=1.0.2"
dash-daq="~=0.1.0"
dash-bootstrap-components="~=0.7.0"
pymongo="~=3.8.0"
gunicorn="~=19.9.0"
numpy = "~=1.17.0"
pandas = "~=0.25.1"
flask-caching = "~=1.7.2"

[requires]
python_version = "3.7"
```

C) Estructura de funciones en connection.py:

```
def findPersonalInformation():
    """
    Get the personal information about the users in a dataframe structure.

    Returns
    -------
    dataframe : pandas Dataframe
        The dataframe contains the date of birth, the first saving date and the houseOwnership type
        from every user.

    See also
    --------
    iterator2dataframes : Turn an iterator into multiple small pandas Dataframe

    """
    try:
        mycol = selectDBandCollection(constants.DATABASE_VALUE,
                                      constants.CUSTOMER_OBJECTS_COLLECTION)
        pipeline = [{
            "$project": {
                "_id": "$userId",
                "dob": "$personalInformation.dateOfBirth",
                "firstSavingDate": "$personalInformation.firstSavingDate",
                "houseOwnership": "$personalInformation.houseOwnership"
            }}]
        # Retrieve all of the customer object data

        dataframe = iterator2dataframes(mycol.aggregate(pipeline), 1000)
        dataframe = dataframe.set_index("_id")

    except errors.OperationFailure as err:

        dataframe = None
        print("PyMongo ERROR:", err, "\n")

    return dataframe
```

```
def iterator2dataframes(iterator, chunk_size: int):
    """Turn an iterator into multiple small pandas.DataFrame
  This is a balance between memory and efficiency
  """
    records = []
    frames = []
    for i, record in enumerate(iterator):
        records.append(record)
        if i % chunk_size == chunk_size - 1:
            frames.append(pd.DataFrame(records))
            records = []
    if records:
        frames.append(pd.DataFrame(records))
    return pd.concat(frames, sort=False)
```