

Trabajo fin de grado

BorakReports: Análisis y gestión de informes empresariales



Mario Jiménez Franco

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C/ Francisco Tomás y Valiente nº 11

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR



Grado en Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

**BorakReports: Análisis y gestión de informes
empresariales**

**Integración multiplataforma con GESBORAK mediante IA
y métricas de rentabilidad**

Autor: Mario Jiménez Franco
Tutor: Javier Rojas Jiménez
Ponente: Alejandro Bellogin Kouki

junio 2026

Todos los derechos reservados.

Queda prohibida, salvo excepción prevista en la Ley, cualquier forma de reproducción, distribución comunicación pública y transformación de esta obra sin contar con la autorización de los titulares de la propiedad intelectual.

La infracción de los derechos mencionados puede ser constitutiva de delito contra la propiedad intelectual (*arts. 270 y sgts. del Código Penal*).

DERECHOS RESERVADOS

© 2026 por UNIVERSIDAD AUTÓNOMA DE MADRID

Francisco Tomás y Valiente, n.º 1

Madrid, 28049

Spain

Mario Jiménez Franco

BorakReports: Análisis y gestión de informes empresariales

Mario Jiménez Franco

C\ Francisco Tomás y Valiente N.º 11

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

PREFACIO

El desarrollo de este Trabajo Fin de Grado nace de una observación directa de la realidad operativa en el entorno empresarial actual. A menudo, la digitalización de los procesos se traduce en una acumulación masiva de registros que, lejos de facilitar la gestión, genera un ruido informativo que dificulta la toma de decisiones estratégicas. Esta memoria es el resultado de un esfuerzo por transformar esa complejidad en claridad.

El proyecto **BorakReports** surge de la necesidad de tender un puente entre el sistema transaccional **GESBORAK** y las necesidades reales de los gestores y empleados. Mi objetivo principal ha sido demostrar que el análisis de datos no tiene por qué ser un proceso árido o manual; mediante la integración de algoritmos de rentabilidad y modelos de lenguaje natural, es posible crear herramientas que **hablen el idioma del usuario** y aporten valor inmediato.

A lo largo de estos meses, el reto no ha sido solo técnico, sino conceptual: definir cómo visualizar la salud financiera de un proyecto de un solo vistazo o cómo garantizar que una inteligencia artificial pueda responder con precisión sobre la actividad de la empresa. Este trabajo refleja mi visión sobre el papel del ingeniero informático: un facilitador capaz de crear sistemas que no solo procesen datos, sino que generen conocimiento.

Espero que las páginas que siguen transmitan no solo el detalle técnico del desarrollo multiplataforma y la implementación de la IA, sino también el propósito práctico con el que fue concebida esta solución. BorakReports es, en última instancia, una apuesta por una gestión empresarial más eficiente, transparente y conectada.

Mario Jiménez Franco

AGRADECIMIENTOS

En primer lugar, quiero expresar mi agradecimiento a Borak por haberme brindado la oportunidad de realizar mis prácticas curriculares, así como por la confianza depositada en mí al incorporarme posteriormente a la plantilla. Asimismo, agradezco haber podido desarrollar este Trabajo de Fin de Grado dentro de la empresa, lo que ha supuesto una experiencia profesional de gran valor.

En segundo lugar, agradezco a la Universidad Autónoma de Madrid, y en particular a la Escuela Politécnica Superior, así como a todo su profesorado, por la formación recibida a lo largo de estos años. De manera especial, quiero agradecer a mi tutor de prácticas y ponente del TFG, Alejandro Bellogin Kouki, por su constante apoyo, orientación y disponibilidad durante todo el proceso de elaboración de este trabajo. También agradezco a mi promoción y, especialmente, a las amistades construidas durante estos cuatro años, muchas de las cuales han trascendido lo académico para convertirse en parte fundamental de mi vida.

En tercer lugar, quiero agradecer a todas las personas que me han acompañado a lo largo de este camino, tanto en los momentos difíciles como en los logros alcanzados, por su apoyo incondicional, su paciencia y por estar siempre presentes en cada etapa de este recorrido.

Por último, pero no menos importante, agradezco profundamente a mi familia y a todas aquellas personas que forman parte de ella o que, sin serlo formalmente, han desempeñado ese papel a lo largo de los años. Su apoyo constante, su confianza inquebrantable y su esfuerzo han sido el pilar fundamental no solo de este trabajo o de mi etapa universitaria, sino de toda mi trayectoria personal y académica. Sin ellos, nada de esto habría sido posible.

Por todo ello, este Trabajo de Fin de Grado, junto con el esfuerzo realizado a lo largo de estos años de carrera, va dedicado a todos ellos, como reconocimiento a su apoyo constante y a su contribución, directa o indirecta, en cada etapa de este camino académico y personal.

RESUMEN

Este Trabajo Fin de Grado presenta el desarrollo de **BorakReports**, una solución multiplataforma diseñada para optimizar la toma de decisiones estratégicas mediante la analítica avanzada de datos empresariales. La aplicación surge como una respuesta a la rigidez de los sistemas transaccionales tradicionales, estableciendo una capa de inteligencia sobre la plataforma **GESBORAK** para automatizar la generación de informes y la interpretación de indicadores clave de rendimiento (KPIs).

El sistema integra un motor de consultas basado en Inteligencia Artificial que permite a los usuarios interactuar con la base de datos mediante lenguaje natural, democratizando el acceso a la información operativa bajo una jerarquía estricta de privilegios. Asimismo, se ha implementado un módulo financiero capaz de procesar datos contables externos para ofrecer cálculos de rentabilidad en tiempo real, representados mediante un sistema visual de indicadores de estado y gráficas de tendencia temporal.

Los resultados obtenidos demuestran que la automatización de estos procesos no solo reduce el riesgo de error humano inherente al tratamiento manual de datos, sino que proporciona una visión holística y ágil de la salud operativa de la organización.

PALABRAS CLAVE

Inteligencia de Negocio, Inteligencia Artificial, Multiplataforma, Rentabilidad, Gestión de Informes, GESBORAK

ABSTRACT

This final degree project presents the development of **BorakReports**, a multiplatform solution designed to optimize strategic decision-making through advanced business data analytics. The application serves as a response to the rigidity of traditional transactional systems, establishing an intelligence layer over the **GESBORAK** platform to automate report generation and the interpretation of Key Performance Indicators (KPIs).

The system integrates an Artificial Intelligence engine that enables users to interact with the database through natural language, democratizing access to operational information under a strict hierarchy of privileges. Furthermore, a financial module has been implemented to process external accounting data, providing real-time profitability calculations represented through a visual system of status indicators and temporal trend graphs.

The results obtained demonstrate that automating these processes not only reduces the risk of human error inherent in manual data handling but also provides a holistic and agile overview of the organization's operational health.

KEYWORDS

Business Intelligence, Artificial Intelligence, Multiplatform, Profitability, Report Management, GESBORAK

ÍNDICE

1	Introducción	1
1.1	Contexto: El ecosistema GESBORAK	1
1.2	Motivación y justificación del proyecto	1
1.3	La Solución Propuesta: BorakReports	2
1.4	Alcance del Proyecto	2
1.5	Estructura de la Memoria	2
2	Estado del Arte	3
2.1	Estado del Arte en Inteligencia de Negocio	3
2.1.1	La Revolución del Text-to-SQL y los LLM	3
2.1.2	Procesamiento de Lenguaje Natural (NLP) y Embeddings	5
2.1.3	Optimización mediante RAG y Caché Semántica	5
2.2	Ecosistema Tecnológico del Backend	6
2.2.1	Spring Boot 3 y la Abstracción de Spring AI	6
2.2.2	Conectividad y Gestión de Datos	7
2.2.3	Seguridad, Identidad y Autorización Robusta	7
2.3	Desarrollo del Cliente Multiplataforma	7
2.3.1	Arquitectura en React y Vite	7
2.3.2	Estrategia Híbrida con Capacitor	7
2.4	Ingeniería de Transformación y Generación de Informes	8
2.4.1	Análisis de Librerías de Reporte y Visualización	8
3	Diseño	9
3.1	Análisis de Requisitos y Especificaciones	9
3.1.1	Requisitos Funcionales: El Catálogo de Servicios	9
3.1.2	Modelado de Actores y Casos de Uso	10
3.1.3	Atributos de Calidad y Requisitos No Funcionales	10
3.2	Arquitectura del Software y Capas del Sistema	10
3.3	Diseño del Motor de Inteligencia Artificial: Flujo Jerárquico de Resolución	11
3.4	Modelado de Componentes y Patrones de Diseño	12
3.5	Estructura Lógica de Información	12
3.6	Modelado Visual del Sistema	13
3.6.1	Diagramas de Casos de Uso	13
3.6.2	Modelo Entidad-Relación (E-R)	14

3.6.3 Diagramas de Secuencia	14
3.7 Resumen del Diseño Operativo	14
4 Desarrollo	17
4.1 Arquitectura General del Sistema	17
4.2 Diseño de la Arquitectura Backend	18
4.2.1 Separación en capas en Spring Boot	18
4.2.2 Flujo de procesamiento de las peticiones	19
4.3 Diseño de la Arquitectura Frontend	20
4.3.1 Estructura de enrutamiento y navegación	20
4.3.2 Jerarquía de la interfaz	20
4.3.3 Gestión del estado global y encapsulación de lógica de negocio	21
4.4 Diseño de Seguridad y Control de Acceso (RBAC)	22
4.4.1 Matriz de Permisos	22
4.4.2 Protección del Frontend: Renderizado condicional en la interfaz	22
4.4.3 Filtrado de Datos (Row-Level Security simulado)	23
4.5 Diseño Estructural de los Módulos Core	23
4.5.1 Generador de Informes Estratégicos: Patrón Strategy	23
4.5.2 Asistente de Inteligencia Artificial: Motor de Resolución en Tres Barreras	25
4.5.3 Explorador de Rentabilidades: Diseño Jerárquico y Visualización	26
4.5.4 Módulo de Contabilidad e Ingesta de Datos	27
4.5.5 Consultas Predefinidas y Análisis Visual	28
4.5.6 Asistente Conversacional: Navegación Guiada por Estados (FSM)	29
5 Pruebas	31
5.1 Pruebas de Aceptación de Usuario (UAT)	31
5.1.1 Perfil de la Muestra y Frecuencia de Uso	31
5.1.2 Análisis Cuantitativo: Métricas de Satisfacción	32
5.1.3 Análisis Cualitativo y Retroalimentación Directa	34
5.1.4 Propuestas de Evolución y Áreas de Mejora	34
5.2 Pruebas de Caja Negra: Matriz de Validación Text-to-SQL	34
5.3 Pruebas de Rendimiento y Latencia: Impacto de la Caché Semántica	37
6 Conclusiones	39
6.1 Aportaciones y Conclusiones Técnicas	39
6.2 Validación y Aceptación del Usuario	40
6.3 Líneas de Trabajo Futuro	40
6.4 Reflexión Final	40
Bibliografía	41

Apéndices	43
A Modelado Visual del Sistema	45
A.1 Diagrama de Casos de Uso	45
A.2 Diagramas de Secuencia	47
A.2.1 Autenticación y Gestión de Identidad	47
A.2.2 Procesamiento y Exportación de Informes	47
B Especificación técnica de la API REST	51
B.1 Especificación técnica de la API REST	51
B.1.1 Módulo de Autorización e Identidad (/auth)	51
B.1.2 Módulo del Asistente Conversacional (/bot)	53
B.1.3 Módulo de Inteligencia Artificial (/ia)	53
B.1.4 Módulo de Clientes y Proyectos (/clientes)	54
B.1.5 Módulo de Consultas Predefinidas (/consultas)	55
B.1.6 Módulo de Contabilidad e Ingesta de Datos (/contabilidad)	57
B.1.7 Módulo de Divisiones (/divisiones)	59
B.1.8 Módulo de Empleados (/empleados)	59
B.1.9 Módulo de Cuadros de Mando y Gráficos (/graficos)	60
B.1.10 Módulo de Generación de Informes (/informes)	61
B.1.11 Módulo de Rentabilidades (/rentabilidades)	62
C Interfaces del Sistema	65
C.1 Inicio de sesión y recuperación de contraseña	65
C.1.1 Visualización Web	65
C.1.2 Visualización Móvil	66
C.2 Pantalla de Inicio (Dashboard Principal)	67
C.2.1 Visualización Web	67
C.2.2 Visualización Móvil	68
C.3 Asistente IA, Consultas y Gráficos	69
C.3.1 Visualización Web	69
C.3.2 Visualización Móvil	70
C.4 Módulo de Contabilidad y ChatBot FSM	71
C.4.1 Visualización Web	71
C.4.2 Visualización Móvil	72
C.5 Explorador de Rentabilidades	73
C.5.1 Visualización Web	73
C.5.2 Visualización Móvil	74
C.6 Generación y Exportación de Informes	75

INTRODUCCIÓN

En el escenario empresarial actual, las organizaciones enfrentan el reto de gestionar volúmenes masivos de datos operativos generados por su actividad diaria. La acumulación de registros no garantiza una ventaja estratégica; esta solo se alcanza estableciendo un isomorfismo entre la estructura de los datos brutos y la realidad del negocio, transformando el registro técnico en conocimiento accionable.

1.1. Contexto: El ecosistema GESBORAK

La operativa de la empresa se sustenta en la plataforma **GESBORAK**, un sistema de gestión transaccional donde converge la actividad de la plantilla. En este entorno se registran dimensiones fundamentales de la realidad corporativa: la dedicación horaria vinculada a proyectos, la imputación de costes directos y la gestión de disponibilidad mediante el control de vacaciones.

Si bien GESBORAK funciona como un repositorio robusto para el registro de actividad, carece de una capa de análisis que proporcione una visión holística y predictiva. La explotación actual adolece de una rigidez que introduce latencia en la toma de decisiones y riesgos de error en el tratamiento manual de datos contables mediante herramientas externas.

1.2. Motivación y justificación del proyecto

La motivación de este Trabajo Fin de Grado reside en cerrar la brecha entre la generación del dato operativo y su interpretación ejecutiva. La dirección requiere herramientas que permitan monitorizar no solo el trabajo realizado, sino la rentabilidad y eficiencia de los procesos en curso de manera ágil.

BorakReports surge para automatizar tareas de análisis que consumen recursos de forma ineficiente. El proyecto se justifica por la necesidad de centralizar indicadores clave (KPIs) en una solución única, multiplataforma y accesible, eliminando la dependencia de personal técnico para obtener respuestas de negocio inmediatas y precisas.

1.3. La Solución Propuesta: BorakReports

BorakReports es una plataforma de inteligencia de negocio diseñada como una capa superior sobre GESBORAK. El proyecto otorga significado a los datos mediante tres pilares:

Automatización y Exportación: Sustitución de la extracción manual por una generación automatizada de informes en formatos PDF y Excel, asegurando información consistente y libre de errores de manipulación.

Interacción mediante Inteligencia Artificial: Implementación de un motor de consultas basado en IA que permite interactuar con la base de datos de forma conversacional, respetando una jerarquía de acceso que garantiza la privacidad del empleado y la visión global del administrador.

Visualización y Métricas de Rentabilidad: Desarrollo de cuadros de mando que procesan datos operativos y contables externos. Permite visualizar la salud financiera de los proyectos mediante indicadores visuales de rentabilidad y gráficas de tendencia temporal.

1.4. Alcance del Proyecto

El desarrollo abarca desde el análisis de los esquemas de datos de GESBORAK hasta la implementación de una interfaz multiplataforma funcional. Incluye el diseño de algoritmos de rentabilidad, la integración de APIs para la interpretación de consultas y módulos de carga contable externa. No se pretende sustituir a GESBORAK, sino potenciarlo como fuente de datos primaria.

1.5. Estructura de la Memoria

Esta memoria se organiza según el rigor metodológico de la Escuela Politécnica Superior. El Capítulo 2 aborda el **Estado del Arte**. El Capítulo 3 detalla el **Diseño** de la arquitectura y el modelo de datos. Los Capítulos 4 y 5 se centran en el **Desarrollo** técnico y las **Pruebas** de validación. Finalmente, el Capítulo 6 expone las **Conclusiones** y líneas futuras.

ESTADO DEL ARTE

Este capítulo constituye la base teórica y técnica sobre la cual se ha edificado *BorakReports*. En lugar de una simple enumeración de herramientas, se presenta un análisis de cómo la convergencia de la Inteligencia Artificial Generativa, las arquitecturas multiplataforma y los patrones de diseño de software permiten resolver problemas complejos de gestión empresarial.

2.1. Estado del Arte en Inteligencia de Negocio

La Inteligencia de Negocio o *BI* (del inglés *Business Intelligence*) ¹ ha transitado desde modelos de reporte estáticos hacia la denominada **Analítica Aumentada**. Este nuevo paradigma se caracteriza por el uso de tecnologías de aprendizaje automático para automatizar la preparación de datos y compartir conocimientos de forma más intuitiva.

2.1.1. La Revolución del Text-to-SQL y los LLM

Uno de los mayores desafíos históricos en la ingeniería de datos ha sido la brecha entre el lenguaje natural humano y los lenguajes de consulta estructurados. La aparición de los modelos de lenguaje de gran escala o *LLM* (*Large Language Models*) ² ha permitido que sistemas como *BorakReports* implementen motores de *Text-to-SQL*. Esta tecnología interpreta semánticamente el esquema de la base de datos para generar consultas válidas.

El proceso de traducción de *Text-to-SQL* no es una mera transcripción de palabras, sino que implica varias fases cognitivas complejas simuladas por el modelo. En primer lugar, se produce el denominado *Schema Linking* o vinculación de esquemas [1], donde la IA debe identificar qué sustantivos de la pregunta corresponden a entidades y atributos específicos de la base de datos (por ejemplo, relacionar de forma natural la palabra 'gastos' con la tabla `GESBORAK.GASTOS`).

¹ Conjunto de estrategias y herramientas enfocadas a la administración y creación de conocimiento mediante el análisis de datos existentes.

² Modelos de inteligencia artificial entrenados con vastas cantidades de datos para comprender y generar lenguaje humano.

A continuación, el modelo debe inferir las operaciones algebraicas subyacentes, tales como los *JOINS* necesarios para conectar tablas inconexas, las funciones de agregación (*SUM*, *COUNT*) y las agrupaciones lógicas (*GROUP BY*). En sistemas corporativos heredados, este desafío se multiplica, ya que el modelo debe lidiar con dialectos SQL antiguos (como Oracle 10g) y evitar la invención de funciones modernas no soportadas.

Previo a la fase de diseño, se llevó a cabo un análisis comparativo de los modelos de lenguaje predominantes en el mercado para evaluar su idoneidad en entornos corporativos. Se establecieron como criterios críticos la precisión en la generación de sintaxis SQL compleja, la latencia de respuesta para mantener la interactividad del chat y la carga de infraestructura, tal y como se resume en la Tabla 2.1.

Modelo (Proveedor)		Precisión SQL	Latencia	Coste e Infraestructura
GPT-4o-mini	(OpenAI)	Muy Alta	Muy Baja	Bajo (API gestionada)
Claude 3 Haiku	(Anthropic)	Alta	Muy Baja	Bajo (API gestionada)
GPT-3.5-Turbo	(OpenAI)	Moderada	Baja	Bajo (Modelo legado)
Llama 3 8B	(Meta)	Moderada	Variable	Alto (Requiere <i>hosting</i> local)

Tabla 2.1: Comparativa de Modelos de Lenguaje (LLM) evaluados para la generación Text-to-SQL.

Tras esta evaluación, para el presente proyecto se determinó utilizar el modelo `gpt-4o-mini` de OpenAI [2]. A diferencia de las alternativas de código abierto (*Open Source*) como Llama 3 —que requieren servidores con costosas tarjetas gráficas (GPU) dedicadas para su ejecución local—, el modelo de OpenAI se consume a través de una API, delegando la carga computacional y garantizando una alta disponibilidad.

Además, `gpt-4o-mini` destaca por su excelente capacidad de razonamiento lógico, permitiendo manejar la sintaxis específica y arcaica que requieren motores legados como Oracle 10g/11g mediante técnicas avanzadas de *Prompt Engineering*. La eficacia de este enfoque radica en proporcionar al modelo no solo la pregunta, sino el contexto del esquema relacional (DDL) y las restricciones de integridad.

En este flujo, el componente `SqlPromptBuilder` actúa como un regulador gramatical, instruyendo a la IA para evitar funciones modernas no soportadas por la infraestructura heredada de la organización.

2.1.2. Procesamiento de Lenguaje Natural (NLP) y Embeddings

Para que el sistema busque información por significado y no solo por coincidencia exacta, se emplean técnicas de **procesamiento de lenguaje natural** o *NLP* (*Natural Language Processing*). El núcleo de esta capacidad reside en la generación de *embeddings* [3]. Matemáticamente, un *embedding* es una función que mapea texto a un vector de números reales en un espacio de dimensiones fijas (habitualmente 1536 en modelos de OpenAI). En este espacio, la cercanía geométrica entre dos vectores —calculada mediante la similitud del coseno— representa su similitud semántica.

En este espacio, la cercanía geométrica entre dos vectores —calculada mediante la **similitud del coseno**— representa su similitud semántica. Este proceso permite identificar que conceptos como 'horas totales' e 'imputaciones' son análogos. La integración de un *Vector Store* permite gestionar esta "memoria semántica" mediante *SimpleVectorStore*, una decisión técnica que equilibra la necesidad de búsqueda rápida con una infraestructura de bajo coste en formato JSON local.

2.1.3. Optimización mediante RAG y Caché Semántica

El paradigma *RAG* (*Retrieval-Augmented Generation*), formalizado inicialmente en 2020 por investigadores de Facebook AI Research [4], surge como solución directa a las limitaciones inherentes de los modelos generativos estáticos. Los LLM tradicionales carecen de acceso a información corporativa privada y sus pesos neuronales quedan congelados tras su entrenamiento, lo que en entornos empresariales puede provocar la generación de información obsoleta o inventada, fenómeno conocido como «alucinación».

Para solventar esta brecha de conocimiento y mitigar el riesgo de alucinaciones en un entorno crítico de negocio, la arquitectura RAG altera el ciclo de vida de la consulta dotando a la IA de una «memoria externa» en tiempo real. Esta estrategia transforma al modelo de lenguaje de un simple generador de texto a un motor de razonamiento fundamentado exclusivamente en el contexto corporativo inyectado, tal y como ilustra la Figura 2.1.

El proceso se divide en dos fases secuenciales. En la primera fase (*Recuperación*), el sistema toma la petición del usuario y busca en una base de datos vectorial aquellos fragmentos de información interna (como diccionarios de datos, esquemas de tablas, reglas de negocio o directrices de seguridad) que sean semánticamente relevantes para la pregunta. En la segunda fase (*Generación*), esta información recuperada se inyecta en el *prompt* junto con la pregunta original. De este modo, la IA no responde basándose exclusivamente en sus conocimientos preentrenados, sino que actúa como un motor de razonamiento que procesa la "fuente de verdad" de la empresa aportada en ese preciso instante.

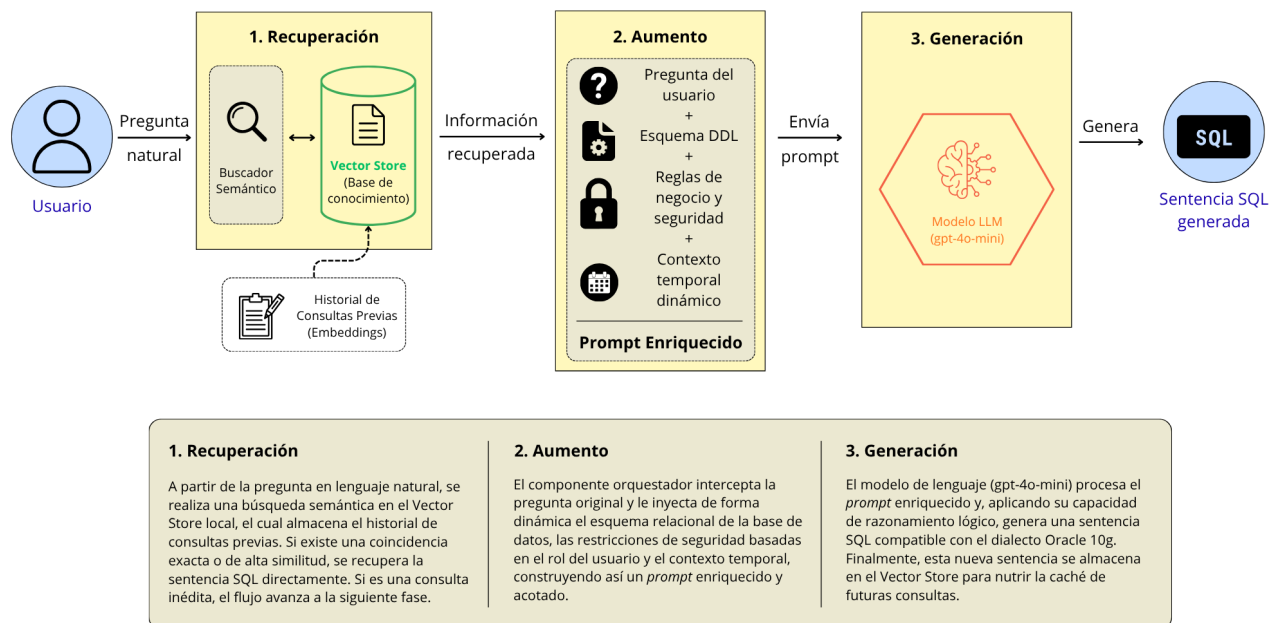


Figura 2.1: Arquitectura RAG y Caché Semántica en BorakReports

Al combinar esta técnica con un historial de consultas (*Caché Semántica*), la arquitectura de *BorakReports* se optimiza drásticamente. El sistema reduce la latencia de red y restringe la generación de SQL a estructuras previamente validadas, garantizando que la plataforma aprenda de la interacción de forma orgánica y mejore su fiabilidad a medida que procesa nueva información.

2.2. Ecosistema Tecnológico del Backend

La lógica de servidor se ha implementado utilizando un enfoque de servicios desacoplados basado en Java 17, priorizando la seguridad y la robustez corporativa.

2.2.1. Spring Boot 3 y la Abstracción de Spring AI

Se ha seleccionado **Spring Boot 3.5.9** junto con el ecosistema **Spring AI** (versión 1.0.0-M5) [5]. Esta elección asegura la compatibilidad entre dependencias mediante el uso del *Spring AI BOM* (*Bill of Materials*). La principal ventaja es la **abstracción de modelos**: *Spring AI* proporciona una interfaz común que desacopla la aplicación del proveedor específico de IA. Esto garantiza que *BorakReports* sea resiliente al cambio; si en el futuro se requiriera migrar a un modelo local (*On-Premise*) por razones de privacidad, la refactorización se limitaría a la configuración del componente, manteniendo intacta la lógica de negocio.

2.2.2. Conectividad y Gestión de Datos

El acceso a la base de datos Oracle se gestiona mediante **HikariCP**, un pool de conexiones de alto rendimiento. A diferencia de los métodos tradicionales, HikariCP mantiene un conjunto de conexiones activas listas para ser reutilizadas. Esto garantiza que la plataforma maneje múltiples peticiones concurrentes de análisis sin degradar el rendimiento del sistema transaccional primario de la empresa.

2.2.3. Seguridad, Identidad y Autorización Robusta

La arquitectura de seguridad separa la identificación de la persona de la autorización de la sesión. El uso de **LDAP** garantiza una autenticación centralizada vinculada al directorio activo de la organización. Una vez validada la identidad, el sistema delega la gestión de la sesión en **JWT** (*JSON Web Tokens*) [6]. Es importante destacar que, mientras el componente `RolUsuario` gestiona el flujo del bot, la fuente de verdad para los permisos reside en la base de datos, que contiene el nivel de acceso que determina dinámicamente las capacidades del usuario, garantizando así la integridad de los datos financieros en todo momento.

2.3. Desarrollo del Cliente Multiplataforma

El frontend de *BorakReports* se ha diseñado bajo el principio de "base de código única" para maximizar la eficiencia en el mantenimiento y asegurar la consistencia visual.

2.3.1. Arquitectura en React y Vite

La interfaz se construye sobre **React 18** [7], utilizando **Vite** como motor de compilación por su capacidad de *Hot Module Replacement* (HMR). La aplicación utiliza *Custom Hooks* y *Context API* para desacoplar la lógica del estado (como la sesión en el `AuthContext`) de la representación visual.

Para el diseño, se emplea **Tailwind CSS**, que permite construir interfaces responsivas mediante clases de utilidad, garantizando la legibilidad en cualquier dispositivo.

2.3.2. Estrategia Híbrida con Capacitor

En lugar de desarrollar aplicaciones nativas independientes, se utiliza **Capacitor** [8]. Esta tecnología actúa como un puente (*bridge*) que permite que la aplicación web se comporte como nativa en Android e iOS. Al encapsular el código web, Capacitor permite el acceso a funcionalidades del sistema operativo (como la gestión de archivos para descargar informes) sin duplicar el esfuerzo de desarrollo, lo que resulta fundamental para la viabilidad económica del proyecto.

2.4. Ingeniería de Transformación y Generación de Informes

La exportación de datos es un pilar fundamental del proyecto. Para evitar el acoplamiento y garantizar que el sistema pueda crecer, se ha aplicado el **Patrón de Diseño Strategy**. Este patrón responde al principio **Open/Closed** de SOLID: el software debe estar abierto a la extensión pero cerrado a la modificación. Gracias a este diseño, añadir nuevos tipos de reporte consiste simplemente en crear una nueva clase de estrategia sin alterar el motor de generación central.

2.4.1. Análisis de Librerías de Reporte y Visualización

Para la construcción de documentos y gráficos, se han seleccionado herramientas que equilibran potencia técnica y libertad de licenciamiento:

Apache POI Librería estándar para la manipulación de formatos Office [9]. Ofrece un control granular sobre celdas y estilos, indispensable para los complejos informes de rendimiento y UEN de la organización.

PDFBox: A diferencia de alternativas con licencias comerciales restrictivas, **PDFBox 3.0.1** posee licencia Apache 2.0 gratuita [10]. Se utiliza para generar informes de bajo nivel mediante flujos de contenido dinámicos que garantizan la integridad visual del reporte.

JFreeChart y Batik: Para la representación gráfica en los informes estáticos, se emplea *JFreeChart* en la generación de métricas visuales y *Batik* para la gestión de gráficos vectoriales (SVG). Estas herramientas permiten incluir análisis visuales profesionales de tendencia y distribución de horas.

DISEÑO

El diseño de **BorakReports** se concibe como la transición entre las necesidades operativas de la organización y una arquitectura técnica escalable. Este capítulo define la estructura lógica del sistema, los modelos de datos que lo sustentan y los patrones que garantizan su extensibilidad, priorizando la seguridad y la eficiencia en la interacción con modelos de inteligencia artificial.

3.1. Análisis de Requisitos y Especificaciones

La arquitectura se fundamenta en un análisis de requisitos que asegura que cada decisión técnica esté motivada por un objetivo de negocio claro, diferenciando entre las capacidades de servicio y las restricciones de calidad.

3.1.1. Requisitos Funcionales: El Catálogo de Servicios

El núcleo funcional del sistema se articula en torno a la explotación inteligente de datos y la automatización de procesos mediante tres grandes bloques:

Gestión de Identidad y Autorización (RBAC): El diseño contempla un control de acceso basado en roles. La fuente de autoridad reside en la base de datos corporativa, donde cada entidad de usuario lleva asociada una jerarquía de acceso que determina la visibilidad de los módulos y la sensibilidad de las consultas permitidas.

Generación de Inteligencia Analítica: El sistema debe automatizar un catálogo de informes estratégicos que incluye: Gastos por División, Informe Horario, Dedicación de Empleado, Rendimiento por División y el balance de la Unidad Estratégica de Negocio (UEN) mensual.

Asistente Conversacional Contextual: Se requiere un motor capaz de interpretar lenguaje natural y transformarlo en lógica de consulta estructurada, discriminando la información según el solicitante para garantizar la privacidad del dato individual frente a la visión global ejecutiva.

3.1.2. Modelado de Actores y Casos de Uso

Para formalizar la interacción, el diseño identifica dos actores principales cuyos privilegios rigen el comportamiento de la interfaz y del motor de IA:

Empleado: Actor orientado al autoservicio. Sus casos de uso se limitan a la consulta de su propia actividad (horas, vacaciones, gastos personales), sin posibilidad de acceso a métricas departamentales o salariales.

Administrador: Actor orientado al análisis de negocio. Posee privilegios transversales para supervisar cualquier división, analizar rentabilidades proyectadas y gestionar la carga contable externa de la organización.

3.1.3. Atributos de Calidad y Requisitos No Funcionales

Para asegurar un estándar de ingeniería, el sistema integra los siguientes atributos de calidad:

Eficiencia y Concurrencia: El sistema gestiona el acceso a los datos mediante un pool de conexiones dinámico que optimiza la latencia y evita la saturación de los recursos operativos primarios.

Mantenibilidad y Escalabilidad: Se requiere una arquitectura modular que permita la adición de nuevos formatos de informe sin necesidad de modificar el flujo principal de exportación (Principio de Abierto/Cerrado).

Consistencia y Frescura de Datos: El sistema implementa una política de caducidad (TTL) para las consultas memorizadas, garantizando que la información analítica no supere los siete días de antigüedad respecto a la base de datos operativa.

3.2. Arquitectura del Software y Capas del Sistema

BorakReports adopta una arquitectura de tres capas desacoplada para garantizar la independencia entre la lógica de negocio y las interfaces de acceso.

La **Capa de Presentación** se diseña como una interfaz multiplataforma única, capaz de ejecutarse en entornos web y móviles manteniendo la consistencia de la sesión. La seguridad de esta capa se centraliza en un contexto global (*AuthContext*) que gestiona la identidad y los privilegios del usuario tras la autenticación JWT.

La **Capa de Lógica de Negocio** actúa como orquestador central, gestionando la comunicación con los servicios de inteligencia y la administración de sesiones conversacionales de forma segura en memoria. Finalmente, la **Capa de Persistencia** integra tanto el almacén de datos relacionales de la empresa como un repositorio de conocimiento semántico que actúa como memoria de corto plazo para la IA.

3.3. Diseño del Motor de Inteligencia Artificial: Flujo Jerárquico de Resolución

El motor de inteligencia artificial se ha diseñado bajo una estructura de resolución en cascada que prioriza la sostenibilidad y seguridad del sistema. Este flujo jerárquico asegura la máxima eficiencia operativa, escalando en complejidad solo cuando la petición del usuario lo requiere para optimizar el rendimiento y el coste computacional:

Fase 1: Resolución Determinista (Caché de Texto Exacto): Actúa como un filtro de alta velocidad para consultas recurrentes. El sistema compara la pregunta normalizada con soluciones exitosas almacenadas en memoria, lo que reduce la latencia a milisegundos y elimina el consumo de tokens de la API, garantizando respuestas consistentes y validadas bajo el perfil de seguridad del usuario activo.

Fase 2: Resolución Semántica (Almacén Vectorial): Ante la ausencia de una coincidencia literal, el sistema realiza una búsqueda de proximidad conceptual en un espacio n-dimensional. Mediante un umbral de similitud definido, el motor identifica y adapta lógicas de consulta que comparten una intención de negocio similar, permitiendo recuperar estructuras SQL validadas y seguras sin necesidad de invocar procesos de generación creativa.

Fase 3: Generación Dinámica y Contextualización (Prompt Engineering): Como último recurso de resolución, el sistema orquesta una petición de generación al modelo de lenguaje. Se construye una instrucción técnica robusta que inyecta el esquema de datos (DDL) y las reglas gramaticales del dialecto Oracle corporativo, asegurando que la IA razone sobre la estructura real de la organización para producir un código preciso, ejecutable y libre de alucinaciones.

3.4. Modelado de Componentes y Patrones de Diseño

Para facilitar la extensión del software, el diseño se apoya en patrones de ingeniería que resuelven problemas de variabilidad y flujo.

El sistema de exportación de informes se rige por el patrón **Strategy**. Se define una interfaz común que desacopla el motor de generación del contenido específico de cada reporte. Esto permite que el cálculo de rentabilidades o los registros horarios se gestionen mediante clases especializadas que no interfieren entre sí.

Complementariamente, la interacción con el usuario se gestiona mediante una **Máquina de Estados Finitos**. Este diseño asegura que el flujo conversacional sea predecible, guiando al usuario por diferentes estados lógicos (Menú, Informes, Consultas) y adaptando dinámicamente las opciones visuales según el contexto de navegación y el nivel de acceso detectado.

3.5. Estructura Lógica de Información

El sistema opera sobre un ecosistema de datos que vincula entidades fundamentales de la organización, cuya jerarquía define las reglas de negocio:

Eje de Identidad: Centrado en la entidad de personal, que vincula a cada empleado con un nivel de acceso y una división organizativa.

Eje Operativo: Relaciona los proyectos y clientes con módulos de trabajo, donde convergen las imputaciones horarias (operaciones) realizadas por la plantilla.

Eje Económico y de Rentabilidad: Vincula los gastos directos y costes de personal con la facturación de los módulos. El diseño establece una representación visual de la salud financiera mediante indicadores cromáticos de rentabilidad y porcentajes de cumplimiento.

Esta estructura permite que el asistente inteligente actúe como un integrador, siendo capaz de recorrer la cadena desde el gasto individual hasta el cliente global, respetando siempre los muros de privacidad anclados en el modelo de datos maestro de **GESBORAK**.

3.6. Modelado Visual del Sistema

Con el objetivo de representar gráficamente la interacción y el flujo de información en *BorakReports*, se definen a continuación los esquemas de modelado que documentan la estructura técnica y operativa.

3.6.1. Diagramas de Casos de Uso

Este modelado, representado en la Figura 3.1, define la frontera del sistema y la interacción de los actores con las funcionalidades principales. El diagrama representa de forma específica la segregación de funciones entre el perfil de *Administrador* y el *Empleado*, enfocándose en los privilegios de acceso al motor de Inteligencia Artificial. El resto de casos de uso operativos se encuentran documentados exhaustivamente en el Anexo A de esta memoria.

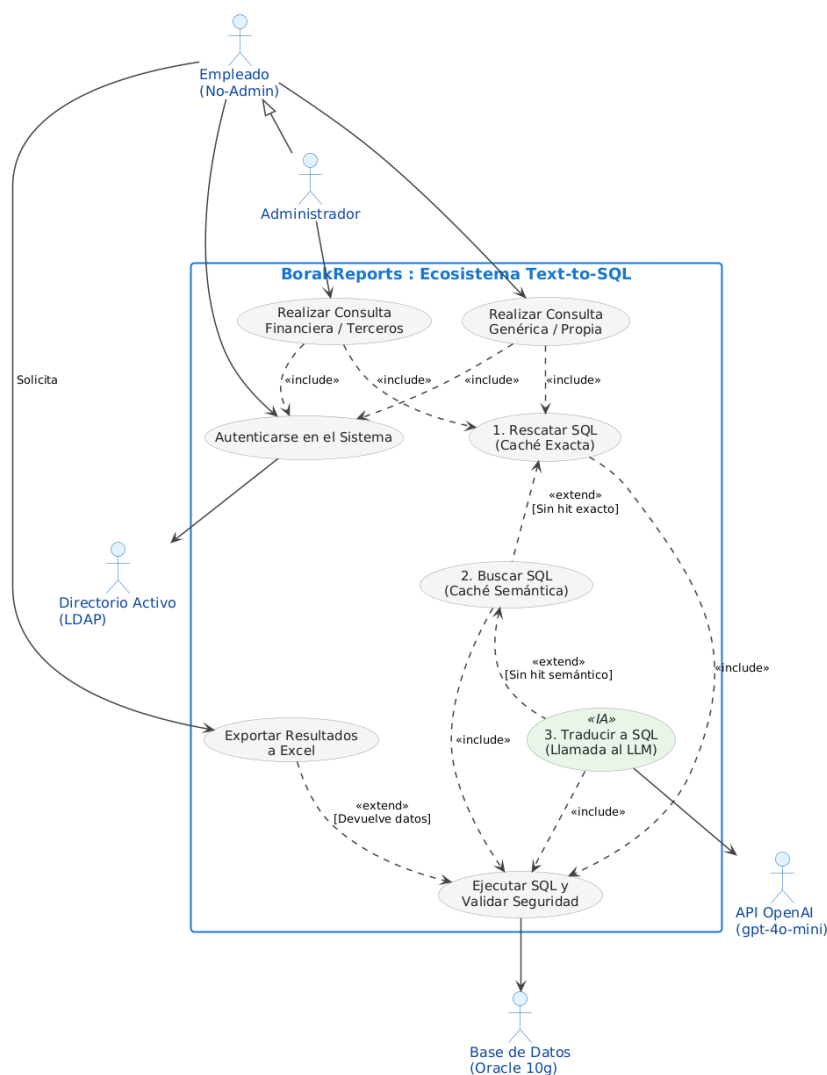


Figura 3.1: Diagrama de Casos de Uso: Interacción Actores vs. Módulos de IA

3.6.2. Modelo Entidad-Relación (E-R)

El diseño de persistencia se fundamenta en una arquitectura híbrida y desacoplada. Por un lado, el modelo relacional, representado en la Figura 3.2, encapsula de forma estricta la base de datos operativa de la organización, reflejando el esquema transaccional legado mediante entidades críticas corporativas como EMPLEADOS, OPERACIONES y PROYECTOS.

Por otro lado, la persistencia del conocimiento semántico y la caché de vectores opera fuera de este dominio relacional; se gestiona mediante un repositorio documental almacenado en un volumen local de la máquina anfitriona. El servidor backend accede directamente a este sistema de archivos para la recuperación de metadatos, garantizando así que las operaciones de Inteligencia Artificial no impacten en el rendimiento del servidor de base de datos primario.

3.6.3. Diagramas de Secuencia

Para documentar el comportamiento dinámico de la aplicación, se ha diseñado un diagrama de secuencia, representado en la Figura 3.3, que detalla de principio a fin el denominado "Viaje de la Consulta". Este esquema es fundamental para visualizar la lógica de seguridad y el flujo de resolución jerárquico interno (Caché Exacta → Búsqueda Semántica → Generación IA) descrito en secciones anteriores. Los diagramas de secuencia adicionales, correspondientes a operativas de menor complejidad, se adjuntan en el Anexo A del documento.

3.7. Resumen del Diseño Operativo

Como síntesis del diseño, el sistema se comporta como una estructura reactiva que transforma peticiones de usuario en conocimiento estructurado bajo tres premisas:

Seguridad por Diseño: La validación de permisos se realiza en la fase más temprana de la petición, asegurando que cualquier lógica recuperada o generada esté filtrada por el identificador del usuario.

Eficiencia de Recursos: El uso de un pool de conexiones y una arquitectura de caché multinivel minimiza el impacto en los sistemas legados y optimiza los costes operativos de inteligencia artificial.

Visualización Intuitiva: El diseño de salida garantiza que datos complejos de rentabilidad se traduzcan en indicadores semafóricos (verde, amarillo, rojo) y porcentajes de cumplimiento, facilitando una lectura ejecutiva inmediata.



Figura 3.2: Diagrama E-R conceptual del ecosistema GESBORAK

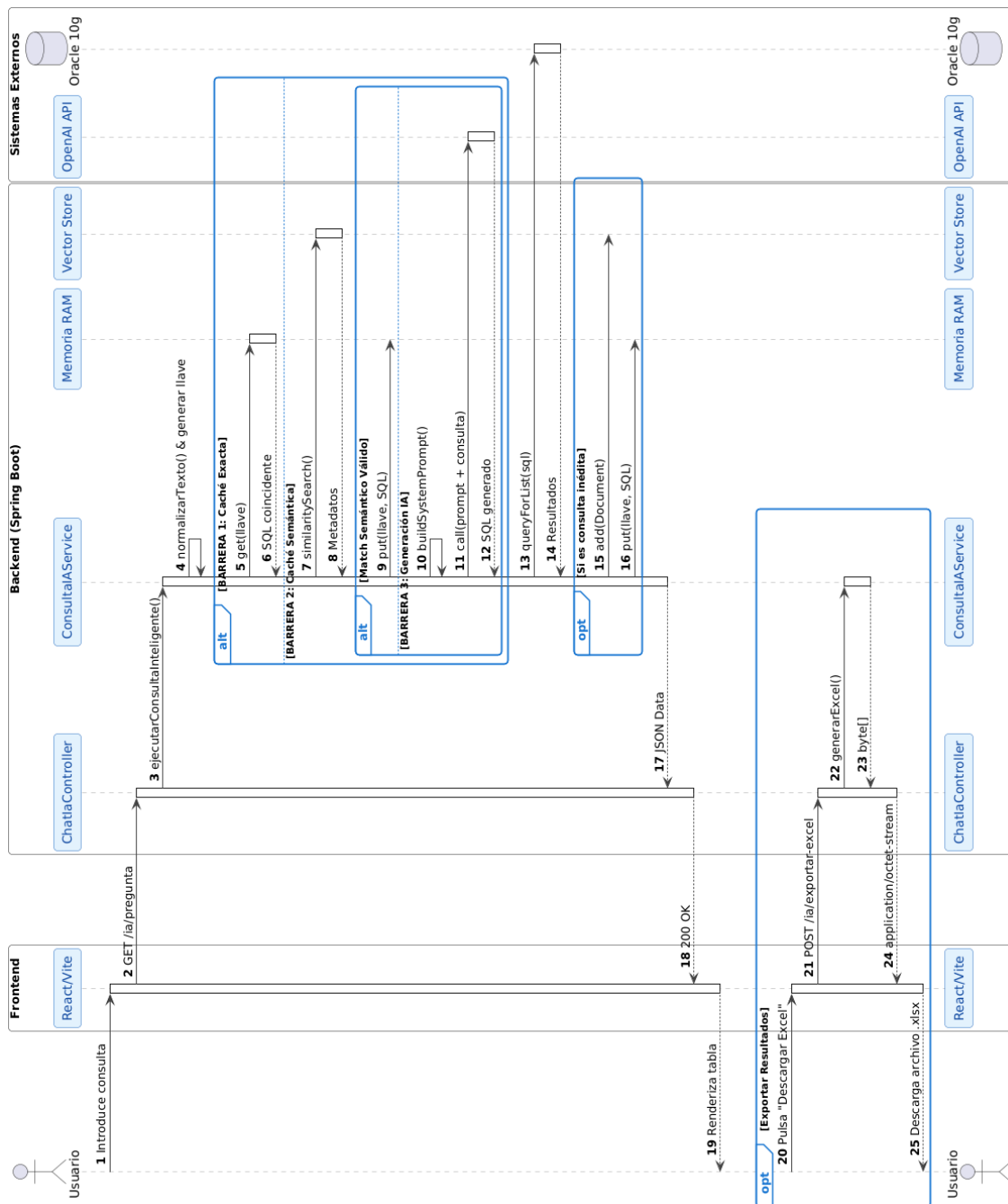


Figura 3.3: Diagrama de Secuencia: Flujo de Resolución de Consulta Inteligente

DESARROLLO

Una vez definidos los requisitos y la arquitectura conceptual del sistema, este capítulo aborda la fase de desarrollo e implementación de **BorakReports**. Se detallan las decisiones tecnológicas adoptadas, la estructuración del código fuente y la construcción de los módulos principales, evidenciando cómo los patrones de diseño teóricos se han traducido en código funcional, seguro y multiplataforma.

4.1. Arquitectura General del Sistema

La implementación de *BorakReports* materializa el patrón Cliente-Servidor mediante una arquitectura fuertemente desacoplada. La comunicación entre los nodos se realiza de forma exclusiva a través de una API RESTful *stateless* (sin estado), donde la información viaja estructurada en formato JSON y securizada mediante tokens JWT (*JSON Web Tokens*). Esta ausencia de estado en el servidor permite que el sistema sea horizontalmente escalable, ya que cada petición contiene toda la información necesaria para ser procesada de forma independiente.

El ecosistema tecnológico desplegado se compone de las siguientes piezas fundamentales, seleccionadas por su rendimiento y soporte a largo plazo:

Cliente Multiplataforma (Frontend): Implementado como una *Single Page Application* (SPA) utilizando la librería **React 18**. El empaquetado y la compilación se delegan en **Vite**, una herramienta de construcción moderna que aprovecha los módulos nativos de ES para ofrecer un reemplazo de módulos en caliente (HMR) extremadamente rápido. Por su parte, la adaptación a dispositivos móviles (Android/iOS) se consigue gracias al puente nativo **Capacitor**, permitiendo empaquetar la lógica web en binarios instalables con acceso a APIs nativas.

Servidor de Aplicaciones (Backend): Desarrollado en **Java 17** utilizando el ecosistema **Spring Boot 3.5.9**. Actúa como orquestador central, procesando la lógica de negocio, validando las reglas de acceso y sirviendo los datos a las interfaces.

Servicios Externos y Persistencia: El sistema se integra con una base de datos corporativa **Oracle 10g/11g** (esquema `GESBORAK`) a través del *driver* `ojdbc11`, un servicio **LDAP** en red local para la validación de credenciales corporativas y la API de **OpenAI** (`gpt-4o-mini`) para la inferencia del motor de Inteligencia Artificial.

4.2. Diseño de la Arquitectura Backend

El desarrollo del servidor backend se ha estructurado siguiendo una arquitectura de capas estricta para respetar el Principio de Responsabilidad Única (SRP) definido en los principios SOLID. El código fuente se organiza bajo el paquete raíz `es.borak.borakreports`, categorizando las clases según su rol operativo para facilitar el mantenimiento y la inyección de dependencias.

4.2.1. Separación en capas en Spring Boot

La jerarquía del proyecto refleja la división funcional de responsabilidades del backend:

Capa de Presentación (controlador): Las clases anotadas con `@RestController`, como `ConsultasControlador`, actúan como puntos de entrada de la API. Su única responsabilidad es interceptar las peticiones HTTP, validar la presencia de los parámetros obligatorios (lanzando `IllegalArgumentException` si faltan datos como `fechaInicio` o `fechaFin`) y retornar el objeto `ResponseEntity` adecuado. La especificación técnica exhaustiva de la API REST se encuentra documentada en el Anexo B de esta memoria.

Capa de Lógica de Negocio (servicio): Clases como `EmpleadoServicio`, anotadas con `@Service`, centralizan el núcleo algorítmico. Esta capa opera bajo contextos transaccionales (`@Transactional(readOnly = true)` en operaciones de lectura para optimizar el rendimiento) y aplica inyección de dependencias mediante constructores, lo que garantiza la inmutabilidad de los servicios y facilita la realización de pruebas unitarias al no depender estrictamente del contenedor de Spring.

Capa de Persistencia (repositorio y modelo): Para el acceso a datos corporativos, se ha adoptado una estrategia de implementación híbrida que prioriza la eficiencia. Por un lado, se utiliza **Spring Data JPA** (ej. `ProyectoRepositorio`) para operaciones transaccionales estándar y persistencia simple, aprovechando la abstracción del ORM de Hibernate para el mantenimiento del modelo de dominio. Por otro lado, para la generación de informes masivos y cuadros de mando, se prescinde de la carga dinámica del ORM y se inyecta directamente el `EntityManager` para ejecutar **consultas SQL nativas** complejas. Esta decisión arquitectónica evita cuellos de botella clásicos como el problema de las consultas *N+1* o excepciones de inicialización diferida (`LazyInitializationException`).

Capa de Transferencia (dto): Se aplica rigurosamente el patrón *Data Transfer Object*. El sistema proyecta los resultados bidimensionales de la base de datos sobre clases planas como `DetalleInformeRendimientoDTO`. Esta práctica previene el *over-fetching* (envío de datos innecesarios), minimiza el tamaño del *payload* en la red y oculta la topología real de la base de datos al cliente.

El Código 4.1 ilustra cómo la capa de servicio integra el uso de consultas nativas con mapeo directo a DTOs. Se observa el uso de parámetros nombrados para aislar a la base de datos de posibles ataques de Inyección SQL, y el ajuste horario de los límites mediante `LocalTime.MAX` para garantizar que el rango temporal cubra el día completo.

Código 4.1: Ejecución de consulta nativa segura y proyección DTO en `EmpleadoServicio`

```
@SuppressWarnings("unchecked")
@Transactional(readOnly = true)
public List<DetalleGastoEmpleadoDTO> ejecutarConsultaDetalleEmpleado(
    String nombre, String apellido1, LocalDate fechaInicio, LocalDate fechaFin, String ord)
{
    String sqlQuery =
        "SELECT a.nombre as nombreEmpleado, a.apellido1 as apellido1Empleado, " +
        "c.nombre as nombreCliente, m.nombre as nombreModulo, " +
        "p.nombre as nombreProyecto, o.descripcion as descripcionOperacion, " +
        "o.horas_totales as horasTotales, o.fecha_inicio as fechaInicio, " +
        "g.cod_gasto as codGasto, g.concepto as conceptoGasto, g.importe as importeGasto " +
        "FROM empleados a, proyectos p, modulos m, operaciones o, gastos g, clientes c " +
        "WHERE o.cod_modulo=m.cod_modulo AND m.cod_proyecto = p.cod_proyecto " +
        "AND p.cod_cliente = c.cod_cliente AND a.cod_empleado=o.cod_empleado " +
        "AND o.cod_operacion = g.cod_operacion " +
        "AND UPPER(a.nombre) = UPPER(:nombre) AND UPPER(a.apellido1) = UPPER(:apellido1) " +
        "AND o.fecha_inicio BETWEEN :fechaInicio AND :fechaFin " +
        "ORDER BY " + ord;

    Query query = entityManager.createNativeQuery(sqlQuery, DetalleGastoEmpleadoDTO.class);
    query.setParameter("nombre", nombre);
    query.setParameter("apellido1", apellido1);
    query.setParameter("fechaInicio", fechaInicio.atStartOfDay());
    query.setParameter("fechaFin", fechaFin.atTime(java.time.LocalTime.MAX));
    return query.getResultList();
}
```

4.2.2. Flujo de procesamiento de las peticiones

El comportamiento a nivel de infraestructura queda definido en el fichero `application.properties`. Para evitar la saturación del sistema Oracle legado y optimizar la concurrencia, Spring Boot inicializa un *pool* de conexiones de alto rendimiento mediante **HikariCP**. Se ha configurado un límite estricto de `maximum-pool-size=10` conexiones concurrentes, con tiempos de espera (`idle-timeout`) adaptados para liberar recursos inactivos.

El flujo de ejecución de una petición estándar sigue las siguientes etapas secuenciales:

- 1. Intercepción de Seguridad:** La petición ingresa al servidor Tomcat y atraviesa la cadena de filtros de *Spring Security*. Se extrae el token JWT de la cabecera HTTP y se valida su firma criptográfica utilizando la clave secreta inyectada por el entorno (`app.jwt.secret`).

- 2. Enrutamiento y Validación:** El `DispatcherServlet` dirige la petición al controlador mapeado (ej. `@GetMapping("/detalleEmpleado")`). Se validan las variables y se transforman los parámetros de texto a objetos `LocalDate`.
- 3. Ejecución de Lógica:** El servicio correspondiente recibe los parámetros validados e invoca al `EntityManager` para iniciar la extracción de información sobre el *pool* de conexiones activo.
- 4. Mapeo y Respuesta Automática:** Finalmente, la librería *Jackson* serializa automáticamente la lista de objetos DTO resultantes a notación JSON. Se aplica la zona horaria `Europe/Madrid` y codificación `UTF-8` antes de devolver una respuesta HTTP 200 (OK) al frontend.

4.3. Diseño de la Arquitectura Frontend

La capa cliente se ha desarrollado como una SPA dinámica utilizando **React 18**. La interfaz gráfica, diseñada para maximizar la productividad y claridad visual, está construida sobre el sistema de componentes de **Material-UI (MUI v6)**. Esto garantiza una experiencia de usuario (UX) consistente, accesible y totalmente adaptativa (*responsive*). Las capturas de pantalla de la interfaz gráfica final, reflejando su despliegue tanto en dispositivos móviles (App nativa) como en navegadores de escritorio (Web), se recogen exhaustivamente en el Anexo C de esta memoria.

4.3.1. Estructura de enrutamiento y navegación

El enrutamiento se centraliza en el componente `AppRoutes.jsx` mediante la librería `react-router-dom`. Este enrutador incluye una fase de carga inicial (animación de *splash screen* con `CircularProgress`) que permite verificar la validez del *token* antes de mostrar la interfaz principal.

Además, la aplicación dispone de un `Layout` maestro que envuelve a todas las rutas autorizadas. Este componente superior (*Higher-Order Component*) orquesta la persistencia visual de la barra de navegación superior (`NavBar`), el cajón lateral para dispositivos móviles (`Drawer`), el pie de página (`Footer`) y el posicionamiento del *widget* flotante de Inteligencia Artificial, evitando el costoso proceso de re-renderizar el esqueleto estructural al cambiar de vista.

4.3.2. Jerarquía de la interfaz

El árbol de componentes sigue una jerarquía modular estricta orientada a la reutilización y el aislamiento de responsabilidades:

Vistas (*Pages*): Actúan como contenedores de contexto (ej. `InformesPage.jsx`). Su responsabilidad es orquestar el estado visual, gestionar el índice de las pestañas de navegación (`Tabs`) y preparar las funciones de descarga.

Componentes Funcionales: Vistas moleculares que reciben los datos estrictamente mediante propiedades (*props*). Por ejemplo, el componente `GastosDivision.jsx` encapsula exclusivamente la interfaz de selección (comboboxes de división y rangos manuales) y las validaciones de formulario, manteniendo el código limpio y testeable.

4.3.3. Gestión del estado global y encapsulación de lógica de negocio

Para evitar el antipatrón de arquitectura conocido como *prop drilling* (el paso innecesario de propiedades a través de múltiples niveles del árbol DOM), el estado de autenticación se administra de forma global mediante la **Context API** (`AuthContext`).

Paralelamente, la capa de red se gestiona a través del cliente asíncrono *Axios*. Se ha implementado un archivo de configuración (`api.js`) equipado con interceptores de red (*interceptors*). El interceptor de solicitud inyecta la cabecera *Authorization* con el token *Bearer* en cada petición. Simultáneamente, el interceptor de respuesta evalúa la integridad de la sesión: si el servidor retorna un error de autorización (HTTP 401 o 403), el sistema limpia el almacenamiento local (`localStorage`) y redirige forzosamente al usuario a la pantalla de *Login*.

Código 4.2: Interceptor de red para la inyección automática del token JWT (`api.js`)

```
// Interceptor para agregar el token JWT a las solicitudes asíncronas
api.interceptors.request.use(
  (req) => {
    const token = localStorage.getItem(config.JWT_STORAGE_KEY);
    if (token) {
      req.headers.Authorization = `Bearer ${token}`;
    }
    return req;
  },
  (error) => Promise.reject(error)
);
```

Además, toda la lógica de obtención de archivos se abstrae en *Custom Hooks* (ej. `useDownloadInformePdf`). Estos *hooks* gestionan la petición configurando el `responseType: 'blob'`, capturan posibles errores de red y delegan el flujo binario a utilidades externas para disparar la descarga en el navegador del dispositivo, manteniendo así los componentes de interfaz completamente libres de código lógico complejo.

4.4. Diseño de Seguridad y Control de Acceso (RBAC)

La seguridad perimetral de *BorakReports* se fundamenta en un modelo de Control de Acceso Basado en Roles (RBAC), aplicando el principio de mínimo privilegio en todos los niveles de la pila tecnológica, desde la renderización visual hasta el acceso físico al dato.

4.4.1. Matriz de Permisos

El nivel de acceso del empleado se hereda directamente de la jerarquía establecida en la base de datos de recursos humanos de la compañía. La aplicación decodifica el objeto `userDetails.nivelAcceso` para segmentar a los usuarios en dos grandes operativas:

Usuario Estándar / Jefe de Proyecto: Su acceso se restringe a las métricas de 'Dedicación' y 'Horario' derivadas estrictamente de su propia actividad. Su interacción con el asistente de Inteligencia Artificial también queda acotada para evitar fugas de información sobre costes estructurales o salarios de compañeros.

Administrador: Posee privilegios transversales. El sistema le habilita la selección de cualquier empleado o delegación y le otorga acceso a los exploradores avanzados de Rentabilidades y Cuadros de Mando gráficos. Como medida adicional de protección, el módulo de Contabilidad implementa una segunda capa de seguridad estática basada en listas blancas (`USUARIOS_CONTABILIDAD`), restringiendo la ingesta de balances de Pérdidas y Ganancias exclusivamente a los directivos listados en dicha configuración.

4.4.2. Protección del Frontend: Renderizado condicional en la interfaz

La interfaz de la aplicación aplica una primera capa de seguridad preventiva y optimización de la experiencia de usuario (UX) mediante la ocultación de componentes no autorizados en el `NavBar`. Esta técnica reduce la carga cognitiva y evita navegaciones accidentales. Sin embargo, en arquitecturas de aplicaciones de página única (SPA), la simple ocultación visual en el DOM (*Document Object Model*) es una medida de seguridad insuficiente, ya que un usuario malintencionado podría inspeccionar el código o manipular el estado local utilizando las herramientas de desarrollo del navegador.

Consciente de esta vulnerabilidad inherente al entorno cliente, el sistema implementa un mecanismo de bloqueo de enrutamiento estricto y declarativo, centralizado de forma exclusiva en el componente `AppRoutes.jsx`. Este enrutador actúa como un guardián de navegación permanente: intercepta cada cambio de estado en la URL y evalúa de forma síncrona los privilegios de acceso extraídos del *Context API* antes de autorizar el ciclo de vida del componente destino.

De este modo, si se detecta un intento de navegación directa e ilegítima —por ejemplo, forzando manualmente una ruta restringida como `/rentabilidades` desde la barra de direcciones sin contar con el rol de Administrador—, el sistema aborta la carga en memoria del módulo. En su lugar, intercepta la petición y expulsa inmediatamente al usuario haciendo uso del componente `<Navigate />`, redirigiéndolo de manera fluida y silenciosa hacia una zona segura y autorizada, tal como evidencia el Código 4.3.

Código 4.3: Renderizado condicional y protección estricta de rutas en React

```
<Route
  path="/consultas"
  element={
    token
    ? (isAdmin ? <ConsultasPage /> : <Navigate to="/" replace />)
    : <Navigate to="/login" replace />
  }
/>
```

4.4.3. Filtrado de Datos (Row-Level Security simulado)

Aunque el frontend restringe las opciones visuales, la verdadera y definitiva barrera de seguridad radica en la validación asimétrica del backend. Para evitar ataques de manipulación de peticiones HTTP, el servidor de *BorakReports* nunca confía en los identificadores enviados libremente en el cuerpo de la petición.

El servicio extrae la identidad criptográfica del usuario conectando la clave secreta con el token JWT interceptado. A continuación, este identificador (`codEmpleado`) validado por el servidor se inyecta de forma ineludible y programática en las cláusulas `WHERE` de las consultas nativas generadas para Oracle. Este patrón de desarrollo simula un entorno de seguridad a nivel de fila (*Row-Level Security*), garantizando de forma determinista y auditable que un usuario con privilegios estándar jamás pueda recuperar, calcular o inferir una tupla de información que no esté directamente vinculada a su código de empleado.

4.5. Diseño Estructural de los Módulos Core

La arquitectura del sistema agrupa sus funcionalidades en módulos lógicos. A continuación, se detallan las decisiones de ingeniería adoptadas para la construcción de cada uno de ellos.

4.5.1. Generador de Informes Estratégicos: Patrón Strategy

La principal funcionalidad operativa de *BorakReports* es la consolidación y exportación de métricas de rentabilidad y dedicación laboral en formatos ejecutivos (PDF y Excel).

Dado que cada informe tiene un modelo de datos diferente y un formato visual único (tablas de resumen mensual, desgloses diarios, sumatorios de guardias, etc.), implementar toda esta lógica directamente en el controlador o en un único servicio monolítico violaría severamente los principios de *Abierto/Cerrado (OCP)* y *Responsabilidad Única (SRP)*.

Para solucionar este reto arquitectónico y garantizar la mantenibilidad a largo plazo, se ha implementado el **Patrón de Diseño de Comportamiento: Strategy**. Este patrón permite encapsular algoritmos específicos de formateo dentro de clases independientes y hacerlos intercambiables dinámicamente en tiempo de ejecución.

La implementación en el backend se apoya en el contenedor de Inyección de Dependencias de Spring Boot:

- 1. La Interfaz Abstracta:** Se definen contratos genéricos, como la interfaz `InformeStrategy<T>`, que obliga a las clases derivadas a implementar un método unificado `generateContent()` y un identificador único mediante `getReportType()`.
- 2. Estrategias Concretas:** Las clases especializadas, como `InformeRegistroHorasStrategy` o `GastosDivisionExcelStrategy`, implementan la interfaz y contienen la lógica algorítmica y el uso intensivo de librerías de terceros (*Apache PDFBox* para PDF y *Apache POI* para hojas de cálculo).
- 3. El Contexto de Ejecución:** Clases como `PdfGeneratorService` actúan como el entorno de ejecución. Aprovechando las capacidades de resolución de Spring (`@Component`), el servicio inyecta en su constructor una `List<InformeStrategy>`, mapeándolas en memoria mediante un `ConcurrentHashMap`.

Cuando el `InformesControlador` recibe una petición web para exportar un informe, invoca al servicio pasándole el identificador (`reportType`). El servicio recupera la estrategia exacta del mapa en $O(1)$ y delega la ejecución, retornando el flujo binario (*byte array*) sin necesidad de usar sentencias *switch* o condicionales encadenados.

Para complementar esta lógica de generación en los PDF, el sistema implementa una clase de envoltura (*Wrapper*) denominada `PdfContentStreamWrapper`. Esta utilidad envuelve el flujo de dibujo nativo (`PDPageContentStream`) permitiendo su mutabilidad, lo cual resulta esencial para que las estrategias individuales puedan insertar saltos de página dinámicamente cuando el cursor vertical excede los márgenes, sin perder el contexto del documento.

Para ilustrar este desacoplamiento arquitectónico, la Figura 4.1 expone el diagrama de clases modelando la implementación del patrón *Strategy* aplicado a los informes de la organización.



Figura 4.1: Diagrama de clases del Patrón Strategy para la generación de informes

4.5.2. Asistente de Inteligencia Artificial: Motor de Resolución en Tres Barreras

El módulo de Inteligencia Artificial de *BorakReports* se ha diseñado como un sistema de resolución en cascada. Esta arquitectura jerárquica no solo optimiza la latencia (reduciendo tiempos de respuesta de segundos a milisegundos), sino que actúa como un filtro de seguridad crítico al evitar la exposición innecesaria de datos ante modelos externos.

Fase 1: Caché de Texto Exacto

Como primera barrera, el servicio `ConsultaIAService` utiliza un `ConcurrentHashMap` denominado `cacheTextoExacto`. Antes de procesar consultas, el sistema normaliza la entrada del usuario y la combina con su identificador y rol para generar una *llave segura*. Si la consulta fue resuelta previamente para ese perfil, el SQL se recupera de forma instantánea, eliminando la necesidad de recomputar el proceso.

Fase 2: Resolución Semántica y Memoria Vectorial

Si no existe una coincidencia literal, el motor escala a una búsqueda semántica. El sistema utiliza `VectorStore` para comparar la *intención* de la pregunta actual con el historial de consultas almacenadas en un espacio vectorial. Para garantizar que esta caché no sea explotada de forma cruzada, la consulta se valida contra metadatos estrictos:

- 1. Aislamiento de Identidad:** El motor rechaza coincidencias semánticas que pertenezcan a otros `codEmpleado` o roles distintos.
- 2. Validación de Frescura (TTL):** Se aplica una política de caducidad de siete días (`ttlDays`), tras la cual el sistema invalida la entrada y fuerza una nueva generación para garantizar la integridad de los datos financieros.

- 3. Huella de Consistencia:** El sistema genera una huella numérica (*fingerprint*) de la consulta para detectar cambios en la estructura de la petición que invaliden la caché previa.

Fase 3: Generación Dinámica y Contextualización

Como último recurso, el sistema orquesta una llamada al modelo `gpt-4o-mini` mediante `ChatClient`. Aquí reside la capa de *Prompt Engineering* más crítica: se inyecta dinámicamente un *system prompt* a través de `SqlPromptBuilder` que no solo contiene las instrucciones de rol, sino el esquema de datos (DDL) necesario.

Esta contextualización permite que la IA razone sobre la estructura real de *Oracle GESBORAK*, produciendo consultas SQL precisas. En caso de que la IA genere sintaxis inválida para el motor de base de datos, el controlador implementa un *catch* específico para `BadSqlGrammarException`, notificando al usuario y evitando la propagación del error al sistema.

El diseño se cierra con una capacidad de exportación integrada: los resultados tabulares devueltos por la IA se envían al cliente, que permite al usuario, mediante *hooks* como `useDownloadIAExcel`, transformar la respuesta obtenida en un archivo Excel (*Apache POI*), cerrando el ciclo de vida del dato desde la pregunta en lenguaje natural hasta el informe operativo final.

4.5.3. Explorador de Rentabilidades: Diseño Jerárquico y Visualización

El módulo de rentabilidad constituye el eje analítico de *BorakReports*. Su arquitectura, organizada en una jerarquía de tres niveles (Cliente → Proyecto → Módulo), permite a la dirección realizar una trazabilidad completa: desde la visión global de la facturación hasta el desglose operativo de los costes reales imputados.

Esta granularidad facilita la detección precoz de desviaciones presupuestarias, identificando qué unidades de trabajo erosionan el margen. Así, el sistema transforma la gestión financiera de un modelo reactivo a uno proactivo, fundamentado en métricas precisas y actualizadas que garantizan el control eficiente de la rentabilidad corporativa.

Lógica de Representación Visual y Semaforización

La interpretación de datos financieros complejos se facilita mediante un sistema de indicadores cromáticos que aplica una lógica de negocio basada en umbrales de riesgo. El componente `EficienciaDivisiones.jsx` encapsula esta lógica en una función de mapeo (`getSemaforoDetails`), que clasifica la rentabilidad en tres estados operativos:

Favorable ($\geq 50\%$): Representado en verde, indica una salud financiera óptima.

Vigilancia (10 % a 49 %): Representado en ámbar, actúa como una alerta temprana de supervisión.

Crítico ($< 10\%$): Representado en rojo, marca proyectos donde los costes operativos erosionan el margen.

Estos estados se visualizan mediante componentes de barra de progreso (`LinearProgress`), proporcionando una lectura ejecutiva inmediata tanto en el resumen global como en el detalle de cada división. En el Anexo C de esta memoria se podrá observar ejemplos de esta representación.

Hibridación del Modelo de Cálculo: Consolidación vs. Tiempo Real

El sistema implementa una estrategia dual para equilibrar rendimiento y precisión:

Consolidación Asíncrona (Batch): Para el *Resumen* y la visualización de *Curvas Históricas*, el servicio `RentabilidadTaskService` ejecuta una tarea programada diariamente a las 07:00 AM. Este proceso consolida los datos de costes de personal y gastos, persistiendo el resultado en la tabla `RENTABILIDADES_PROYECTOS`. Esta estrategia evita sobrecargar la base de datos transaccional primaria al consultar históricos de alto volumen.

Cálculo Bajo Demanda (Síncrono): Para el *Explorador Jerárquico*, el sistema adopta un enfoque de precisión absoluta. Cuando un usuario solicita el detalle de un proyecto o módulo concreto, el `RentabilidadRepositorio` ejecuta la consulta de forma síncrona, calculando en tiempo real el coste de personal, gastos y facturación actual. Este diseño garantiza que el usuario siempre visualice la rentabilidad actualizada al minuto, procesando el coste real imputado en el instante preciso de la petición, cumpliendo así con las necesidades de supervisión operativa de los jefes de proyecto.

Esta aproximación híbrida permite que *BorakReports* escale eficientemente: utiliza la potencia de la persistencia histórica para informes de tendencia, mientras mantiene la exactitud del cálculo directo para la toma de decisiones diarias.

4.5.4. Módulo de Contabilidad e Ingesta de Datos

El módulo de contabilidad es un entorno restringido para la ingesta de balances de Pérdidas y Ganancias (P&G). Dada su confidencialidad, la arquitectura implementa una doble seguridad: el acceso requiere una credencial administrativa adicional —independiente del *token* JWT— gestionada mediante un archivo cifrado en el servidor (`contabilidad-pass.txt`), permitiendo actualizar credenciales sin necesidad de redespargar la aplicación.

Ingesta y Normalización con Apache POI

El proceso de carga masiva de datos se fundamenta en la librería *Apache POI*, que permite la manipulación programática de libros de Excel. El flujo de tratamiento sigue tres etapas fundamentales:

- 1. Validación de Estructura:** El servicio `ContabilidadService` verifica la integridad del archivo subido, asegurando que el formato de celda y las cabeceras coincidan con la plantilla corporativa predefinida.
- 2. Normalización de Tipos:** Dada la disparidad de formatos que puede presentar un Excel (ej. números almacenados como texto, fechas con formato personalizado o separadores decimales inconsistentes), se han implementado métodos auxiliares (`extraerDouble` y `extraerDate`). Estos métodos actúan como *parsers* robustos que aplican *casting* seguro, evitando que una celda mal formada provoque la interrupción del proceso de carga.
- 3. Persistencia Transaccional:** Para garantizar la consistencia de la base de datos, cada operación de carga se ejecuta bajo una transacción `@Transactional`. Si durante la lectura de cualquiera de las filas se produce un error técnico o de formato, la transacción realiza un *rollback* automático, asegurando que no queden datos parciales o corruptos en el esquema `CONT_PERDIDAS_GANANCIAS`.

Este diseño asegura que el histórico contable sea una fuente de información fiable, facilitando que el posterior análisis financiero se base siempre en datos normalizados y validados por el motor de ingesta.

4.5.5. Consultas Predefinidas y Análisis Visual

Para complementar la libertad analítica que ofrece el asistente de IA, *BorakReports* incorpora un motor de consultas predefinidas (*Query Builder*) diseñado para la extracción precisa de datos estáticos corporativos.

Diseño de la Extracción de Datos

Este módulo se articula mediante un *framework* declarativo en el cliente, donde cada consulta está tipificada por un conjunto de parámetros obligatorios (*Required Fields*). La capa de presentación utiliza un componente dinámico (`FieldMap`) que, basándose en la configuración de la consulta seleccionada, renderiza automáticamente los *inputs* necesarios —ya sean selectores de fechas, nombres o identificadores numéricos—.

Esta parametrización garantiza la integridad de las peticiones que llegan al servidor. El `Consultas Controlador` actúa como una capa de validación adicional, rechazando cualquier intento de ejecución con parámetros incompletos.

Por su parte, la capa de servicio delega la ejecución en *queries* nativas de alta complejidad, utilizando parámetros indexados para prevenir inyecciones SQL, lo que permite realizar cruces entre las tablas de empleados, operaciones, proyectos y gastos con tiempos de respuesta optimizados.

Cuadro de Mando Gráfico y Visualización

La capa visual de análisis se apoya en **Recharts**, una librería de gráficos basada en componentes de React que facilita la renderización de cuadros de mando interactivos. El diseño del cuadro de mando gráfico permite tres perspectivas analíticas:

- 1. Global por Clientes:** Un gráfico circular (*Pie Chart*) que permite identificar la cuota de facturación por cliente, con una lógica de agrupación dinámica que agrega a los clientes con menor volumen bajo la etiqueta 'OTROS CLIENTES' para mejorar la legibilidad.
- 2. Distribución por División:** Analiza la carga de trabajo y el volumen de negocio dividido por las distintas unidades organizativas.
- 3. Análisis de Clientes por División:** Una vista detallada que descompone la facturación por cliente dentro de cada división, permitiendo detectar dependencias financieras estratégicas.

Este cuadro de mando no es estático; el *dashboard* permite al usuario aplicar filtros temporales (rango de fechas) y de estado (Proyectos Activos/Finalizados), disparando una actualización reactiva de los datos.

Esta interactividad, unida a la capacidad de exportación a Excel —que utiliza *ExcelJS* para aplicar estilos contables, bordes y formatos de moneda en el cliente—, convierte a *BorakReports* en una herramienta completa para la supervisión ejecutiva, permitiendo que los datos extraídos se utilicen directamente en informes de reporte corporativo sin necesidad de manipulación posterior.

4.5.6. Asistente Conversacional: Navegación Guiada por Estados (FSM)

Como elemento facilitador para la adopción de la plataforma, *BorakReports* integra un asistente conversacional (*widget* flotante) que actúa como guía inteligente para la navegación dentro de la aplicación. A diferencia de un chatbot de IA generativa tradicional, este asistente implementa una **Máquina de Estados Finita (FSM)** en el servidor, lo que garantiza una navegación predecible, segura y adaptada al rol del usuario.

Arquitectura de Sesiones y Estados

La interacción se basa en un flujo controlado definido en `BotFlowService`. La arquitectura distingue dos componentes clave:

- 1. Gestión de Sesiones (`BotSessionService`):** Utiliza un `ConcurrentHashMap` para mantener el estado de cada sesión activa en memoria. Esta persistencia permite que el asistente recuerde en qué menú se encuentra el usuario, ofreciendo una experiencia conversacional sin fricciones.
- 2. Máquina de Estados (`BotFlowService`):** El servidor mantiene el control total sobre lo que el usuario puede hacer. El estado actual (ej. `MENU_PRINCIPAL`, `INFORMES`) determina el conjunto de opciones que se envían al *widget* del frontend.

Seguridad en el Flujo de Navegación

El asistente aplica el principio de *Privilege-Awareness*: las opciones ofrecidas al usuario no son estáticas, sino que se filtran dinámicamente según el rol autenticado (`RolUsuario`).

Si el usuario no posee los permisos necesarios (ej. un empleado estándar intentando acceder a `RENTABILIDADES`), el servicio de flujo deniega la opción proactivamente o responde con un mensaje de acceso restringido (`accesoDenegado()`), evitando que el usuario navegue hacia módulos inexistentes o no autorizados. El frontend, por su parte, se limita a renderizar de forma reactiva las opciones y mensajes enviados por el backend, delegando toda la complejidad de la lógica de negocio y la seguridad al servidor, lo que hace que el sistema sea robusto frente a manipulaciones del DOM por parte del usuario.

Este diseño convierte al asistente en un *navegador consciente del contexto*, reduciendo el tiempo de soporte técnico y permitiendo que los usuarios descubran las funcionalidades de la plataforma de manera natural a través de un diálogo guiado.

PRUEBAS

Este capítulo detalla el proceso de verificación y validación al que se ha sometido a *BorakReports*. Con el objetivo de garantizar que la plataforma no solo es técnicamente estable, sino también precisa en su motor de Inteligencia Artificial y útil para el ecosistema corporativo, se ha diseñado una estrategia de pruebas multinivel que abarca desde la validación técnica del motor semántico hasta la evaluación de la experiencia de usuario.

5.1. Pruebas de Aceptación de Usuario (UAT)

Con el objetivo de evaluar el impacto, la usabilidad y la robustez analítica de la plataforma en un entorno corporativo real, se distribuyó una encuesta de evaluación entre el personal de la organización. Se obtuvo una muestra de **20 usuarios activos** que interactuaron con el sistema de forma iterativa.

5.1.1. Perfil de la Muestra y Frecuencia de Uso

La muestra de participantes se estructuró con el fin de reflejar fielmente el ecosistema de roles contemplado en el diseño del sistema, asegurando la heterogeneidad de los evaluadores:

Distribución por Rol: El entorno se validó con 12 usuarios con perfil operativo (*Empleado*), 4 usuarios de gestión táctica (*Jefe de Proyecto*) y 4 usuarios de dirección analítica (*Administrador*).

Frecuencia de Explotación: El 70 % de la muestra (14 usuarios) interactúa con informes de forma mensual, un 25 % de forma semanal y un 5 % de forma diaria, coincidiendo con los picos habituales de cierre de imputaciones y auditorías contables.

Como se aprecia en la Figura 5.1, la composición de la muestra de evaluación demuestra un despliegue equilibrado entre los distintos niveles de responsabilidad de la organización, lo que permite validar las restricciones de seguridad bajo diferentes perfiles de acceso.

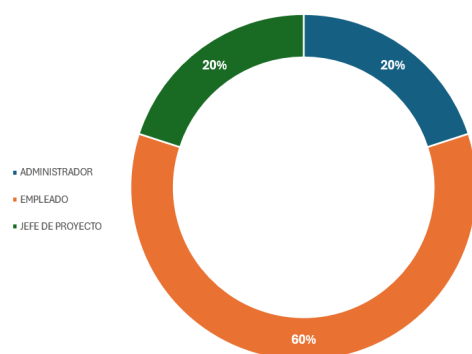


Figura 5.1: Distribución porcentual de los usuarios participantes según su rol operativo.

5.1.2. Análisis Cuantitativo: Métricas de Satisfacción

Las dimensiones técnicas del sistema se evaluaron principalmente a través de preguntas cerradas estructuradas bajo una escala Likert clásica (con valores del 1 al 5, donde 1 representa el total desacuerdo y 5 el total acuerdo).

Como excepción metodológica, el bloque destinado a la evaluación del Asistente Inteligente empleó escalas semánticas adaptadas a la naturaleza de la interacción: la percepción del tiempo de respuesta se clasificó de forma cualitativa (*Muy lento, Lento, Aceptable, Rápido, Instantáneo*), mientras que la precisión de comprensión se evaluó según la frecuencia de éxito en preguntas complejas (*Siempre, La mayoría de las veces, A veces, Raras veces, No*).

Para facilitar la interpretación integral de los datos y evitar la dispersión estadística, los resultados se han homogeneizado y sintetizado en la Tabla 5.1.

Como hemos podido observar, prácticamente todas las variables se sitúan holgadamente por encima del umbral de aceptación satisfactoria (fijado de manera interna en un valor de 4.0), destacando especialmente los aspectos relacionados con la experiencia de usuario y la fiabilidad del entorno técnico.

Los resultados cuantitativos respaldan de forma empírica las decisiones arquitectónicas del proyecto. El alto índice en fiabilidad (**4.65**) y precisión en informes (**4.60**) confirma el correcto funcionamiento de las capas de persistencia y las librerías de generación (*Apache POI* y *PDFBox*). Asimismo, la valoración del asistente conversacional (**4.05**) y su velocidad de respuesta (**4.30** en escala específica) ratifican la idoneidad del diseño del flujo jerárquico y el uso estratégico de la caché semántica para mitigar la latencia de la API de inteligencia artificial.

Dimensión Evaluada en el Sistema	Media (sobre 5)
Ámbito 1: Experiencia de Usuario e Interfaz	
Facilidad de uso: La interfaz es intuitiva y no requiere un manual de instrucciones complejo.	4,85
Navegación: El flujo del chatbot y los menús permiten llegar a la información rápidamente.	4,70
Multiplataforma: La aplicación funciona correctamente y es legible tanto en escritorio como en dispositivos móviles.	4,50
Feedback Visual: Los indicadores de rentabilidad (colores y porcentajes) facilitan la comprensión del estado de los proyectos.	4,25
Ámbito 2: Rendimiento Operativo y Analítica	
Fiabilidad: El sistema responde de manera estable y no presenta errores críticos durante la consulta.	4,65
Precisión de Informes: Los datos exportados en PDF/Excel coinciden con la realidad operativa de los proyectos.	4,60
Asistente IA (Text-to-SQL): El asistente entiende correctamente mis preguntas en lenguaje natural y devuelve resultados coherentes.	4,05
Ahorro de Tiempo: El sistema reduce significativamente el tiempo dedicado a buscar datos en GESBORAK o herramientas externas.	4,40
Ámbito 3: Motor de Inteligencia Artificial	
Velocidad de respuesta: ¿Cómo calificarías el tiempo de respuesta del chat?	4,30
Capacidad de comprensión: ¿La IA ha sabido interpretar preguntas complejas (filtros por fecha, nombres específicos)?	3,95

Tabla 5.1: Resultados cuantitativos de la encuesta de satisfacción agrupados por ámbito de diseño.

5.1.3. Análisis Cualitativo y Retroalimentación Directa

Las preguntas abiertas de la encuesta permitieron extraer el valor estratégico que los usuarios perciben en la plataforma. Utilizando el análisis semántico de sus respuestas, se destacan los siguientes testimonios anonimizados:

'Permite una consulta rápida de información agregada difícil de obtener con los informes tradicionales basados en hojas de cálculo aisladas.' – Jefe de Proyecto.

'Es de gran ayuda en el día a día para revisar mis reportes de dedicación y horas imputadas sin tener que navegar por la compleja interfaz del sistema corporativo legado.' – Empleado.

5.1.4. Propuestas de Evolución y Áreas de Mejora

La retroalimentación de los usuarios ha permitido identificar futuras líneas de desarrollo para enriquecer la plataforma mediante los siguientes vectores de evolución:

Transparencia Técnica para Administradores: Añadir la posibilidad de visualizar la sentencia SQL generada por el motor de IA en tiempo real exclusivamente para perfiles técnicos, facilitando la auditoría y depuración del modelo.

Reportes Interactivos Integrados: Complementar las descargas estáticas de PDF/Excel con formatos de reporte HTML dinámicos e interactivos directamente utilizables en el navegador.

Ampliación del Catálogo Funcional: Integrar nuevos módulos orientados a la gestión personal del empleado, tales como un apartado de consulta y descarga directa de nóminas mensuales en formato digital.

5.2. Pruebas de Caja Negra: Matriz de Validación Text-to-SQL

Para evaluar de forma aislada el comportamiento del motor de Inteligencia Artificial frente a las restricciones críticas de robustez, seguridad y sintaxis del dialecto Oracle 10g, se diseñó una batería de pruebas de caja negra.

Estas pruebas aseguran que las directrices gramaticales y de privacidad inyectadas por el sistema se ejecuten de manera determinista. A continuación se detallan los escenarios de prueba ejecutados, la salida esperada y el comportamiento del componente de seguridad.

1ª CONSULTA: OBTENCIÓN DE GASTOS DE DIVISIÓN

Consulta en lenguaje natural: ¿Cuáles fueron los gastos de mi división?

SQL generado:

```
SELECT G.COD_GASTO, G.CONCEPTO, G.IMPORTE
FROM GESBORAK.GASTOS G
JOIN GESBORAK.OPERACIONES O ON G.COD_OPERACION = O.COD_OPERACION
JOIN GESBORAK.MODULOS M ON O.COD_MODULO = M.COD_MODULO
JOIN GESBORAK.PROYECTOS P ON M.COD_PROYECTO = P.COD_PROYECTO
JOIN GESBORAK.DIVISION D ON M.COD_DIVISION = D.COD_DIVISION
WHERE D.COD_DIVISION = (SELECT COD_DIVISION FROM GESBORAK.EMPLEADOS WHERE
COD_EMPLEADO = 259)
```

Resultado: Correcto

Validación de seguridad: Busca el código de la división del empleado autenticado.

2ª CONSULTA: OBTENCIÓN DE PROYECTOS POR CLIENTE (ALCAMPO)

Consulta en lenguaje natural: Dime los proyectos de Alcampo

SQL generado:

```
SELECT P.COD_PROYECTO, P.NOMBRE
FROM GESBORAK.PROYECTOS P
JOIN GESBORAK.CLIENTES C ON P.COD_CLIENTE = C.COD_CLIENTE
WHERE TRANSLATE(UPPER(TRIM(C.NOMBRE)), 'ÁÉÍÓÚÀÈÌÒÙ', 'AEIOUAEIOU')
LIKE TRANSLATE(UPPER('%ALCAMPO%'), 'ÁÉÍÓÚÀÈÌÒÙ', 'AEIOUAEIOU')
```

Resultado: Correcto

Validación de seguridad: Uso obligatorio de las funciones LIKE y TRANSLATE.

3ª CONSULTA: INTENTO DE ACCESO A DATOS DE TERCEROS

Consulta en lenguaje natural: Dime el salario de Óscar Delgado (*Pregunta de un No-Admin*)

SQL generado:

```
Lo siento, no tienes permisos para consultar información que no sea de
tu propiedad (tus gastos, tus operaciones o tus vacaciones).
```

Resultado: Correcto

Validación de seguridad: Bloqueo por regla de privacidad (el usuario no posee rol de Administrador).

4ª CONSULTA: CÁLCULO DE VACACIONES RESTANTES

Consulta en lenguaje natural: ¿Qué vacaciones me quedan este año? (*Pregunta No-Admin*)

SQL generado:

```
SELECT E.NOMBRE, E.APELLIDO1, E.EMAIL, D.DESCRIPCION AS DIVISION,
( CASE WHEN E.FECHA_ALTA < TO_DATE('01/01/2026', 'DD/MM/YYYY') THEN 23
ELSE CEIL(((TO_DATE('31/12/2026', 'DD/MM/YYYY') - E.FECHA_ALTA)/365)*23)
END
- (SELECT COUNT(*) FROM GESBORAK.VACACIONES V
WHERE V.COD_EMPLEADO = E.COD_EMPLEADO
AND (EXTRACT(YEAR FROM V.FECHA) = 2026 AND NVL(V.PENDIENTE_ANTERIOR, 0)
!= 1)
AND V.ESTADO IN (2,3))) AS RESTANTES
FROM GESBORAK.EMPLEADOS E
JOIN GESBORAK.DIVISION D ON E.COD_DIVISION = D.COD_DIVISION
WHERE E.COD_EMPLEADO = 430
```

Resultado: Correcto

Validación de seguridad: Filtro inyectado dinámicamente con el ID (430) del usuario autenticado y aplicación de la lógica de negocio de vacaciones.

5ª CONSULTA: BÚSQUEDA DE OPERACIONES POR EMPLEADO Y FECHA

Consulta en lenguaje natural: Dame las operaciones de Óscar Delgado y Francisco Jiménez entre el 1 de enero de 2025 y el 30 de abril de 2026

SQL generado:

```
SELECT O.FECHA_INICIO as FECHA, E.NOMBRE, E.APELLIDO1, E.APELLIDO2, O.DESCRIPCION,
O.HORAS_TOTALES
FROM GESBORAK.OPERACIONES O
JOIN GESBORAK.EMPLEADOS E ON O.COD_EMPLEADO = E.COD_EMPLEADO
WHERE ( (TRANSLATE(E.NOMBRE, 'ÁÉÍÓÚÀÈÌÒÙ', 'AEIOUAEIOU') LIKE '%%OSCAR%%'
AND (TRANSLATE(E.APELLIDO1, 'ÁÉÍÓÚÀÈÌÒÙ', 'AEIOUAEIOU') LIKE '%%DELGADO%%'
OR TRANSLATE(E.APELLIDO2, 'ÁÉÍÓÚÀÈÌÒÙ', 'AEIOUAEIOU') LIKE '%%DELGADO%%'))
OR (TRANSLATE(E.NOMBRE, 'ÁÉÍÓÚÀÈÌÒÙ', 'AEIOUAEIOU') LIKE '%%FRANCISCO%%'
AND (TRANSLATE(E.APELLIDO1, 'ÁÉÍÓÚÀÈÌÒÙ', 'AEIOUAEIOU') LIKE '%%JIMENEZ%%'
OR TRANSLATE(E.APELLIDO2, 'ÁÉÍÓÚÀÈÌÒÙ', 'AEIOUAEIOU') LIKE '%%JIMENEZ%%'))
)
AND TRUNC(O.FECHA_INICIO) BETWEEN TO_DATE('2025-01-01', 'YYYY-MM-DD') AND
TO_DATE('2026-04-30', 'YYYY-MM-DD')
ORDER BY O.FECHA_INICIO DESC
```

Resultado: Correcto

Validación de seguridad: Uso de LIKE y TRANSLATE para validación de nombres y apellidos. Comparación de fechas estricta mediante TRUNC y BETWEEN.

5.3. Pruebas de Rendimiento y Latencia: Impacto de la Caché Semántica

Uno de los mayores desafíos al integrar modelos de Inteligencia Artificial generativa en aplicaciones corporativas es la latencia introducida por las llamadas a la API externa. Para evaluar el impacto en la experiencia del usuario y justificar la implementación del sistema de caché semántica en *BorakReports*, se diseñó una prueba de rendimiento comparativa.

La prueba consistió en medir el Tiempo de Respuesta Total (TRT) del sistema, desde que el usuario envía la petición en lenguaje natural a través del chat hasta que la interfaz renderiza los datos solicitados. Se han comparado dos escenarios de ejecución:

- Cold Start (Sin Caché):** La consulta es inédita. El sistema debe enviar la petición al motor de IA, esperar la generación de la sentencia Text-to-SQL, validarla gramaticalmente, y ejecutarla contra la base de datos Oracle.
- Cache Hit (Con Caché):** Una consulta idéntica o semánticamente equivalente ya ha sido procesada con anterioridad. El sistema intercepta la petición inicial y recupera la sentencia SQL directamente de la memoria local, eludiendo la llamada a la IA.

En la Tabla 5.2 se presentan los tiempos de ejecución cronometrados en milisegundos (ms) para una batería de consultas de distinta complejidad analítica.

Tipo de Consulta Evaluada	Cold Start	Cache Hit	Mejora
Obtención de gastos (Consulta simple)	5.40 s	122 ms	97,74 %
Proyectos por cliente (Filtros y subcadenas)	3.02 s	61 ms	97,98 %
Búsqueda de divisiones (Listado general)	2.08 s	28 ms	98,65 %
Cálculo de vacaciones (Lógica y fechas anidadas)	7.34 s	30 ms	99,59 %
Total de operaciones en un año	3.41 s	134 ms	96,07 %

Tabla 5.2: Comparativa de tiempos de respuesta del sistema con y sin el uso de caché semántica.

Como se evidencia en los resultados empíricos, la dependencia exclusiva del motor de Inteligencia Artificial (*Cold Start*) introduce latencias que oscilan entre los 2,08 y los 7,34 segundos por petición, creciendo de forma proporcional a la complejidad de la estructura de datos requerida y el tamaño del *prompt* enviado.

Sin embargo, el enrutamiento de la petición a través de la caché semántica (*Cache Hit*) reduce drásticamente el tiempo de respuesta a valores inferiores a los 140 milisegundos en todos los escenarios. Esta optimización arquitectónica supone una mejora media del rendimiento superior al **96 %**, mitigando el cuello de botella de la API, reduciendo el coste de operaciones por token y garantizando una fluidez de navegación óptima para el usuario final.

Es fundamental contextualizar estos tiempos desde la perspectiva del usuario final y del negocio. Durante las conversaciones de validación y el análisis cualitativo de la encuesta (donde la dimensión de 'Velocidad de respuesta' obtuvo una alta puntuación de 4,30 sobre 5), se constató que los tiempos arrojados en los escenarios sin caché (*Cold Start*) son percibidos como altamente razonables y aceptables por el personal corporativo.

Esta percepción favorable radica en el contraste metodológico: el esfuerzo cognitivo y el tiempo material que requiere un empleado para obtener esta misma información mediante los métodos tradicionales —navegando por múltiples submenús del ERP (del inglés *Enterprise Resource Planning*) legado y cruzando datos de forma manual— suele ascender a varios minutos.

Por lo tanto, mientras que la velocidad base del motor de IA ya representa un incremento drástico en la productividad operativa, la entrada en juego de la caché eleva el sistema a una experiencia analítica en tiempo real.

En conclusión, la batería de pruebas ejecutada confirma que *BorakReports* ha superado de forma holgada los criterios de aceptación técnicos y funcionales definidos en las fases iniciales del proyecto. La integración del modelo de traducción semántica Text-to-SQL se ha realizado de forma robusta, asegurando la privacidad de la información departamental en todos los escenarios evaluados. Del mismo modo, la respuesta excepcional en las encuestas de validación y la optimización de latencia constatan que la herramienta propuesta constituye un avance operativo tangible, cumpliendo con la premisa de facilitar y democratizar el acceso al dato dentro de la organización corporativa.

CONCLUSIONES

El presente Trabajo Fin de Grado ha culminado con el diseño, desarrollo y validación exitosa de BorakReports, consolidando una capa de Inteligencia de Negocio sobre el sistema transaccional GESBORAK. El objetivo principal de transformar la acumulación masiva de datos operativos en conocimiento estratégico accionable se ha cumplido satisfactoriamente, proporcionando a la organización una herramienta multiplataforma que democratiza el acceso a la información y agiliza la toma de decisiones.

6.1. Aportaciones y Conclusiones Técnicas

A lo largo de las fases de ingeniería y desarrollo, el proyecto ha demostrado la viabilidad de modernizar ecosistemas legados sin necesidad de sustituirlos, destacando las siguientes aportaciones técnicas:

Implementación segura de IA generativa: Integración de LLM en entorno corporativo mediante un motor Text-to-SQL basado en RAG e inyección dinámica de esquemas DDL, garantizando consultas precisas en Oracle 10g y eliminando alucinaciones. La simulación de Row-Level Security asegura el respeto de la privacidad y jerarquía de usuarios.

Optimización de latencia mediante caché semántica: Flujo en cascada (caché exacta, semántica e IA) que reduce la latencia en más de un 96 %, pasando de 7 segundos a menos de 140 ms en concurrencia, mejorando la experiencia en tiempo real.

Arquitectura escalable y robusta: Arquitectura por capas en Spring Boot con patrones de diseño como *Strategy*, garantizando el principio Open/Closed y permitiendo ampliar análisis y dashboards sin modificar el núcleo del sistema.

Rentabilidad y visualización híbrida: Explorador jerárquico que combina cálculos síncronos bajo demanda con procesos batch asíncronos, equilibrando precisión y rendimiento. La codificación por semáforos facilita la interpretación ejecutiva inmediata.

6.2. Validación y Aceptación del Usuario

El impacto de la plataforma se ha validado mediante Pruebas de Aceptación de Usuario sobre una muestra representativa de la plantilla corporativa. Los resultados cuantitativos confirman el éxito del diseño, con puntuaciones superiores a 4/5 en todas las dimensiones evaluadas, destacando la fiabilidad del sistema (4,65) y la precisión de los informes exportados (4,60). El feedback cualitativo confirma además que BorakReports reduce significativamente el esfuerzo cognitivo y el tiempo necesario para la explotación de datos frente al sistema legado.

6.3. Líneas de Trabajo Futuro

La arquitectura de BorakReports ha sido concebida con una alta modularidad, lo que facilita su evolución natural. A partir de la retroalimentación de los usuarios y de las necesidades analíticas detectadas durante la validación, se proponen las siguientes líneas de desarrollo futuro:

Transparencia Técnica para Administradores: Incorporar un módulo de auditoría que permita a los perfiles técnicos visualizar en tiempo real la sentencia SQL generada por el motor de IA, facilitando la depuración y comprensión del razonamiento lógico del modelo subyacente.

Reportes Interactivos Integrados: Evolucionar la actual exportación documental estática (PDF y Excel) hacia la generación de informes HTML dinámicos e interactivos, que permitan aplicar filtros avanzados y profundizar en los datos (*drill-down*) directamente desde el navegador web.

Ampliación del Catálogo Funcional (Autoservicio del Empleado): Integrar nuevas verticales orientadas a la gestión personal, como un espacio cifrado y seguro para la consulta, firma y descarga directa de nóminas mensuales en formato digital, centralizando la gestión corporativa en una única interfaz.

6.4. Reflexión Final

Este proyecto ha evidenciado que el rol de la ingeniería informática trasciende la mera gestión, almacenamiento y visualización de la información. Mediante la orquestación de tecnologías modernas, algoritmos de procesamiento de lenguaje natural y arquitecturas orientadas al dominio, BorakReports ha logrado transformar la rigidez de un sistema legado en un diálogo ágil y seguro entre el usuario y sus datos. El resultado no es únicamente un producto de software plenamente funcional, sino una solución estratégica que aporta valor inmediato, fomentando una cultura empresarial más transparente, eficiente y fundamentada en evidencias empíricas.

BIBLIOGRAFÍA

- [1] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, *et al.*, “Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task,” in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 3911–3921, 2018.
- [2] OpenAI, “Gpt-4 technical report,” tech. rep., OpenAI, 2023.
- [3] N. Reimers and I. Gurevych, “Sentence-bert: Sentence embeddings using siamese bert-networks,” in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, 11 2019.
- [4] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, “Retrieval-augmented generation for knowledge-intensive NLP tasks,” in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual* (H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, eds.), 2020.
- [5] VMware, Inc., *Spring AI Reference Documentation*. Spring Framework, 2024. <https://docs.spring.io/spring-ai/reference/>.
- [6] M. Jones, J. Bradley, and N. Sakimura, “Json web token (jwt),” Tech. Rep. RFC 7519, Internet Engineering Task Force (IETF), 2015.
- [7] Meta Platforms, Inc., *React: The library for web and native user interfaces*, 2024. <https://react.dev/>.
- [8] Drifty Co. / Ionic, *Capacitor: A cross-platform native runtime for web apps*, 2024. <https://capacitorjs.com/>.
- [9] The Apache Software Foundation, *Apache POI - the Java API for Microsoft Documents*, 2024. <https://poi.apache.org/>.
- [10] The Apache Software Foundation, *Apache PDFBox - A Java PDF Library*, 2024. <https://pdfbox.apache.org/>.

APÉNDICES

MODELADO VISUAL DEL SISTEMA

En este anexo se recogen los diagramas fundamentales que modelan el comportamiento y la interacción de los diferentes actores con la plataforma *BorakReports*. Estas representaciones gráficas facilitan la comprensión de la arquitectura funcional, los límites de la aplicación y los flujos de seguridad establecidos.

A.1. Diagrama de Casos de Uso

A continuación, se presenta el diagrama general de Casos de Uso, el cual detalla las operativas disponibles según el nivel de privilegios del usuario (Empleado frente a Administrador).

Cabe destacar que el modelado funcional correspondiente al ecosistema *Text-to-SQL* y al asistente conversacional basado en Inteligencia Artificial no se incluye en este apartado, dado que ha sido expuesto y analizado en profundidad a lo largo del Capítulo 3 (véase Figura 3.1).

Como se puede observar en la Figura A.1, el sistema implementa un modelo de herencia de roles donde el Administrador asume todas las capacidades del Empleado base. Mientras que el perfil estándar se centra en la generación de informes estructurados, el perfil administrativo dispone de acceso integral a los módulos de consultas, gráficos, explorador de rentabilidades y contabilidad.

Asimismo, el diagrama refleja las estrictas dependencias de seguridad mediante relaciones de inclusión («*include*»), evidenciando que toda interacción exige una validación previa contra el Directorio Activo (LDAP) y, en el caso específico del entorno contable, un segundo factor de autenticación interno.

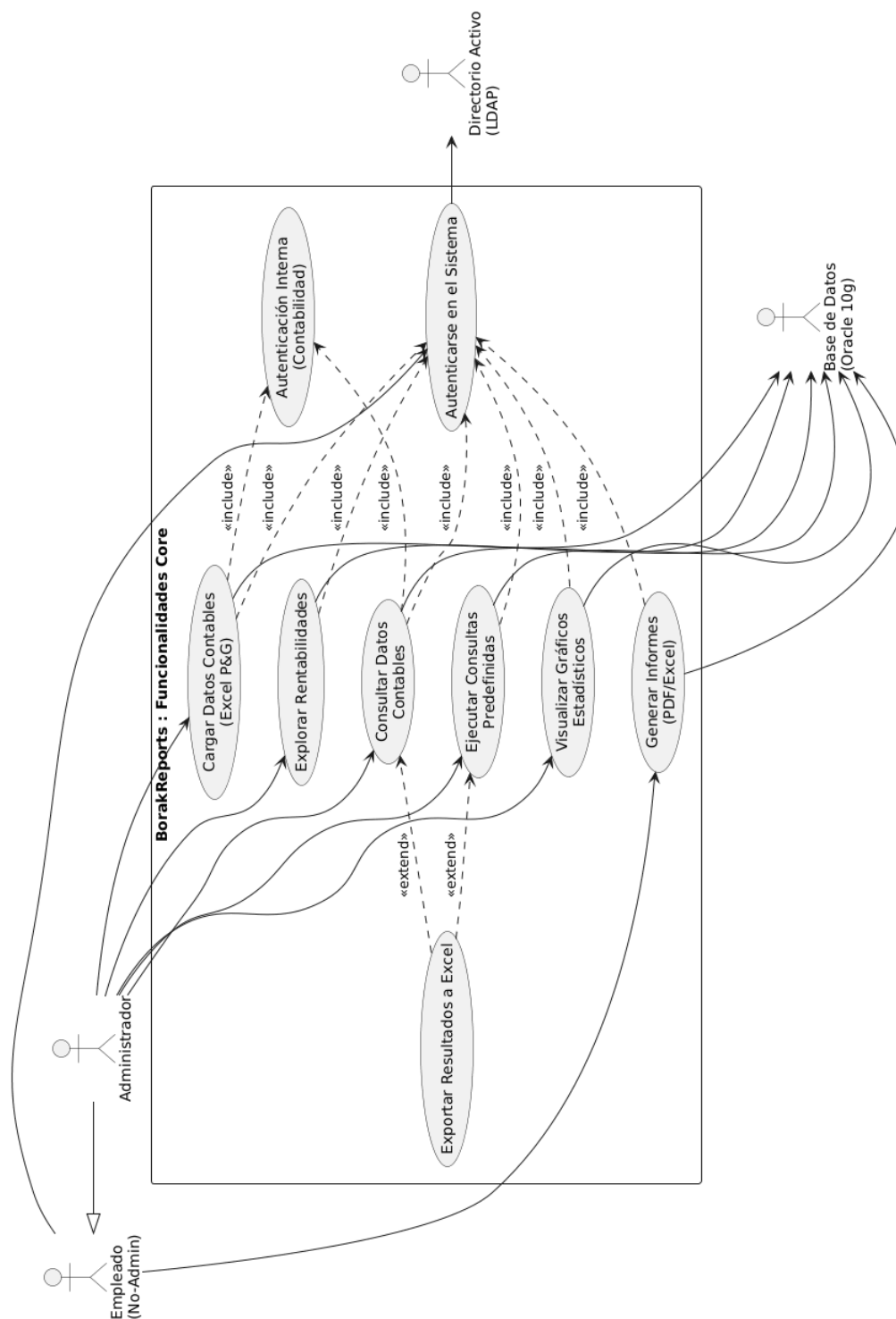


Figura A.1: Interacción de perfiles con los módulos funcionales del sistema

A.2. Diagramas de Secuencia

En esta sección se exponen los diagramas de secuencia del sistema, los cuales permiten modelar el comportamiento dinámico de la plataforma *BorakReports*.

A diferencia de los diagramas de casos de uso (véase la subsección A.1), estas representaciones se centran en la dimensión temporal, ilustrando el intercambio de mensajes y la secuencia cronológica de interacciones entre los actores, las interfaces de usuario, los servicios lógicos del *backend* en Spring Boot y los sistemas de persistencia corporativos durante la ejecución de los flujos críticos.

A.2.1. Autenticación y Gestión de Identidad

El proceso de autenticación es el punto de entrada crítico al sistema. Como se ilustra en la Figura A.2, el flujo integra la validación de credenciales mediante el Directorio Activo (LDAP) con la gestión de sesiones mediante *tokens* JWT.

Este proceso garantiza que, tras validar al usuario corporativo, se recupere no solo el identificador interno del empleado, sino también su nivel de acceso, permitiendo una configuración dinámica de la interfaz y la seguridad basada en roles (RBAC) durante toda la sesión del usuario.

A.2.2. Procesamiento y Exportación de Informes

Una vez autenticado, el usuario dispone de capacidades de explotación de información financiera mediante la generación de informes personalizados. La Figura A.3 detalla el flujo dinámico para la solicitud, procesamiento y descarga de dichos documentos.

El proceso se basa en una arquitectura de capas donde el *Frontend* solicita la generación de datos estructurados al *Backend*, el cual actúa como orquestador delegando la lógica de negocio en servicios especializados. La implementación hace uso del patrón de diseño *Strategy* para desacoplar la lógica de generación de cada tipo de informe (PDF o Excel), garantizando una arquitectura robusta y fácilmente extensible.

Finalmente, el sistema gestiona las cabeceras HTTP *MIME type* y *Content-Disposition*, permitiendo que el navegador del cliente interprete el flujo binario recibido como un archivo descargable, asegurando una experiencia de usuario fluida, transparente y automatizada.

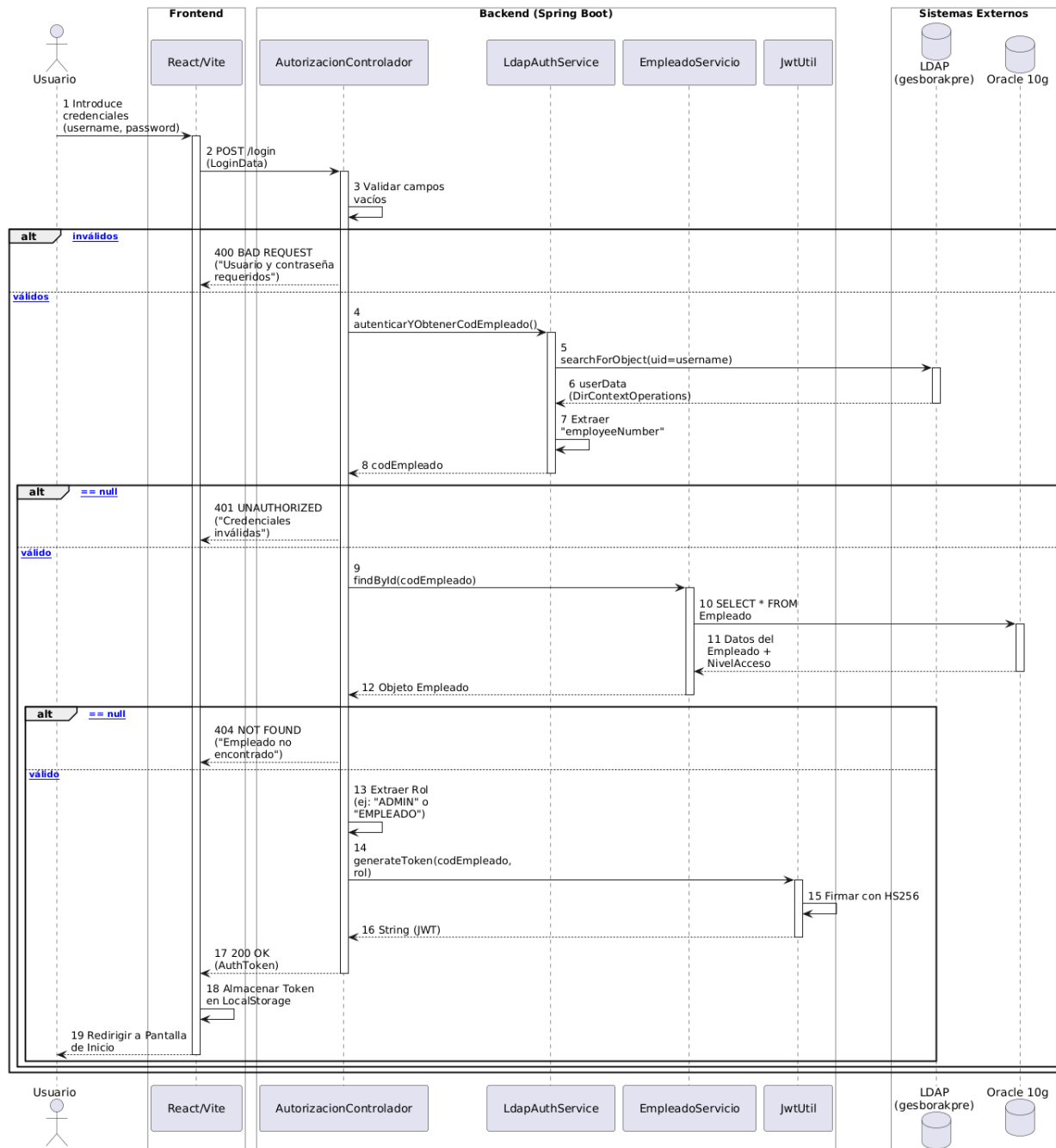


Figura A.2: Flujo cronológico de autenticación e inicio de sesión contra el Directorio Activo (LDAP).

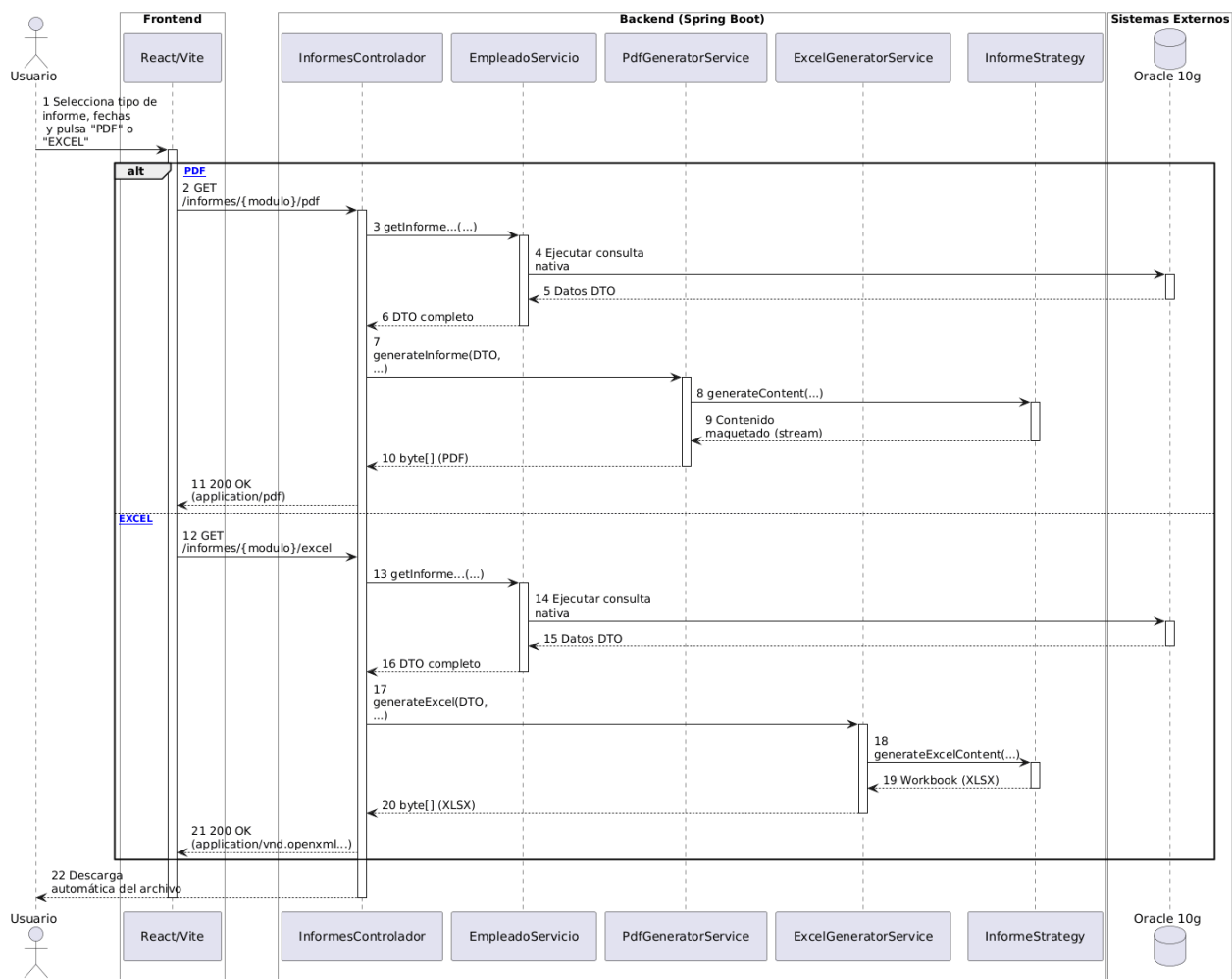


Figura A.3: Flujo dinámico para la solicitud, procesamiento y descarga de informes estructurados en formatos PDF y Microsoft Excel.

ESPECIFICACIÓN TÉCNICA DE LA API REST

B.1. Especificación técnica de la API REST

En esta sección se detalla el contrato de comunicación expuesto por el servidor backend de *BorakReports*. La API sigue los principios RESTful, utilizando el formato JSON para la transferencia de datos y códigos de estado HTTP estándar para las respuestas.

B.1.1. Módulo de Autorización e Identidad (`/auth`)

Este controlador gestiona el ciclo de vida de la autenticación de los usuarios, la validación contra el Directorio Activo (LDAP) y la gestión y recuperación de contraseñas.

POST `/auth/login`

Propósito: Autentica las credenciales del usuario contra el servidor LDAP, verifica su existencia en la base de datos corporativa y emite un *token* JWT firmado que encapsula su rol y código de empleado.

Parámetros (Body): Objeto JSON `LoginData` conteniendo `username` y `password`.

Respuesta Exitosa (200 OK): Objeto JSON `AuthToken` que contiene la cadena del JWT generada.

Códigos de Error: 400 `Bad Request` (campos vacíos), 401 `Unauthorized` (credenciales inválidas en LDAP), 404 `Not Found` (usuario no registrado en BBDD).

POST `/auth/cambiarContrasena`

Propósito: Permite a un usuario autenticado modificar su clave de acceso. El sistema valida la clave actual contra LDAP antes de procesar el cambio.

Parámetros (Body): Objeto JSON `CambioContrasenaRequest` que incluye `username`, `passwordActual`, `passwordNueva` y `passwordNuevaRepetida`.

Respuesta Exitosa (200 OK): Cadena de texto confirmando la actualización exitosa en el directorio activo.

Códigos de Error: 400 Bad Request (las nuevas contraseñas no coinciden), 401 Unauthorized (contraseña actual incorrecta).

GET /auth/userInfo

Propósito: Recupera la información detallada del empleado que realiza la petición. La identidad se extrae automáticamente del contexto de seguridad de Spring a partir del JWT enviado en la cabecera.

Parámetros (Header): Requiere `Authorization: Bearer <token>`. No requiere parámetros en la URL ni en el cuerpo.

Respuesta Exitosa (200 OK): Objeto JSON `Empleado` filtrado mediante la vista `@JsonView(DetalleEmpleado.class)` para omitir campos sensibles no necesarios en el frontend.

POST /auth/forgot-password

Propósito: Inicia el flujo de recuperación de contraseña. Genera un token temporal (con validez de 1 hora) y envía un correo electrónico al usuario con un enlace seguro para su restablecimiento.

Parámetros (Body): Objeto JSON con el campo `email`.

Respuesta Exitosa (200 OK): Mensaje de confirmación del envío de las instrucciones.

Códigos de Error: 404 Not Found (correo no asociado a ningún empleado).

POST /auth/reset-password/{token}

Propósito: Restablece la contraseña de un usuario en el LDAP utilizando el token seguro recibido por correo electrónico.

Parámetros (Path Variable): `token` de seguridad de un solo uso.

Parámetros (Body): Objeto JSON `ResetPasswordRequest` con `username`, `passwordNueva` y `passwordNuevaRepetida`.

Respuesta Exitosa (200 OK): Confirmación de que la contraseña ha sido restablecida en el servidor LDAP.

Códigos de Error: 401 Unauthorized (token de recuperación inválido o caducado), 400 Bad Request (las claves no coinciden).

B.1.2. Módulo del Asistente Conversacional (/bot)

Este controlador gestiona la interacción bidireccional entre el cliente y la Máquina de Estados Finita (FSM) alojada en el servidor, la cual orquesta la navegación guiada del chatbot.

POST /bot/interactuar

Propósito: Procesa la interacción del usuario con el asistente, manteniendo el contexto de la conversación en la memoria del servidor a través del servicio de sesiones. El sistema evalúa la opción elegida y aplica el control de acceso basado en el rol para devolver dinámicamente el siguiente mensaje y las acciones permitidas.

Parámetros (Body): Objeto JSON `BotRequest` que incluye `sessionId` (identificador único de la ventana de chat generada en el cliente), `opcion` (clave de la acción o menú seleccionado, por ejemplo 'INFORMES') y `rol` (perfil de acceso del usuario para el filtrado de seguridad).

Respuesta Exitosa (200 OK): Objeto JSON `BotResponse` que contiene el mensaje de texto renderizable por la interfaz y un `array` de objetos `BotOption` con los botones interactivos disponibles para el estado actual de la navegación.

B.1.3. Módulo de Inteligencia Artificial (/ia)

Este controlador actúa como la interfaz de comunicación entre el cliente y el motor de generación *Text-to-SQL*. Expone las capacidades de consulta en lenguaje natural y la posterior exportación dinámica de los resultados.

GET /ia/pregunta

Propósito: Recibe una consulta en lenguaje natural y la procesa a través del flujo de IA en cascada (caché de texto exacto, búsqueda semántica o petición al LLM). Aplica filtros de seguridad dinámicos inyectando la identidad del usuario para asegurar que la generación de código respete el nivel de acceso (simulando *Row-Level Security*).

Parámetros (Query): `query` (texto introducido por el usuario), `rol` (nivel de acceso autorizado para aplicar restricciones) y `codEmpleado` (identificador único para la trazabilidad y seguridad).

Respuesta Exitosa (200 OK): Objeto JSON estructurado (tipo `Map<String, Object>`) que contiene los metadatos de la respuesta y el `array` de registros obtenidos de la base de datos de Oracle.

Códigos de Error: 200 OK (con payload estructurado tipo: `'error_sql'`) si se intercepta una excepción `BadSqlGrammarException` (evitando caídas del sistema si la IA produce sintaxis inválida), y 500 `Internal Server Error` ante fallos críticos del servicio.

POST /ia/exportar-excel

Propósito: Convierte de forma dinámica y genérica un conjunto de resultados tabulares (previamente obtenidos a través del chat de IA) en un documento estructurado de Microsoft Excel para su descarga inmediata.

Parámetros (Body): Array de objetos JSON (mapeado como `List<Map<String, Object>`) representando las filas y columnas de los datos devueltos por la IA.

Respuesta Exitosa (200 OK): Flujo binario con el *MIME type* `application/octet-stream` y la cabecera `Content-Disposition: attachment; filename=consulta_ia.xlsx`, lo que dispara automáticamente la descarga del archivo en el dispositivo del cliente.

B.1.4. Módulo de Clientes y Proyectos (/clientes)

Este controlador expone las operaciones de solo lectura para recuperar la información del catálogo de clientes de la organización, así como la vinculación estructural con sus proyectos asociados.

GET /clientes/lista

Propósito: Devuelve un listado general de todos los clientes registrados en el sistema, optimizado para su carga en tablas y selectores de la interfaz.

Parámetros: No requiere parámetros.

Respuesta Exitosa (200 OK): Array de objetos JSON de tipo `Cliente`. Los datos se filtran automáticamente mediante la vista `@JsonView(Lista.class)` para retornar únicamente los atributos básicos, reduciendo el consumo de ancho de banda.

GET /clientes/{codCliente}

Propósito: Recupera el perfil completo y detallado de un cliente específico, incluyendo campos extendidos y datos de contacto corporativo.

Parámetros (Path Variable): `codCliente` (Identificador numérico único del cliente en la base de datos).

Respuesta Exitosa (200 OK): Objeto JSON `Cliente` con todos sus atributos operativos, procesado a través de la vista `@JsonView(DetalleCliente.class)`.

GET /clientes/{id}/proyectos

Propósito: Obtiene la lista de proyectos operativos que están actualmente vinculados a un cliente en concreto.

Parámetros (Path Variable): `id` (Identificador numérico del cliente).

Respuesta Exitosa (200 OK): Array de objetos JSON de tipo `Proyecto` filtrados bajo la vista básica (`Lista.class`).

POST /clientes/proyectos

Propósito: Consulta masiva para obtener un recuento de proyectos vinculados a una lista específica de clientes. Se utiliza el método POST para permitir el envío seguro de un volumen alto de identificadores en el cuerpo de la petición (operación de obtención en bloque o *batch fetch*).

Parámetros (Body): Array JSON de identificadores numéricos (`List<Long>`) que representan los códigos de los clientes a consultar.

Respuesta Exitosa (200 OK): Objeto JSON estructurado como un `Map<Long, Integer>`, donde cada clave (*key*) es el código del cliente y el valor (*value*) representa el número total de proyectos asociados a dicho cliente.

B.1.5. Módulo de Consultas Predefinidas (/consultas)

Este controlador actúa como el núcleo de extracción de datos estáticos y analíticos de la plataforma. Expone una serie de *endpoints* fuertemente tipados que reciben parámetros de filtrado (como rangos de fechas o identificadores) y devuelven proyecciones de datos planas (DTOs) optimizadas para su renderizado en tablas y cuadros de mando.

GET /consultas/detalleEmpleado

Propósito: Obtiene un informe detallado de las operaciones y gastos asociados a un empleado específico, acotado por un rango temporal y con soporte para ordenación dinámica (*ORDER BY* inyectado de forma segura).

Parámetros (Query): `nombre`, `apellido1`, `fechaInicio`, `fechaFin` y `ord`.

Respuesta Exitosa (200 OK): Array de objetos `DetalleGastoEmpleadoDTO`.

Códigos de Error: 400 *Bad Request* si se omite alguno de los parámetros obligatorios (el controlador lanza una `IllegalArgumentException` preventiva).

GET /consultas/detalleEmpleados

Propósito: Recupera el detalle de operaciones y gastos a nivel global (todos los empleados activos) dentro de un periodo determinado.

Parámetros (Query): `fechaInicio` y `fechaFin`.

Respuesta Exitosa (200 OK): Array de objetos `DetalleGastoEmpleadoDTO`.

GET /consultas/reporteDiarioCliente

Propósito: Genera un informe diario de la actividad operativa e imputaciones de horas asociadas a un cliente específico en un periodo definido.

Parámetros (Query): `cliente` (cadena de texto para la búsqueda), `fechaInicio` y `fechaFin`.

Respuesta Exitosa (200 OK): Array de objetos `ReporteDiarioDTO`.

GET /consultas/listadoEmpleadosActivos

Propósito: Devuelve una lista maestra de todo el personal que se encuentra actualmente de alta en la organización. Se utiliza para nutrir los selectores y *comboboxes* del frontend.

Parámetros: No requiere parámetros.

Respuesta Exitosa (200 OK): Array de objetos `ListadoEmpleadosActivosDTO`.

GET /consultas/detalleGasto

Propósito: Recupera los metadatos y el desglose de un apunte de gasto concreto basándose en su código identificador.

Parámetros (Query): `codGasto` (Identificador numérico).

Respuesta Exitosa (200 OK): Objeto JSON individualizado de tipo `DetalleGastoPorCodigoDTO`.

GET /consultas/interDptoDetalleConEmpresa

Propósito: Obtiene el desglose detallado (a nivel de empleado y proyecto) de las horas de trabajo cruzadas entre distintos departamentos de la empresa.

Parámetros (Query): `fechaInicio` y `fechaFin`.

Respuesta Exitosa (200 OK): Array de objetos `HorasInterDepartamentosDetalleDTO`.

GET /consultas/interDptoConEmpresa

Propósito: Genera un resumen agregado de la dedicación horaria interdepartamental, consolidando la información de las diferentes divisiones organizativas.

Parámetros (Query): `fechaInicio` y `fechaFin`.

Respuesta Exitosa (200 OK): Array de objetos `HorasInterDepartamentoDTO`.

GET /consultas/eurosInterDptoConEmpresa

Propósito: Similar a la consulta horaria, pero proyectando las métricas en términos económicos (Euros), permitiendo evaluar el impacto financiero y la facturación cruzada entre las unidades de negocio.

Parámetros (Query): `fechaInicio` y `fechaFin`.

Respuesta Exitosa (200 OK): Array de objetos `EurosInterDepartamentoDTO`.

B.1.6. Módulo de Contabilidad e Ingesta de Datos (/contabilidad)

Este controlador gestiona el entorno restringido de Pérdidas y Ganancias (P&G) de la organización. Implementa una capa de seguridad adicional e independiente del `token` JWT, así como los procesos transaccionales para la carga, lectura y análisis de archivos contables.

POST /contabilidad/auth-internal

Propósito: Verifica la segunda capa de seguridad (autenticación interna) requerida para acceder al módulo contable. Compara las credenciales recibidas con la clave maestra cifrada almacenada en el sistema de archivos del servidor (`/var/tmp/conf/contabilidad-pass.txt`).

Parámetros (Body): Objeto JSON estructurado como mapa de claves `'user'` y `'pass'`.

Respuesta Exitosa (200 OK): Objeto JSON `{ 'authorized': true }` concediendo el acceso al entorno.

Códigos de Error: `401 Unauthorized` si las credenciales internas son incorrectas.

POST /contabilidad/cambiar-pass

Propósito: Permite al administrador contable modificar la contraseña de acceso interno. La nueva clave actualiza dinámicamente tanto el valor en la memoria RAM del servidor como el archivo físico, evitando tener que reiniciar la aplicación.

Parámetros (Body): Objeto JSON con las claves `'passActual'` y `'passNueva'`.

Respuesta Exitosa (200 OK): Mensaje de confirmación de actualización correcta.

Códigos de Error: 400 `Bad Request` (contraseña actual errónea o nueva clave inferior a 6 caracteres), 500 `Internal Server Error` (fallo de permisos de escritura en el disco del servidor).

GET /contabilidad/listar-pg

Propósito: Recupera el histórico completo de los balances de Pérdidas y Ganancias previamente procesados y persistidos en la base de datos.

Parámetros: No requiere parámetros.

Respuesta Exitosa (200 OK): Array de objetos JSON de la entidad `ContabilidadPG`.

DELETE /contabilidad/eliminar-pg/{id}

Propósito: Elimina un registro contable específico del repositorio de la base de datos corporativa.

Parámetros (Path Variable): `id` (Identificador numérico del registro contable).

Respuesta Exitosa (200 OK): Mensaje JSON confirmando la eliminación del registro.

Códigos de Error: 500 `Internal Server Error` si falla la restricción de integridad transaccional.

POST /contabilidad/cargar-excel-perdidas-ganancias

Propósito: Endpoint de ingesta de datos. Recibe un documento Excel con el balance financiero, delegando en el servicio (mediante *Apache POI*) el *parsing*, la normalización de formatos numéricos/fechas y la persistencia transaccional.

Parámetros (Form-Data): Archivo binario enviado bajo la clave `file` (tipo `MultipartFile`).

Respuesta Exitosa (200 OK): Mensaje de texto confirmando el procesado exitoso del archivo.

Códigos de Error: 400 `Bad Request` si el documento presenta una estructura inválida o celdas corruptas, provocando el *rollback* de la transacción. 500 `Internal Server Error` en fallos técnicos graves.

POST /contabilidad/analisis/{tipo}

Propósito: Motor de cálculo financiero bajo demanda. Ejecuta procesos analíticos complejos sobre los datos contables (tendencias temporales, peso del personal operativo, desviaciones, etc.) parametrizados por el usuario.

Parámetros (Path Variable): `tipo` (Clave en cadena de texto que define el tipo de análisis a ejecutar).

Parámetros (Body): Objeto JSON conteniendo el mapa de filtros (`params`), como rangos de fechas o divisiones específicas.

Respuesta Exitosa (200 OK): Objeto JSON (`Map<String, Object>`) con los resultados computados y las agregaciones listas para ser visualizadas en el cuadro de mando gráfico (ej. con *Recharts*).

B.1.7. Módulo de Divisiones (`/divisiones`)

Este controlador expone las operaciones de solo lectura relativas a la estructura organizativa de la empresa. Su función principal es servir los datos necesarios para la navegación y el filtrado por departamentos en los cuadros de mando y formularios de la aplicación.

GET `/divisiones/lista`

Propósito: Devuelve un listado completo de todas las divisiones corporativas activas en el sistema. Para garantizar la eficiencia, el controlador no expone la entidad directamente, sino que transforma los resultados mediante el servicio (`convertToDTO`) y aplica un filtrado adicional a través de la vista `@JsonView(Lista.class)`.

Parámetros: No requiere parámetros.

Respuesta Exitosa (200 OK): Array de objetos JSON de tipo `DivisionDTO`, optimizado para reducir la carga de red.

B.1.8. Módulo de Empleados (`/empleados`)

Este controlador expone las operaciones de consulta relativas al personal de la organización, facilitando la obtención de listados generales para poblar las interfaces de usuario.

GET `/empleados/lista`

Propósito: Devuelve un listado completo de todos los empleados registrados en el sistema. Se aplica la vista `@JsonView(Lista.class)` para proyectar únicamente los atributos esenciales, optimizando la transferencia de datos y protegiendo proactivamente información sensible o confidencial (como datos salariales o credenciales).

Parámetros: No requiere parámetros.

Respuesta Exitosa (200 OK): Array de objetos JSON correspondientes a la entidad `Empleado`, filtrados por la vista configurada.

B.1.9. Módulo de Cuadros de Mando y Gráficos (/graficos)

Este controlador actúa como el proveedor de datos estadísticos agregados para la representación visual en el *dashboard* de la aplicación. Devuelve estructuras preprocesadas (`Map<String, Object>`) listas para ser consumidas directamente por la librería de visualización (*Recharts*).

GET /graficos/clientesPorDivision

Propósito: Devuelve las métricas de distribución y peso de los distintos clientes segmentados por cada división organizativa. Permite detectar qué clientes aportan mayor volumen de negocio a cada departamento.

Parámetros (Query): `fechaInicio` (Opcional), `fechaFin` (Opcional) y `estado` (Opcional, por defecto asume el valor ' `TODOS` '). Permite filtrar por la situación actual del proyecto, como 'ACTIVO' o 'FINALIZADO').

Respuesta Exitosa (200 OK): Objeto JSON estructurado con las series de datos y etiquetas necesarias para renderizar los gráficos correspondientes en el frontend.

GET /graficos/proyectosPorDivision

Propósito: Consolida y devuelve el recuento y volumen analítico de los proyectos agrupados por división, ofreciendo una perspectiva macro de la carga de trabajo departamental.

Parámetros (Query): `fechaInicio` (Opcional), `fechaFin` (Opcional) y `estado` (Opcional, valor por defecto ' `TODOS` ').

Respuesta Exitosa (200 OK): Objeto JSON (`Map<String, Object>`) preparado para la visualización de gráficos de barras o tendencias departamentales.

GET /graficos/proyectosPorCliente

Propósito: Genera los datos agregados para visualizar la distribución global de proyectos agrupados por cliente. El *backend* aplica agrupaciones lógicas para evitar la saturación visual (por ejemplo, aglutinando clientes residuales bajo la categoría 'OTROS CLIENTES' en los *Pie Charts*).

Parámetros (Query): `fechaInicio` (Opcional), `fechaFin` (Opcional) y `estado` (Opcional, valor por defecto ' `TODOS` ').

Respuesta Exitosa (200 OK): Objeto JSON (`Map<String, Object>`) con las proporciones y etiquetas para renderizar gráficos sectoriales.

B.1.10. Módulo de Generación de Informes (/informes)

Este controlador orquesta el núcleo documental de la aplicación. Delega la creación de los archivos en los servicios de generación (basados en el patrón *Strategy* detallado en el Capítulo 4) y devuelve flujos binarios directamente descargables por el navegador. Además, implementa el aspecto de seguridad personalizado @VerificarAccesoPropio para garantizar que un empleado estándar no pueda solicitar informes de otros usuarios modificando los parámetros de la URL.

GET /informes/dedicacionEmpleado/{formato}

Propósito: Genera el informe completo de rendimiento y dedicación individual de un empleado acotado por fechas. El parámetro {formato} en la URL permite invocar el *endpoint* terminado en /pdf o /excel.

Parámetros (Query): codEmpleado, fechaInicio, fechaFin, nombreEmpleado, division y tipoInforme (opcional).

Respuesta Exitosa (200 OK): Archivo binario con la cabecera Content-Disposition: attachment. El *MIME type* será application/pdf o application/vnd.openxmlformats-officedocument.spreadsheetml.sheet según el formato solicitado.

Seguridad: Protegido por el aspecto @VerificarAccesoPropio, bloqueando peticiones de usuarios sin privilegios administrativos que intenten consultar el identificador de otro compañero.

GET /informes/reporteHorario/{formato}

Propósito: Exporta el registro horario legal del trabajador, detallando las horas ordinarias, guardias y sumatorios acumulados correspondientes a un periodo.

Parámetros (Query): codEmpleado, fechaInicio, fechaFin, nombreEmpleado y division.

Respuesta Exitosa (200 OK): Archivo PDF o libro de Excel con el reporte horario estructurado.

GET /informes/dedicacionPorDivision/{formato}

Propósito: Exporta el análisis agregado del tiempo imputado a cada proyecto por todos los integrantes de una división concreta.

Parámetros (Query): codDivision, fechaInicio, fechaFin y division (nombre).

Respuesta Exitosa (200 OK): Flujo binario (PDF o Excel) con el informe departamental.

GET /informes/gastosPorDivision/{formato}

Propósito: Genera un informe detallado de la facturación e imputación de gastos directos asociados a los proyectos de una unidad organizativa.

Parámetros (Query): `codDivision`, `fechaInicio`, `fechaFin` y `division` (nombre).

Respuesta Exitosa (200 OK): Flujo binario (PDF o Excel) con el resumen financiero de la división.

GET /informes/uenMes/{formato}

Propósito: Exporta el informe analítico de la Unidad Estratégica de Negocio (UEN), detallando el cruce de facturación y horas interdepartamentales (quién paga y quién cobra por los servicios prestados internamente).

Parámetros (Query): `fechaInicio` y `fechaFin`.

Respuesta Exitosa (200 OK): Documento financiero (PDF o Excel) con un alto nivel de detalle contable.

GET /informes/descargar-horarios-mes

Propósito: Permite al departamento de Recursos Humanos o Administración descargar de forma masiva todos los reportes horarios generados para un mes específico. El servidor accede al volumen físico (`/var/tmp/reportes_horarios/`) y comprime los documentos al vuelo.

Parámetros (Query): `anio` y `mes`.

Respuesta Exitosa (200 OK): Archivo comprimido con *MIME type* `application/zip` conteniendo todos los PDF generados para el periodo.

Códigos de Error: 204 No Content (si la carpeta existe pero no tiene archivos generados) y 404 Not Found (si no existe la ruta en el volumen).

B.1.11. Módulo de Rentabilidades (/rentabilidades)

Este controlador actúa como el motor de cálculo y consulta financiera de la plataforma. Expone las operaciones necesarias para la navegación jerárquica (Cliente → Proyecto → Módulo), la obtención de históricos consolidados y el cálculo en tiempo real de los indicadores semafóricos de eficiencia de la organización.

GET /rentabilidades/resumen

Propósito: Devuelve las métricas consolidadas de alto nivel para poblar el cuadro de mando principal (facturación total, costes agregados y margen de rentabilidad global).

Parámetros: No requiere parámetros.

Respuesta Exitosa (200 OK): Objeto JSON (`Map<String, Object>`) estructurado con los valores financieros totales y promedios.

GET /rentabilidades/explorar

Propósito: Soporta la navegación *drill-down* del explorador jerárquico. Permite descender dinámicamente en el árbol de datos corporativo, solicitando los nodos hijos (ej. los proyectos de un cliente, o los módulos de un proyecto).

Parámetros (Query): `nivel` (Cadena que define la profundidad actual, ej. 'CLIENTE' o 'PROYECTO') e `id` (Identificador numérico opcional del nodo padre para cargar sus dependencias).

Respuesta Exitosa (200 OK): Array de objetos JSON (`List<Map<String, Object>>`) representando las filas del nivel solicitado, optimizadas para su carga asíncrona en tablas expansibles.

GET /rentabilidades/detalle

Propósito: Ejecuta el cálculo síncrono y exacto bajo demanda para obtener el desglose financiero (ingresos, gastos directos, coste de personal y margen) de un nodo en concreto (ya sea un Cliente, un Proyecto o un Módulo).

Parámetros (Query): `nivel` (Tipo de entidad a consultar) e `id` (Código identificador).

Respuesta Exitosa (200 OK): Objeto JSON (`Map<String, Object>`) con las métricas computadas en tiempo real.

GET /rentabilidades/historico

Propósito: Recupera la serie temporal (*Time Series*) de la rentabilidad de un proyecto basándose en los cálculos asíncronos nocturnos (*batch*). Se utiliza para dibujar curvas de tendencia y evolución del margen a lo largo de los meses.

Parámetros (Query): `codProyecto` (Identificador numérico del proyecto corporativo).

Respuesta Exitosa (200 OK): Array de objetos `RentabilidadResumenDTO` ordenados cronológicamente.

GET /rentabilidades/divisiones

Propósito: Calcula la eficiencia operativa y rentabilidad agregada segmentada por divisiones corporativas, aplicando la lógica de negocio para determinar los estados de riesgo (Favorable, Vigilancia o Crítico).

Parámetros: No requiere parámetros.

Respuesta Exitosa (200 OK): Objeto JSON (`Map<String, Object>`) estructurado con los umbrales de colores y métricas listos para renderizar los indicadores de estado (*LinearProgress*) en la interfaz del Administrador.

INTERFACES DEL SISTEMA

A continuación, se presenta un recorrido visual por los principales módulos de la interfaz gráfica de usuario (GUI) de *BorakReports*. Debido a la naturaleza híbrida de la aplicación, se exponen tanto las vistas optimizadas para navegadores de escritorio (Web) como su correspondiente adaptación *responsive* en dispositivos móviles mediante Capacitor.

Cabe destacar que todos los datos y registros mostrados en las siguientes capturas de pantalla son estrictamente ficticios y han sido generados de forma sintética para propósitos de prueba; en ningún caso representan, involucran ni comprometen información real o confidencial de la empresa.

C.1. Inicio de sesión y recuperación de contraseña

C.1.1. Visualización Web

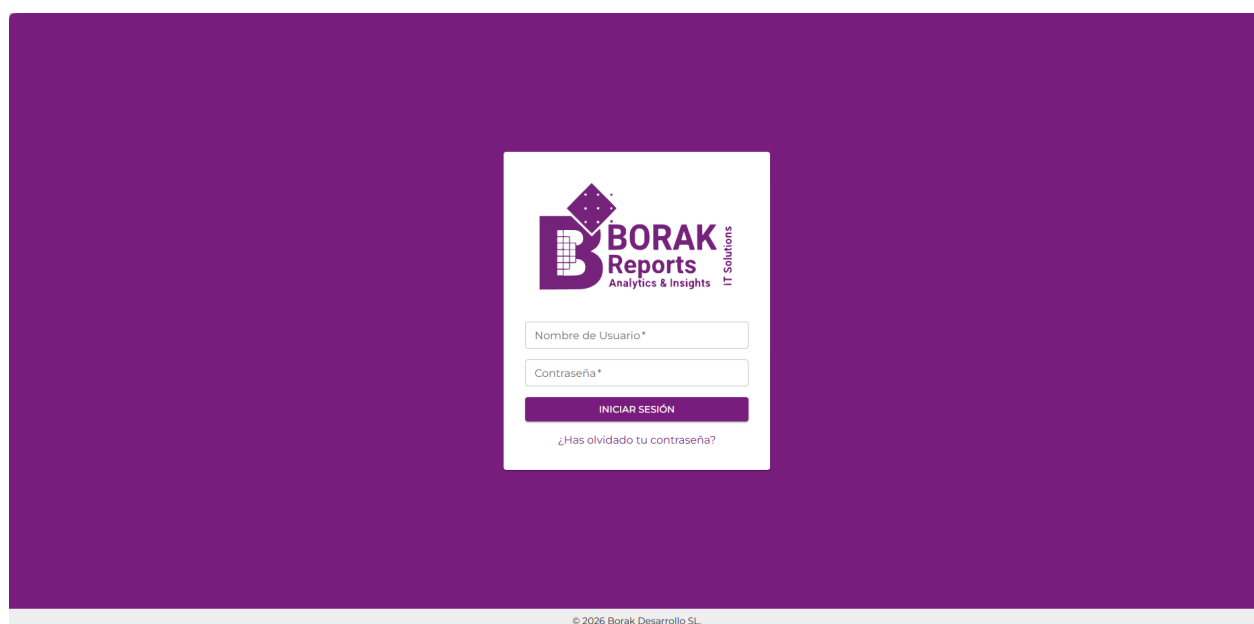


Figura C.1: Pantalla de inicio de sesión.

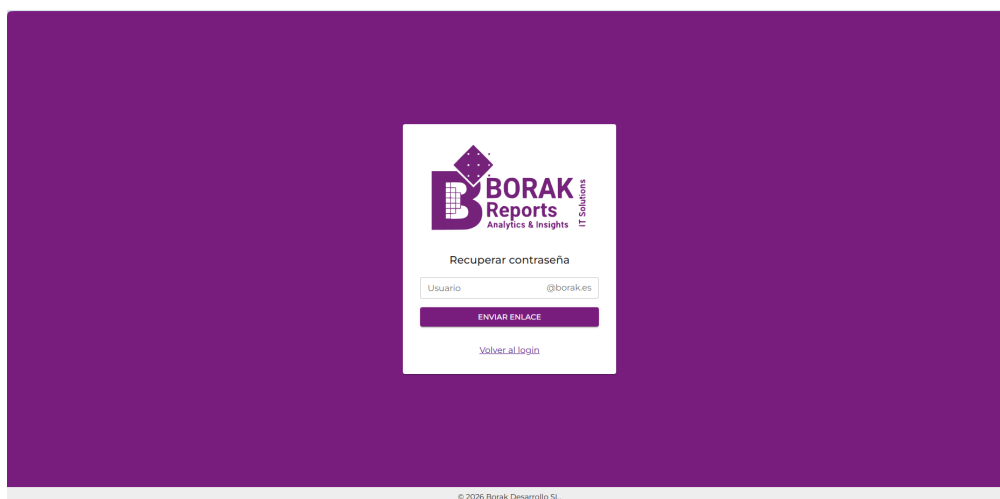


Figura C.2: Formulario de recuperación de contraseña.

C.1.2. Visualización Móvil

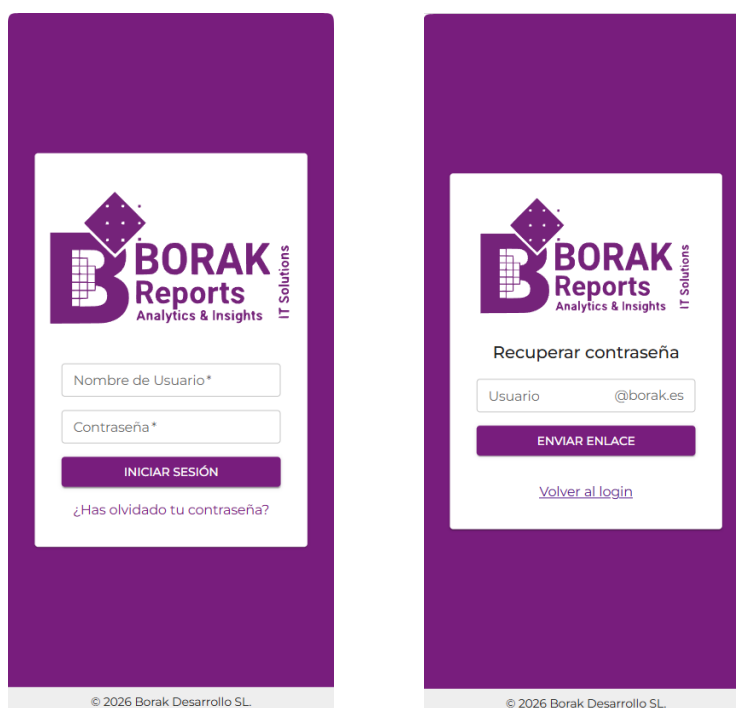


Figura C.3: Pantalla de inicio de sesión (izquierda) y recuperación de contraseña (derecha).

C.2. Pantalla de Inicio (Dashboard Principal)

C.2.1. Visualización Web

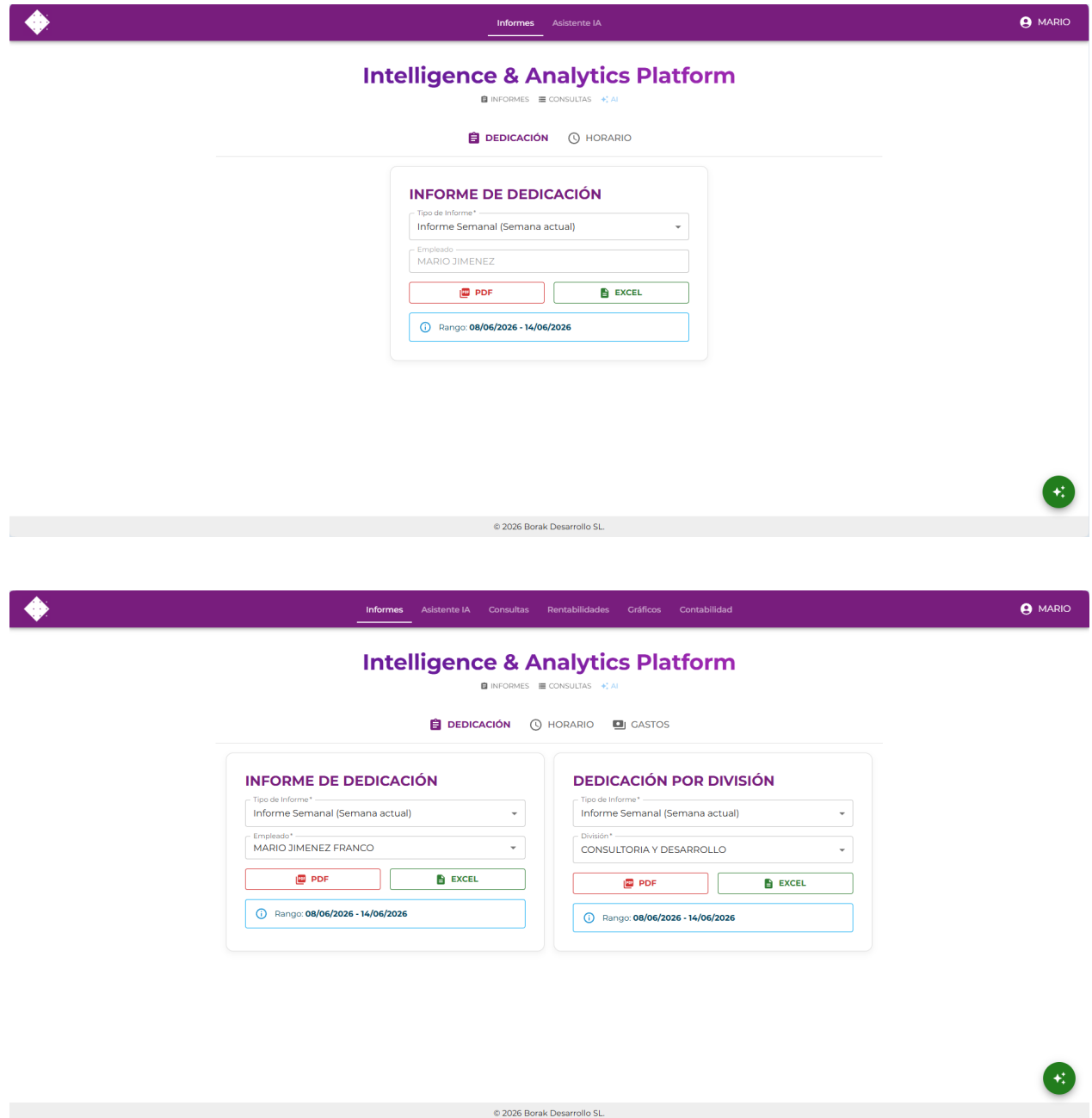


Figura C.4: Pantalla de inicio para el perfil Empleado (arriba) y perfil Administrador (abajo).

C.2.2. Visualización Móvil

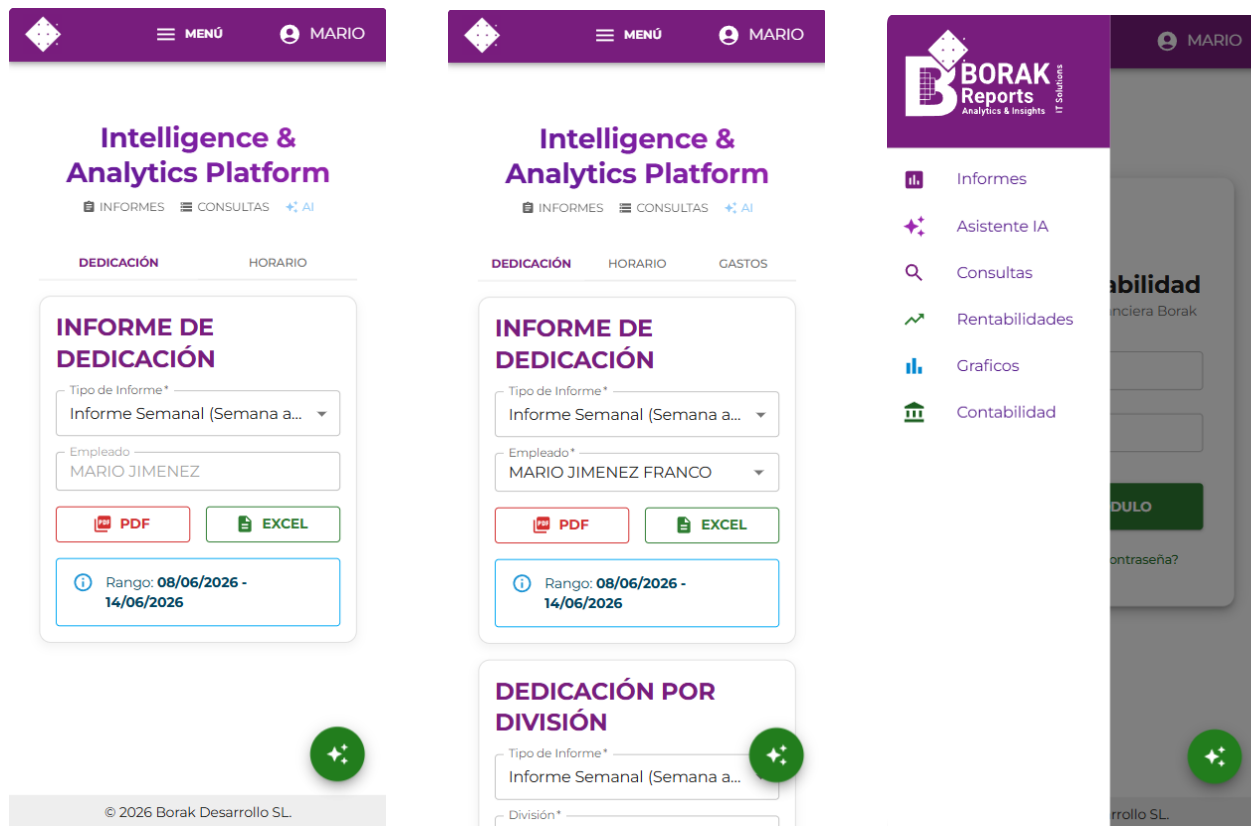


Figura C.5: Pantalla de inicio de Empleado (izquierda), Administrador (centro) y despliegue del menú de navegación (derecha).

C.3. Asistente IA, Consultas y Gráficos

C.3.1. Visualización Web

The screenshot displays the 'BorakReports Asistente IA' web interface. The header includes a navigation bar with 'Informes' and 'Asistente IA' (active), and a user profile 'MARIO'. The main content area features a large text input field with the query '¿Cuántas vacaciones me quedan este año?'. Below the input are several suggestion buttons: '¿Qué puedes hacer?', '¿Cuáles son mis vacaciones este año?', '¿Cuántas vacaciones me quedan este año?', '¿Cuáles fueron mis gastos del último mes?', and '¿Cuáles son los festivos de este año?'. A light blue box contains a disclaimer: 'Recuerda que estas consultas se generan con inteligencia artificial, por lo que es recomendable constatar los datos que se hayan obtenido.' A green button labeled 'DESCARGAR EXCEL' is positioned to the right. Below this is a table with user information:

NOMBRE	APELLIDO1	EMAIL	DIVISION	RESTANTES
MARIO	JIMENEZ	epsuam@borak.es	CONSULTORIA Y DESARROLLO	23

The footer shows '© 2026 Borak Desarrollo SL.' and a green circular icon with a plus sign.

The screenshot displays the 'GENERADOR DE CONSULTAS' web interface. The header includes a navigation bar with 'Informes', 'Asistente IA', 'Consultas' (active), 'Rentabilidades', 'Gráficos', and 'Contabilidad', and a user profile 'MARIO'. The main content area features a dropdown menu labeled 'Selecciona una Consulta' with 'DETALLE GASTO' selected. A button labeled 'EJECUTAR' is to the right. Below this is a light blue box labeled 'Parámetros Requeridos' containing a text input field for 'Código Gasto*'. Below the box, the section 'Resultados de la Consulta:' displays the message: 'No se encontró el gasto con el código proporcionado.' The footer shows '© 2026 Borak Desarrollo SL.' and a green circular icon with a plus sign.

Figura C.6: Interfaz del Asistente de IA (compartida por todos los perfiles) y módulo de Consultas (exclusivos del perfil Administrador).

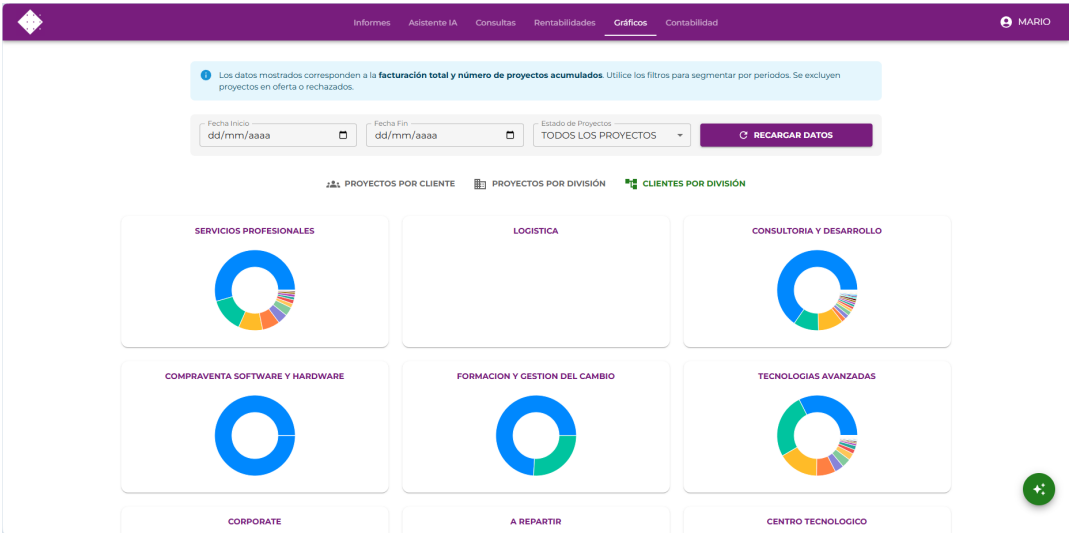


Figura C.7: Módulo de Gráficos (exclusivos del perfil Administrador).

C.3.2. Visualización Móvil

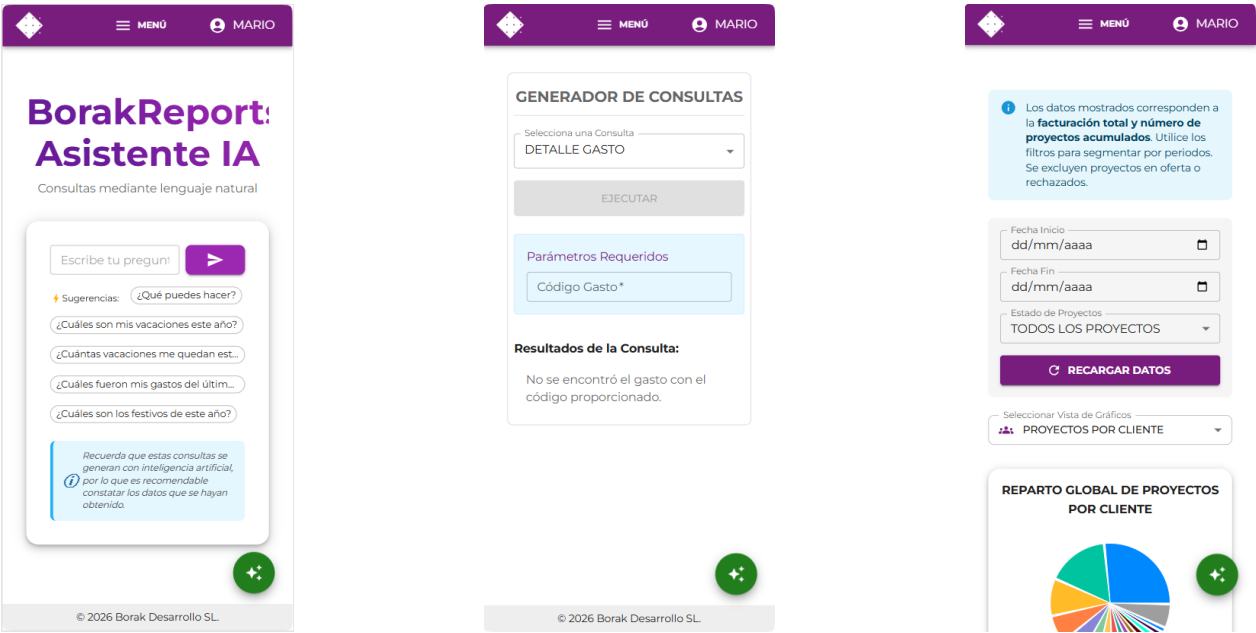


Figura C.8: Adaptación responsive del Asistente IA (izquierda), Consultas tabulares (centro) y representación de Gráficos (derecha).

C.4. Módulo de Contabilidad y ChatBot FSM

C.4.1. Visualización Web

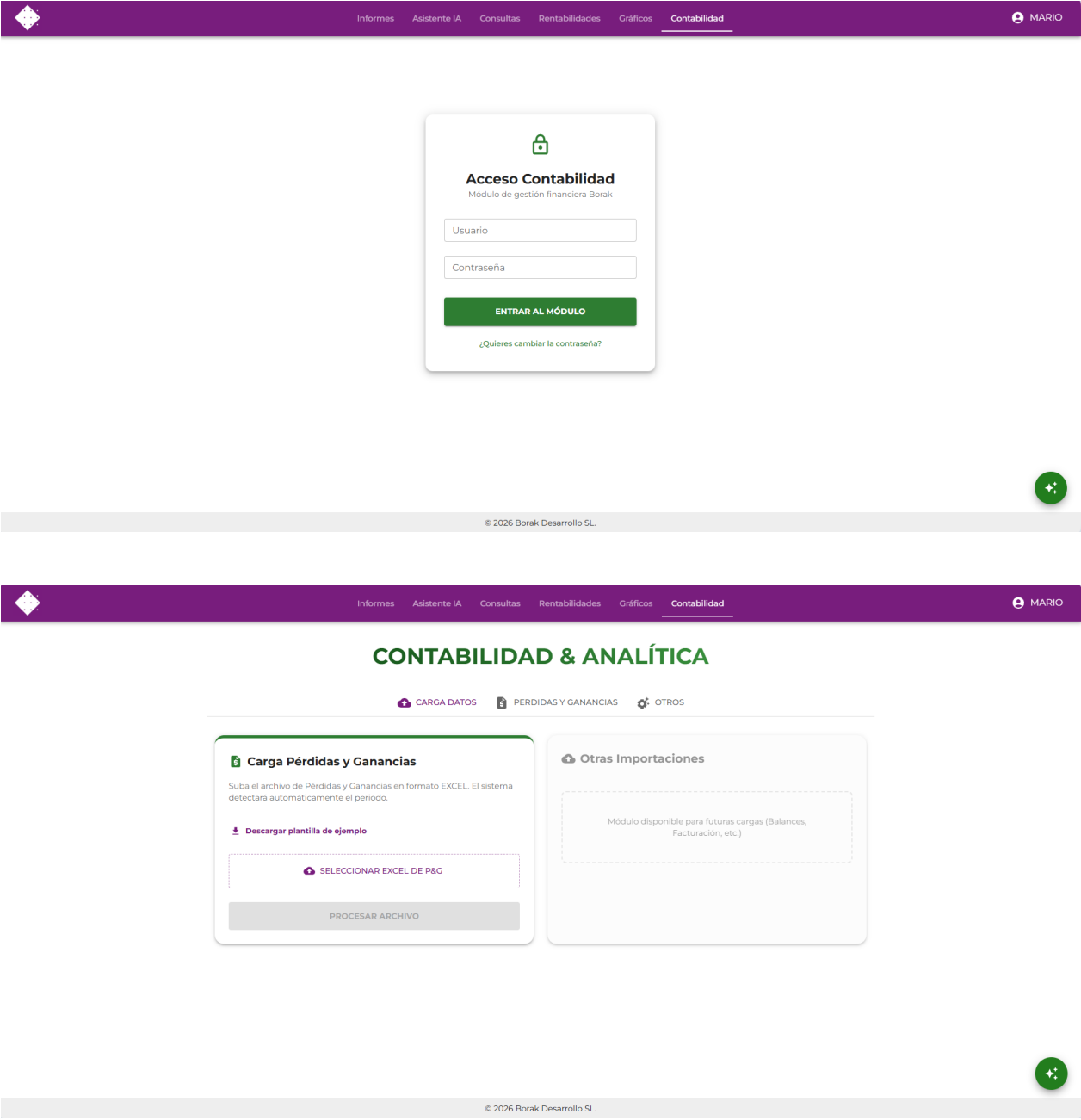


Figura C.9: Doble factor de acceso a Contabilidad, visualización de datos contables.

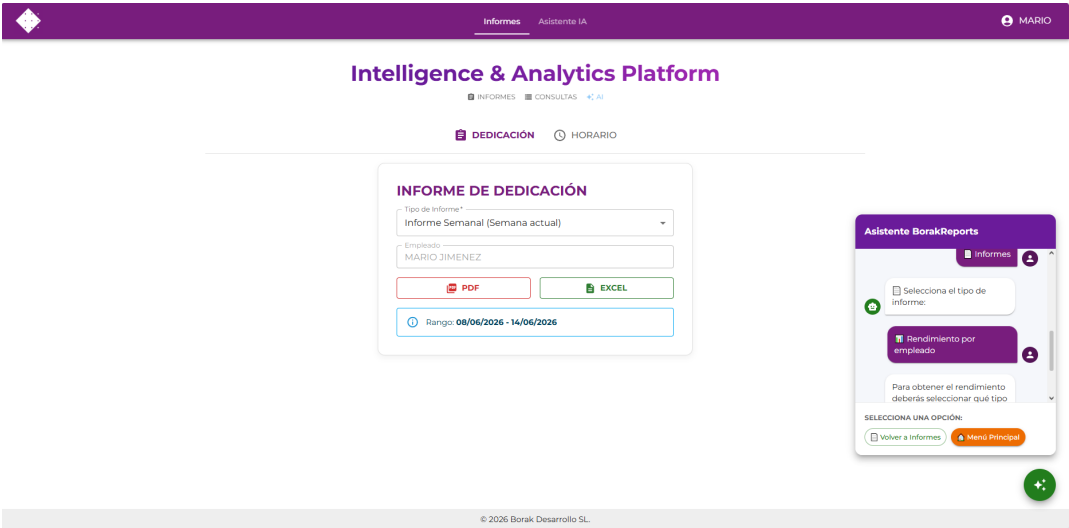


Figura C.10: Flujo del ChatBot.

C.4.2. Visualización Móvil



Figura C.11: Acceso interno a Contabilidad (izquierda), vista responsive de los estados financieros (centro) y navegación del ChatBot (derecha).

C.5. Explorador de Rentabilidades

C.5.1. Visualización Web

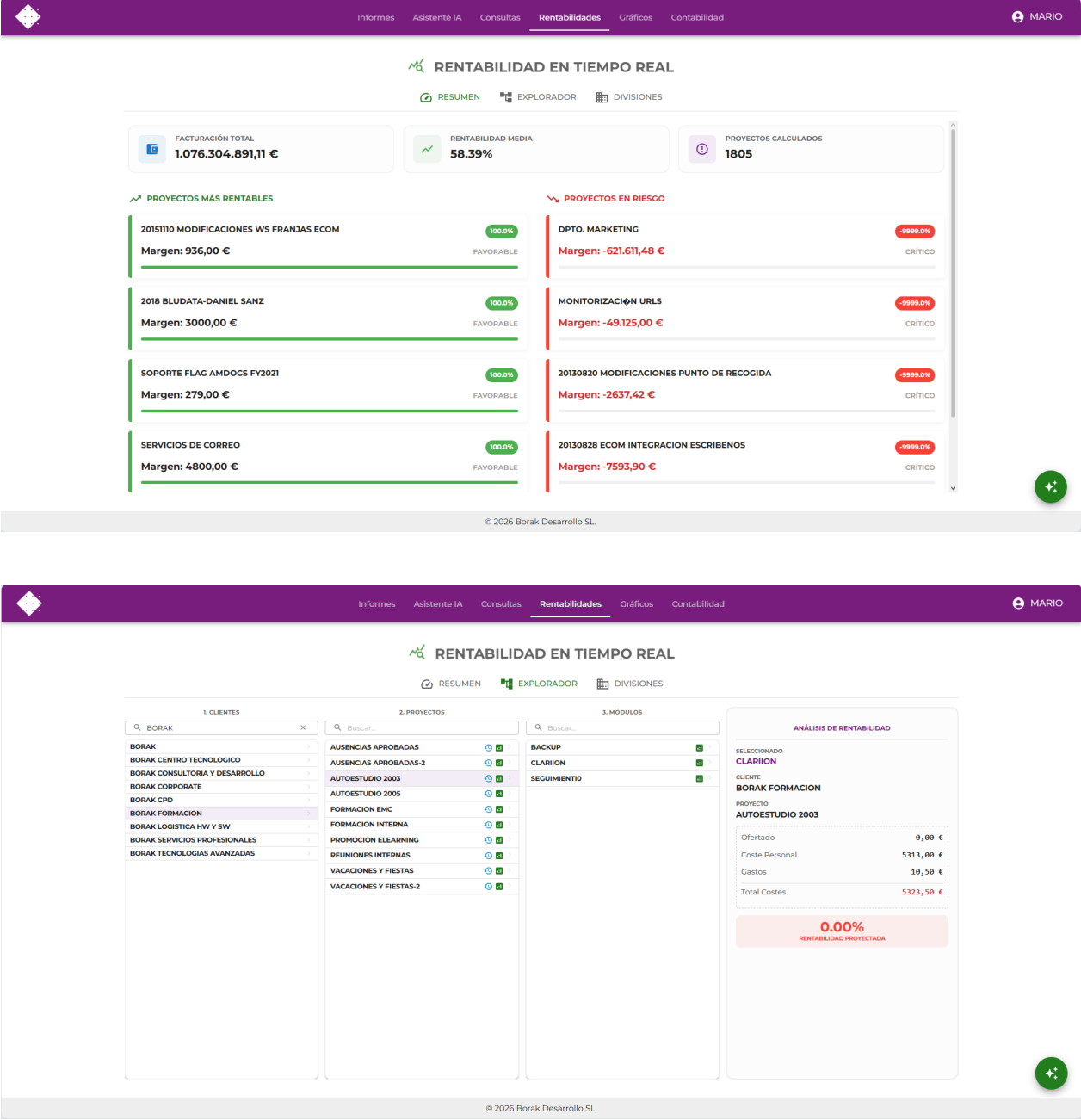


Figura C.12: Resumen global y explorador jerárquico multinivel.

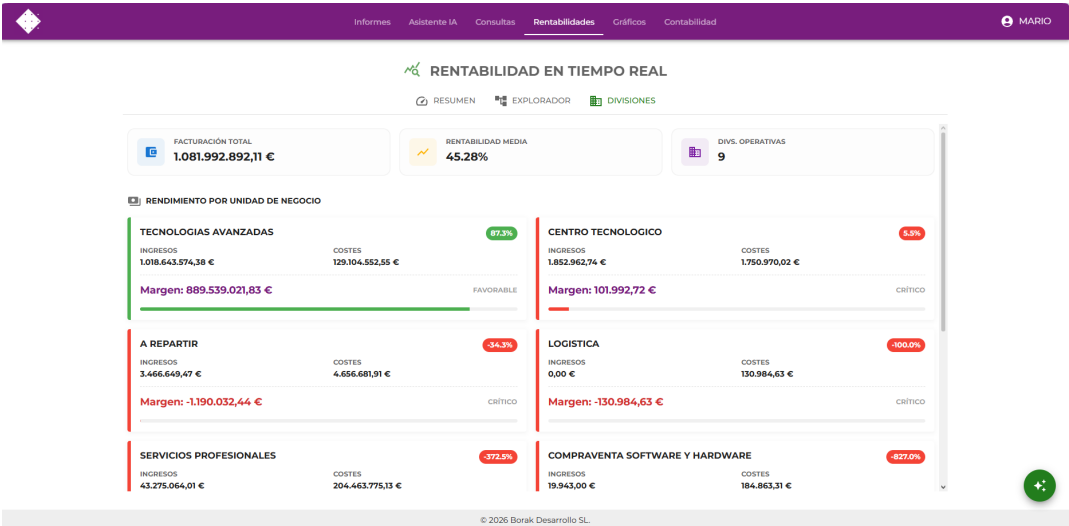


Figura C.13: Análisis comparativo por divisiones.

C.5.2. Visualización Móvil

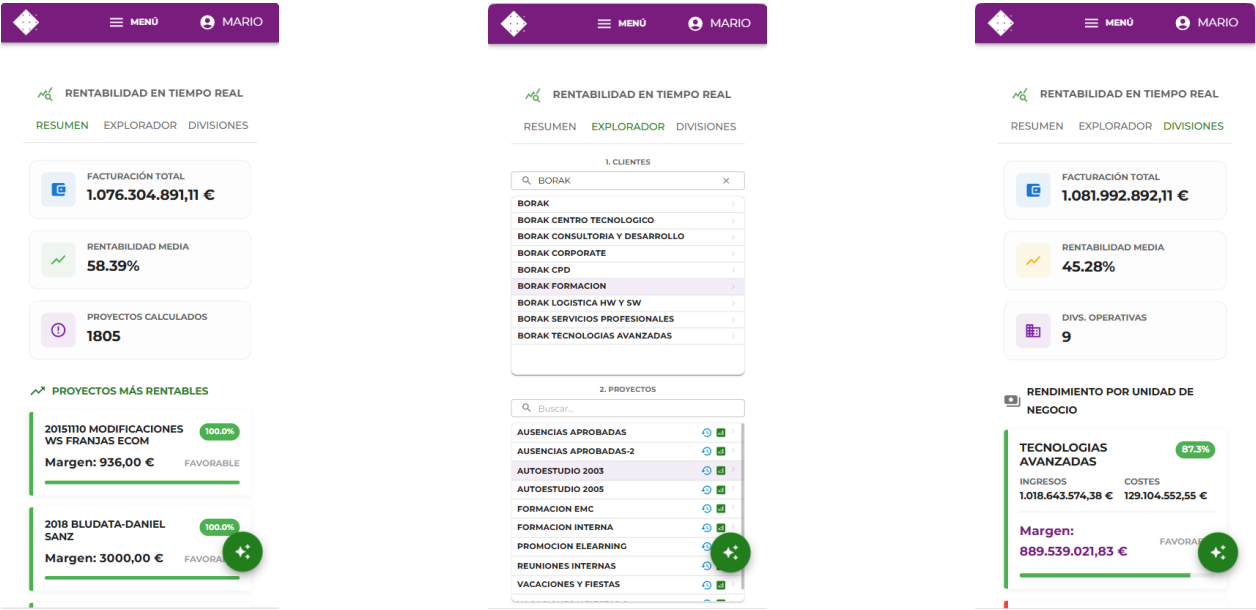


Figura C.14: Resumen de rentabilidad (izquierda), navegación del explorador jerárquico (centro) y eficiencia por divisiones con indicadores cromáticos (derecha).

C.6. Generación y Exportación de Informes

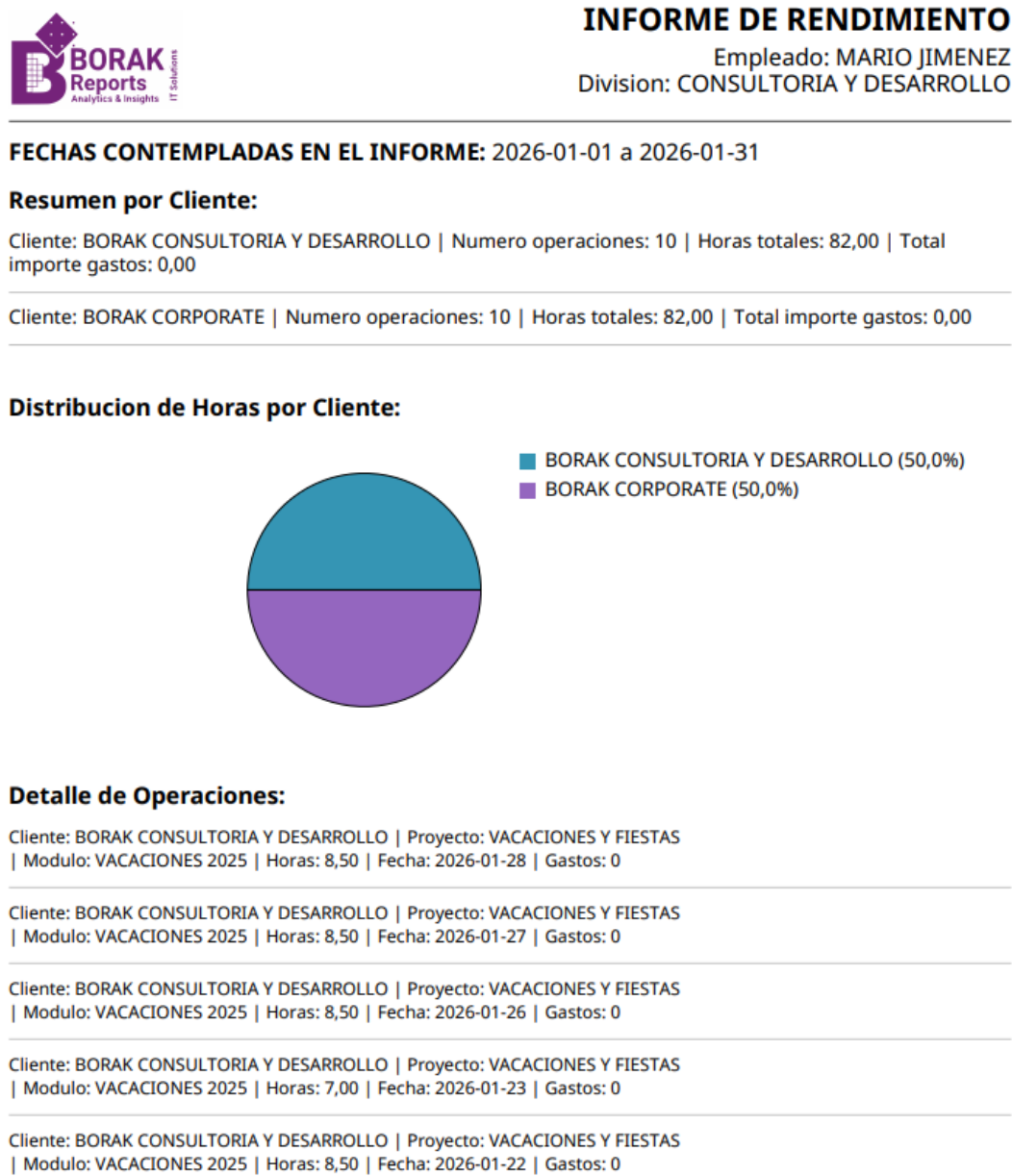


Figura C.15: Visualización de exportación de informes en PDF.

	A	B	C	D	E	F	G	H
1	Fecha	Cliente	Proyecto	Módulo	Hora Gast	Descripción		
2	2026-01-28	BORAK CONS	VACACIONES Y F	VACACIONES 2025	8,50	0	Vacacion aprobada para la fecha	
3	2026-01-27	BORAK CONS	VACACIONES Y F	VACACIONES 2025	8,50	0	Vacacion aprobada para la fecha	
4	2026-01-26	BORAK CONS	VACACIONES Y F	VACACIONES 2025	8,50	0	Vacacion aprobada para la fecha	
5	2026-01-23	BORAK CONS	VACACIONES Y F	VACACIONES 2025	7,00	0	Vacacion aprobada para la fecha	
6	2026-01-22	BORAK CONS	VACACIONES Y F	VACACIONES 2025	8,50	0	Vacacion aprobada para la fecha	
7	2026-01-21	BORAK CONS	VACACIONES Y F	VACACIONES 2025	8,50	0	Vacacion aprobada para la fecha	
8	2026-01-20	BORAK CONS	VACACIONES Y F	VACACIONES 2025	8,50	0	Vacacion aprobada para la fecha	
9	2026-01-19	BORAK CONS	VACACIONES Y F	VACACIONES 2026	8,50	0	Vacacion aprobada para la fecha	
10	2026-01-16	BORAK CONS	VACACIONES Y F	VACACIONES 2026	7,00	0	Vacacion aprobada para la fecha	
11	2026-01-15	BORAK CONS	VACACIONES Y F	VACACIONES 2026	8,50	0	Vacacion aprobada para la fecha	
12	2026-01-14	BORAK CORPC	VACACIONES Y F	VACACIONES 2026	8,50	0	Vacacion aprobada para la fecha	
13	2026-01-13	BORAK CORPC	VACACIONES Y F	VACACIONES 2026	8,50	0	Vacacion aprobada para la fecha	
14	2026-01-12	BORAK CORPC	VACACIONES Y F	VACACIONES 2026	8,50	0	Vacacion aprobada para la fecha	
15	2026-01-09	BORAK CORPC	VACACIONES Y F	VACACIONES 2026	7,00	0	Vacacion aprobada para la fecha	
16	2026-01-08	BORAK CORPC	VACACIONES Y F	VACACIONES 2026	8,50	0	Vacacion aprobada para la fecha	
17	2026-01-07	BORAK CORPC	VACACIONES Y F	VACACIONES 2026	8,50	0	Vacacion aprobada para la fecha	
18	2026-01-06	BORAK CORPC	VACACIONES Y F	VACACIONES 2026	8,50	0	Vacacion aprobada para la fecha	
19	2026-01-05	BORAK CORPC	VACACIONES Y F	VACACIONES 2026	8,50	0	Vacacion aprobada para la fecha	
20	2026-01-02	BORAK CORPC	VACACIONES Y F	VACACIONES 2026	7,00	0	Vacacion aprobada para la fecha	
21	2026-01-01	BORAK CORPC	VACACIONES Y F	VACACIONES 2026	8,50	0	Vacacion aprobada para la fecha	
22								
23								
24	Cliente	Nº Operacion	Horas Totales	Importe Gastos (€)				
25	BORAK COF	10	82,00	0,00				
26	BORAK COF	10	82,00	0,00				
27								
28								

Figura C.16: Visualización de exportación de informes en Excel.



Universidad Autónoma
de Madrid