



**Trabajo de Fin de Grado**

**Grado en Ingeniería del Software**

**Departamento de Lenguajes y Sistemas Informáticos**

**ESTUDIO TECNOLÓGICO Y DESARROLLO DE APLICACIÓN BANCARIA VVBA**

**Realizado por:**

Miguel Ángel Romalde Dorado

**Dirigido por:**

Juan Antonio Ortega Ramírez

Sevilla, junio 2024

Tabla de contenidos

1. INTRODUCCIÓN 9

2. ESTUDIO DEL ARTE 10

2.1 Tecnologías de FrontEnd 10

2.1.1 React 10

2.1.2 Angular 10

2.1.3 Vue.js 10

2.1.4 Svelte 11

2.1.5 Ember.js 11

2.1.6 Backbone.js 11

2.1.7 Preact 11

2.1.8 Tabla comparativa sobre estudios de FrontEnd 12

2.2 Tecnologías del BackEnd 13

2.2.1 Node.js 13

2.2.2 Django 13

2.2.3 Ruby on Rails 13

2.2.4 Spring boot 13

2.2.5 Express.js 14

2.2.6 Flask 14

2.2.7 ASP.NET Core 14

2.2.8 Tabla comparativa sobre tecnologías de BackEnd 14

2.3 Selección de tecnologías 16

2.3.1 FrontEnd 16

2.3.2 BackEnd 17

2.4 Estudio sobre las aplicaciones bancarias 18

2.4.1 Banco La Caixa 18

2.4.2 Banco Santander 20

2.4.3 Banco BBVA 21

2.4.4 Banco Bankinter 22

2.4.5 Banco Sabadell 23

2.4.6 Tabla comparativa de las aplicaciones bancarias. 25

3. ALCANCE DEL PROYECTO 25

3.1 Elicitación de requisitos del producto 26

3.1.1 Requisitos de información 26

3.1.2 Requisitos funcionales 26

3.1.3 Requisitos no funcionales 27

3.1.3.1 Requisitos de calidad 27

3.1.3.2 Requisitos de implementación 27

3.1.3.3 Requisitos de seguridad 27

3.2 Riesgos 28

3.3 Viabilidad 29

3.4 Objetivos de la aplicación 33

4. VERSIÓN DE COSTES 33

4.1 Coste salarial del desarrollo 33

4.2 Costes derivados del desarrollo 34

4.3 Costes de licencias 34

4.4 Coste del material de trabajo 35

4.5 Desglose total de los costes de desarrollo 36

4.6 Reservas de contingencia 36

5. Evolución de la aplicación 36

5.1 Fase de diseño del producto 36

5.1.1 Mockups y prototipo 38

5.2 Fase de implementación 45

5.2.1 Herramientas y servicios. 45

5.2.2 Desarrollo en ASP .NET Core. 45

5.2.2.1 Arquitectura de ASP .NET Core 46

5.2.2.2 Layers. 47

5.2.2.3 Ciclo de vida 49

5.2.2.4 Estructuración del proyecto. 49

5.2.2.5 Pruebas 51

5.2.3 Desarrollo en Angular 51

5.2.3.1 Arquitectura de Angular 52

5.2.3.2 Layers en Angular 54

5.2.3.3 Ciclo de vida 55

5.2.3.4 Estructuración del proyecto 56

5.2.3.5 Pruebas 57

6. Resultados experimentales. 57

7. Conclusiones y trabajo futuro 64

8. Bibliografía 65

Lista de tablas

[Tabla 1. Tabla comparativa FrontEnd 12](#_Toc178074947)

[Tabla 2. Tabla comparativa BackEnd 15](#_Toc178074948)

[Tabla 3. Tabla comparativa bancos 25](#_Toc178074949)

[Tabla 4. REQUISITOS DE INFORMACIÓN 26](#_Toc178074950)

[Tabla 5. Requisitos funcionales 27](#_Toc178074951)

[Tabla 6. Requisitos de calidad 27](#_Toc178074952)

[Tabla 7. Requisitos de implementación 27](#_Toc178074953)

[Tabla 8. Tabla de riesgos 29](#_Toc178074954)

[Tabla 9. Coste de desarrollo del equipo 33](#_Toc178074955)

[Tabla 10. Coste total salarial del desarrollo del proyecto 34](#_Toc178074956)

[Tabla 11. Costes derivados en el desarrollo 34](#_Toc178074957)

[Tabla 12. Costes derivados en el desarrollo totales 34](#_Toc178074958)

[Tabla 13. Coste de licencias mensual 35](#_Toc178074959)

[Tabla 14. Costes de licencias totales 35](#_Toc178074960)

[Tabla 15. Precio material de trabajo 35](#_Toc178074961)

[Tabla 16. Amortización de los equipos 35](#_Toc178074962)

[Tabla 17. Amortización durante el desarrollo 35](#_Toc178074963)

[Tabla 18. Costes totales del desarrollo 36](#_Toc178074964)

[Tabla 19. Reservas de contingencia 36](#_Toc178074965)

Lista de ilustraciones

[Ilustración 1. Principales bancos en españa 9](#_Toc178074966)

[Ilustración 2. Logotipo Angular 16](#_Toc178074967)

[Ilustración 3. Logotipo ASP .NET Core 17](#_Toc178074968)

[Ilustración 4. Logotipo Caixa 18](file:///C:\Users\MRoma\Desktop\BancoVVBA%20new\TFG\Documentación\Memoria.docx#_Toc178074969)

[Ilustración 5. Logotipo Santander 20](file:///C:\Users\MRoma\Desktop\BancoVVBA%20new\TFG\Documentación\Memoria.docx#_Toc178074970)

[Ilustración 6. Logotipo BBVA 21](file:///C:\Users\MRoma\Desktop\BancoVVBA%20new\TFG\Documentación\Memoria.docx#_Toc178074971)

[Ilustración 7. Logotipo Bankinter 22](file:///C:\Users\MRoma\Desktop\BancoVVBA%20new\TFG\Documentación\Memoria.docx#_Toc178074972)

[Ilustración 8. Logotipo Sabadell 23](file:///C:\Users\MRoma\Desktop\BancoVVBA%20new\TFG\Documentación\Memoria.docx#_Toc178074973)

[Ilustración 9. Matriz de probabilidad e impacto 28](#_Toc178074974)

[Ilustración 10. Encuesta pregunta 1 30](#_Toc178074975)

[Ilustración 11. Encuesta Pregunta 2 30](#_Toc178074976)

[Ilustración 12. Encuesta pregunta 3 31](#_Toc178074977)

[Ilustración 13. Encuesta pregunta 4 31](#_Toc178074978)

[Ilustración 14. Encuesta pregunta 5 32](#_Toc178074979)

[Ilustración 15. Encuesta pregunta 6 32](#_Toc178074980)

[Ilustración 16. Encuesta pregunta 7 33](#_Toc178074981)

[Ilustración 17. Login 38](#_Toc178074982)

[Ilustración 18. Registro 39](#_Toc178074983)

[Ilustración 19. Detalles cuenta usuario. 39](#_Toc178074984)

[Ilustración 20. Ingresar o retirar dinero 40](#_Toc178074985)

[Ilustración 21. Transferencias 40](#_Toc178074986)

[Ilustración 22. lista de usuarios 41](#_Toc178074987)

[Ilustración 23. creacion usuario 42](#_Toc178074988)

[Ilustración 24. Creacion operacion 42](#_Toc178074989)

[Ilustración 25. Creacion comisión 43](#_Toc178074990)

[Ilustración 26. Creacion comision en cuenta 44](#_Toc178074991)

[Ilustración 27. Cuenta administrador 44](#_Toc178074992)

[Ilustración 28. Arquitectura ASP .NET Core 47](#_Toc178074993)

[Ilustración 29. Layers ASP .NET Core 48](#_Toc178074994)

[Ilustración 30. Ciclo de vida ASP .NET Core 49](#_Toc178074995)

[Ilustración 31. Estructura proyecto ASP .NET Core 50](#_Toc178074996)

[Ilustración 32. Ejemplo Creación proyecto Angular 52](#_Toc178074997)

[Ilustración 33. Capa Dominio Angular 53](#_Toc178074998)

[Ilustración 34. Layers Angular 54](#_Toc178074999)

[Ilustración 35. Ciclo de vida Angular 55](#_Toc178075000)

[Ilustración 36. Estructura proyecto Angular 56](#_Toc178075001)

[Ilustración 37.Login final 58](#_Toc178075002)

[Ilustración 38. Registro final 58](#_Toc178075003)

[Ilustración 39. Usuarios final 59](#_Toc178075004)

[Ilustración 40. Creacion usuario 59](#_Toc178075005)

[Ilustración 41. Cuentas final 60](#_Toc178075006)

[Ilustración 42. Editar cuenta 60](#_Toc178075007)

[Ilustración 43. Operaciones final 60](#_Toc178075008)

[Ilustración 44. Editar operaciones final 61](#_Toc178075009)

[Ilustración 45. Comisiones final 61](#_Toc178075010)

[Ilustración 46. Editar comisiones 61](#_Toc178075011)

[Ilustración 47. Cuentas comisiones final 62](#_Toc178075012)

[Ilustración 48. Editar cuentas comisiones 62](#_Toc178075013)

[Ilustración 49. Mi cuenta final 62](#_Toc178075014)

[Ilustración 50. Ingresar retirar dinero final 63](#_Toc178075015)

[Ilustración 51. Transferencias final 63](#_Toc178075016)

Resumen

Este trabajo pretende analizar y explorar alternativas en el mercado correspondiente al ámbito tecnológico con una aplicación de las mismas en una solución especifica, enfocada a la ayuda y facilitación a la población a usar soluciones tecnológicas desde la comodidad de su hogar.

Por otra parte, para completar el caso de estudio y objetivo del trabajo en su forma práctica, se busca desarrollar una aplicación para la plataforma de escritorio Windows, que, mediante el uso de las tecnologías estudiadas, se podrá desarrollar una aplicación con los casos de uso más usados en las aplicaciones bancarias, y sobre todo, es una aplicación de prueba que sirve para que la población que no está acostumbrada a usar aplicaciones bancarias o tienen miedo del uso de la misma, puedan probarla con total libertad y sin ningún riesgo.

Abstract

This work aims to analyse and explore alternatives in the technology market with an application focused on helping and facilitating the population to use banking applications from the comfort of their homes.

On the other hand, to complete the case study and objective of the work in its practical form, the aim is to develop an application for the Windows desktop platform, which, through the use of the technologies studied, it will be possible to develop an application with the most used use cases in banking applications, and above all, it is a test application that serves so that the population that is not used to using banking applications or is afraid of using them, can try it with total freedom and without any risk.

Agradecimientos

En primer lugar, me gustaría agradecer a mi tutor, Juan Antonio Ortega Ramírez, por haber tutorizado este proyecto y por la enorme paciencia y flexibilidad que ha tenido conmigo. Me ha sabido guiar y darme respuesta a las necesidades que surgían a lo largo del desarrollo de este.

También, me gustaría agradecer a todos aquellos que, voluntariamente, participaron en la encuesta inicial del proyecto y en algunas de las pruebas.

1. INTRODUCCIÓN

Este trabajo pretende la realización de un análisis de las tecnologías más usadas por las empresas, en conjunto con aplicación de estas tecnologías analizadas dichas anteriormente.

Por otra parte, para completar el caso de estudio y objetivo del trabajo en su forma práctica, se busca desarrollar una aplicación para la plataforma de escritorio Windows, de forma colateral, con la aplicación se pretende fomentar la independencia de los usuarios a la hora de trabajar con una aplicación de banco y reducir la incertidumbre o el miedo a la utilización de aplicaciones del mismo estilo.

Cabe destacar que se usarán tecnologías actuales, las cuales ayuden al propietario de la aplicación a su entrada a diferentes empresas. Para ello, previamente se ha realizado un estudio detallado sobre la descripción, características, ventajas y desventajas de cada una de las tecnologías existentes de FrontEnd y BackEnd, con el fin de obtener una correcta selección tecnológica especifica y adecuada para el desarrollo de la aplicación.

Por último, previo a la puesta en marcha de la aplicación bancaria (VVBA), y junto al estudio de las tecnologías anteriormente mencionadas, también se ha realizado una investigación mínima sobre las diferentes aplicaciones bancarias de los cinco grandes bancos existentes a nivel nacional como son Santander, BBVA, CaixaBank, Bankinter y Sabadell y cuyas aplicaciones están puesta a disposición de la sociedad.



Ilustración 1. Principales bancos en españa

1. ESTUDIO DEL ARTE

En este apartado vamos a hacer un estudio del arte con las 7 tecnologías más usadas actualmente en las empresas.

* 1. Tecnologías de FrontEnd
     1. React
* **Descripción:** Biblioteca de JavaScript para construir interfaces de usuario.
* **Características:**
* Componentes reutilizables.
* Virtual DOM para mejorar el rendimiento.
* Amplio ecosistema de herramientas y bibliotecas.
* **Ventajas:** Flexibilidad, gran comunidad, buen rendimiento.
* **Desventajas:** Puede ser complejo de aprender para principiantes debido a su ecosistema.
  + 1. Angular
* **Descripción:** Framework de aplicaciones web basado en TypeScript.
* **Características:**
* Estructura MVC.
* Inyección de dependencias.
* Herramientas integradas como Angular CLI.
* **Ventajas:** Estructurado, soporte para aplicaciones grandes, dos vías de enlace de datos.
* **Desventajas:** Curva de aprendizaje empinada, complejo.
  + 1. Vue.js
* **Descripción:** Framework progresivo de JavaScript para construir interfaces de usuario.
* **Características:**
* Integración gradual.
* Reactividad.
* Componentes.
* **Ventajas:** Fácil de aprender, flexible, buena documentación.
* **Desventajas:** Menos recursos y herramientas en comparación con React y Angular.
  + 1. Svelte
* **Descripción:** Framework para construir interfaces de usuario que compila eficientemente.
* **Características:**
* Compilación en tiempo de desarrollo.
* Sin virtual DOM.
* Código más simple.
* **Ventajas:** Rendimiento muy alto, menos código.
* **Desventajas:** Comunidad más pequeña, no está tan desarrollada como otras opciones.
  + 1. Ember.js
* **Descripción:** Framework de JavaScript para construir aplicaciones web ambiciosas, similar a Vue.js pero con más dificultad y complejidad.
* **Características:**
* Convención sobre configuración.
* Herramientas de desarrollo integradas.
* Enrutamiento robusto.
* **Ventajas:** Productividad alta, estructura sólida.
* **Desventajas:** Curva de aprendizaje empinada, menos flexible.
  + 1. Backbone.js
* **Descripción:** Biblioteca de JavaScript para estructurar aplicaciones web con modelos y vistas.
* **Características:**
* Ligero.
* MVP (Model-View-Presenter).
* Sincronización con servidor RESTful.
* **Ventajas:** Ligero, flexible.
* **Desventajas:** Requiere más configuración, menos soporte oficial.
  + 1. Preact
* **Descripción:** Alternativa ligera a React.
* **Características:**
* API similar a React.
* Tamaño muy pequeño y eficiente.
* Virtual DOM.
* **Ventajas:** Muy ligero y un gran rendimiento.
* **Desventajas:** Menos características integradas, menos comunidad y menos información.
  + 1. Tabla comparativa sobre estudios de FrontEnd

A continuación, vamos a ver una tabla comparativa de las características de las tecnologías de FrontEnd.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **FRONTEND** | **React** | **Angular** | **Vue.js** | **Svelte** | **Ember.js** | **Backbone.js** | **Preact** |
| Componentes reutilizables |  |  |  |  |  |  |  |
| Amplio ecosistema de herramientas y bibliotecas |  |  |  |  |  |  |  |
| Estructura MVC |  |  |  |  |  |  |  |
| Inyección de dependencias |  |  |  |  |  |  |  |
| Herramientas integradas |  |  |  |  |  |  |  |
| Enrutamiento robusto |  |  |  |  |  |  |  |
| Productividad alta |  |  |  |  |  |  |  |
| Flexibilidad |  |  |  |  |  |  |  |
| Ligero |  |  |  |  |  |  |  |
| Soporte para aplicaciones grandes |  |  |  |  |  |  |  |
| Dos vías de enlace de datos |  |  |  |  |  |  |  |
| Comunidad |  |  |  |  |  |  |  |
| Buen rendimiento |  |  |  |  |  |  |  |
| Curva de aprendizaje empinada |  |  |  |  |  |  |  |

Tabla 1. Tabla comparativa FrontEnd

* 1. Tecnologías del BackEnd
     1. Node.js
* **Descripción:** Entorno de ejecución para JavaScript del lado del servidor.
* **Características:**
* Event-driven.
* Non-blocking I/O.
* Gran ecosistema (NPM).
* **Ventajas:** Trabajo de forma asíncrona, gran comunidad.
* **Desventajas:** No adecuado para tareas intensivas en CPU.
  + 1. Django
* **Descripción:** Framework web de alto nivel en Python.
* **Características:**
* Estructura MVC.
* ORM incorporado.
* Herramientas de administración.
* **Ventajas:** Seguridad, desarrollo rápido y API REST integrado.
* **Desventajas:** Pesado, menos flexible.
  + 1. Ruby on Rails
* **Descripción:** Framework web en Ruby.
* **Características:**
* Convención sobre configuración.
* Herramientas integradas.
* ORM (ActiveRecord).
* **Ventajas:** Productividad, orientado a prototipos rápidos.
* **Desventajas:** Rendimiento, curva de aprendizaje elevada.
  + 1. Spring boot
* **Descripción:** Framework para aplicaciones Java.
* **Características:**
* Microservicios.
* Configuración automática.
* Seguridad incorporada.
* **Ventajas:** Potente, escalable.
* **Desventajas:** Configuración compleja, pesado.
  + 1. Express.js
* **Descripción:** Similar a Node.js, minimalista y ligero.
* **Características:**
* Middleware.
* Ruteo sencillo.
* Gran comunidad.
* **Ventajas:** Ligero, flexible.
* **Desventajas:** Requiere más configuración, menos estructurado.
  + 1. Flask
* **Descripción:** Microframework web en Python, similar a Django pero menos complejo.
* **Características:**
* Minimalista.
* Extensible.
* WSGI.
* **Ventajas:** Ligero, fácil de aprender.
* **Desventajas:** No tiene muchas herramientas integradas.
  + 1. ASP.NET Core
* **Descripción:** Framework para construir aplicaciones web modernas en .NET.
* **Características:**
* Cross-platform.
* Inyección de dependencias.
* Buen rendimiento.
* **Ventajas:** Integración con el ecosistema .NET además de un buen rendimiento.
* **Desventajas:** Curva de aprendizaje y más complejo para principiantes.
  + 1. Tabla comparativa sobre tecnologías de BackEnd

A continuación, vamos a ver como en el apartado anterior, una tabla comparativa sobre las tecnologías de backend.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **BACKEND** | **Node.js** | **Django** | **Ruby on Rails** | **Spring Boot** | **Express.js** | **Flask** | **ASP.NET**  **Core** |
| Estructura MVC |  |  |  |  |  |  |  |
| ORM incorporado |  |  |  |  |  |  |  |
| Herramientas de administración |  |  |  |  |  |  |  |
| Microservicios |  |  |  |  |  |  |  |
| Configuración automática |  |  |  |  |  |  |  |
| Seguridad incorporada |  |  |  |  |  |  |  |
| Ruteo sencillo |  |  |  |  |  |  |  |
| Extensible |  |  |  |  |  |  |  |
| Cross-platform |  |  |  |  |  |  |  |
| Inyección de dependencias |  |  |  |  |  |  |  |
| Buen rendimiento |  |  |  |  |  |  |  |
| Ligero |  |  |  |  |  |  |  |
| Curva de aprendizaje |  |  |  |  |  |  |  |
| Trabajo de forma asíncrona |  |  |  |  |  |  |  |
| Gran comunidad |  |  |  |  |  |  |  |

Tabla 2. Tabla comparativa BackEnd

* 1. Selección de tecnologías

Las tecnologías de FrontEnd y BackEnd que se han analizado representan lo último en herramientas y frameworks utilizados por empresas para desarrollar aplicaciones web modernas. La elección de unas sobre otras depende de varios factores como pueden ser: requisitos del proyecto o necesidades de escalabilidad y rendimiento.

* + 1. FrontEnd

En nuestro caso se ha elegido utilizar **Angular** en el apartado de FrontEnd.



Ilustración 2. Logotipo Angular

Hemos seleccionado esta tecnología por las siguientes características:

1. **Desarrollo de SPAs (Single Page Applications)**

Angular es ideal para construir aplicaciones de una sola página (SPA) que proporcionan una experiencia de usuario fluida y rápida sin necesidad de recargar la página.

1. **Estructura Modular**

Utiliza una arquitectura basada en módulos que permite la organización del código, facilitando mantenimiento y escalabilidad del proyecto.

1. **TypeScript**

Angular está construido sobre TypeScript, similar a JavaScript con tipos estáticos y observables. Esto mejora la calidad del código y facilita la detección de errores durante el desarrollo.

1. **Inyección de dependencias**

Angular tiene un sistema de inyección de dependencias que mejora el modularidad y facilita la reutilización de componentes.

1. **CLI (Command Line Interface)**

Angular CLI facilita la creación de nuevos proyectos, la generación de componentes, servicios, y más, además de simplificar diferentes tareas como pruebas o despliegues.

1. **Soporte y comunidad**

Angular es desarrollado y mantenido por Google, lo que garantiza su continuidad y evolución. Además, cuenta con una gran comunidad de desarrolladores y recursos de aprendizaje.

* + 1. BackEnd

Para la parte del BackEnd se ha decidido utilizar **ASP.NET Core.**

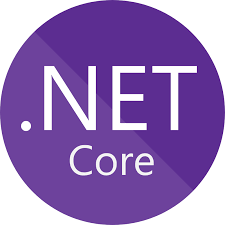


Ilustración 3. Logotipo ASP .NET Core

Hemos decidido usar esta tecnología dadas las siguientes características:

1. **Rendimiento**

ASP.NET Core está diseñado para ser un framework de alto rendimiento. Su arquitectura modular y ligera contribuye a mejorar la velocidad y eficiencia de las aplicaciones.

1. **Cross-platform**

A diferencia de su predecesor, ASP.NET Core es completamente multiplataforma, permitiendo que las aplicaciones se ejecuten en Windows, macOS y Linux, aunque en nuestro caso, no es relevante dado que vamos a hacer la aplicación solo para Windows.

1. **Inyección de dependencias integrada**

ASP.NET Core tiene un sistema de inyección de dependencias incorporado que facilita la gestión de dependencias (paquetes Nuggets).

1. **Modularidad y flexibilidad**

ASP.NET Core permite agregar solo los componentes necesarios con lo que, gracias a esta característica, podemos mejorar mucho el rendimiento y ligereza del proyecto.

1. **Compatibilidad con Microservicios**

Es ideal para arquitecturas de microservicios, permitiendo la construcción de aplicaciones escalables.

1. **Seguridad**

ASP.NET Core proporciona características de seguridad robustas, como protección contra ataques CSRF.

1. **Ecosistema .NET**

Se integra perfectamente con el ecosistema .NET, permitiendo el uso de bibliotecas y herramientas existentes y aprovechando las capacidades de C#.

Además de todas las características mencionadas anteriormente, otro motivo importante es que, para hacer este TFG, era necesario usar tecnologías que ayudaran personalmente a la incorporación al mundo laboral. Son tecnologías nuevas y de las más usadas, ambas tienen un gran rendimiento y un soporte a largo plazo, por lo que son ideales para la realización de este proyecto.

* 1. Estudio sobre las aplicaciones bancarias
     1. Banco La Caixa
* **Descripción:** La página web de “La Caixa” es la plataforma digital que ofrece a sus usuarios una amplia gama de servicios bancarios en línea, incluyendo gestión de cuentas, transferencias, pagos, inversiones y servicios de financiación, tanto para particulares como para empresas. Además, proporciona información detallada sobre los productos y servicios de CaixaBank, así como acceso a recursos de apoyo y contacto.

Ilustración 4. Logotipo Caixa

* **Características**
* Gestión de cuentas: permite a los usuarios gestionar sus cuentas bancarias, realizar transferencias y ver movimientos.
* Servicios de inversión: ofrece información y herramientas para la gestión de inversiones y fondos.
* Préstamos y financiación: proporciona opciones de financiación y préstamos con detalles sobre condiciones y solicitudes.
* Pagos y transferencias: facilita el pago de facturas y la realización de transferencias tanto nacionales como internacionales.
* Soporte y atención al cliente: incluye recursos de ayuda, FAQs, y contacto con atención al cliente.
* Seguridad: implementa medidas de seguridad avanzadas para proteger las transacciones y datos de los usuarios.
* **Ventajas**
* Comodidad: permite realizar operaciones becarias desde cualquier lugar y en cualquier momento.
* Ahorro de tiempo: los usuarios pueden gestionar sus finanzas sin necesidad de acudir a una sucursal física.
* Acceso a información: ofrece un acceso fácil y rápido a información sobre productos financieros y servicios.
* Seguridad: alta seguridad en las transacciones, protegiendo los datos personales y financieros de los usuarios.
* **Desventajas**
* Dependencia tecnológica: requiere una conexión a internet y dispositivos adecuados para el acceso.
* Curva de aprendizaje: algunos usuarios pueden encontrar complicado navegar y utilizar todas las funciones disponibles.
* Limitaciones en la interacción personal: la falta de interacción cara a cara puede ser un inconveniente para aquellos que prefieren el trato personal en sus gestiones bancarias.
  + 1. Banco Santander
* **Descripción**: la página web de Santander proporciona una plataforma completa para la gestión de servicios bancarios y financieros. Está diseñada para facilitar el acceso a cuentas personales y empresariales, realizar transacciones y gestionar productos financieros de manera eficiente.

Ilustración 5. Logotipo Santander

* **Características:**
* Acceso a cuentas y productos: permite a los usuarios consultar saldos, movimientos y gestionar tarjetas, préstamos y seguros.
* Transacciones en línea: facilita la realización de transferencias, pagos de recibos e impuestos, y envío de dinero a través de Bizum.
* Seguridad: implementa medidas de seguridad avanzadas como la autenticación con huella dactilar y la generación de códigos para operaciones seguras.
* Herramientas financieras: ofrece simuladores y opciones de financiación como fraccionamiento de compras y pagos a plazos.
* **Ventajas:**
* Comodidad y accesibilidad: los usuarios pueden gestionar sus finanzas desde cualquier lugar y en cualquier momento.
* Amplia gama de servicios: proporciona una gran variedad de servicios financieros y herramientas de gestión.
* Integración móvil: la app de Santander permite realizar todas las operaciones bancarias desde dispositivos móviles, incluyendo pagos y retiradas de efectivo.
* Alertas personalizadas: los usuarios pueden configurar alertas para mantenerse informados sobre movimientos y transacciones.
* **Desventajas:**
* Curva de aprendizaje: la complejidad de algunas funciones puede requerir tiempo para que los usuarios menos familiarizados se adapten.
* Pesadez del sitio: algunas funcionalidades pueden resultar lentas o complicadas debido a la amplia cantidad de servicios y herramientas disponibles.
* Menor flexibilidad: la página web puede resultar menos flexible y más estructurada.
  + 1. Banco BBVA
* **Descripción:** la página web de BBVA ofrece una plataforma digital integral que permite a los usuarios gestionar sus finanzas personales y empresariales en línea. Proporciona acceso a una variedad de servicios bancarios, incluidos cuentas, préstamos, inversiones y herramientas de planificación financiera.

Ilustración 6. Logotipo BBVA

* **Características:**
* Banca online y móvil: acceso a servicios bancarios a través de la web y aplicaciones móviles.
* Seguridad: múltiples capas de seguridad, incluyendo autenticación de dos factores y cifrado SSL.
* Gestión de cuentas: visualización y gestión de cuentas, transferencias y pagos.
* Inversiones: acceso a productos de inversión y herramientas de análisis.
* Atención al cliente: soporte a través de chat, teléfono y correo electrónico.
* **Ventajas**
* Comodidad: permite a los usuarios a realizar transacciones y gestionar sus finanzas desde cualquier lugar.
* Seguridad: protocolos de seguridad robustos para proteger la información del usuario.
* Variedad de servicios: amplia gama de servicios financieros disponibles en línea.
* Interfaz intuitiva: Diseño fácil de usar.
* **Desventajas:**
* Curva de aprendizaje: algunos usuarios pueden encontrar complejas las funcionalidades avanzadas.
* Requiere conexión a internet: la dependencia de internet puede ser un inconveniente en áreas con mala conectividad.
* Limitaciones regionales: algunos servicios pueden no estar disponibles en todas las regiones.
  + 1. Banco Bankinter
* **Descripción:** La página web de Bankinter es una plataforma digital que ofrece servicios bancarios a individuos, empresas y clientes de banca privada. Está diseñada para proporcionar acceso a una amplia gama de productos y servicios financieros, desde cuentas bancarias y tarjetas hasta inversiones y seguros.

Ilustración 7. Logotipo Bankinter

* **Características:**
* Interfaz amigable: navegación sencilla y accesible
* Acceso seguro: inicia sesión en el área privada con opciones de seguridad mejoradas.
* Servicios integrados: herramientas para la gestión de cuentas, transferencias y pagos.
* Calculadoras financieras: diversas herramientas para calcular hipotecas, inversiones y seguros.
* Información detallada: acceso a información y comparación de productos financieros.
* Soporte al cliente: opciones de contacto y asistencia en línea.
* **Ventajas:**
* Amplia gama de herramientas: calculadora y simuladores que facilitan la planificación financiera.
* Acceso móvil: versión optimizada para dispositivos móviles.
* Personalización: ofertas y productos adaptados a las necesidades del usuario.
* Seguridad: protocolos robustos para la protección de datos y transacciones.
* Facilidad de uso: interfaz intuitiva que facilita la navegación y el acceso a servicios.
* **Desventajas:**
* Curva de aprendizaje: puede ser abrumador para nuevos usuarios debido al amplia gama de servicios.
* Dependencia de internet: requiere una conexión estable para el uso óptimo de todas sus funcionalidades.
* Actualizaciones necesarias: requiere actualizaciones periódicas del navegador para garantizar la seguridad y funcionalidad.
  + 1. Banco Sabadell
* **Descripción:** la página web de Banco Sabadell es una plataforma digital que ofrece una amplia gama de servicios bancarios y financieros tanto para clientes particulares como para empresas. A través de su portal, los usuarios pueden acceder a cuentas bancarias, tarjetas, hipotecas, préstamos, inversiones, seguros, etc.

Ilustración 8. Logotipo Sabadell

* **Características:**
* Cuentas y tarjetas: los usuarios pueden abrir y gestionar cuentas, así como solicitar tarjetas de débito y crédito.
* Préstamos e hipotecas: simulación y solicitud de préstamos personales e hipotecas.
* Servicios digitales: incluye servicios como Bizum para transferencias instantáneas y la posibilidad de gestionar cuentas y pagos a través de la aplicación móvil.
* Gestión de inversiones y ahorros: opciones para gestionar inversiones y planes de ahorro.
* Atención al cliente: asistencia a través de chat en vivo, teléfono y formularios de contacto.
* **Ventajas:**
* Acceso rápido y fácil: los usuarios pueden abrir cuentas y gestionar sus finanzas en menos de 10 minutos a través de su móvil.
* Variedad de servicios: ofrece una amplia gama de productos financieros adaptados a diferentes necesidades.
* seguridad: dispone de medidas de seguridad robustas para proteger las transacciones y la información del usuario.
* Integración con Bizum: facilita transferencias rápidas y sencillas.
* Alertas y notificaciones: los usuarios reciben notificaciones personalizadas sobre sus transacciones y movimientos bancarios.
* **Desventajas:**
* Curva de aprendizaje: algunos usuarios pueden encontrar compleja la navegación y utilización inicial del sitio web.
* Limitaciones en servicios profesionales: algunas cuentas, como las de autónomos, tienen restricciones específicas.
* Menor personalización: aunque ofrece muchas herramientas, puede no ser tan flexible o personalizada como el contacto directo en una sucursal.
  + 1. Tabla comparativa de las aplicaciones bancarias.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **SERVICIOS** | **CaixaBank** | **Santander** | **BBVA** | **Bankinter** | **Sabadell** |
| **Características** | | | | | |
| Gestión de cuentas |  |  |  |  |  |
| Préstamos y financiación |  |  |  |  |  |
| Pagos y transferencias |  |  |  |  |  |
| Soporte de atención al cliente |  |  |  |  |  |
| Seguridad |  |  |  |  |  |
| Información detallada |  |  |  |  |  |
| **Ventajas** | | | | | |
| Comodidad |  |  |  |  |  |
| Ahorro de tiempo |  |  |  |  |  |
| Integración móvil |  |  |  |  |  |
| Personalización |  |  |  |  |  |
| **Desventajas** | | | | | |
| Dependencia tecnológica |  |  |  |  |  |
| Pesadez del sitio |  |  |  |  |  |
| Limitaciones regionales |  |  |  |  |  |
| Limitaciones profesionales |  |  |  |  |  |

Tabla 3. Tabla comparativa bancos

1. ALCANCE DEL PROYECTO

Antes de comenzar con la parte práctica del trabajo, es conveniente hacer un estudio del propio desarrollo del producto, para lograrlo se ha hecho uso de tres prácticas estándar extendidas en la industria del desarrollo de software, la elicitación de requisitos, la evaluación de riesgos y un pequeño análisis de viabilidad, que sirve para proporcionar un poco más de información sobre qué usuarios podrían llegar a hacer uso de la aplicación.

* 1. Elicitación de requisitos del producto
     1. Requisitos de información

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Descripción | Prioridad | Criterio de aceptación |
| RI-01 | Debe almacenar los siguientes datos: nombre, apellidos, nombre de usuario, contraseña, avatar, dirección de correo electrónico e IBAN | Crítica | La base de datos debe permitir almacenar los datos mencionados y consultarlos |
| RI-02 | Debe permitir almacenar, descripción, cantidad, persona que hace, persona que recibe y fecha sobre una transferencia | Critica | Todas las transferencias deben tener estos datos |
| RI-03 | Debe almacenar un registro de todas las transferencias del usuario | Alta | Debe poder revisar en todo momento sus transferencias, así como sus movimientos |
| RI-04 | Debe almacenar datos de ingresos y extracciones, de dinero a la cuenta del usuario, con su fecha y cantidad correspondiente | Alta | Debe poder revisar en todo momento los ingresos y extracciones de la cuenta del usuario |

Tabla 4. REQUISITOS DE INFORMACIÓN

* + 1. Requisitos funcionales

A continuación, veremos los requisitos funcionales de nuestra aplicación.

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Descripción | Prioridad | Criterio de aceptación |
| RF-01 | Debe permitir un inicio de sesión para el acceso a los datos del usuario | Crítica | Debe controlarse el acceso a los datos mediante la sesión activa |
| RF-02 | Debe permitir la modificación o eliminación de los datos de la cuenta de un usuario | Critica | El usuario debe tener control sobre los datos de su cuenta |
| RF-03 | Debe permitir al usuario hacer ingresos o extracciones en la cuenta | Alta | Debe existir un formulario para llevar a cabo estas acciones |
| RF-04 | Debe posibilitar hacer transferencias de una cuenta a otra registrada en la aplicación | Alta | Las transferencias tendrán un máximo y un mínimo de dinero, nunca podrá ser superior al dinero íntegro en la cuenta |
| RF-05 | Debe tener una lista de movimientos para la revisión de estos por parte del usuario | Alta | Se hará una lista paginada la cual sea agradable y sencilla |
| RF-06 | Debe poder revisar en detalle los datos de cualquier movimiento en la cuenta | Alta | El usuario puede ver en cualquier momento los datos de los movimientos de su cuenta |
| RF-07 | Debe mostrar información de contacto del administrador de la aplicación | Baja | Ha de existir un apartado que cuente con la dirección de correo electrónico del administrador |
| RF-08 | El administrador debe poder modificar y eliminar cualquier dato de la aplicación | Media | El administrador tiene total acceso y poder sobre la aplicación |
| RF-09 | Debe poder exportar los datos de los movimientos de una cuenta en formato PDF | Media | El usuario puede exportar los datos de los movimientos de la cuenta |

Tabla 5. Requisitos funcionales

* + 1. Requisitos no funcionales
       1. Requisitos de calidad

En este apartado se verán los requisitos de calidad.

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Descripción | Prioridad | Criterio de aceptación |
| RC-01 | La disponibilidad del sistema será la más alta posible | Alta | El sistema deberá ser estable, sin problemas y cierres inesperados |
| RC-02 | El sistema deberá ser rápido y eficiente para una mayor facilidad del uso | Media | El sistema será rápido para un mejor uso de este |

Tabla 6. Requisitos de calidad

* + - 1. Requisitos de implementación

En este apartado se expondrán los requisitos de implementación.

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Descripción | Prioridad | Criterio de aceptación |
| RIM-01 | Se implementara en una versión de escritorio para escritorio (Windows) | Alta | Será desarrollada en Angular y ASP .NET Core |
| RIM-02 | Empleará una base de datos para almacenar los datos de la aplicación | Media | Será una base de datos relacional y contendrá todos los datos de la aplicación |

Tabla 7. Requisitos de implementación

* + - 1. Requisitos de seguridad

En este apartado veremos los requisitos de seguridad necesarios en la aplicación.

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Descripción | Prioridad | Criterio de aceptación |
| RS-01 | La aplicación no tendrá ningún código ejecutable salvando librerías de terceros | Media | Aunque es código libre, el código oficial de la aplicación ha de ser protegido |
| RS-02 | No se podrá modificar movimientos por parte del usuario, solo por parte del administrador y en caso de ser necesario | Alta | La aplicación mantendrá la integridad de los datos del usuario |
| RS-03 | Debe garantizar la confidencialidad, integridad y autenticidad de la información transmitida | Crítica | Especialmente los datos del usuario estarán protegidos de forma robusta y no saldrán de la aplicación |
| RS-04 | No se debe ejecutar operaciones con privilegios en modo súper usuario | Alta | No se debe solicitar el modo administrador (root) en ningún momento |
| RS-06 | Se validarán los formularios para evitar inyecciones de código malicioso | Critica | Todos los formularios contarán con validación |
| RS-07 | La aplicación no contendrá ningún software malicioso o perjudicial para el dispositivo del usuario | Alta | La aplicación no contendrá ningún software malicioso, con fines lucrativos o dañinos |

* 1. Riesgos

La clasificación de los riesgos que pueden afectar a la aplicación se estimará multiplicando la probabilidad por el impacto del riesgo. Para determinar el impacto de un riesgo se tienen en cuenta tres parámetros, la modificación de alcance de la aplicación, repercusión en la calidad y necesidad de revisiones de la planificación. Para realizar la estimación se ha utilizado la siguiente figura:

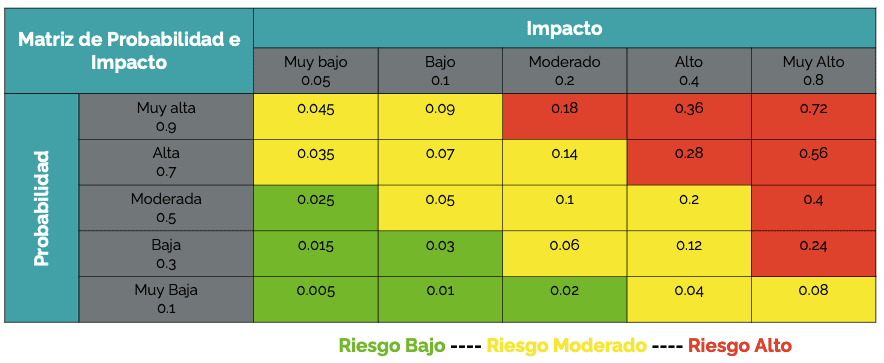


Ilustración 9. Matriz de probabilidad e impacto

Donde la probabilidad e impacto máxima es 0.72.

Para su evaluación, primero se consideran los riesgos entre 10% y 90% de probabilidades de que ocurra, luego se decidirá el impacto de ese riesgo, y se hará una multiplicación entre la columna y la fila correspondiente para obtener la puntuación.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Descripción | Probabilidad | Impacto | Puntuación | Riesgo |
| 1 | Estimación incorrecta de la planificación | Moderada | Muy Alto | 0.4 | Alto |
| 2 | Subestimar el alcance | Alta | Alto | 0.28 | Alto |
| 3 | Cambio de requisitos al avance del proyecto | Media | Alto | 0.2 | Medio |
| 4 | Incompatibilidad en funcionalidades tecnológicas | Baja | Bajo | 0.03 | Bajo |
| 5 | Alta complejidad en el desarrollo del modelo | Medio | Alto | 0.2 | Medio |
| 6 | Bajo rendimiento en la aplicación | Bajo | Medio | 0.06 | Medio |
| 7 | Baja aceptación de la aplicación | Bajo | Bajo | 0.03 | Bajo |

Tabla 8. Tabla de riesgos

Como podemos ver en la tabla 5, hay varios riesgos a considerar en el desarrollo de la aplicación, aunque vemos que la mayoría de ellos suponen un riesgo bajo/medio, hay algunos riesgos altos, aunque sean en la parte organizativa de la aplicación.

* 1. Viabilidad

En este caso se ha hecho una encuesta anónima en la cual hay 7 preguntas realizadas mediante la plataforma de Google Forms. Se debe destacar que en dicha en cuentas han participado 75 personas, de las cuales se han revelado los siguientes resultados.

Es importante mencionar que el profesor Juan Antonio Andrés Lalueza, perteneciente al Centro Educativo Altair, ubicado en la provincia de Sevilla, utilizará este proyecto, es decir, esta aplicación bancaria VVBA para sus clases de economía que imparte en dicho centro.

A continuación, con respecto a la gráfica 1, se puede observar que un 41´9%, es decir, 31 de 75 persona se muestran “satisfechos” con el uso de las aplicaciones bancarias de manera general. Por otro lado, hay un 2´7% y un 16,2% que se muestran “muy insatisfechos” e “insatisfechos” por lo que existe un número elevado de usuarios a los cuales, les resulta complicado el manejo de las aplicaciones bancarias.

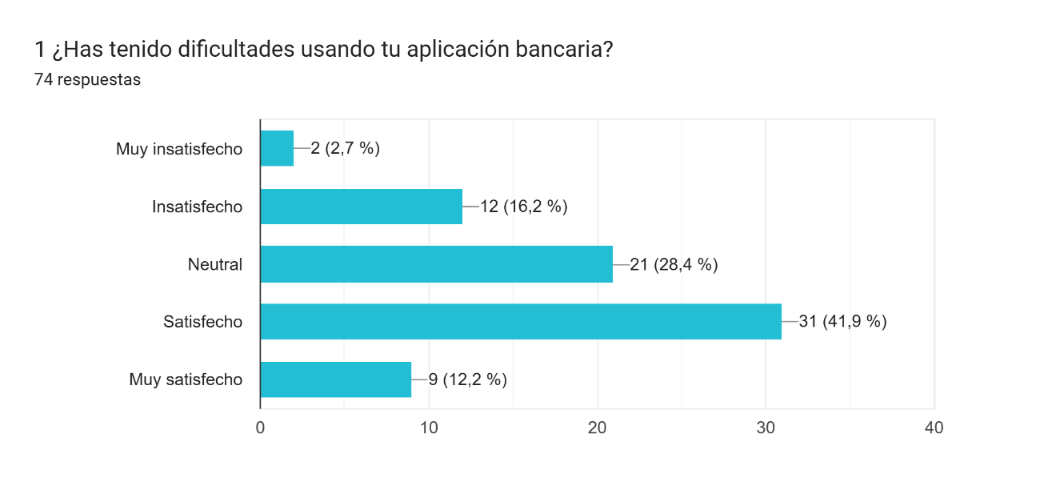


Ilustración 10. Encuesta pregunta 1

En relación a la gráfica 2, existe un 60,8% que se muestra “satisfecho” con las aplicaciones bancarias en general, por lo que se puede intuir que el uso de estas aplicaciones, facilita a los usuarios las operaciones y tramites bancarios.

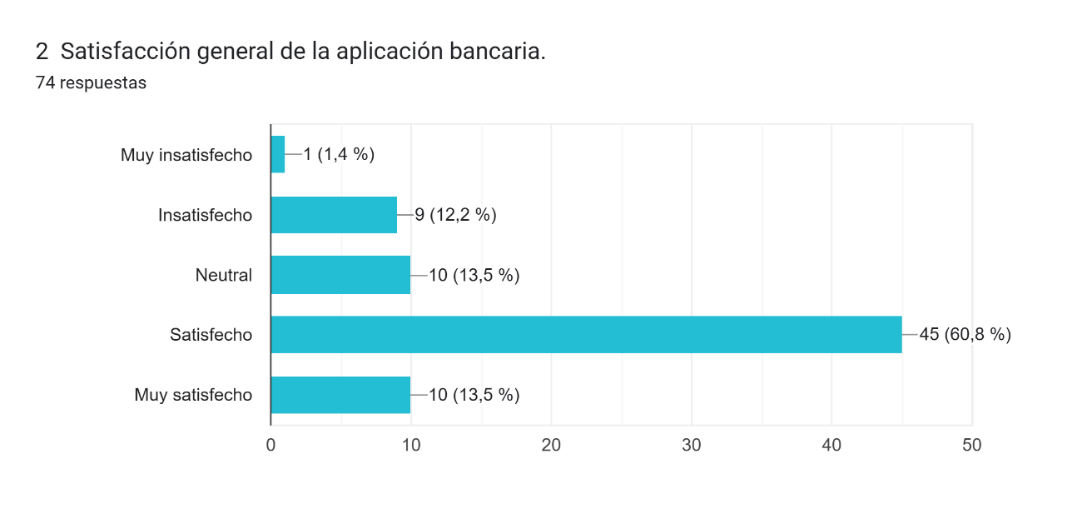


Ilustración 11. Encuesta Pregunta 2

En la gráfica 3, los usuarios se muestran “satisfechos” con un 52´7% y “muy satisfecho” con un 27%, esto significa que un gran número de personas sienten seguridad a la hora de realizar transacciones a través de las aplicaciones bancarias. El resto de personas se muestran “neutral” o se sienten inseguras a la hora de realizar estas operaciones.

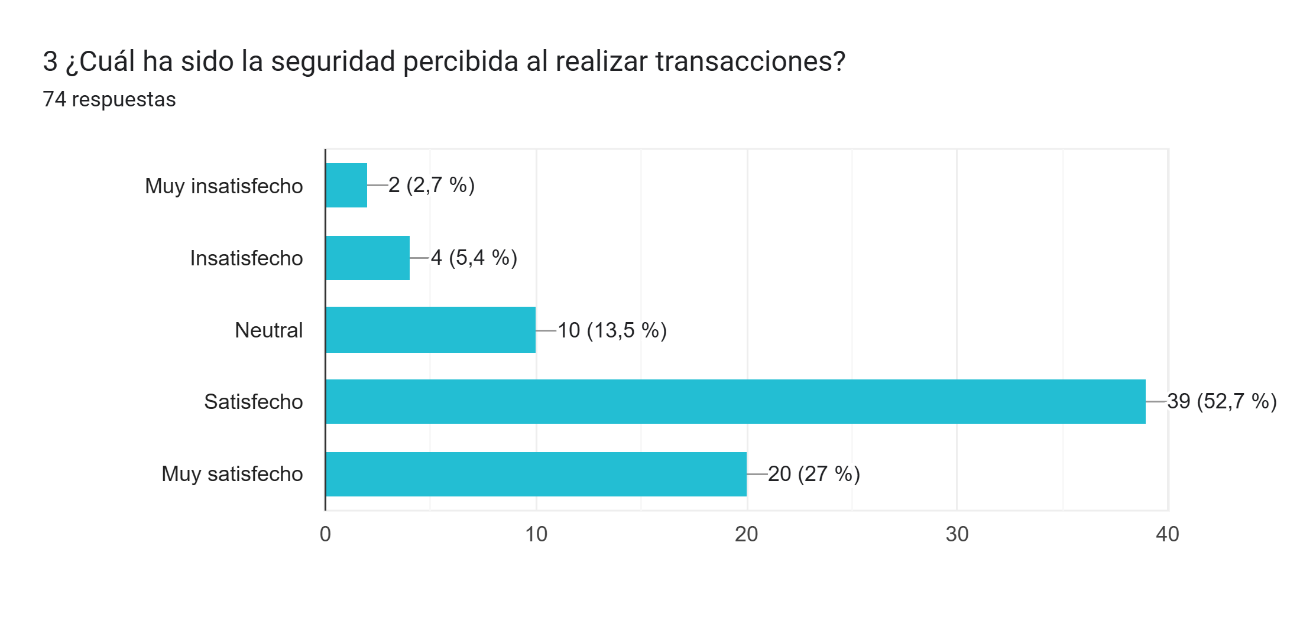


Ilustración 12. Encuesta pregunta 3

En la gráfica 4, se puede observar que un 81´1% de las personas no necesitan una aplicación bancaria de prueba para poder entender mejor su uso, por otro lado, un 20´3% necesitarían una aplicación de prueba que les simule cada una de las operaciones que pueden realizar en una aplicación bancaria de manera real, para que de esta manera puedan practicar.

Se debe destacar que la muestra obtenida ha sido pequeña, por lo que estos datos podrían fluctuar dependiendo de número de personas, edad o la población a la que se ha dirigido la encuesta.

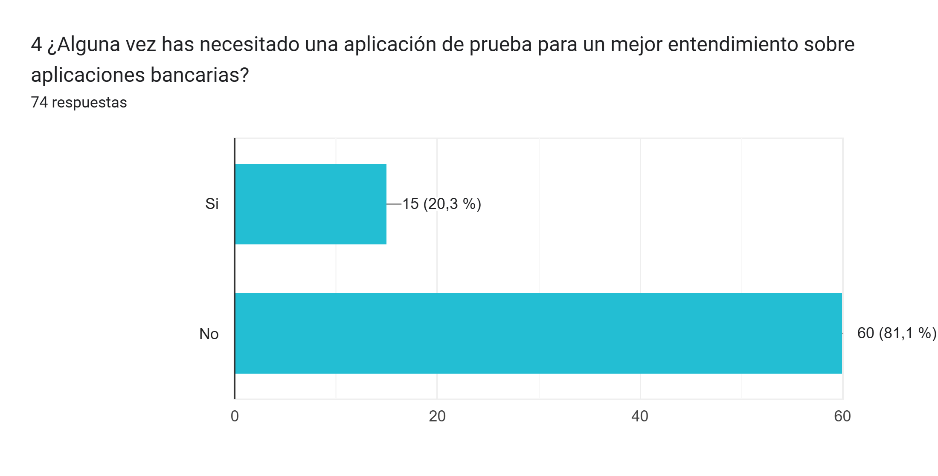


Ilustración 13. Encuesta pregunta 4

En la gráfica 5, se puede observar que como un 39´7% de los usuarios que han realizado la encuesta se muestran “neutral”, esto significa que durante el uso de las aplicaciones bancarias todavía no se han encontrado con alguna incidencia o problemas técnicos que los lleve a valorar este aspecto. Aun así, se puede observar que un 24´7% se muestran “insatisfechos”, son usuarios que, ante un incidente o problemas técnicos, el banco a través de la aplicación bancaria no les ha podido ofrecer una solución. Por otro lado, un 28´8% se muestra “satisfecha” con las soluciones obtenidas a través de la aplicación.

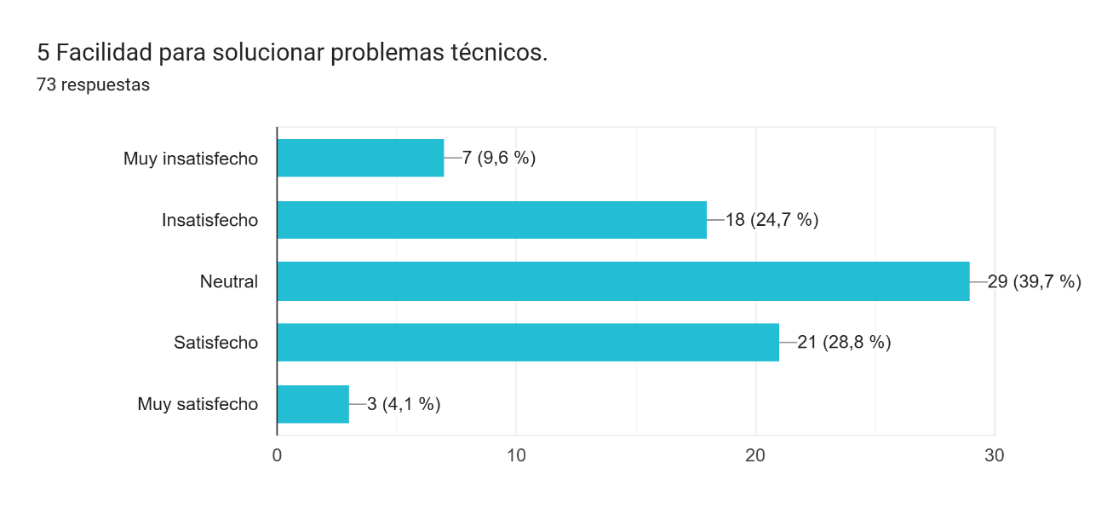


Ilustración 14. Encuesta pregunta 5

En la gráfica 6, se puede percibir que un 62´2% de los usuarios que han realizado la encuesta, no necesitan tener una aplicación bancaria, sin embargo, hay un 41´9% que si desearía obtener una aplicación bancaria de prueba para practicar o simular sus operaciones bancarias.

Cabe destacar que, al presentar una muestra pequeña de participantes, estos valores pueden variar dependiendo de la cantidad de participantes que interactúen y según a la población que vaya dirigida la encuesta. Aún así se pretende mostrar de manera general unos resultados previos para realizar un análisis.

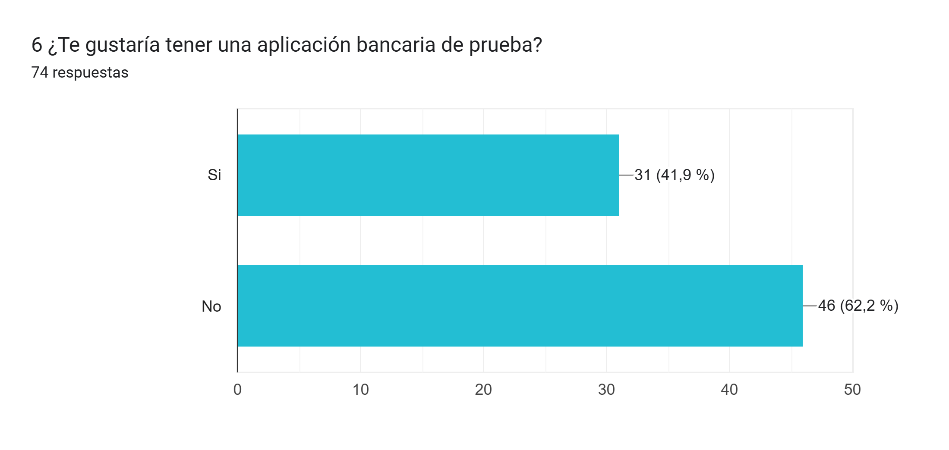


Ilustración 15. Encuesta pregunta 6

Por último, en la gráfica 7, existe un alto porcentaje de usuarios 83´8% que no invertiría 2€ para probar una aplicación de prueba, por otro lado, un 17´6% si pagaría este dinero por poseer una aplicación bancaria de prueba que les permitiera aprender cada uno de los movimientos u operaciones bancarias.

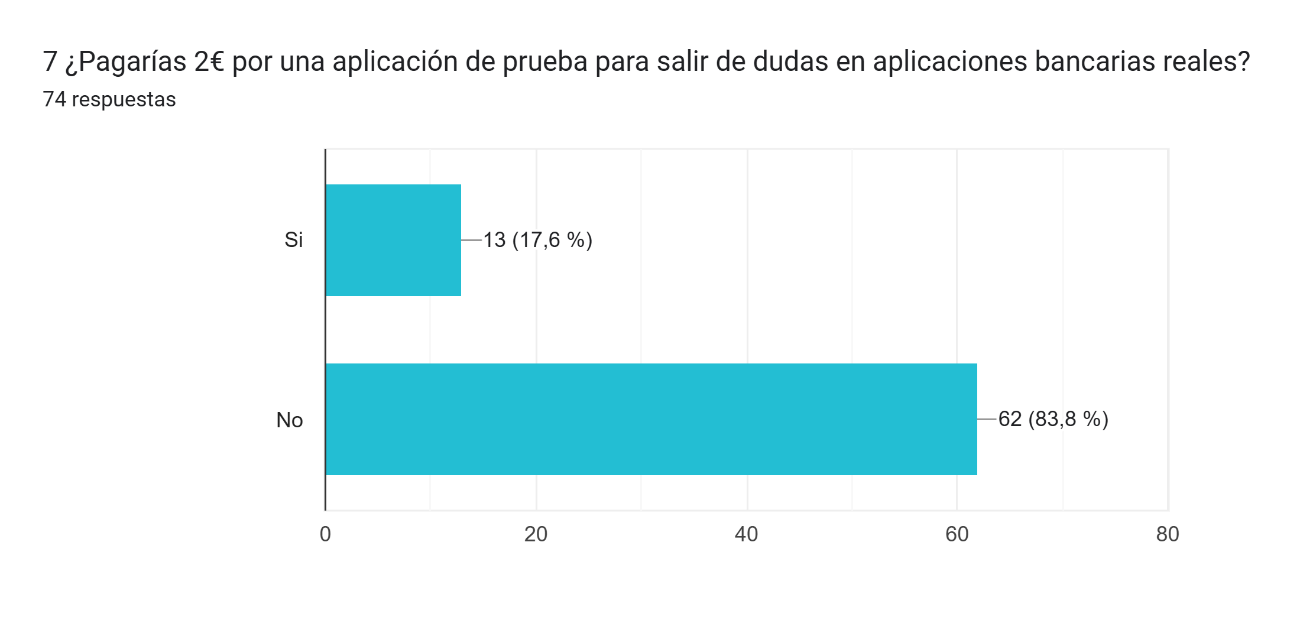


Ilustración 16. Encuesta pregunta 7

* 1. Objetivos de la aplicación

El objetivo principal del trabajo es la investigación de las tecnologías más útiles ya que tienen un mayor porcentaje de uso en las empresas y una aplicación de estas tecnologías basada en una solución especifica. Esta solución contará del desarrollo de una aplicación web orientada a ser un ejemplo general de un banco el cual tiene todas las funcionalidades más usadas, para que los usuarios, puedan practicar la utilización de aplicaciones del mismo estilo y haya una mayor comprensión general de estas.

1. VERSIÓN DE COSTES
   1. Coste salarial del desarrollo

En este apartado se desglosan los costes salariales del equipo de desarrollo que contempla desde la primera semana de vida del proyecto hasta su puesta en producción, en este caso, 15 semanas. Cabe destacar de que se va a trabajar unas 20 horas semanales en el proyecto.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Sueldo por hora | | Sueldo por semana | | Coste Total mensual | |
| Rol | Bruto | Neto | Bruto | Neto | Bruto | Neto |
| Project Manager | 23,50€ | 16,45€ | 470€ | 329€ | 1880€ | 1316€ |
|  |  |  |  |  |  |  |

Tabla 9. Coste de desarrollo del equipo

Como solo es una persona la que realiza el proyecto, el coste salarial del equipo mensual es igual al coste mensual del integrante.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Coste total | |
| Evento | Duración (semanas) | Bruto | Neto |
| Preparación del proyecto | 4 | 1880€ | 1316€ |
| Sprint 1 | 3 | 1410€ | 987€ |
| Sprint 2 | 3 | 1410€ | 987€ |
| Sprint 3 | 3 | 1410€ | 987€ |
| Preparación | 2 | 940€ | 658€ |
| Total | **15** | **7.050€** | **4.935€** |

Tabla 10. Coste total salarial del desarrollo del proyecto

* 1. Costes derivados del desarrollo

Como su propio nombre indica, aquí se describirán y definirán los costes derivados del desarrollo, en el mismo periodo de tiempo que hemos definido hasta la puesta en producción, 15 semanas.

|  |  |  |
| --- | --- | --- |
| Tipo de gastos | Gasto semanal | Gasto mensual |
| Gasto eléctrico | 15€ | 60€ |
| Gasto de internet | 15€ | 60€ |

Tabla 11. Costes derivados en el desarrollo

Estos gastos se deben a la compensación de que sea un trabajo remoto.

|  |  |  |
| --- | --- | --- |
| Evento | Duración (semanas) | Coste total |
| Preparación del proyecto | 4 | 60€ |
| Sprint 1 | 3 | 45€ |
| Sprint 2 | 3 | 45€ |
| Sprint 3 | 3 | 45€ |
| Preparación | 2 | 30€ |
| Total | **15** | **225€** |

Tabla 12. Costes derivados en el desarrollo totales

* 1. Costes de licencias

A continuación, trataremos el desglose de los costes asociados a las licencias usadas durante el periodo de desarrollo. Teniendo en cuenta los precios sacados de la página oficial de Microsoft, y obteniendo el Microsoft 365 E3 EEA (sin teams).

|  |  |
| --- | --- |
| Licencia | Gasto por uso mensual |
| Microsoft Office | 35,70€ |
| ASP .NET Core | 0€ |
| Angular | 0€ |
| Total | **35,70€** |

Tabla 13. Coste de licencias mensual

Para el desarrollo y conteo de horas también usaremos Clockify, el cual también es gratuito y no necesitaremos pagar la licencia.

|  |  |  |
| --- | --- | --- |
| Evento | Duración (semanas) | Coste total |
| Preparación del proyecto | 4 | 35,70€ |
| Sprint 1 | 3 | 26,80€ |
| Sprint 2 | 3 | 26,80€ |
| Sprint 3 | 3 | 26,80€ |
| Preparación | 2 | 17,85€ |
| Total | **15** | **133,95€** |

Tabla 14. Costes de licencias totales

* 1. Coste del material de trabajo

Aquí se describen los gastos de coste de material de trabajo adquiridos durante el periodo de desarrollo, debido a que su vida útil será de más de 15 semanas de vida, se considerará las amortizaciones de los equipos y materiales adquiridos durante el periodo de desarrollo durante un año de vida.

|  |  |
| --- | --- |
| Item | Precio/unidad |
| Ordenador portátil | 600€ |

Tabla 15. Precio material de trabajo

|  |  |
| --- | --- |
| Amortización de los equipos | |
| Suma total | Periodo de amortización |
| 600 | 1 año |
| Amortización semanal | Amortización mensual |
| 12,50€ | 50€ |

Tabla 16. Amortización de los equipos

|  |  |
| --- | --- |
| Amortización durante el desarrollo | |
| Periodo de desarrollo | 15 semanas |
| Amortización semanal | 12,50€ |
| Total | **187,50€** |

Tabla 17. Amortización durante el desarrollo

Teniendo en cuenta los cálculos finales, en casi 4 meses se va a amortizar unos 187,50€, todo con el I.V.A incluido.

* 1. Desglose total de los costes de desarrollo

En este subapartado están resumidos todos los costes asociados al periodo de desarrollo del producto.

|  |  |
| --- | --- |
| Costes totales del desarrollo (sin reservas de contingencia) | |
| Costes salariales totales | 4935€ |
| Costes derivados totales | 225€ |
| Costes de licencia totales | 133,95€ |
| Costes de material total | 600€ |
| Total | **5893,95€** |

Tabla 18. Costes totales del desarrollo

Si le sumamos la reserva de contingencia del siguiente apartado se nos quedaría en **6.483,35€.**

* 1. Reservas de contingencia

Aquí se definirán las reservas de contingencia establecidas para el proyecto, que serán de un 10% del coste total (sin contar la propia reserva).

|  |  |
| --- | --- |
| Reservas de contingencia | |
| Porcentaje de contingencia | 10% |
| Presupuesto total | 5893,95€ |
| Total | **589,40€** |

Tabla 19. Reservas de contingencia

1. Evolución de la aplicación
   1. Fase de diseño del producto

Al ser una aplicación que va a tener un uso real con diferentes personas (tanto como aplicación de pruebas en el ámbito bancario, como aplicación escolar para el profesor Juan Antonio Andrés Lalueza, en sus clases de economía) se ha tenido una preocupación por la estética y el nivel de accesibilidad.

Para realizar la fase de diseño se ha utilizado “Bootstrap”, en conjunto con el material de “Icons de Google”, pues facilitan la representación del diseño y la funcionalidad de la aplicación para los usuarios.

Para la creación del diseño, la interfaz de usuario conocida con las iniciales UI y la experiencia de usuario reconocida como UX, se han empleado las siguientes características:

* **Planteamiento y maquetación:** En lo referente a los diferentes apartados de la aplicación, podemos destacar los bocetos o mockups,los cuales han dado un esquema de diseño a la misma, la herramienta con la cual se ha trabajado para la realización de dichos mockups es una herramienta online llamada Figma. Haciendo referencia al nombre, se ha formado haciendo un juego de palabras, cambiando letras del nombre de uno de los bancos más importantes en España.
* **Diseño del concepto**: Para el diseño se ha optado por una alternativa sencilla, donde los usuarios podrán reconocer de manera fácil cada uno de los conceptos y funciones que componen la aplicación. Para una mejor visualización de la aplicación, previamente se ha creado mockups de su diseño, estos se mostrarán posteriormente.

Por último, en esta fase han ajustado las diferentes paletas de colores para crear un aspecto visual agradable y observar con antelación el aspecto de la versión final.

* **Prototipado y diseño detallado**: En este apartado, se han realizado todos los cambios definitivos que han definido el aspecto de la versión final de la aplicación y el funcionamiento de la misma.

En esta fase se ha concretado también los iconos e imágenes usados a lo largo de las vistas de la aplicación. Estos recursos visuales externos se han llevado a cabo detallando su código tal y como se especifica en la página donde se han obtenido.

Las páginas principales de donde se han obtenido estos recursos, como hemos mencionado anteriormente son:

* **Material Icons de Google:** Estos son los iconos que se han usado en la aplicación provenientes de la API oficial de fuentes de Google.
* **Bootstrap:** Esto es una página a la que se hace referencia para obtener un CSS (Cascading Style Sheets) externo el cual nos ayuda a diseñar y estilar la página de una manera más fácil y eficiente.

En todo el proceso se ha llevado un nivel de armonía en los diferentes elementos de cada vista, introduciendo iconos, placeholders y una gama de colores agradable a la vista del usuario. Aparte se han utilizado imágenes acordes a las vistas de la aplicación.

* + 1. Mockups y prototipo

A continuación, se muestran cada una de las vistas y funciones que representa la aplicación, a través de mockups. Se va a proceder a hacer una división del prototipado de la aplicación, esta división se compone de tres partes: no logeados, usuario normal y usuario administrador.

Primeramente, se procederá a enseñar la parte de usuarios no logeados, que consta de una vista inicial, en la que se muestra la fase de acceso a la aplicación.

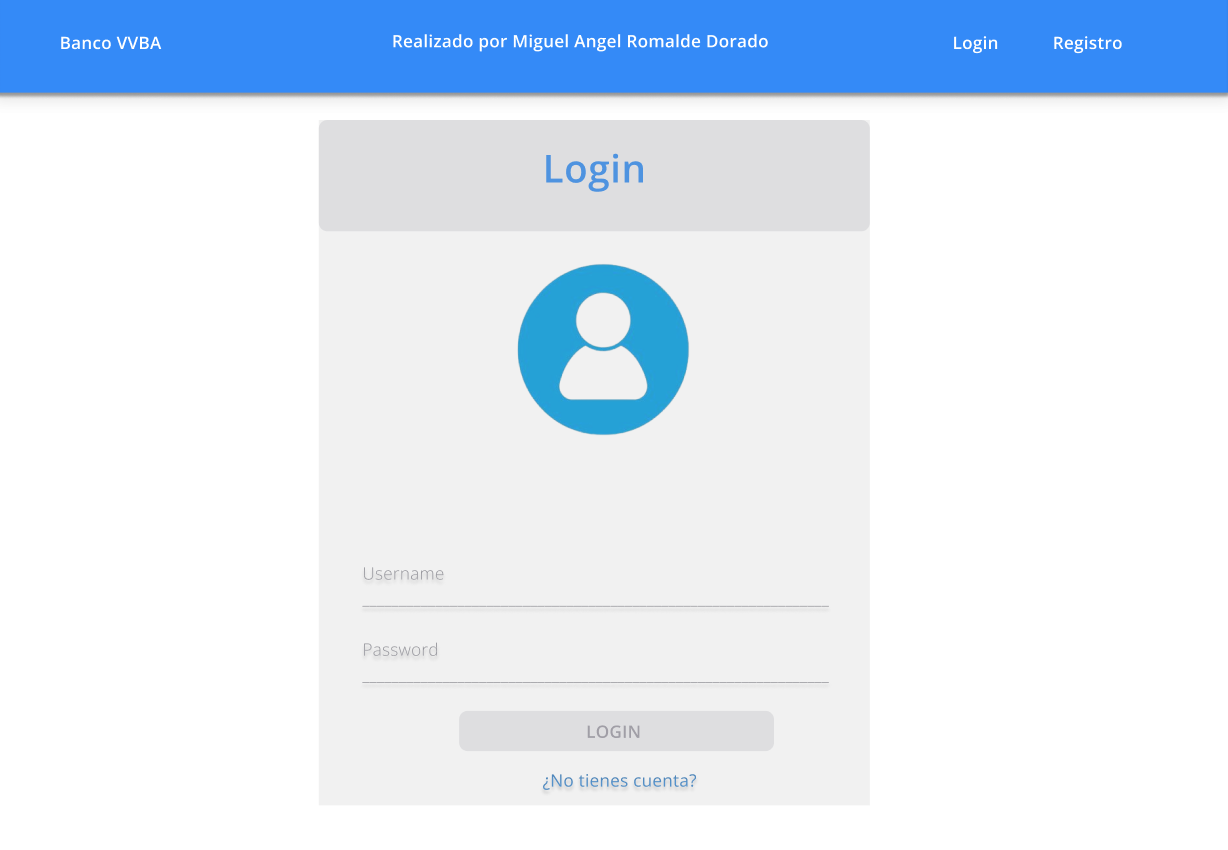


Ilustración 17. Login

Seguidamente, se muestra la fase de registro del usuario, en la cual, se obtienen todos los datos necesarios de la persona para poder acceder correctamente, usando las credenciales del usuario.

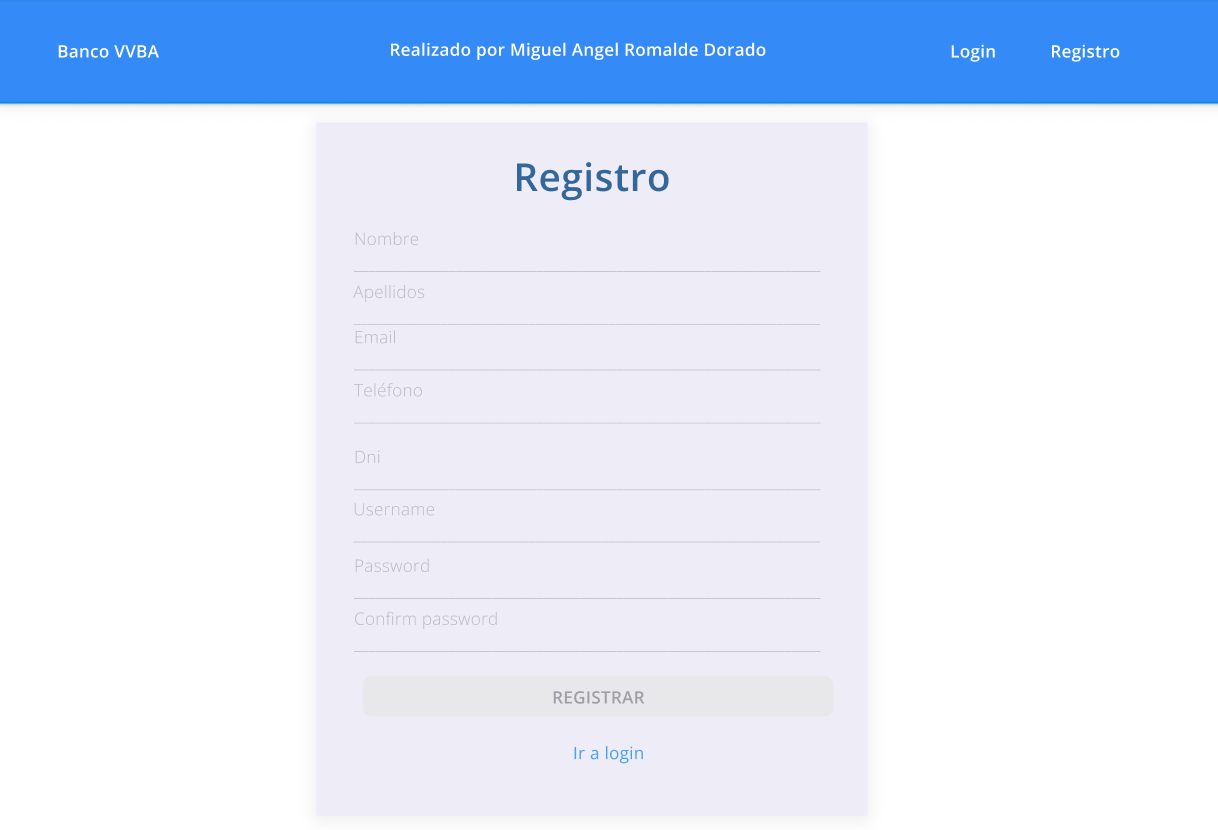


Ilustración 18. Registro

Se entra ahora en la fase de usuarios normales, en la cual se accede una vez que el usuario se ha logeado en la aplicación. Esta primera pantalla hace referencia a los datos de la cuenta del usuario y las operaciones de la misma.

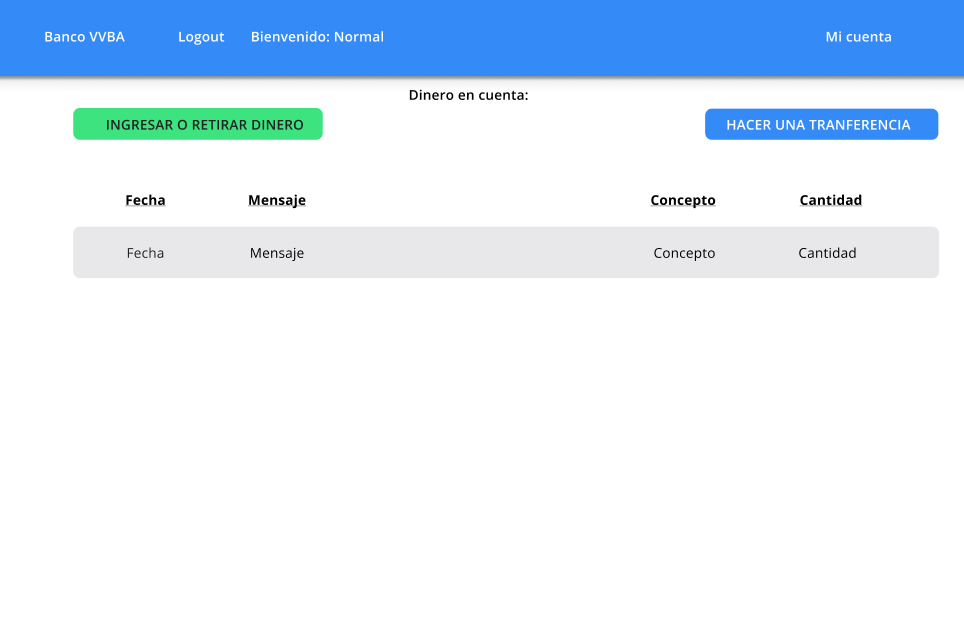


Ilustración 19. Detalles cuenta usuario.

En la siguiente ilustración se muestra el formulario referente al ingreso o retirada de dinero en el banco. Los usuarios accederán a esta vista pulsando el botón “ingresar o retirar dinero” que se observa en la imagen anterior.

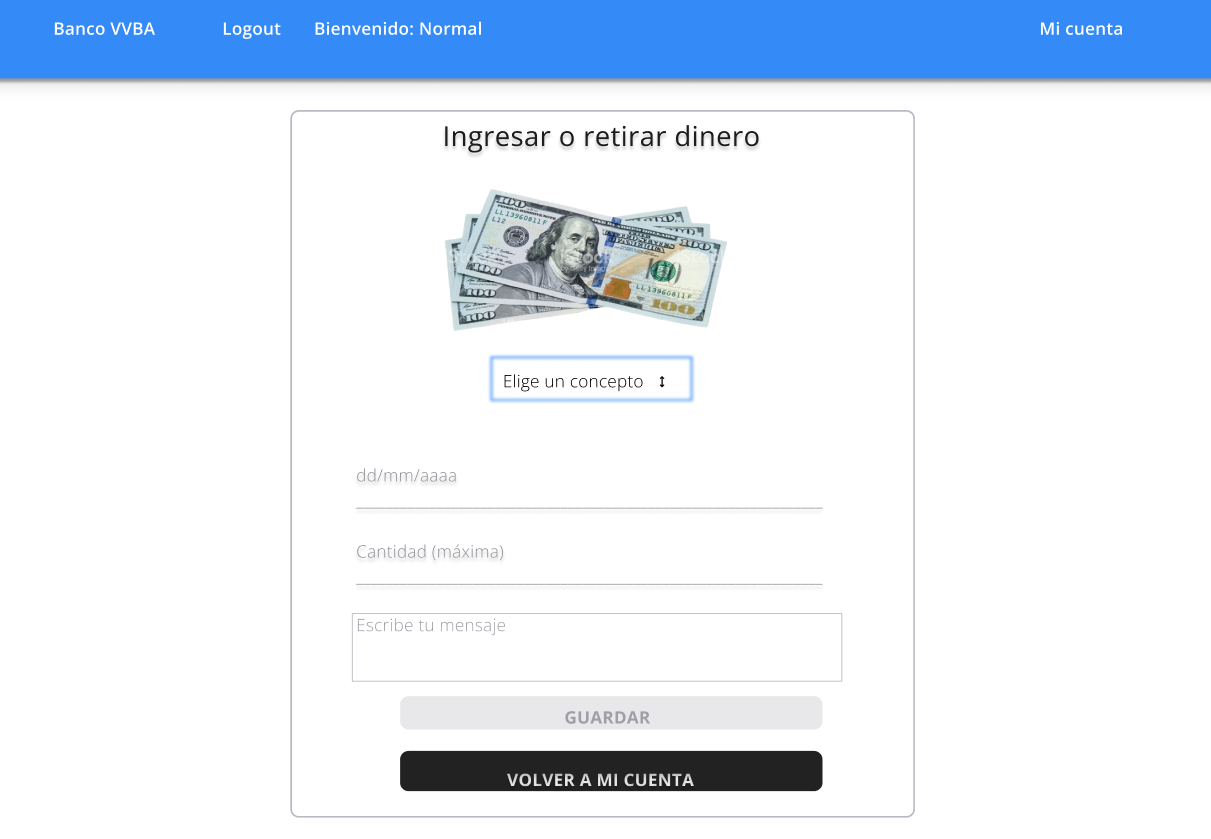


Ilustración 20. Ingresar o retirar dinero

Además de ingresar o retirar dinero, los usuarios podrán realizar transferencias de dinero hacia otra cuenta registrada en la aplicación. Los usuarios accederán a esta vista pulsando el botón “hacer una transferencia” que se observa en la ilustración 19.

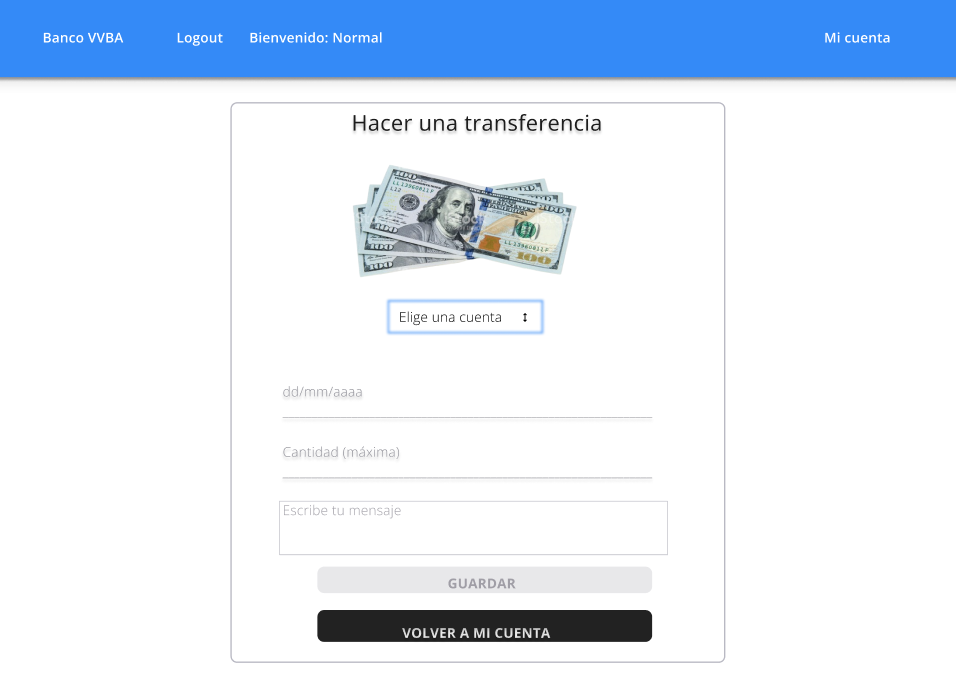


Ilustración 21. Transferencias

Por último, se entra en la fase de usuarios administradores, esta es la mayor fase y consta de 9 vistas, dado que hay algunas parecidas, pondremos una imagen de ejemplo y haremos referencia a las vistas a las que hace referencia.

Esta es la vista de usuarios, la cual muestra diferentes datos de los usuarios de la aplicación, esta vista es la que, mencionada anteriormente, hace referencia a 5 vistas, ya que la vista de usuarios, cuentas, operaciones, comisiones y cuentas comisiones son similares a esta con cambios menores.

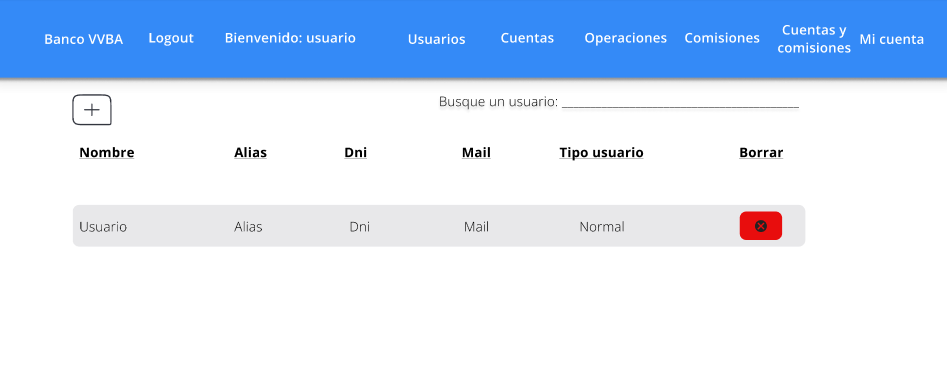


Ilustración 22. lista de usuarios

En la siguiente vista podemos ver la creación de un usuario hecha por el usuario administrador, en esta se puede elegir el tipo de usuario en diferencia al registro. Podemos navegar a esta vista pulsando el botón de la esquina izquierda en la pantalla de usuarios.

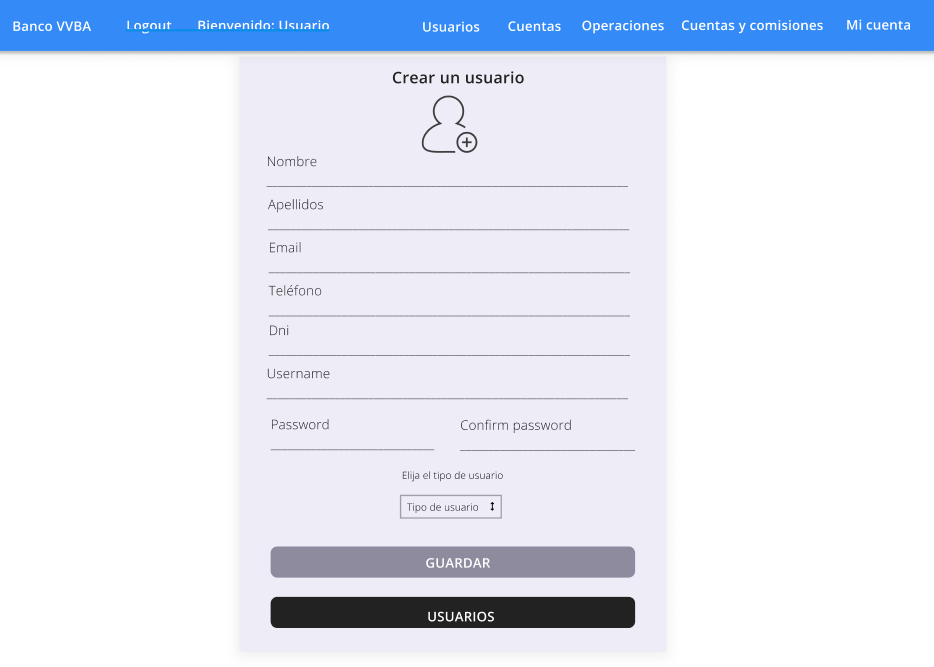


Ilustración 23. creacion usuario

En la posterior vista podemos ver la creación de una operación dentro de una cuenta, se pondrá en el placeholder, la cantidad máxima a ingresar o a retirar de forma dinámica dependiendo de la cuenta elegida.

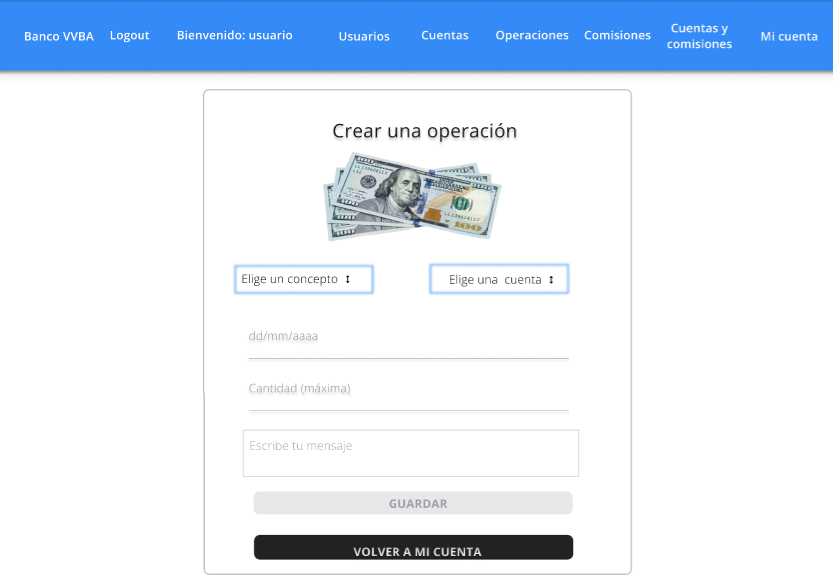


Ilustración 24. Creacion operacion

En la siguiente pantalla podemos crear un tipo de comisión dicha comisión podrá ser usada en la aplicación.

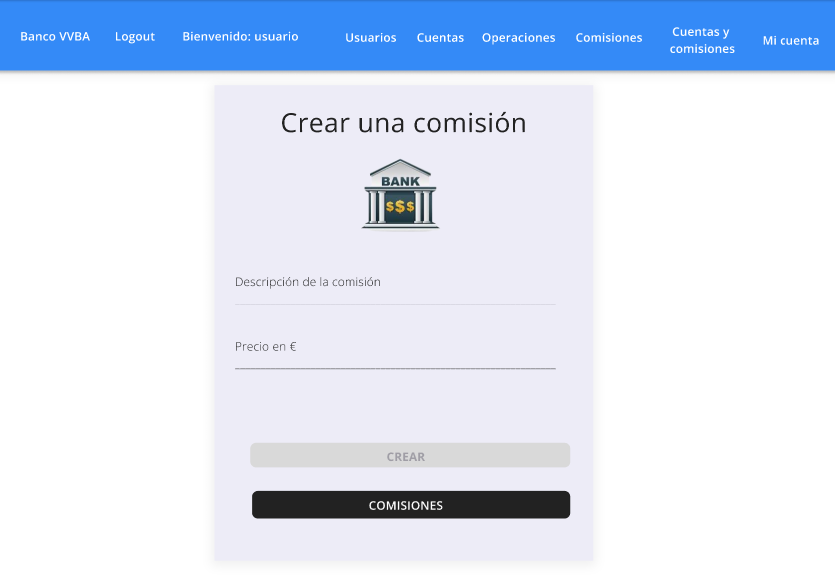


Ilustración 25. Creacion comisión

Podemos adjuntar una comisión a una cuenta, para ello deberemos elegir el tipo de cuenta y el tipo de la comisión que se vaya a añadir.

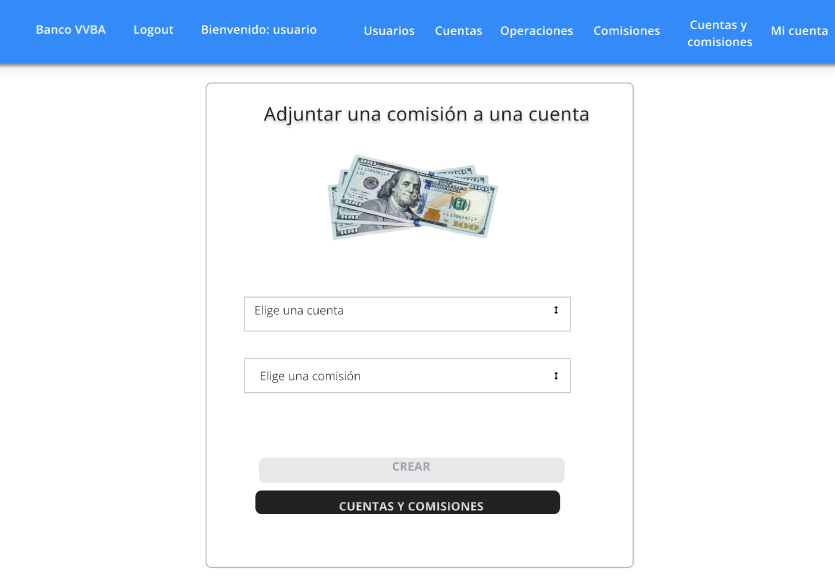


Ilustración 26. Creacion comision en cuenta

Por último, el propio usuario administrador que esté logeado, también es un usuario de la aplicación, por lo que, también dispone de su cuenta y es capaz de hacer las mismas operaciones que un usuario normal.

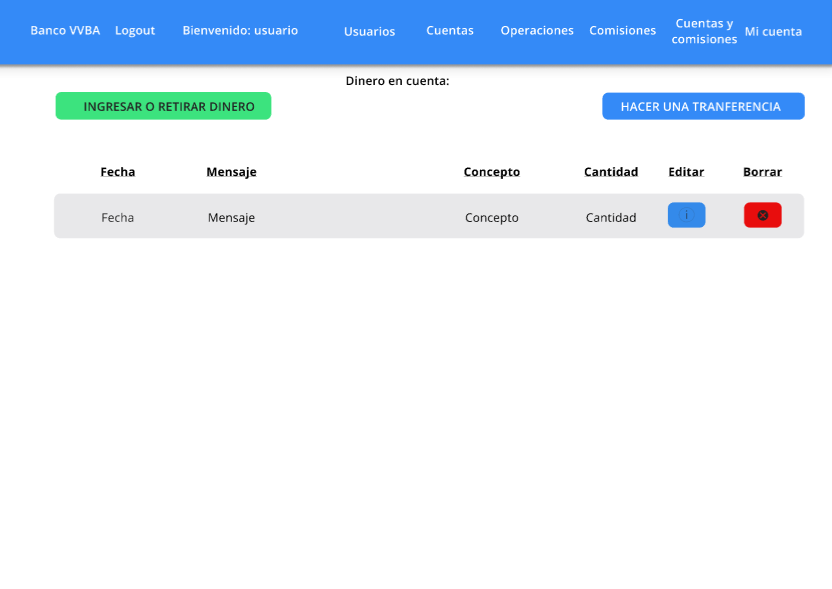


Ilustración 27. Cuenta administrador

* 1. Fase de implementación

Dentro de la fase de implementación procederemos a explicar todos los subapartados que se han llevado a cabo.

* + 1. Herramientas y servicios.

Para un correcto desarrollo de la aplicación se han seleccionado las siguientes herramientas para desarrollo y gestión.

* Git: esta herramienta la utilizamos para un control del versionado del proyecto en sí. Es una herramienta estandarizada la cual usaremos en concreto Github que cuenta con todas las funcionalidades buscadas.
* Visual Studio Code: es el IDE seleccionado para la realización del Front-End de la aplicación, ya hemos utilizado múltiples veces este IDE en la carrera lo cual nos ayuda a tener un trabajo más fácil y eficiente. Además, disponemos de plugins los cuales nos ayudan al desarrollo y a una fácil comprensión de las carpetas internas del proyecto.
* Visual Studio: se va a utilizar este IDE para la realización del Back-End de nuestro proyecto, dado que queremos usar ASP .NET Core este es el IDE recomendado para su uso, aunque no hemos usado este IDE en la universidad, si lo he usado anteriormente en proyectos personales por lo que no gastaremos tiempo en aprender cómo utilizarlo.
* SQL Server: se ha elegido SQL Server, utilizado con SQL Server Management Studio como forma de acceso. Esta base de datos relacional permitirá almacenar todos los datos necesarios de la aplicación con un fácil acceso y manejo.
* Postman: esta herramienta nos permite hacer llamadas HTTP a la aplicación.

Todas las herramientas mencionadas anteriormente no han tenido una fase de aprendizaje previa ya que, se conocía su funcionalidad.

En los siguientes dos apartados, se va a explicar cómo ha sido el desarrollo en cada una de las tecnologías utilizadas, su arquitectura, layers, ciclo de vida y estructura de la aplicación.

* + 1. Desarrollo en ASP .NET Core.

Una vez seleccionada la base para el desarrollo del proyecto, se ha comenzado con una investigación, ya que, aunque ya había una experiencia con el IDE no existía la misma experiencia con la tecnología a utilizar, por ello, primero se investigó como realizar la conexión a la base de datos (DB), como se han de almacenar objetos en dicha DB, su persistencia y cómo usar el patrón MVC el cual está dirigido ASP .NET Core.

Para ello, primero explicaremos que son los paquetes Nugget, esto es un archivo ZIP con extensión .nupkg que contiene código compilado en su interior, con una funcionalidad parecía a Dynamic-Link Library (DLL), este paquete se descarga mediante una herramienta interna de Visual Studio y se descomprime dentro del proyecto, el cual nos permite utilizar las funcionalidades de la librería externa.

Aparte de los paquetes Nugget esenciales de los que ya dispondremos cuando creamos un proyecto ASP .NET Core, necesitaremos de otros tres paquetes externos para un correcto funcionamiento de nuestra aplicación, estos tres paquetes son: Swashbuckle.AspNetCore , Microsoft.EntityFrameworkCore.tools y Microsoft.EntityFrameworkCore.SqlServer.

* Swashbuckle.AspNetCore: este paquete Nugget nos proporciona herramientas para poder documentar la API usada mediante Swagger. Esto es una herramienta que nos muestra todos los endpoints con información de como funciona, atributos y resultados de respuesta de nuestra API.
* Microsoft.EntityFrameworkCore.tools: este es un paquete necesario para poder utilizar el siguiente paquete, sin ello no podríamos tener un correcto funcionamiento del paquete.
* Microsoft.EntityFrameworkCore.SqlServer: este es el paquete esencial por el cual, con las herramientas que trae este paquete, podremos conectarnos fácilmente a la DB que utilizamos en nuestro proyecto.

En el siguiente apartado se profundizará más a fondo en la arquitectura de esta tecnología.

* + - 1. Arquitectura de ASP .NET Core

La arquitectura seguida por esta tecnología funciona como una aplicación monolítica, ya que, toda la lógica está contenida en un solo proyecto, se compila en una solución e implementa como una unidad.

Contiene todo el comportamiento de la aplicación como se muestra en la siguiente ilustración.

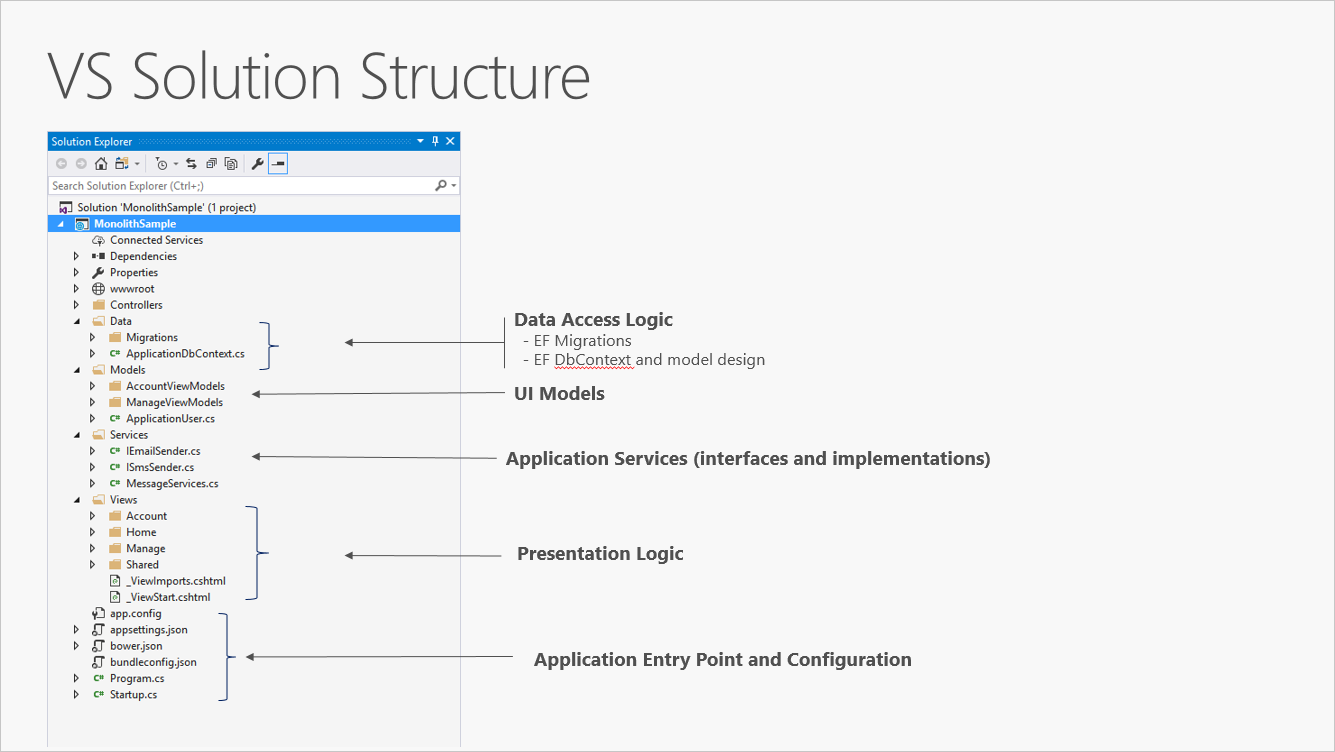


Ilustración 28. Arquitectura ASP .NET Core

En la ilustración anterior se pueden pueden distinguir los siguientes elementos:

* UI Models: esta parte hace referencia a los modelos de la aplicación, estos modelos una vez se haga una migración de la DB se convertirán en las tablas.
* Data Access Logic: podemos diferenciar las migraciones de la aplicación y el contexto de la DB, este contexto contiene toda la información y elementos de nuestra DB.
* Application Services: Contiene la implementación de la lógica de nuestra aplicación.
* Entry Point and Configuration: estos son archivos de configuración que utilizaremos para configurar nuestro proyecto.

El elemento de Presentation Logic, no se ha explicado ya que en nuestro proyecto utilizaremos Angular para la parte del Front-End y no utilizamos esta parte.

* + - 1. Layers.

Se va a proceder a explicar los layers que se han utilizado en nuestro proyecto como se ve en la siguiente imagen.

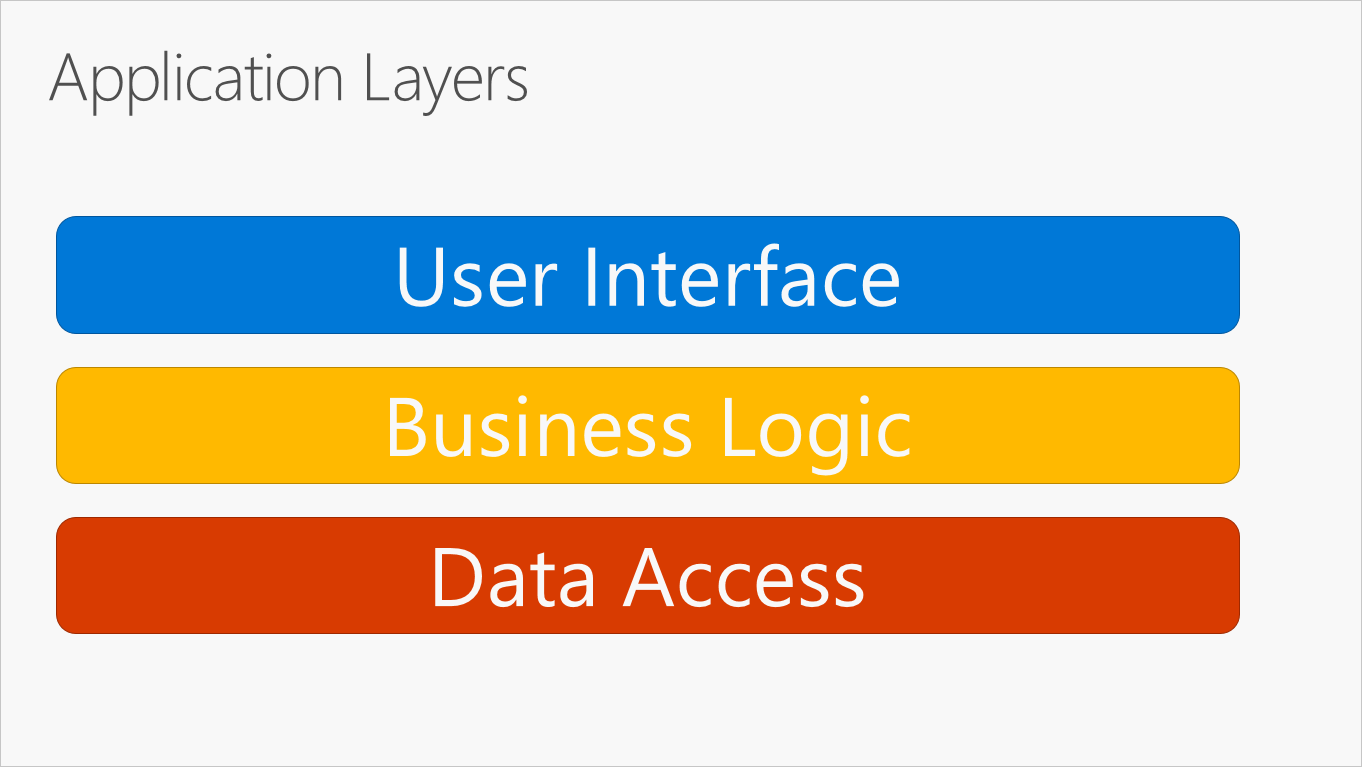


Ilustración 29. Layers ASP .NET Core

En nuestro proyecto hemos utilizado diferentes layers en el funcionamiento de la lógica de negocio, diferenciaremos tres layers esenciales: Controller, Service y Repository respectivamente.

* Controller: esta es la primera capa de nuestra aplicación donde llegarán los datos del Front-End, cabe destacar que los datos que viajan entre las dos tecnologías viajan en formato JSON, este layer se encarga de recibir datos y enviárselos al Service.
* Service: esta es la capa intermedia, tiene una importante función, realizar la lógica (en caso de que sea necesario) dentro del código y traspasar los datos a la última capa, la de Repository.
* Repository: la última capa de la aplicación, esta capa es muy importante, ya que, es la encargada de conectar con la DB, actualizarla y devolver los datos necesarios hacia las capas posteriores.

Estas son las capas en las cuales el proyecto se ha basado, en el siguiente punto explicaremos el ciclo de vida de ASP .NET Core.

* + - 1. Ciclo de vida

Para comprender nuestra aplicación vamos a hacer un ciclo de vida de los datos que son utilizados en el proyecto.

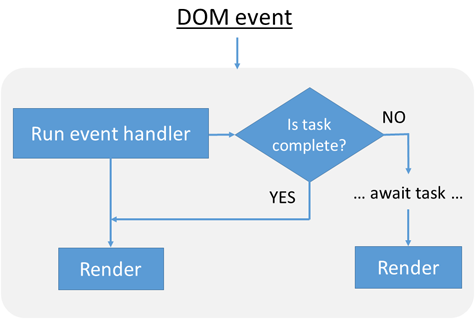


Ilustración 30. Ciclo de vida ASP .NET Core

Como se muestra en la ilustración anterior, ASP .NET Core una vez arrancado, queda a la espera de algún cambio en el DOM, al haber una llamada a nuestra aplicación, se arranca un manejador de eventos, y sigue el proceso de la imagen, al trabajar en formato asíncrono, tenemos que añadir await a las tareas para que devuelva los datos de forma segura, este proceso vuelve a repetirse hasta que para la aplicación.

A continuación, vamos a ver la estructura de nuestro proyecto.

* + - 1. Estructuración del proyecto.

La estructuración del proyecto es similar a la ilustración 28, ya que, hay archivos que no pueden ser borrados debido a que son necesarios para el arranque de la aplicación, como se mostrará en la siguiente imagen.

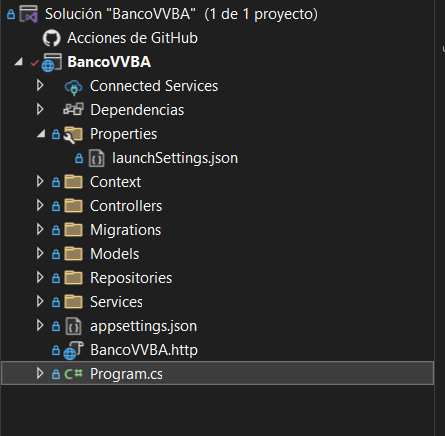


Ilustración 31. Estructura proyecto ASP .NET Core

Como podemos observar en la ilustración 31 y como hemos ido comentando anteriormente, el proyecto ha sido dividido en diferentes carpetas.

* Controller, Service y Repository: estos son las capas en las que hemos dividido la lógica de negocio.
* Models: son los modelos, en los que se basa el contexto para crear las tablas de la DB.
* Configuración: los últimos tres archivos son los archivos de configuración de la aplicación en estos esta la conexión a la base de datos y la configuración de arranque de la aplicación.
* Migrations: contiene las migraciones de nuestra DB con los cambios que haya habido en esta.
* Context: contiene el archivo de contexto de la base de datos donde hace referencia a los modelos para crear las tablas de nuestra DB.

Se procederá en el siguiente punto a ver el estado de las pruebas.

* + - 1. Pruebas

Debido a la magnitud del proyecto no hay pruebas unitarias de ninguno de los dos tipos, sin embargo, esto no quiere decir que la aplicación no haya sido probada, si no, que las pruebas no han sido automatizadas, pero si se han hecho de forma manual y han sido bastante intensas a lo largo del desarrollo de la aplicación.

Cabe destacar que, aunque se hubieran hecho pruebas automatizadas, la aplicación hubiese tenido que probarse de forma manual de todas formas. También cabe destacar que en la parte del Front-End desarrollado en angular están todos los datos probados para que lleguen en un formato adecuado y de forma para un correcto funcionamiento de la aplicación en la parte del Back-End.

* + 1. Desarrollo en Angular

Como hemos hablado anteriormente, la parte de la aplicación correspondiente al Front-End ha sido desarrollada en Angular, esta ha sido una tecnología totalmente nueva y se han utilizado varios tutoriales para aprender sobre su uso, concretamente uno interno que nos proporciona la página oficial de Angular llamado “Tour of Heroes”, siguiendo este tutorial obtenemos una aproximación bastante buena sobre cómo funciona esta tecnología.

Primero hay que explicar el funcionamiento de esta tecnología, para no extendernos mucho, vamos a explicar por encima como funciona. Primero creas el proyecto y se crean varias carpetas, nuestro código del proyecto vendrá definido en la carpeta que se llame como nuestro proyecto, una vez ingresado a esta carpeta vienen diferentes archivos.

Como podemos ver en la siguiente ilustración vienen los archivos correspondientes al proyecto app (es como hemos llamado nuestro proyecto) vamos a hacer una breve explicación de cada uno de estos archivos.

* **app.html:** este html funciona como el html global el cual dentro importamos las diferentes vistas de los componentes que creemos para nuestra aplicación.
* **app.css:** es el css que se le impondrá a la aplicación de forma global.
* **app.spec.ts:** autogenerado por angular.
* **app.ts:** esta es el archivo interno, es un archivo typescript, parecido a javascript, hace referencia al html del mismo componente.
* **app.config.ts:** archivo de configuración donde se importarán librerías y elementos externos a nuestra aplicación.
* **app.routes.ts:** este archivo es muy importante ya que se definirán las rutas que utilicen nuestros componentes en la aplicación.

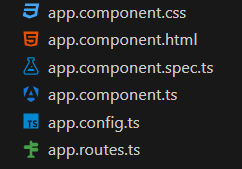


Ilustración 32. Ejemplo Creación proyecto Angular

Cada vez que creemos un componente en Angular se crearan los archivos css, html, spec.ts y ts, como hemos visto anteriormente haciendo referencia a los archivos de su componente y trabajaremos importando estos componentes creados al componente proyecto llamado app.

Para concretar un poco más a fondo se ha de mencionar que se ha utilizado la última versión, Angular 17, el cual se hacen importaciones de componentes con el formato standalone. Ahora se va a proceder a explicar cómo anteriormente hemos hecho con ASP .NET Core la arquitectura, los layers, ciclo de vida, estructuración del proyecto y pruebas.

* + - 1. Arquitectura de Angular

Angular utiliza una arquitectura llamada “Clean Arquitecture” la cual su función es centralizar su solución en el dominio, esto significa, centrarse en su capa de negocio y no quedarse en un concepto técnico. Para ello utilizamos las siguientes partes:

* **Entidades:** Objetos de la aplicación esta capa es como la definición de las tablas en ASP .NET Core, tiene que haber una semejanza entre los modelos usados en el Back-End y las entidades del Front-End. Ningún cambio en la navegación de la aplicación ni utilización de esta debería someter un cambio a esta capa.
* **Casos de uso:** Esta capa contiene reglas comerciales y lógica de la aplicación, estos organizan el flujo de datos desde las entidades y hacia estas mismas. Se espera que los cambios en el funcionamiento de la aplicación generen cambios en la esta capa.

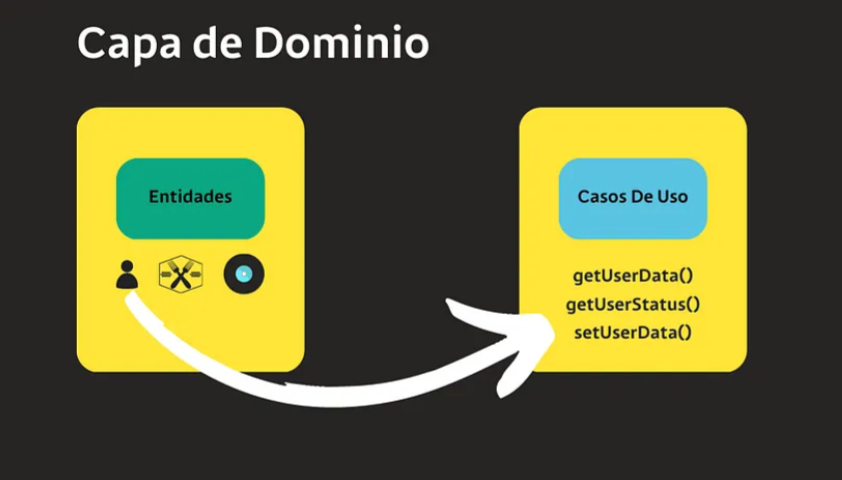


Ilustración 33. Capa Dominio Angular

* **Capa de infraestructura:** Contiene los siguientes elementos:
  + **Driven adapters:** son los adaptadores que nos permiten tener una conexión con el exterior.
  + **Entry points:** es la capa en la que se exponen los servicios, es común encontrarla en soluciones que conecten con Back-End.
  + **Helpers:** capa dedicada a ayudar a las demás con transformaciones de datos, operaciones o funciones útiles.
* **Capa de presentación:** implementa todo lo relacionado con elementos visuales con los que el usuario trabajará.

En el siguiente apartado vamos a explicar los layers que se ha usado en el proyecto.

* + - 1. Layers en Angular

En nuestro proyecto se han utilizado diferentes layers, se va a explicar uno por uno cuales han sido los esenciales: Modelos, Components y Services.

* **Modelos:** esta capa genera todas las entidades necesarias para el uso de nuestro proyecto, como hemos mencionado anteriormente debe de haber una correspondencia entre los modelos del Front-End y el Back-End.
* **Components:** donde vendrá la mayor parte de la lógica de nuestra aplicación, se crean los componentes que compondrán las vistas y el funcionamiento de nuestro proyecto. Cabe destacar que dentro de estos componentes utilizamos un Guard, es un comprobador para que gente no logeada o que esté sin permisos no pueda acceder a la aplicación.
* **Services:** esta capa es la encargada de hacer todas las llamadas a la API que utilizamos en el Back-end, transmitir y recibir los datos. Como ya hemos mencionado anteriormente en la memoria, estos datos se transmiten en formato JSON.

Mostramos en la ilustración posterior como resultaría la utilización de estas capas dentro de la carpeta app.

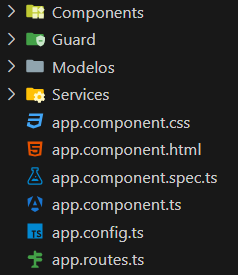


Ilustración 34. Layers Angular

* + - 1. Ciclo de vida

Angular tiene un ciclo de vida complejo así que explicaremos las 3 partes más usadas en nuestro proyecto en el ciclo de vida: constructor, ngOnInit y ngAfterViewInit.

* constructor: en esta parte se definen los servicios y componentes externos necesarios para el buen uso del propio componente.
* ngOnInit: se ejecuta una vez el componente se ha inicializado, este es el evento utilizado para generar los datos iniciales de los componentes.
* ngAfterViewInit: se ejecuta cuando el componente se ha incializado al completo con sus vistas primarias y secundarias.

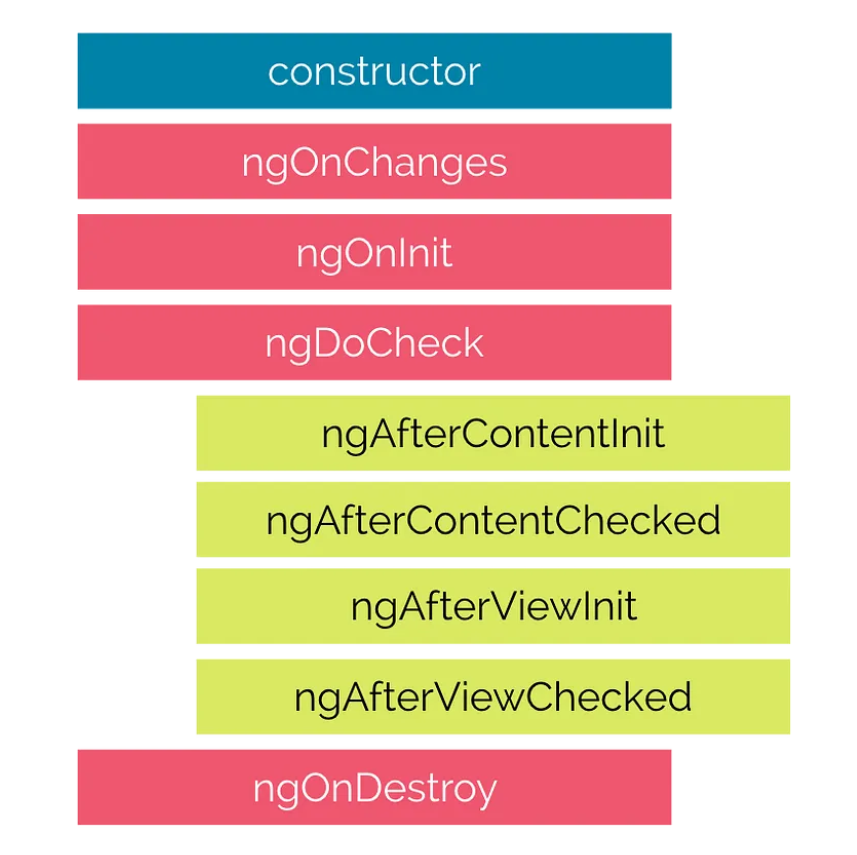


Ilustración 35. Ciclo de vida Angular

Como se puede observar en la imagen anterior hay más partes, pero dado que no las hemos utilizado en nuestro proyecto no van a ser explicadas.

* + - 1. Estructuración del proyecto

La estructuración del proyecto es clara dentro de la carpeta app, es donde hemos generado todos los componentes necesarios de la aplicación y utilizados en el componente app que es el global de la aplicación. El resto de archivos son autogenerados y completados por el propio Angular así que no se han explicado ni centrado en estos, como podemos ver en la siguiente ilustración.

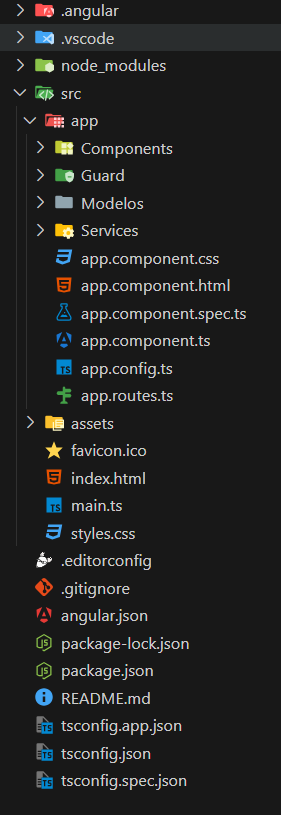


Ilustración 36. Estructura proyecto Angular

Como hemos mencionado anteriormente, se trabaja mediante componentes importados unos a otros. El funcionamiento del componente es el siguiente.

El html del componente contiene la vista del usuario final, complementado por el css y el ts, del que saca los valores necesarios a ser llamados por nuestra vista. Si se hace algúna navegación o alteración en la vista está llama al componente typescript asociado y hace la acción correspondiente, en el caso de que se necesiten valores del Back-End, el archivo typesctipt llama al servicio necesario, este comunica con el Back-End y se transfieren los valores necesarios.

* + - 1. Pruebas

Tenemos el mismo caso que anteriormente en ASP .NET Core, debido a la magnitud del proyecto, no hemos podido implementar pruebas unitarias de ningún tipo, sin embargo, esto no significa que no esté probado el proyecto, ya que, se ha ido probando a lo largo de todo su desarrollo mediante varias personas en funcionalidad y riesgo.

Cabe destacar que, por el objetivo del proyecto, ha tenido que haber siempre pruebas humanas dado que una de los objetivos de este proyecto es que el profesor Juan Antonio Andrés Lalueza, va a hacer uso de este proyecto para impartir clases en su asignatura.

1. Resultados experimentales.

En general, enfocándonos en el resultado práctico, en el que implicaba encadenar dos tecnologías nunca usadas, y usarlas a la vez, ha sido muy satisfactoria por los siguientes motivos:

* **Conexión:** en el principio del proyecto había demasiadas dudas, ya que, al ser tecnologías nunca usadas anteriormente, era probable que fueran poco compatibles o que la dificultad fuera demasiado alta para su uso, sin embargo, una vez terminado el proyecto se puede decir que, aunque ha habido momentos de dificultad, en general se ha podido trabajar muy bien y se han conectado las dos tecnologías de una manera fluida y precisa.
* **Requisitos:** se han completado todos los requisitos que se han mencionado en la memoria por lo que, ha sido un proyecto satisfactorio con buenas sensaciones.
* **Aprendizaje:** las tecnologías usadas en nuestra aplicación son tecnologías consideradas de las más importantes hoy en día por lo que su aprendizaje además de haber sido muy divertido ha sido una parte importante en el proyecto.

Por último, les vamos a mostrar en las siguientes ilustraciones como han quedado las vistas finales de nuestra aplicación Banco VVBA.

Ventana de login

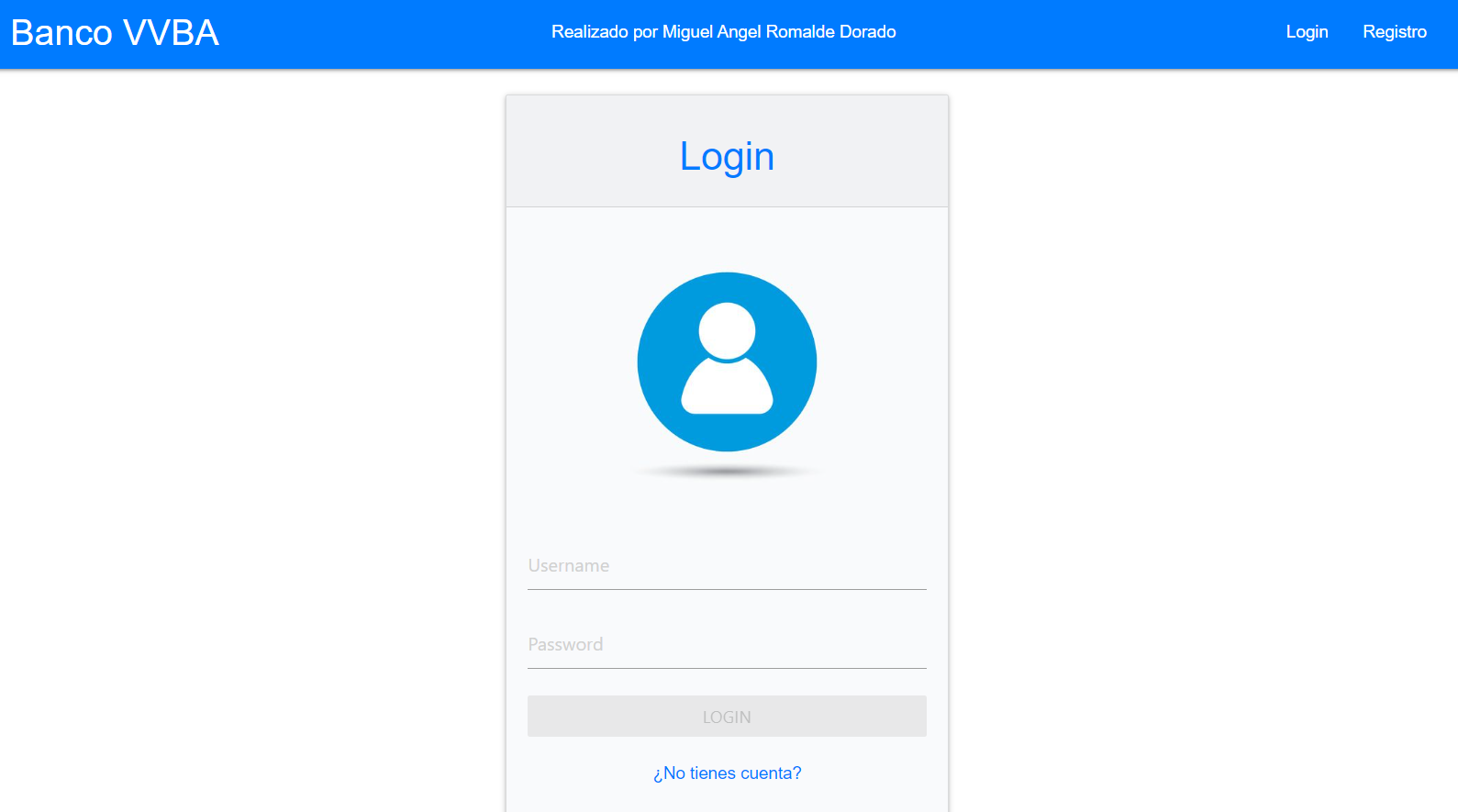


Ilustración 37.Login final

Registro

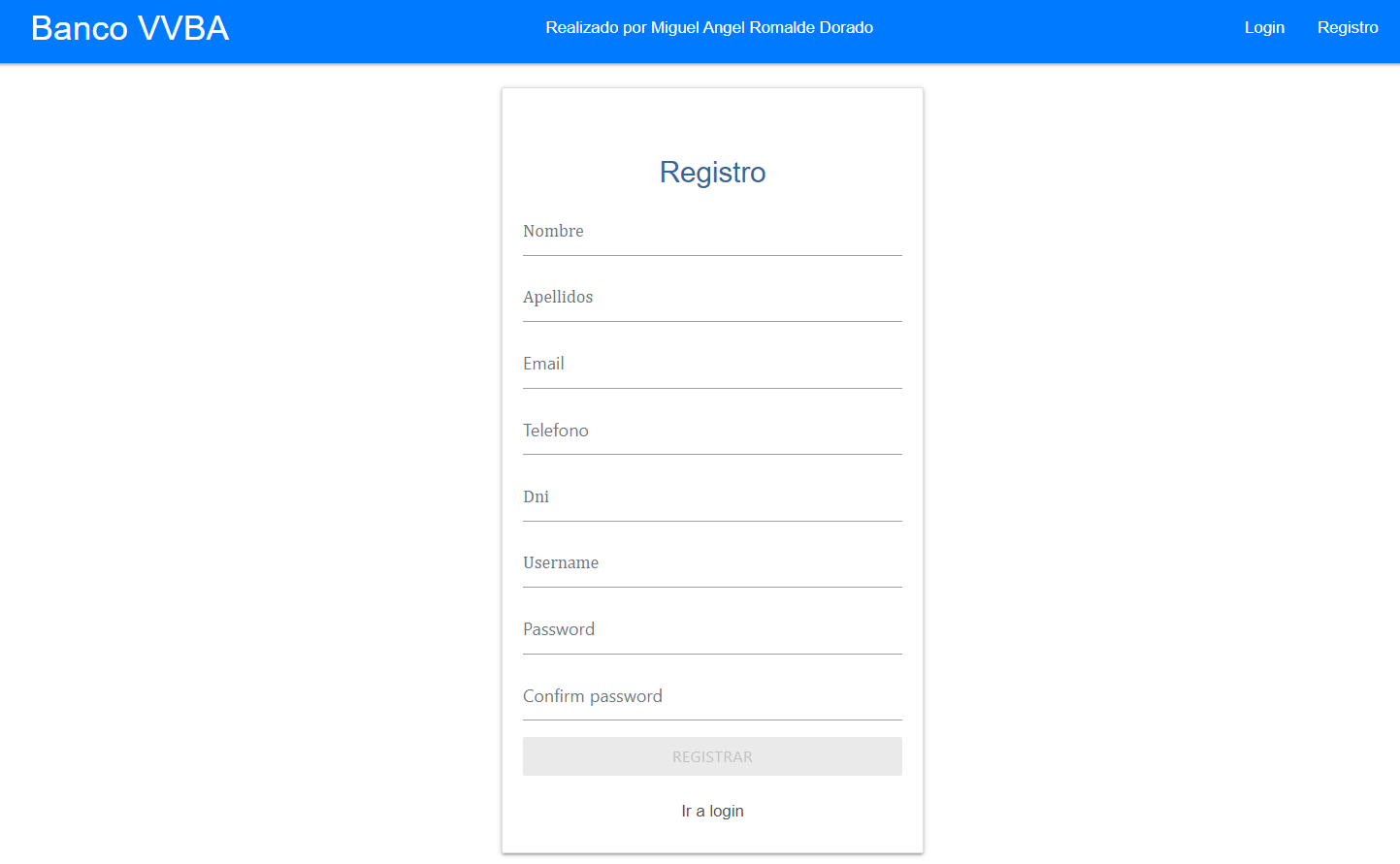


Ilustración 38. Registro final

Usuarios

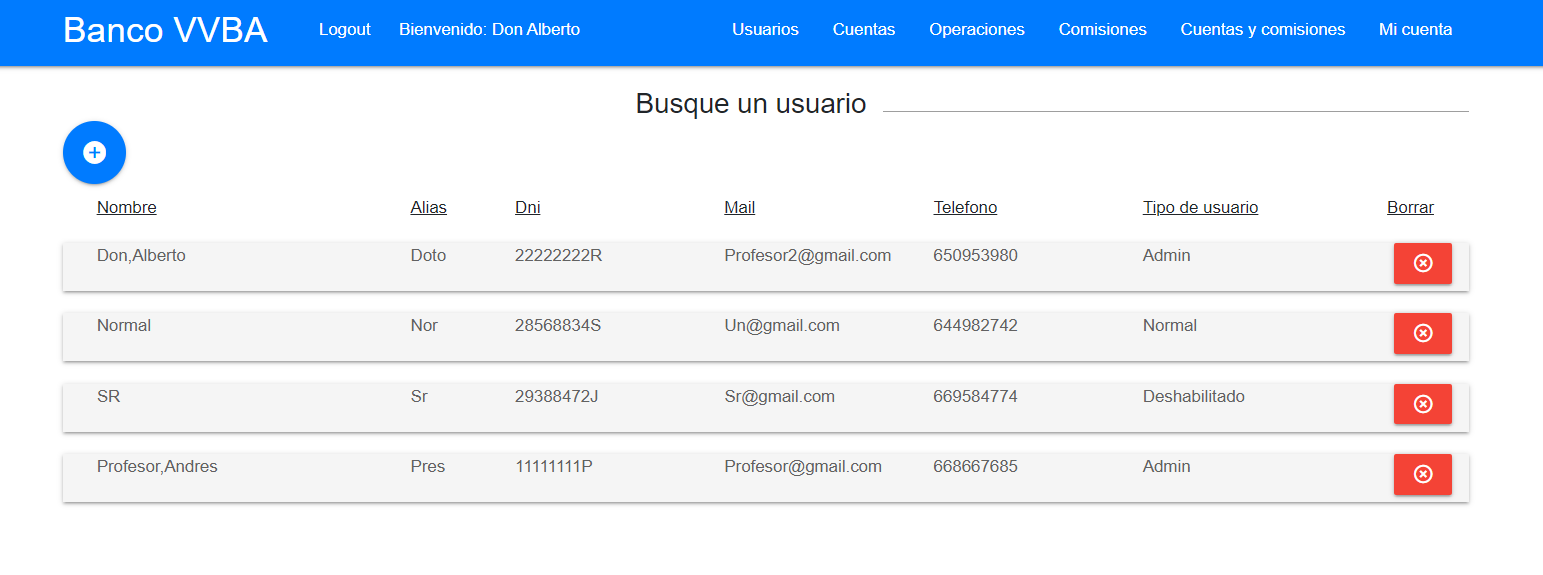


Ilustración 39. Usuarios final

Creación/edit del usuario por parte del administrador

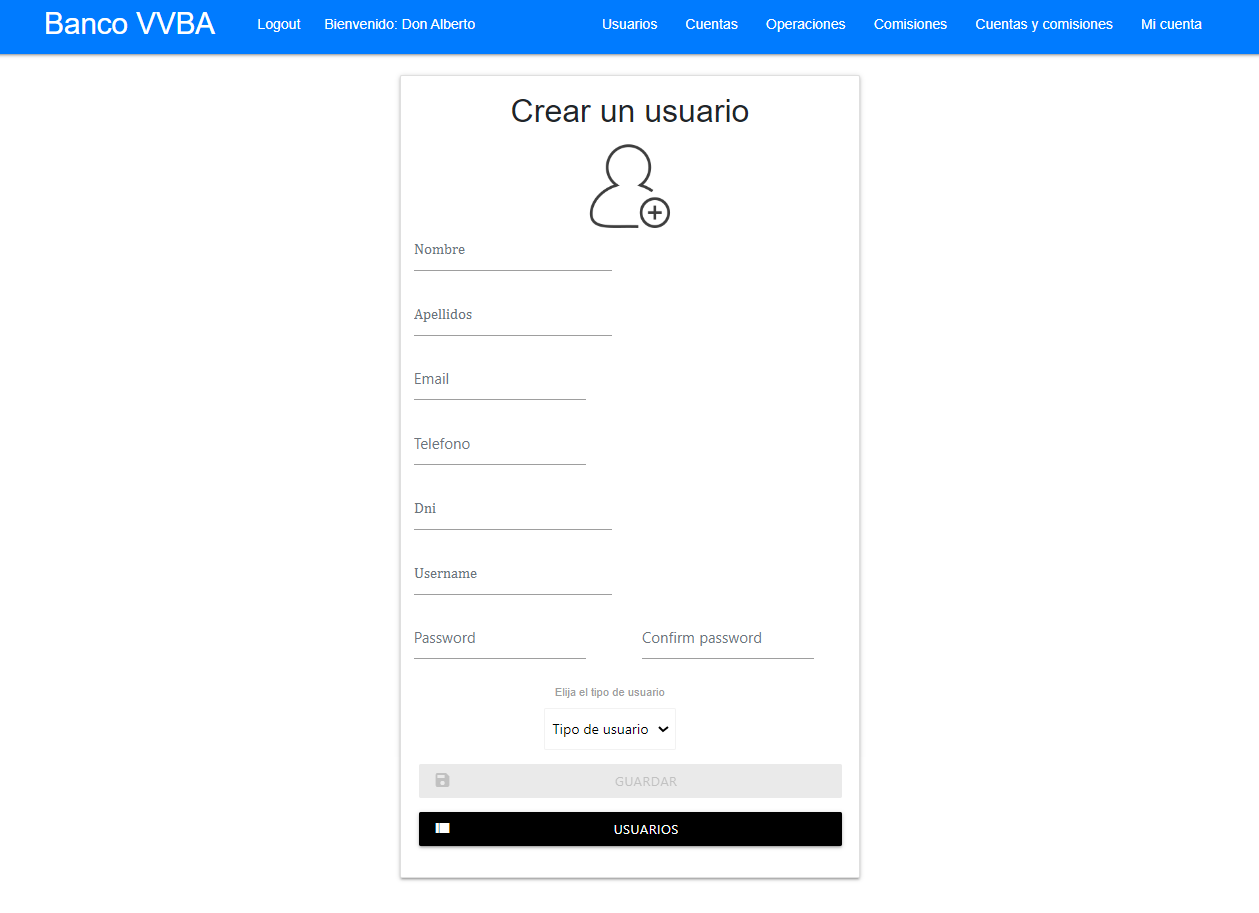


Ilustración 40. Creacion usuario

Cuentas

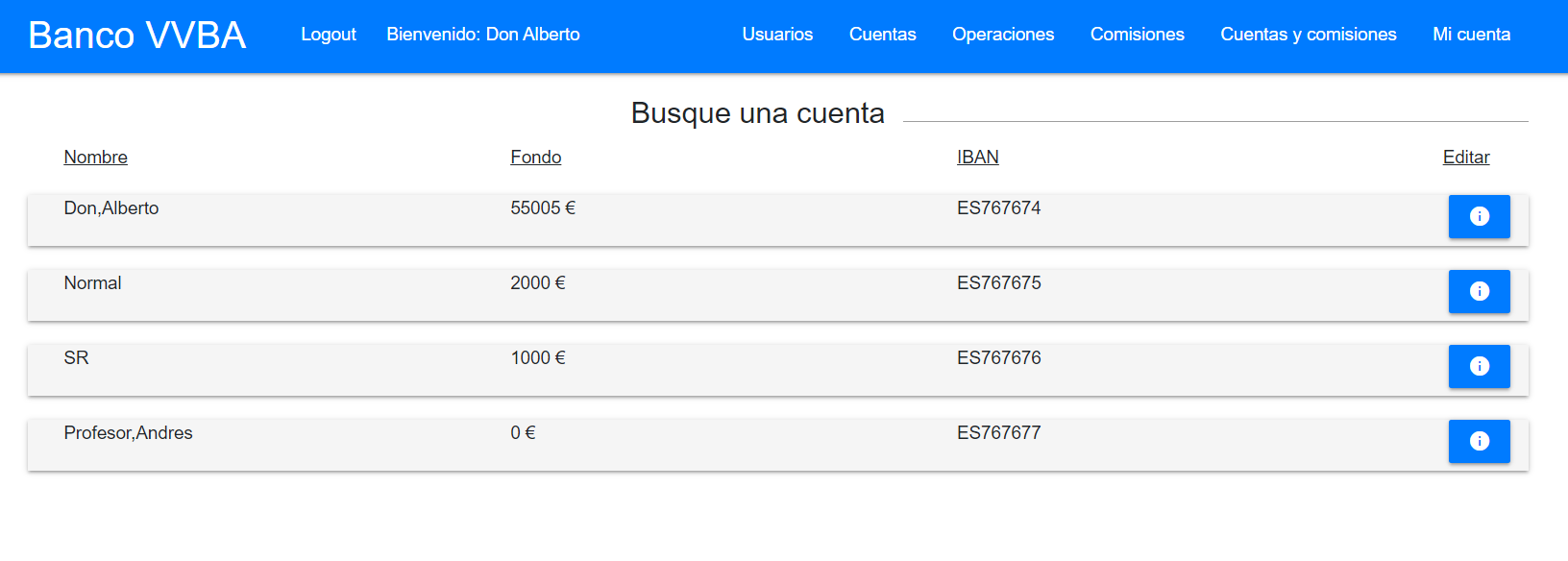


Ilustración 41. Cuentas final

Editar cuenta



Ilustración 42. Editar cuenta

Operaciones

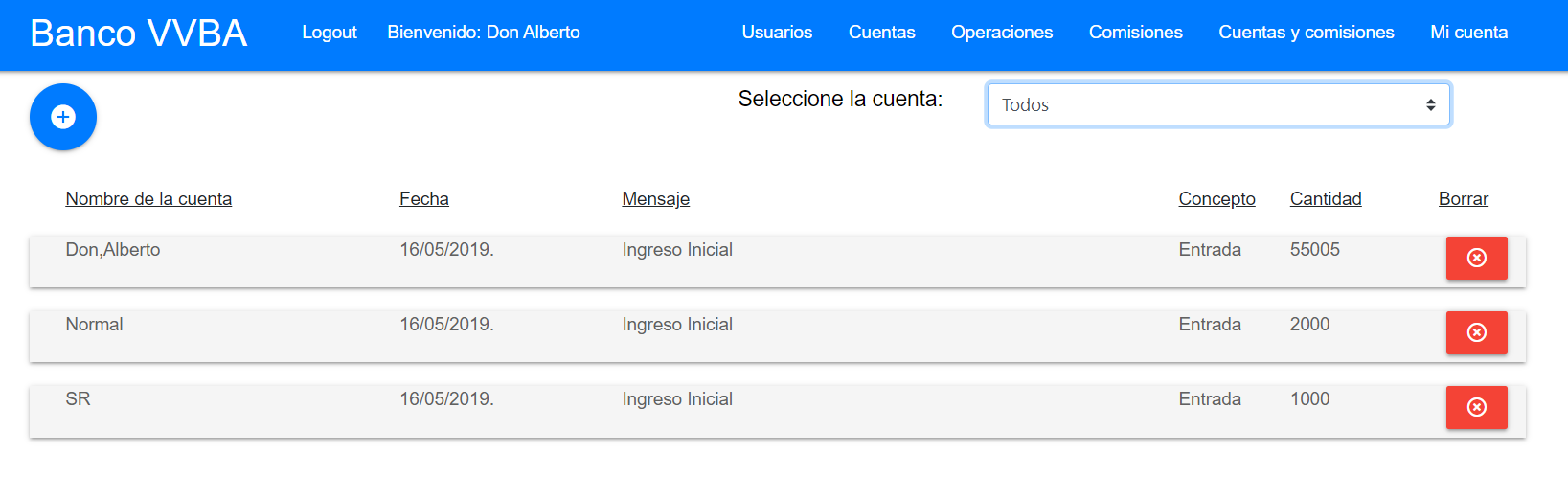


Ilustración 43. Operaciones final

Editar/crear operaciones

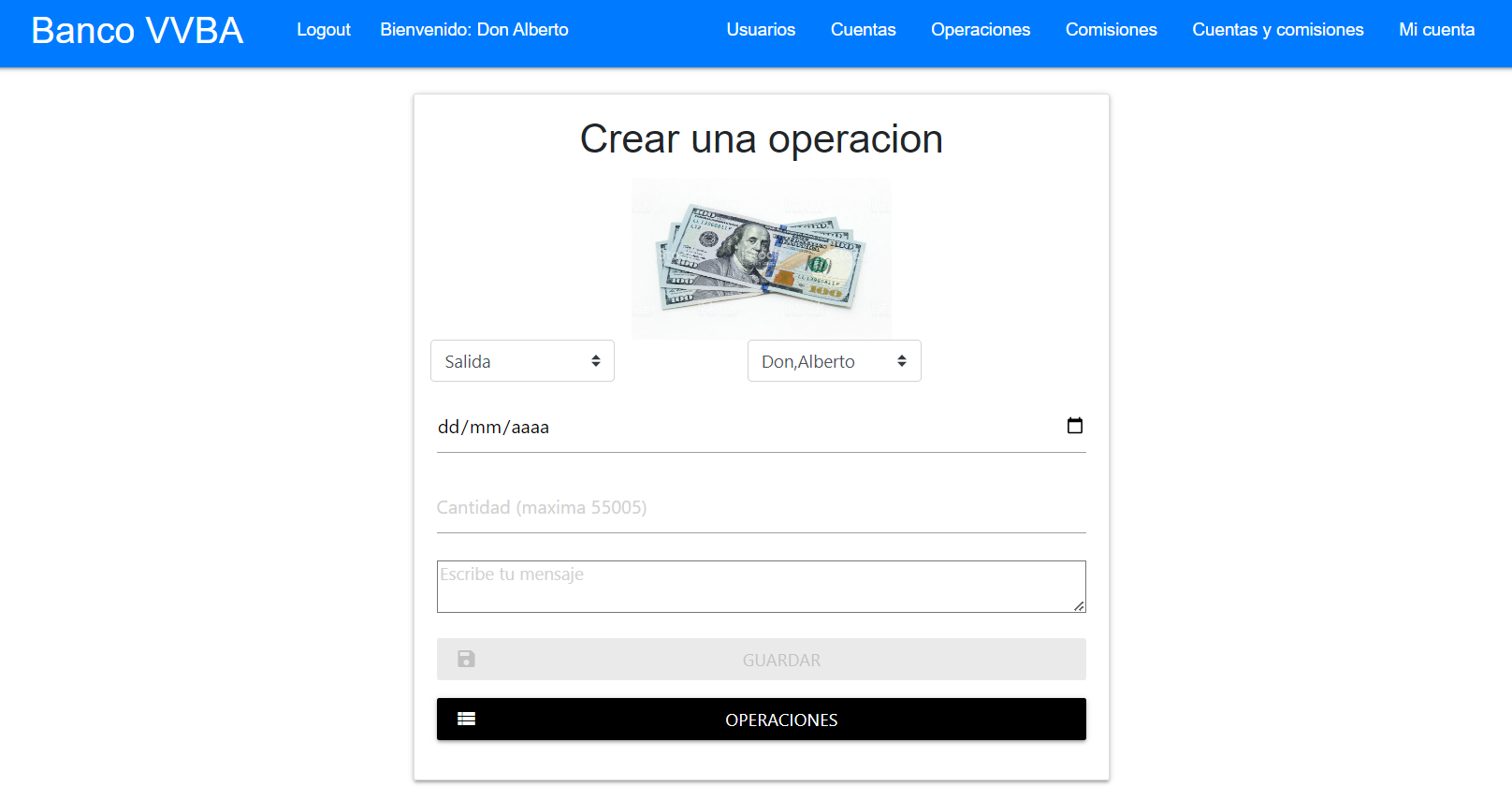


Ilustración 44. Editar operaciones final

Comisiones

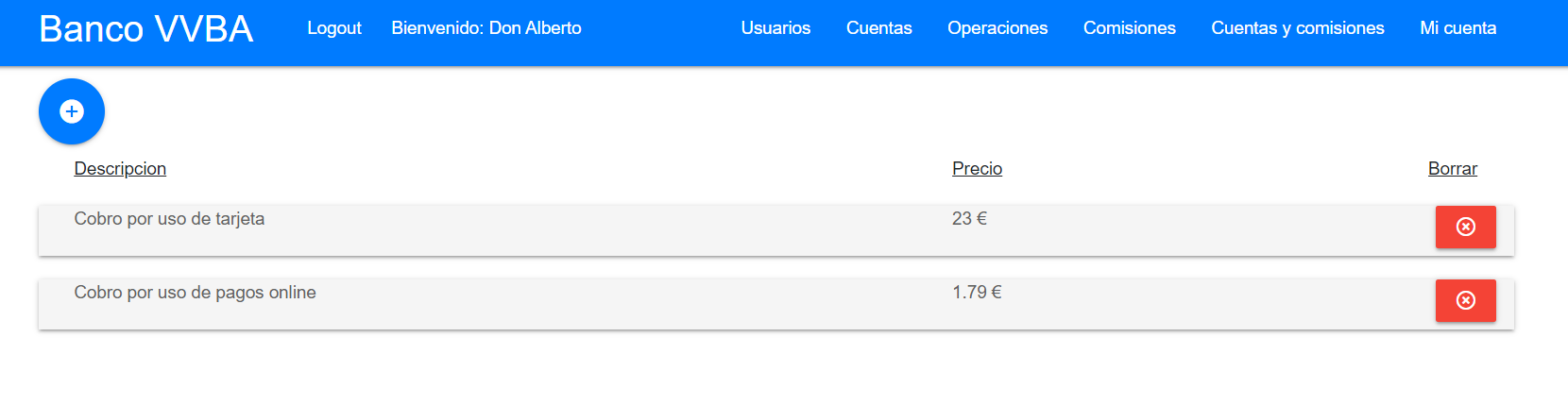


Ilustración 45. Comisiones final

Creacion/editar comisiones

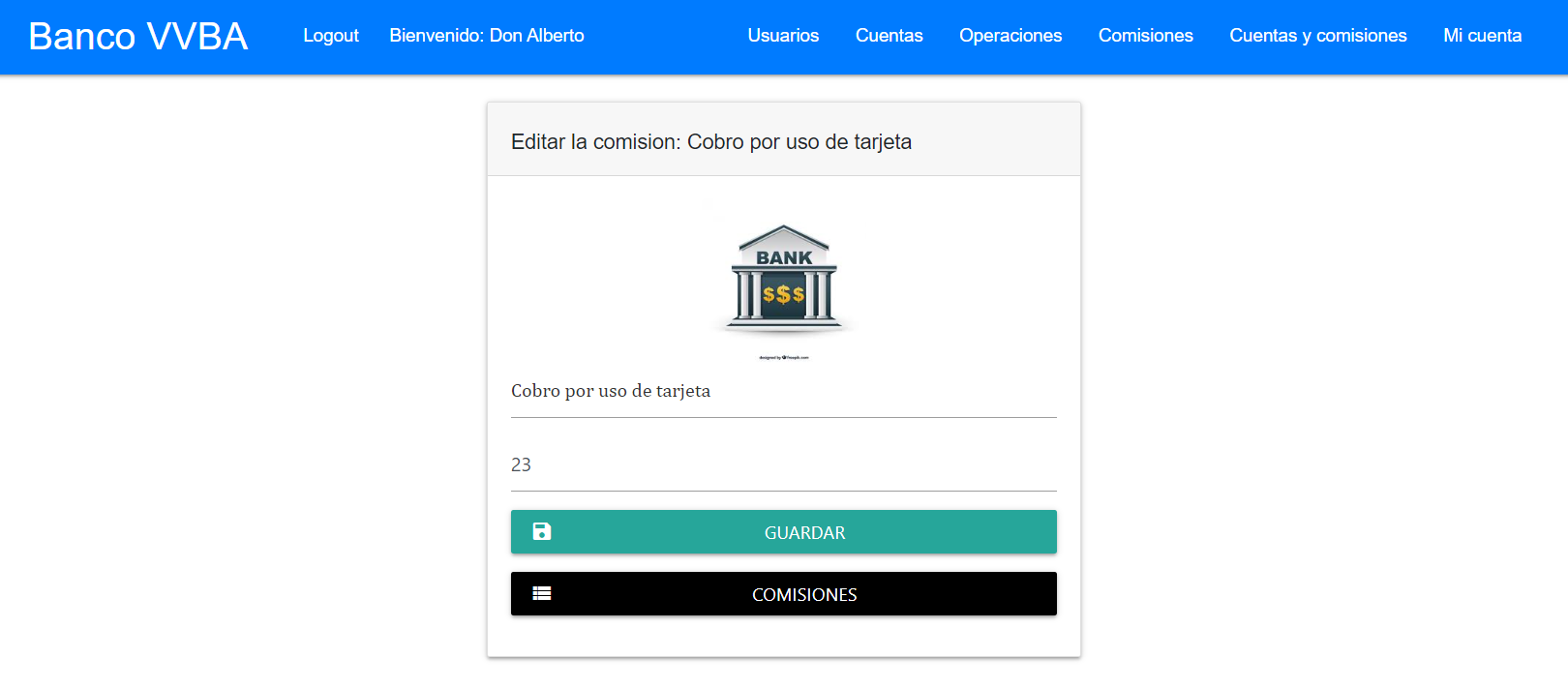


Ilustración 46. Editar comisiones

Cuentas comisiones

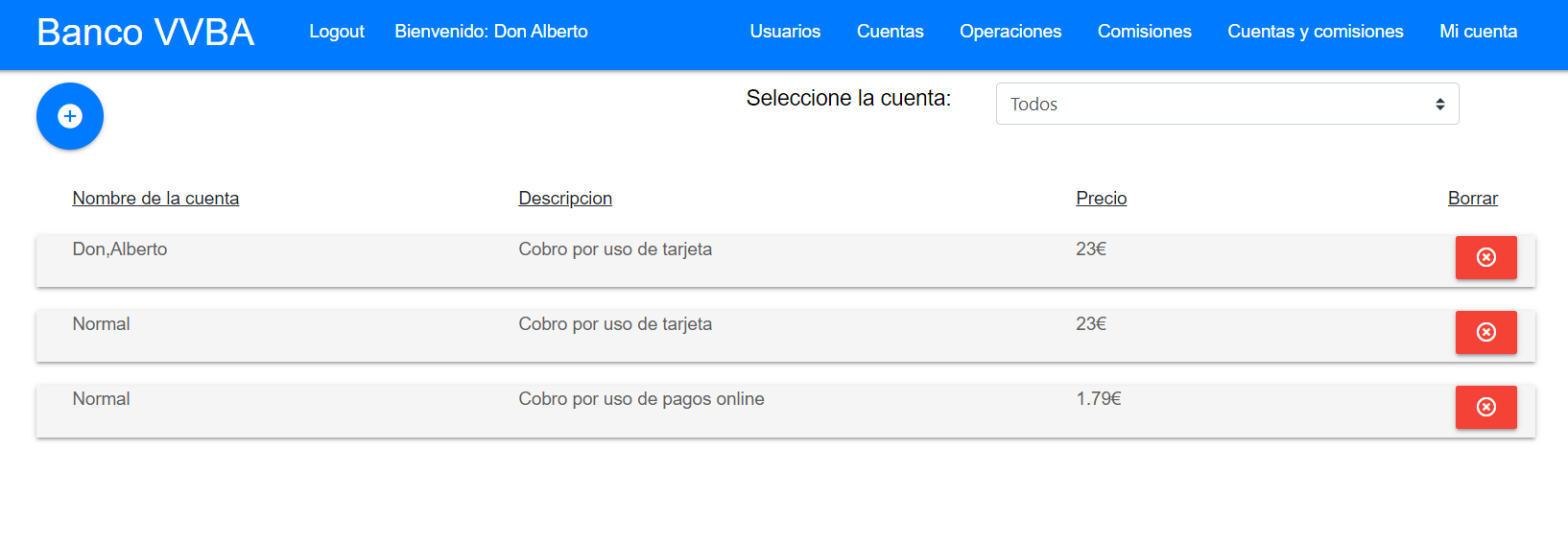


Ilustración 47. Cuentas comisiones final

Creación/editar cuentas comisiones

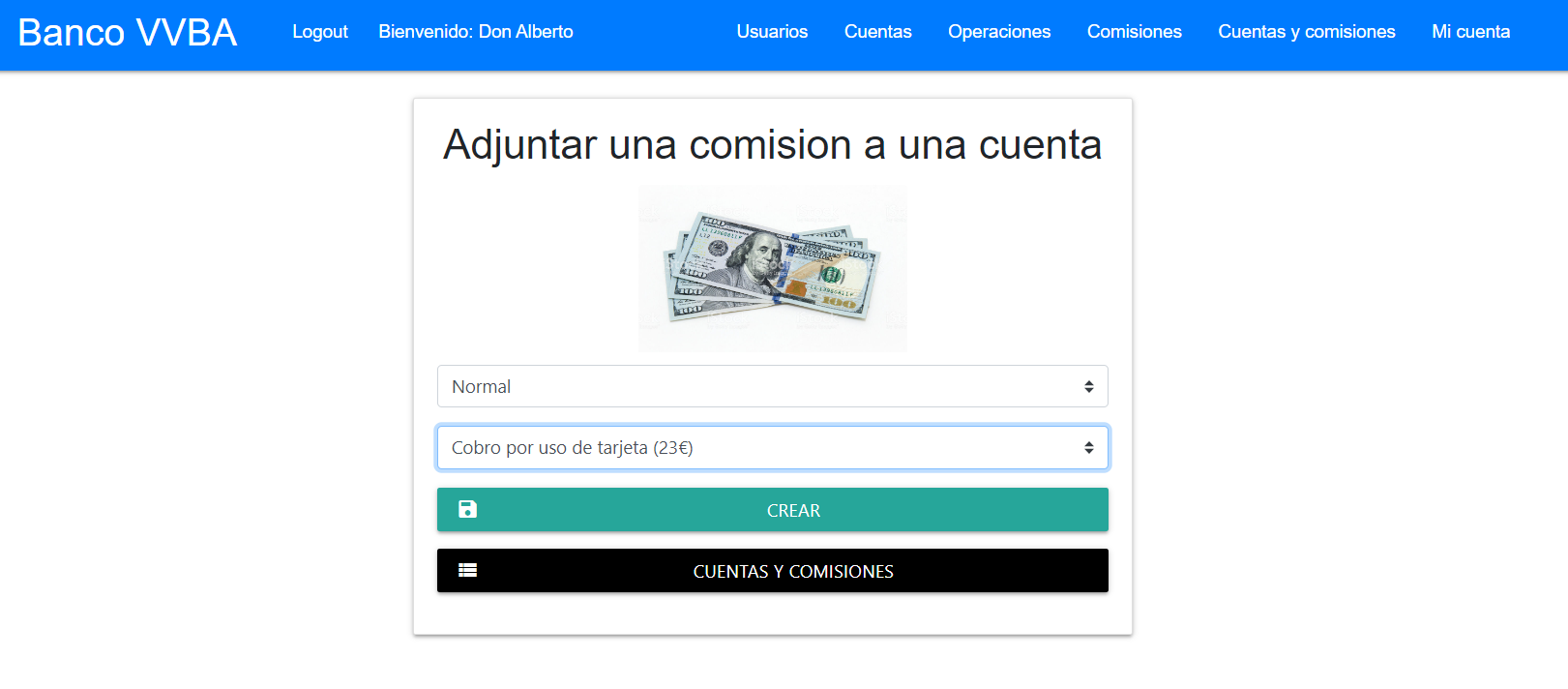


Ilustración 48. Editar cuentas comisiones

Mi cuenta

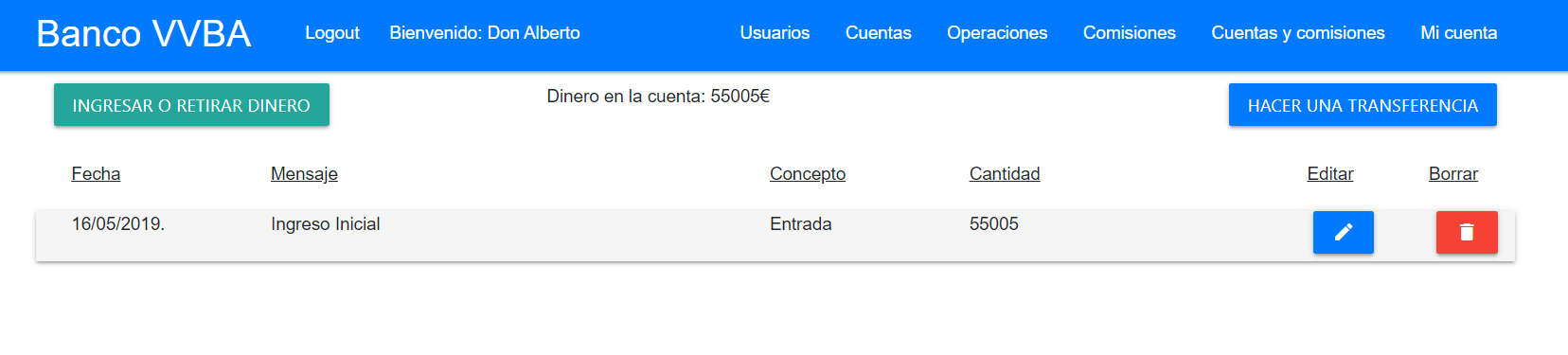


Ilustración 49. Mi cuenta final

Ingresar/retirar dinero en mi cuenta

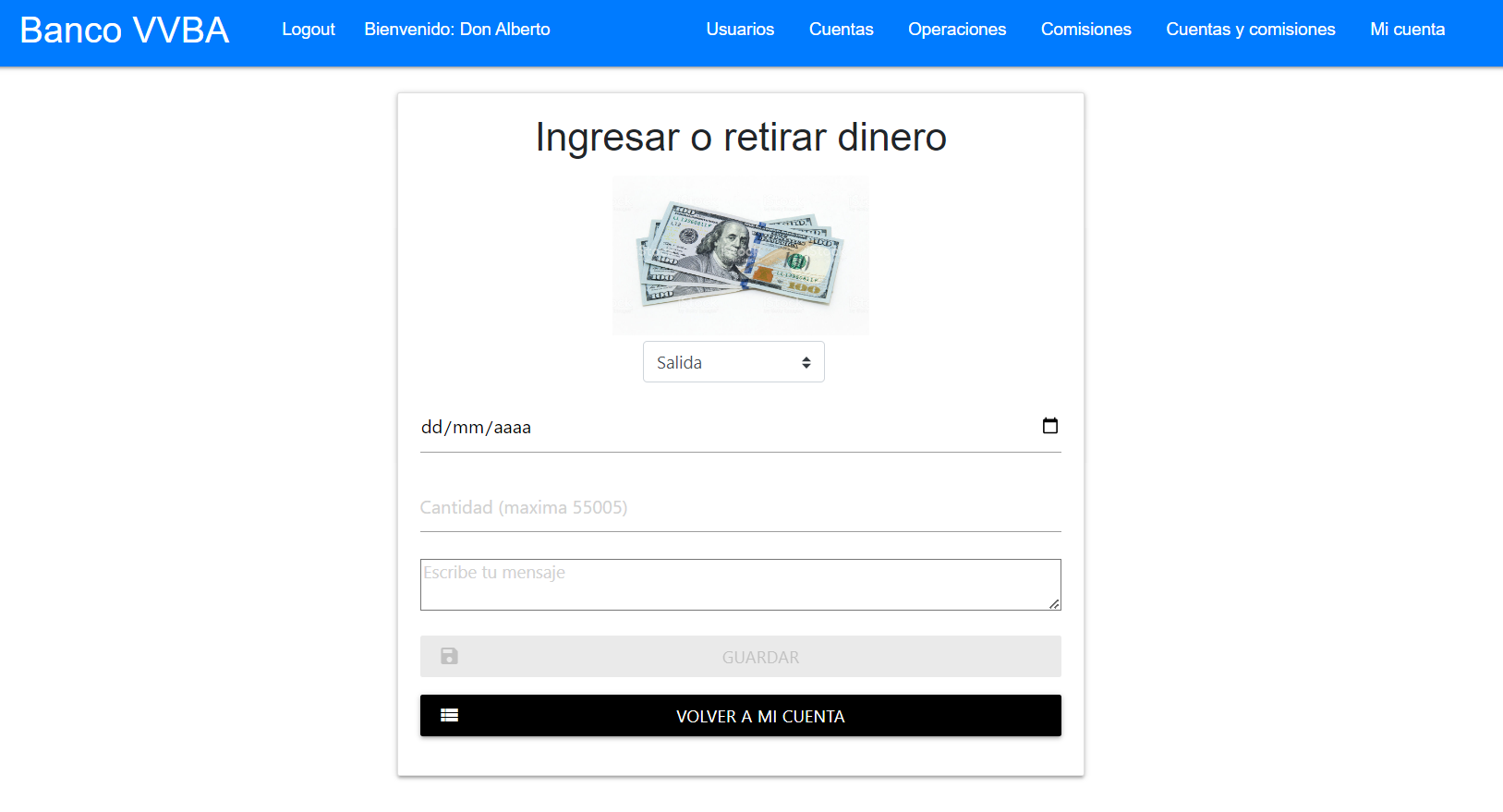


Ilustración 50. Ingresar retirar dinero final

Transferencia

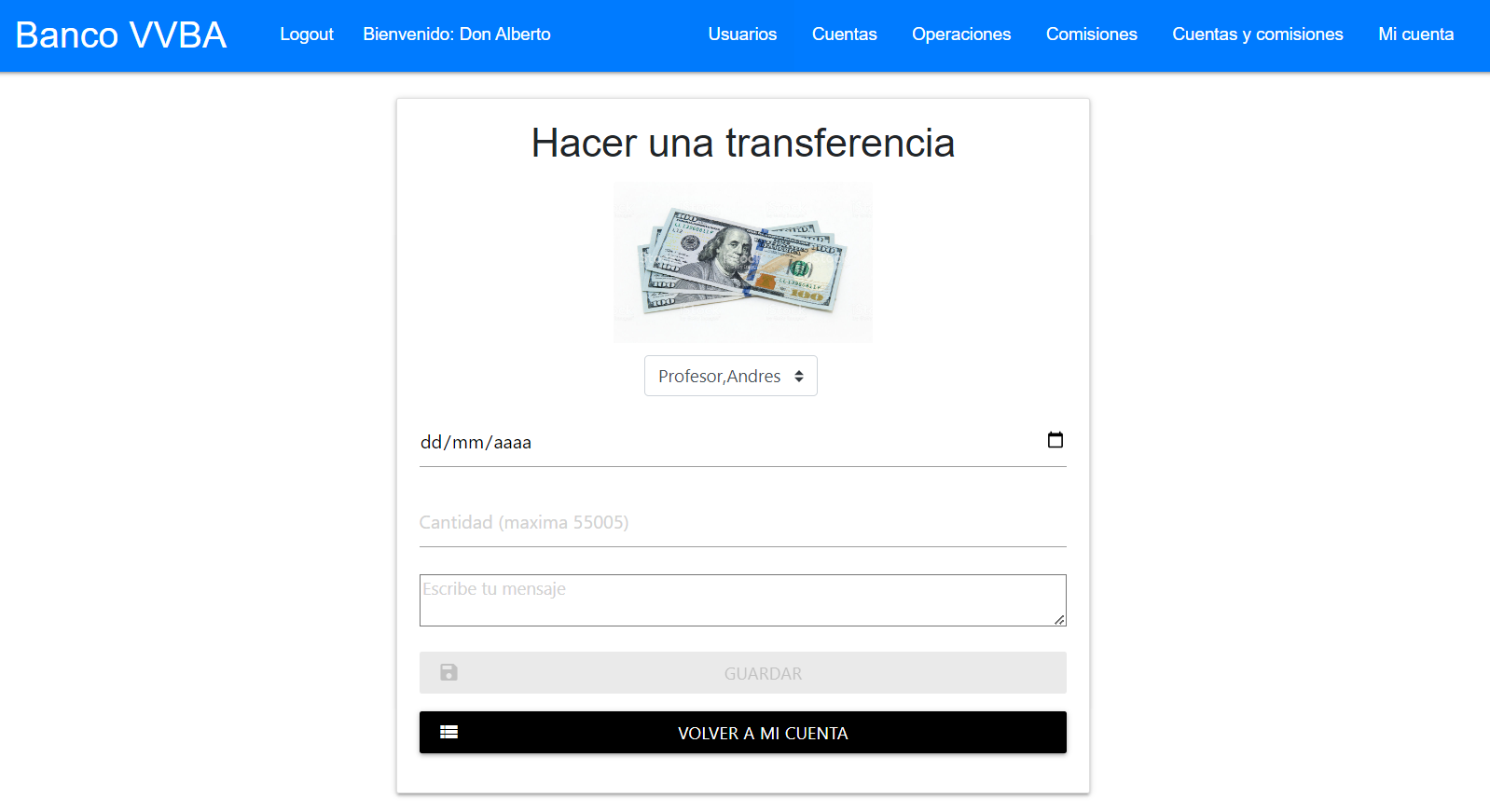


Ilustración 51. Transferencias final

1. Conclusiones y trabajo futuro

Para comenzar, se ha cumplido el objetivo propuesto por el proyecto, el cual era realizar un estudio de las tecnologías más usadas en las empresas hoy en día y hacer una aplicación utilizando estas tecnologías. Se ha logrado realizando una aplicación bancaria el cual servirá como mockup para personas que no están socializadas con las aplicaciones bancarias que se utilizan hoy en día y puedan practicas a gusto y antojo. Además, que la aplicación será utilizada en el ámbito de la enseñanza por el profesor ya mencionado anteriormente Juan Antonio Andrés Lalueza, al habérsele enseñado la aplicación le ha encantado y le dará un buen uso.

A mencionar que al haber desarrollado la aplicación en ASP .NET Core, la aplicación Banco VVBA servirá de referencia y de estudio para otro profesor también del centro educativo Altair, Alberto Arjona Pozo, para aprender esta tecnología y enseñarla en los próximos años en este centro.

Gracias a la buena estimación ha sido un proyecto considerado exitoso, ya que, al haberse hecho un estudio previo de los requisitos y todo lo necesario para la aplicación se han podido cumplir bien los tiempos.

Por último, una línea de trabajo futuro, serían implementar Jobs en la base de datos los cuales pudieran tener un “cobro” mensual de las comisiones en el salario de tu cuenta. También se podría implementar un buscador el cual busques lo que necesites dentro de la aplicación y te lleve a la página necesaria para ello.

1. Bibliografía

[1] Creación de proyecto angular y Tour of héroes <https://v17.angular.io/tutorial/tour-of-heroes>

[2] Tutorial de creación de proyecto ASP .NET Core con Entity Framework Core  
<https://www.youtube.com/watch?v=fqR-WbmX8PM&t=2025s>

[3] Conectividad con ASP .NET Core

<https://www.youtube.com/watch?v=OZGdKYzUYvU&list=PLjC4UKOOcfDQtElvsn1ZCAHatLtqDrTgQ>

[4] Uso de paquetes Nugget <https://learn.microsoft.com/en-us/nuget/consume-packages/install-use-packages-powershell>

[5] Importación de componentes Standalone <https://stackoverflow.com/questions/78035343/toastr-implementation-in-angular17-at-standalone-components>

[6] Conexiones http con el Back-End desde Angular <https://angular.dev/guide/http/setup>

[7] Mapear modelos en angular <https://medium.com/@mik3lov3st3ch/mapping-api-responses-to-models-in-angular-using-a-model-adapter-38a82b348fe5>

[8] Tutorial de uso de Guard <https://www.youtube.com/watch?v=dsVKoSyQZ98>

[9] Microsoft. (n.d.). ASP.NET Core documentation. Microsoft Docs. Retrieved September 24, 2024, from <https://docs.microsoft.com/aspnet/core/?view=aspnetcore-7.0>

[10] MacDonald, A. (2019). ASP.NET Core 3.0: The complete guide to building web applications with .NET Core 3.0. Packt Publishing.

[11] Kranthi, K. (2020). Implementing security in ASP.NET Core applications. Code Project. Retrieved September 24, 2024, from <https://www.codeproject.com/Articles/5256377/Implementing-Security-in-ASP-NET-Core-App>

[12] Angular. (n.d.). Angular documentation. Angular.io. Retrieved September 24, 2024, from https://angular.io/docs

[13] Tslim, G. (2021). Angular 11 by Example: Build real-world applications with Angular 11 and TypeScript. Packt Publishing.

[14] Vasan, S. (2021). 10 best practices for Angular developers. Medium. Retrieved September 24, 2024, from <https://medium.com/swlh/10-best-practices-for-angular-developers-87f6875e3c92>

[15] Lasak, D. (2022). Building microservices with ASP.NET Core. InfoQ. Retrieved September 24, 2024, from <https://www.infoq.com/articles/microservices-aspnet-core/>

[16] Johnson, J. (2020). Dependency Injection in ASP.NET Core. Dev.to. Retrieved September 24, 2024, from <https://dev.to/johnjohnston/dependency-injection-in-asp-net-core-3i68>

[17] Reilly, S. (2023). Using the repository pattern in ASP.NET Core applications. Medium. Retrieved September 24, 2024, from <https://medium.com/@stevenreilly/using-the-repository-pattern-in-asp-net-core-applications-b3c15167df30>

[18] Marzocco, J. (2020). Master lazy loading in Angular. AngularInDepth. Retrieved September 24, 2024, from <https://indepth.dev/master-lazy-loading-in-angular/>

[19] Corley, N. (2023). Understanding state management in Angular applications. Dev.to. Retrieved September 24, 2024, from <https://dev.to/ncorley/understanding-state-management-in-angular-applications-2d0p>

[20] Desjardins, P. (2022). Improve Angular performance with trackBy and onPush strategy. Netlify Blog. Retrieved September 24, 2024, from https://www.netlify.com/blog/improve-angular-performance/