



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



TFG del Grado en Ingeniería Informática

Seguimiento de
requerimientos para gestionar,
medir, evaluar y mejorar
Documentación Técnica



Presentado por Pablo Ahíta del Barrio
en Universidad de Burgos — 18 de abril
de 2024

Tutor: Pedro Luis Sánchez Ortega

Índice general

| | |
|--|-------------|
| Índice general | i |
| Índice de figuras | iii |
| Índice de tablas | viii |
| Apéndice A Plan de Proyecto Software | 1 |
| A.1. Edición del informe | 1 |
| A.2. Formato del informe | 1 |
| A.3. Comprobación de la ortografía del informe | 2 |
| A.4. Precedente | 2 |
| A.5. Planificación temporal | 23 |
| A.6. Estudio de viabilidad | 24 |
| Apéndice B Especificación de Requisitos | 29 |
| B.1. Introducción | 29 |
| B.2. Objetivos generales | 29 |
| B.3. Catálogo de requisitos | 31 |
| B.4. Especificación de requisitos | 33 |
| Apéndice C Especificación de diseño | 41 |
| C.1. Introducción | 41 |
| C.2. Diseño de datos | 41 |
| C.3. Diseño procedimental | 53 |
| C.4. Diseño arquitectónico | 62 |
| Apéndice D Documentación técnica de programación | 97 |

| | |
|--|------------|
| D.1. Introducción | 97 |
| D.2. Estructura de directorios | 97 |
| D.3. Manual del programador | 98 |
| D.4. Compilación, instalación y ejecución del proyecto | 98 |
| Apéndice E Documentación de usuario | 153 |
| E.1. Requisitos de usuarios | 153 |
| E.2. Instalación | 153 |
| E.3. Manual del usuario | 153 |
| Bibliografía | 155 |

Índice de figuras

| | |
|---|----|
| A.1. Menú principal de OTEA | 3 |
| A.2. Apartado de introducción de datos de la organización o del servicio | 4 |
| A.3. Apartado de realización del test de indicadores. Primer indicador | 6 |
| A.4. Ventana flotante de finalización del test de indicadores antes de guardar los datos | 7 |
| A.5. Ventana flotante de finalización del test de indicadores después de guardar los datos | 7 |
| A.6. Gráfico de muestra de resultados del servicio u organización . . | 8 |
| A.7. Informe con la puntuación del test de indicadores | 9 |
| A.8. Apartado de informe final con los datos del centro, las observaciones y las conclusiones | 11 |
| A.9. Página principal de <i>Print to PDF Pro</i> | 12 |
| A.10. Mensaje de bienvenida del programa de instalación de <i>Print to PDF Pro</i> | 13 |
| A.11. Contrato de licencia de <i>Print to PDF Pro</i> , de obligada aceptación para la instalación del programa | 14 |
| A.12. Selección del directorio de instalación de <i>Print to PDF Pro</i> . . | 15 |
| A.13. Registro de <i>Print to PDF Pro</i> , de cumplimiento opcional. | 16 |
| A.14. Finalización del asistente de instalación de <i>Print to PDF Pro</i> . . | 17 |
| A.15. Ventana emergente donde se pregunta si se desea visualizar el fichero recién generado | 17 |
| A.16. Fichero de prueba de <i>Print to PDF Pro</i> | 18 |
| A.17. Informe final con el botón de la impresora seleccionado | 19 |
| A.18. Informe en PDF generado por el programa en <i>Microsoft Access</i> | 20 |
| A.19. Información general de la suscripción en Azure | 24 |
| A.20. Velocidad y previsión de gastos junto con los costes por cada recurso | 25 |

| | |
|---|----|
| A.21.Ejemplo del registro de un usuario de la <i>Fundación Miradas</i> , con la casilla de la aceptación de los datos de acuerdo con la ley | 27 |
| B.1. Diagrama de casos de uso de la aplicación | 30 |
| C.1. Paquete user | 42 |
| C.2. Paquete organization | 46 |
| C.3. Paquete indicator | 47 |
| C.4. Paquete orgData | 49 |
| C.5. Diagrama de flujo para el inicio de sesión | 54 |
| C.6. Diagrama de flujo para el registro del usuario | 55 |
| C.7. Diagrama de flujo para el registro de la organización | 57 |
| C.8. Diagrama de flujo para el registro del equipo evaluador | 59 |
| C.9. Diagrama de flujo para la realización del test de indicadores | 61 |
| C.10. Esquema de comunicación entre el cliente (lado derecho) y el servidor (lado izquierdo), para la clase Address | 89 |
| C.11. Esquema de comunicación entre el cliente (lado derecho) y el servidor (lado izquierdo), para la clase Center | 90 |
| C.12. Esquema de comunicación entre el cliente (lado derecho) y el servidor (lado izquierdo), para la clase City | 90 |
| C.13. Esquema de comunicación entre el cliente (lado derecho) y el servidor (lado izquierdo), para la clase Country | 91 |
| C.14. Esquema de comunicación entre el cliente (lado derecho) y el servidor (lado izquierdo), para la clase EvaluatorTeamMember . | 91 |
| C.15. Esquema de comunicación entre el cliente (lado derecho) y el servidor (lado izquierdo), para la clase EvaluatorTeam | 92 |
| C.16. Esquema de comunicación entre el cliente (lado derecho) y el servidor (lado izquierdo), para la clase Evidence | 92 |
| C.17. Esquema de comunicación entre el cliente (lado derecho) y el servidor (lado izquierdo), para la clase Indicator | 93 |
| C.18. Esquema de comunicación entre el cliente (lado derecho) y el servidor (lado izquierdo), para la clase IndicatorsEvaluationReg | 93 |
| C.19. Esquema de comunicación entre el cliente (lado derecho) y el servidor (lado izquierdo), para la clase IndicatorsEvaluation . . | 94 |
| C.20. Esquema de comunicación entre el cliente (lado derecho) y el servidor (lado izquierdo), para la clase Organization | 94 |
| C.21. Esquema de comunicación entre el cliente (lado derecho) y el servidor (lado izquierdo), para la clase Province | 95 |
| C.22. Esquema de comunicación entre el cliente (lado derecho) y el servidor (lado izquierdo), para la clase Region | 95 |

| | |
|---|-----|
| C.23.Esquema de comunicación entre el cliente (lado derecho) y el servidor (lado izquierdo), para la clase User | 96 |
| D.1. Página de descarga del instalador de Android Studio | 99 |
| D.2. Sección de términos y condiciones anterior a la descarga del instalador de Android Studio | 99 |
| D.3. El fichero de instalación ya ha sido descargado | 100 |
| D.4. Mensaje de bienvenida del instalador de Android Studio | 101 |
| D.5. Selección de componentes a instalar en el instalador de Android Studio | 102 |
| D.6. Selección del directorio de instalación | 103 |
| D.7. Selección del directorio de creación de accesos directos en el menú de inicio | 104 |
| D.8. Inicio de instalación de Android Studio | 105 |
| D.9. Finalización de la instalación de Android Studio | 106 |
| D.10.Mensaje de finalización de la instalación | 107 |
| D.11.Ventana emergente de importación de configuración | 108 |
| D.12.Mensaje de bienvenida del asistente de configuración incial de Android Studio | 109 |
| D.13.Selección del tipo de instalación de los componentes de Android Studio | 110 |
| D.14.Selección del directorio predeterminado del JDK | 111 |
| D.15.Selección del diseño de la interfaz de Android Studio | 112 |
| D.16.Selección de componentes SDK a instalar en Android Studio | 113 |
| D.17.Advertencia por falta de componentes a instalar | 113 |
| D.18.Configuración de la memoria RAM utilizada por el emulador de Android | 114 |
| D.19.Verificación de componentes a instalar en Android Studio | 115 |
| D.20.Licencia de software del paquete SDK | 116 |
| D.21.Licencia de software del acelerador de emulación de Intel | 117 |
| D.22.Instalación de los componentes en la configuración inicial | 118 |
| D.23.Finalización del asistente de instalación de componentes | 119 |
| D.24.Pantalla de selección de proyecto | 120 |
| D.25.Pantalla de selección de actividad principal | 121 |
| D.26.Pantalla de configuración de la actividad | 122 |
| D.27.Desarrollo del proceso de instalación del soporte JDK | 123 |
| D.28.Finalización del proceso de instalación del soporte JDK | 123 |
| D.29.Interfaz de Android Studio con el proyecto Android recién creado | 124 |
| D.30.Acceso a las opciones de Android Studio | 124 |
| D.31.Pantalla de control de cuentas de GitHub | 125 |

| | |
|--|-----|
| D.32.Pantalla de tokens de usuario antes de ir a la pantalla de generación de tokens | 126 |
| D.33.Pantalla de generación de tokens con los permisos de usuario | 126 |
| D.34.Siguientes permisos de usuario | 127 |
| D.35.Últimos permisos de usuario | 127 |
| D.36.Pantalla de tokens de usuario con el token de usuario generado | 128 |
| D.37.Pantalla de control de cuentas de GitHub lista para añadir usuario | 128 |
| D.38.Añadir token de usuario de GitHub en Android Studio | 129 |
| D.39.Cuenta de GitHub recién añadida | 129 |
| D.40.Acceso al administrador de dispositivo mediante el menú desplegable de dispositivos | 130 |
| D.41.Administrador de dispositivos | 131 |
| D.42.Selección del nuevo dispositivo virtual | 131 |
| D.43.Selección de sistema operativo para el nuevo dispositivo virtual | 132 |
| D.44.Proceso de instalación del sistema operativo y de configuración del dispositivo virtual | 133 |
| D.45.Finalización del proceso de instalación del sistema operativo y de configuración del dispositivo virtual | 133 |
| D.46.Finalización del asistente de creación del dispositivo y comprobación de la configuración | 134 |
| D.47.Acceso a la configuración del nuevo commit | 135 |
| D.48.Configuración del nuevo commit | 135 |
| D.49.Soluciones a warnings durante el proceso de commit | 136 |
| D.50.Menú de publicación de cambios realizados | 136 |
| D.51.Commit y push realizados satisfactoriamente | 137 |
| D.52.Acceso a la creación de la web app desde el menú principal del portal de <i>Azure</i> | 137 |
| D.53.Configuración de los detalles del proyecto y de la aplicación web del nuevo <i>App Service</i> con <i>Azure SQL</i> | 138 |
| D.54.Configuración de la base de datos del nuevo <i>App Service</i> con <i>Azure SQL</i> | 139 |
| D.55.La web app ya ha empezado a crearse | 139 |
| D.56.Implementación en curso | 140 |
| D.57.Página oficial de descarga de <i>Visual Studio 2022</i> | 140 |
| D.58.Antes de empezar hay que preparar el instalador | 141 |
| D.59.Preparando el instalador... | 141 |
| D.60.Selección de las cargas de trabajo | 142 |
| D.61.Instalación en marcha | 143 |
| D.62.Tareas iniciales | 144 |
| D.63.Selección de plantilla | 144 |
| D.64.Configuración de proyecto | 145 |

| | |
|--|-----|
| D.65.Selección de framework | 146 |
| D.66.Ir a publicar | 147 |
| D.67.Agregar perfil de publicación | 147 |
| D.68.Selección de publicación en Azure | 148 |
| D.69. | 149 |
| D.70.Selección de web service | 150 |
| D.71.Despliegue completado | 151 |

Índice de tablas

| | |
|---|----|
| A.1. Tiempo invertido en cada actividad | 23 |
| B.1. CU-1 Mostrar actividad reciente | 34 |
| B.2. CU-2 Administrar base de datos. | 35 |
| B.3. CU-3 Gestión de indicadores. | 36 |
| B.4. CU-4 Gestión de organizaciones. | 37 |
| B.5. CU-4 Gestión de usuarios. | 38 |
| B.6. CU-6 Realizar evaluación de indicadores. | 39 |
| B.7. CU-6 Realizar evaluación de indicadores. | 40 |

Apéndice A

Plan de Proyecto Software

A.1. Edición del informe

En cuanto a la edición del informe se han tenido en cuenta diferentes aspectos para poder escoger el formato que debe tener informe, como por ejemplo la personalización del documento, la preocupación por los márgenes de cada una de las páginas, el control de la ortografía, la introducción de fórmulas matemáticas, de tablas, de imágenes y de gráficos y el acabado profesional de este informe. Por tanto, dichos aspectos son los expuestos en los sub-apartados de este punto.

A.2. Formato del informe

El editor de documentos que se ha decidido utilizar para realizar dicho informe es *LaTeX* por encima de editores de documentos convencionales como *OpenOffice Write* o *Microsoft Word*. El motivo por el cual se ha decidido utilizar *LaTeX* es por su gran cantidad de funcionalidades y por la libertad con la que se puede diseñar el documento en todos los aspectos mencionados con anterioridad. Para poder crear los documentos pdf en (*LaTeX*) es necesaria la instalación de un compilador. Existen diferentes opciones para poder realizar la compilación del documento, como es el caso de *Overleaf*, el cual es un editor en línea de código en *LaTeX*, de uso compartido y con compilador integrado. En el caso de este informe, se ha utilizado el compilador *MiKTeX*, que es un compilador de libre distribución el cual permite instalar todos los paquetes que el usuario necesite para su documento. Como editor de *LaTeX* se ha utilizado *Microsoft Visual*

Studio Code. Se ha escogido ese editor por encima del editor predefinido de *MikTeX* (de nombre *TexWorks*), puesto que es un editor bastante más completo, ya que se puede editar el código fuente de la aplicación además del propio documento, puesto que soporta la gran mayoría de los lenguajes de programación más utilizados de la actualidad, aparte de dar soporte al propio *LaTeX* y a su compilador *MikTeX*.

A.3. Comprobación de la ortografía del informe

Gracias al editor *Microsoft Visual Studio Code* es posible agregar complementos que faciliten el soporte a todo tipo de lenguajes y que faciliten también un buen uso para todo tipo de usuarios. Uno de estos complementos es el que se ha utilizado para facilitar la comprobación de la ortografía de todo tipo de lenguajes, entre ellos *LaTeX*, el cual recibe el nombre de ***Code Spell Checker***, para el idioma español, el cual se instala de una manera muy sencilla en *Visual Studio Code*, presionando el botón de Instalar dentro de la página web del *Marketplace*, el cual está disponible también para su búsqueda dentro del propio editor. Posteriormente, para activarlo se puede hacer de diferentes maneras:

1. El primer método consiste en presionar el botón *F1* y posteriormente escribir el comando *Enable Spanish Spell Checker Dictionary* y se presiona el botón *Enter* para que empiece a trabajar.
2. El segundo método consiste en ir al menú de la barra superior de tareas *Ver* y posteriormente al sub-menú *Paleta de comandos*. Posteriormente se escribe el comando *Enable Spanish Spell Checker Dictionary* y se presiona el botón *Enter* para que empiece a trabajar.

La versión original de este complemento realiza las comprobaciones ortográficas para el idioma **inglés**, el cual se instala en *Visual Studio Code* de la misma manera, presionando el botón de Instalar dentro de la página web del *Marketplace*. Al tener instalada la versión en español, que es una extensión de la versión original, no precisa instalarlo después de la versión en español.

A.4. Precedente

Para dar contexto a la aplicación desarrollada se tiene que tener en cuenta el punto de partida de este proyecto, el cual se trata de un programa *.mdb*,

el cual está implementado sobre *Microsoft Access*, el cual estaba grabado en un CD-ROM. Dicho programa recibe el nombre de *Guía de indicadores de calidad para Organizaciones que presentan apoyo a personas con Trastorno del Espectro Autista*, cuyo acrónimo es **OTEA**.

OTEA. Guía de indicadores de calidad para Organizaciones que presentan apoyo a personas con Trastorno del Espectro Autista

OTEA, como se ha expuesto con anterioridad es una aplicación implementada en Access implementada por la *Fundación Miradas* en el año 2009. Dicha aplicación estaba implementada en un CD-ROM, el cual debía introducirse en una bandeja conectada al equipo para que posteriormente pudiese ejecutar la aplicación. El menú principal de dicha aplicación es el siguiente:



Figura A.1: Menú principal de OTEA

El menú principal de la aplicación tiene una apariencia bastante sencilla para la época, con diferentes apartados para realizar el diagnóstico de la forma indicada. Dichos apartados son los siguientes:

- **Datos de la organización o servicio:** Este apartado consiste en un formulario en el que se introducen los datos de la organización

evaluadora, los datos del equipo evaluador y las fechas en las que se realizaron las cuatro evaluaciones. Su interfaz es la siguiente:

The screenshot shows a software window titled "tbl_datos_centro" with a background illustration of a person holding a child. The form is titled "Datos de la organización o servicio". It contains the following fields:

- CENTRO O SERVICIO:** [Redacted]
- DIRECCIÓN:** [Redacted]
- TELÉFONO:** [Redacted]
- E-MAIL:** [Redacted]
- OTROS DATOS DE INTERÉS:** Terapias para niños y orientación a padres
- CONSULTOR/A EXTERNO/A:** Miguel Gómez Gentil
- ORGANIZACIÓN A LA QUE PERTENECE:** Fundación Miradas (Burgos)
- DIRECTOR/A DE LA ORGANIZACIÓN:** [Redacted]
- PROFESIONAL DE ATENCIÓN DIRECTA:** [Redacted]
- FAMILIAR:** [Redacted]
- OTROS PARTICIPANTES:** [Redacted]
- CONSTITUCIÓN DEL EQUIPO EVALUADOR:** 08/09/2022
- VISITA AL CENTRO:** 14/11/2022
- PRIMERA SESIÓN DE VALORACIÓN:** 16/01/2023
- SEGUNDA SESIÓN DE VALORACIÓN:** 14/04/2023
- TERCERA SESIÓN DE VALORACIÓN:** 14/07/2023
- CUARTA SESIÓN DE VALORACIÓN:** 16/10/2023

At the bottom are buttons for "Guardar" (Save), "Cancelar" (Cancel), and a print icon. The footer reads "OTEA, José Luis Cuesta Gómez, jl.cuesta@ubu.es".

Figura A.2: Apartado de introducción de datos de la organización o del servicio

Como se puede comprobar en la captura anterior, los datos a rellenar son los siguientes:

- **Centro o servicio:** En este campo se introduce el nombre del centro o servicio al que se va a evaluar.
- **Dirección:** En este campo se introduce la dirección donde se ubica el centro a evaluar.
- **Teléfono:** En este campo se introduce el número telefónico de la organización a la que se va a evaluar.

- **Email:** En este campo se introduce la dirección de correo electrónico del centro a evaluar.
- **Otros datos de interés:** En este campo se introduce información adicional sobre el centro o servicio a evaluar
- **Consultor/a externo/a:** En este campo se introduce el nombre completo de la persona que lidera el equipo que va a realizar la valoración a ese centro o servicio en concreto.
- **Organización a la que pertenece:** En este campo se introduce la organización a la que pertenece el consultor externo que lidera el equipo de valoración
- **Director/a de la organización:** En este campo se introduce el nombre del director o directora del centro a evaluar.
- **Profesional de atención directa:** En este campo se introduce el nombre del responsable del profesional del centro a evaluar que actúa de intermediario entre la organización y el equipo evaluador.
- **Familiar:** En este campo se introduce el nombre del familiar que conoce la organización o servicio a evaluar.
- **Otros participantes:** En este campo se introducen los nombres de los demás componentes del equipo evaluador.
- **Fecha de constitución del equipo evaluador:** En este campo se introduce el día en el que se constituyó el equipo evaluador.
- **Fecha de visita al centro:** En este campo se introduce el día en el que se realizó la primera visita al centro a evaluar
- **Fechas de las correspondientes sesiones de evaluación:** En estos campos se introducen todas las fechas de evaluación

Al igual que con otros aspectos del programa, se dispone de una interfaz bastante intuitiva para la época, ayudando a introducir correctamente los diferentes tipos de datos, como el caso de las fechas, obligando a forzar la introducción de ese tipo de dato. Un problema bastante importante a la hora de introducir los datos es la escasa cantidad máxima de caracteres en algunos campos que lo necesitan, como es el caso de la dirección de la organización y de los otros datos de interés, puesto que, tal y como se ha podido comprobar, no se ha podido introducir correctamente la dirección completa de, en este caso, el *Centro de Organización Neurológica Neocortex* de Majadahonda (Madrid), ya que por la longitud de la misma se han tenido que abbreviar

el tipo de vía y la palabra bloque, aparte de que no se ha podido introducir el código postal ni el municipio donde se encuentra, siendo este último introducido junto con en el nombre de la organización. Además de eso, no existe control sobre el tipo de datos introducidos en cada uno de los campos, puesto que toma todos los datos como tipo string.

- **Comenzar test:** En este apartado se realiza el test de indicadores de la organización a evaluar. El test consta de 68 indicadores, con cuatro evidencias cada uno. Un indicador es una categoría que abarca diferentes aspectos de la organización que alberga a personas con Trastorno del Espectro Autista, siendo cada una de las cuatro evidencias un aspecto específico relacionado con dicha categoría de evaluación.

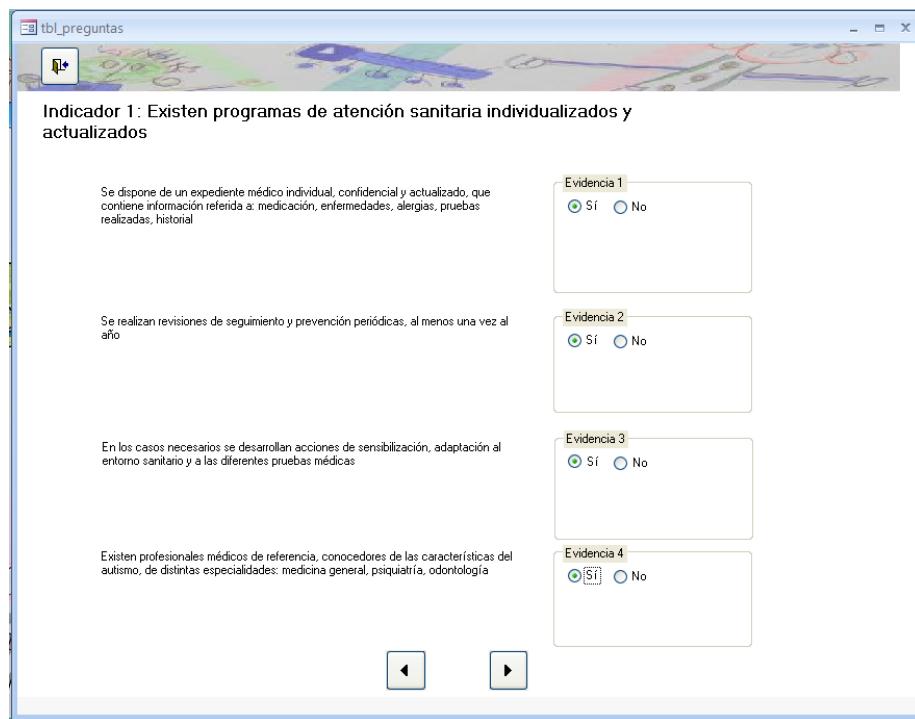


Figura A.3: Apartado de realización del test de indicadores. Primer indicador

Como se puede comprobar en la captura anterior, la interfaz del apartado del test tiene un formato bastante intuitivo para la época. Pero a medida de que se van analizando más indicadores, se vuelve bastante pesado en su uso, puesto que hay que mover el ratón hasta la respuesta seleccionada. Cuando ya se han evaluado los 68 indicadores,

aparece la siguiente ventana flotante invitando al usuario a que guarde los datos introducidos:

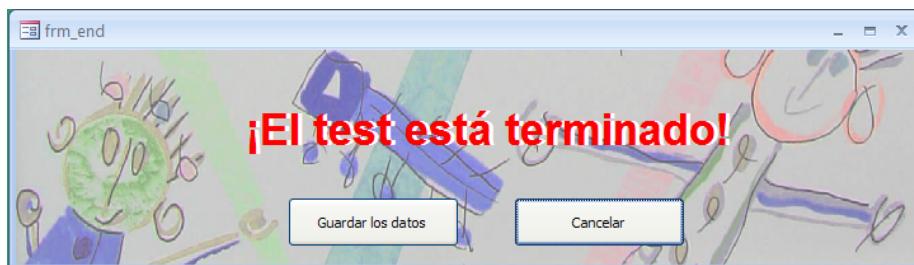


Figura A.4: Ventana flotante de finalización del test de indicadores antes de guardar los datos

Para finalizar se presiona en *Guardar datos* y posteriormente en *Sair*.

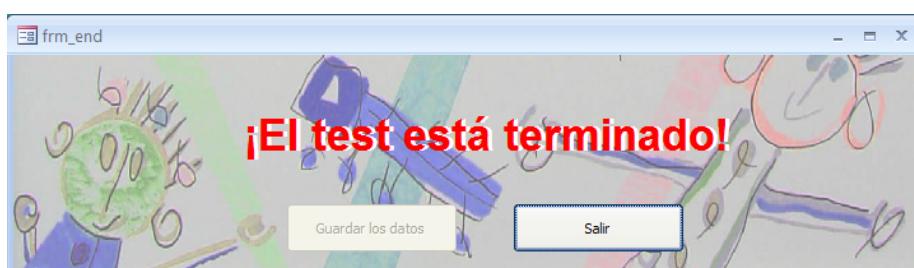


Figura A.5: Ventana flotante de finalización del test de indicadores después de guardar los datos

- **Gráfico del servicio u organización:** Este gráfico se utiliza como muestra de resultados del test de indicadores y evidencias que se realiza con anterioridad. Este gráfico es una tabla en la que las filas reflejan el interés que tiene el indicador y las columnas reflejan la clasificación de cada indicador dependiendo de cada aspecto a evaluar de esa organización. Cada indicador es identificado como un cuadrado en el que se muestra el número del mismo y el color verde, amarillo o rojo dependiendo del nivel mejor, promedio o peor de cumplimiento de cada indicador.

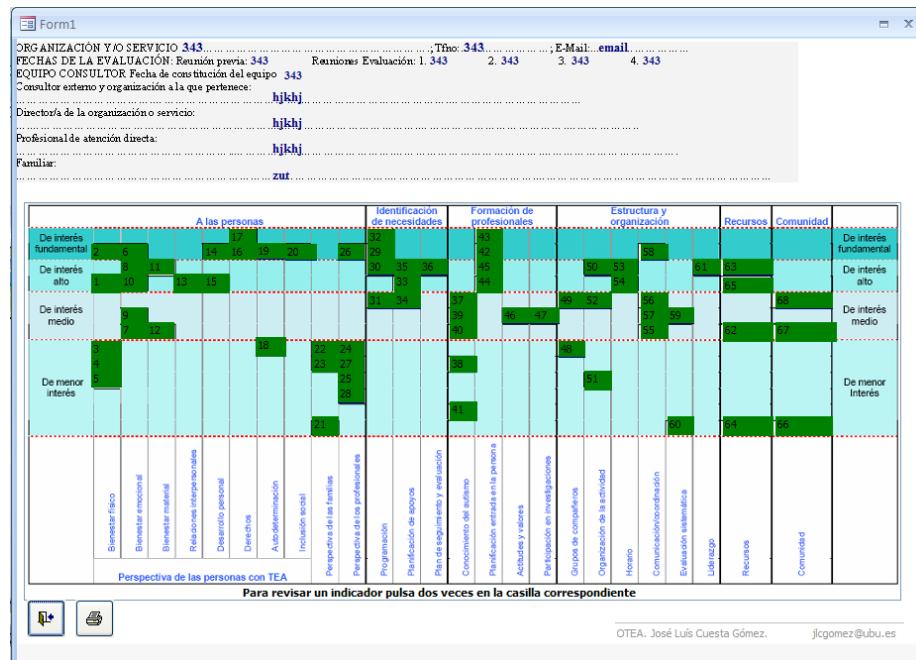


Figura A.6: Gráfico de muestra de resultados del servicio u organización

Se puede comprobar en este gráfico que no aparece la información introducida en el apartado de Datos de Información y Servicio, sino que se muestran valores aleatorios para cada uno de los campos. También se puede comprobar que esta evaluación de los indicadores ha sido perfecta como refleja cada uno de los cuadrados de los indicadores, los cuales son rellenos de color verde. Si se da doble clic encima de cualquiera de los cuadrados del gráfico, se puede mostrar información de dicho indicador, la cual se muestra en la interfaz del apartado de realización de test.

- **Perfil general:** En este apartado se obtiene un informe general con la información obtenida desde Datos de organización o servicio, con la puntuación obtenida en el test de indicadores y en el rango en el que se ubica.

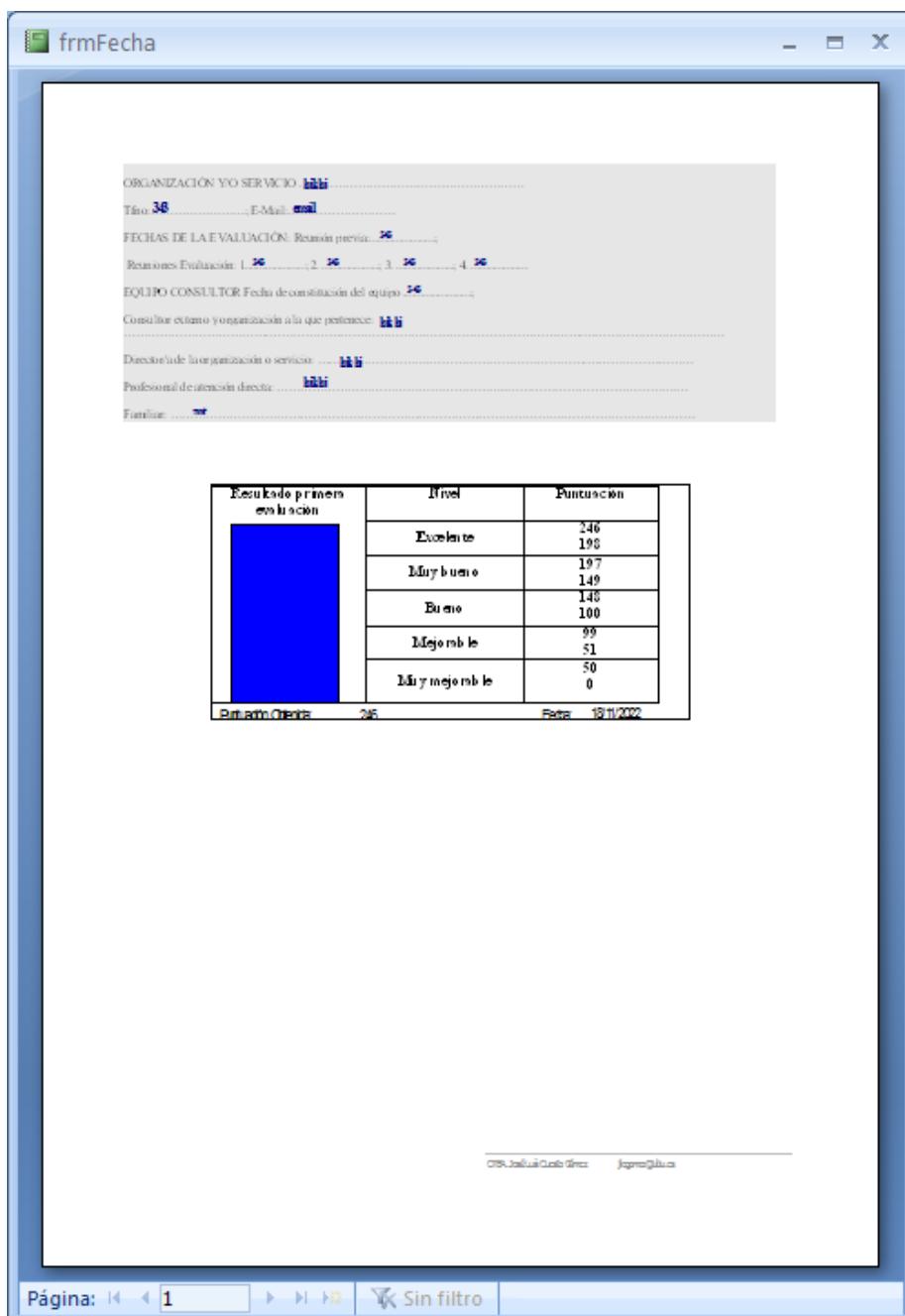


Figura A.7: Informe con la puntuación del test de indicadores

- **Informe final:** Este apartado es similar al de Datos de la organización o servicio, con los mismos campos que se han introducido con

anterioridad, con la diferencia de que hay dos cuadros de relleno de párrafo, los cuales son

- **Observaciones:** En este apartado se escriben las observaciones que se han tenido en cuenta durante las cuatro sesiones de valoración.
- **Informe final:** En este apartado se escriben las conclusiones de todas las sesiones de evaluación, así como algún apunte sobre el resultado del test de indicadores.

Dicho apartado tiene la siguiente interfaz:

Datos de la organización o servicio

| | |
|--|---|
| CENTRO O SERVICIO | CON Neocortex (Majadahonda, Madrid) |
| DIRECCIÓN | Ctra. Boadilla del Monte, 8-44 Local 1 Bl.3 |
| TELÉFONO | 916390390 |
| E-MAIL | cita@neocortex.com |
| OTROS DATOS DE INTERÉS | Terapias para niños y orientación a padres |
| CONSULTOR/A EXTERNO/A | Miguel Gómez Gentil |
| ORGANIZACIÓN A LA QUE PERTENECE | Fundación Miradas (Burgos) |
| DIRECTOR/A DE LA ORGANIZACIÓN | Mª Jesús López Juez |
| PROFESIONAL DE ATENCIÓN DIRECTA | Mª Jesús López Juez |
| FAMILIAR | Sonia Rodríguez Cano |
| OTROS PARTICIPANTES | Sonia Rodríguez Cano |
| CONSTITUCIÓN DEL EQUIPO EVALUADOR | 08/09/2022 |
| VISITA AL CENTRO | 14/11/2022 |
| PRIMERA SESIÓN DE VALORACIÓN | 16/01/2023 |
| SEGUNDA SESIÓN DE VALORACIÓN | 14/04/2023 |
| TERCERA SESIÓN DE VALORACIÓN | 14/07/2023 |
| CUARTA SESIÓN DE VALORACIÓN | 16/10/2023 |

Observaciones:

En este centro se realizan terapias que estimulan el desarrollo cerebral de los niños. Dichas terapias constan de programas tanto fisiológicos y sensoriales como físicos e intelectuales. Para diseñar de forma individualizada los programas que cada niño necesita, el niño y su familia realizan una visita de dos días al centro. El primer día es dedicado a la evaluación, audiometría e historial del menor por la mañana y la realización de la revisión médica y el resumen de todo lo realizado durante el día por la tarde. Al día siguiente se realiza una conferencia de formación y se presenta el programa fisiológico y sensorial por la mañana y por la tarde se presentan los programas físico e intelectual. Dichas visitas son realizadas cada 6 meses como seguimiento de la evolución del niño.

Informe Final:

La dinámica del centro hacia los niños con Trastorno del Espectro Autista, al tratarse de visitas de dos días, cubren al completo todas las necesidades que tiene el niño a tratar, desde las fisiológicas y sensoriales hasta las físicas e intelectuales. Al tener dos días seguidos de visita, de los cuales se dedican bastantes horas a todo el proceso de evaluación, se realizan de manera personalizada los programas que el niño necesita y las pautas que deben seguir los padres y/o tutores del menor en

OTEA. José Luis Cuesta Gómez. jlcgomez@ubu.es

Figura A.8: Apartado de informe final con los datos del centro, las observaciones y las conclusiones

Para obtener el PDF del informe, no es posible hacerlo de forma nativa desde Windows XP, puesto que no posee la herramienta *Microsoft Print to PDF* al ser un sistema operativo bastante antiguo. Para solucionar este problema se ha descargado el programa *Print to PDF*

Pro, de la desarrolladora *Traction Software Limited*, cuya instalación sigue los siguientes pasos:

- En primer lugar, se accede a la página web de descarga del programa y se presiona el botón *Download* para descargar el fichero de instalación:

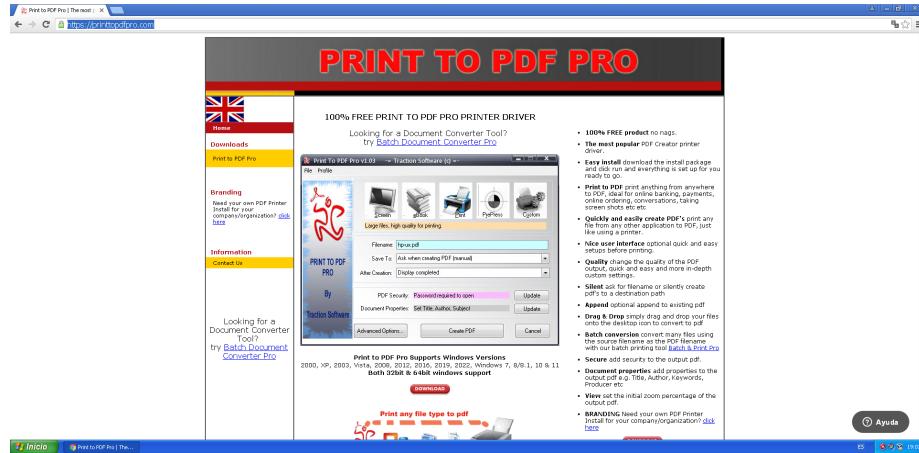


Figura A.9: Página principal de *Print to PDF Pro*

- Posteriormente, se ejecuta el fichero de instalación *PrintToPDF-ProSetup.exe*, el cual ejecuta el asistente de instalación del mismo:
 1. Al ejecutarse el fichero, da un mensaje de bienvenida donde también se recomienda que se cierren otros programas antes de ejecutar el asistente de instalación.

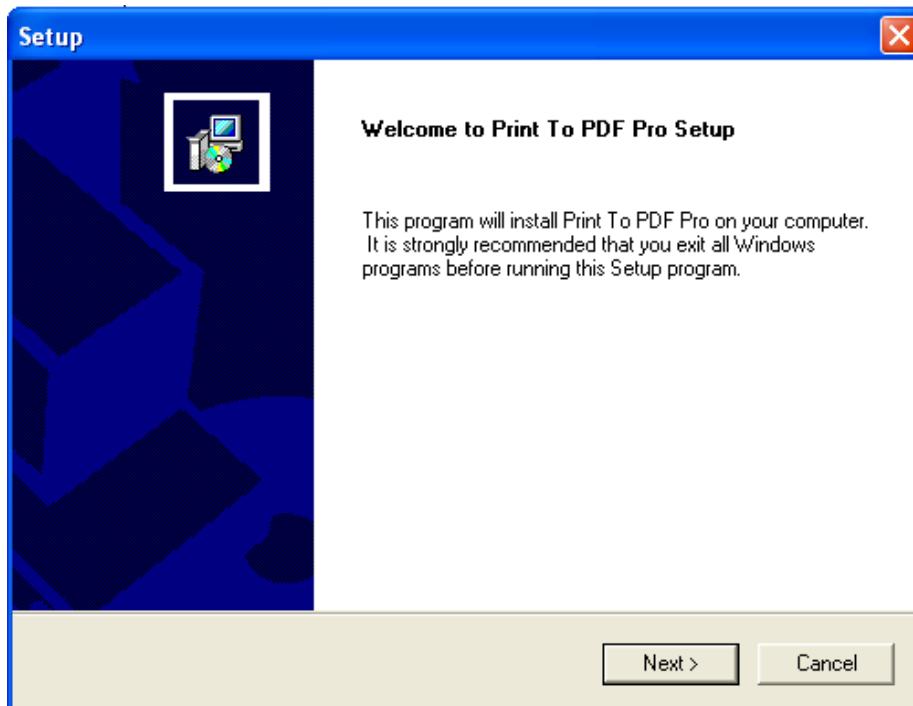


Figura A.10: Mensaje de bienvenida del programa de instalación de *Print to PDF Pro*

2. Tras presionar el botón *Next*, se tiene que aceptar el contrato de licencia para poder instalar *Print to PDF Pro* correctamente. El propio asistente recomienda también la lectura de dicho contrato antes de su aceptación.

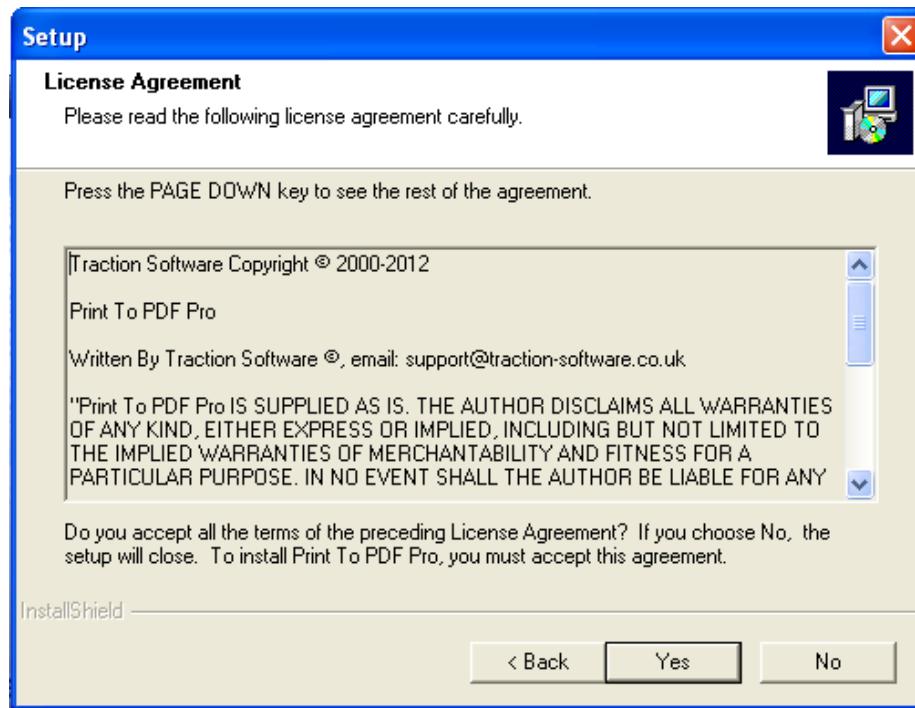


Figura A.11: Contrato de licencia de *Print to PDF Pro*, de obligada aceptación para la instalación del programa

3. Más adelante se procede a elegir el directorio donde se instalará el programa. Se puede optar por mantener el directorio por defecto, el cual es *C:/Program Files/Traction Software/Print to PDF Pro*, o en su defecto elegir otro directorio diferente donde instalarlo.

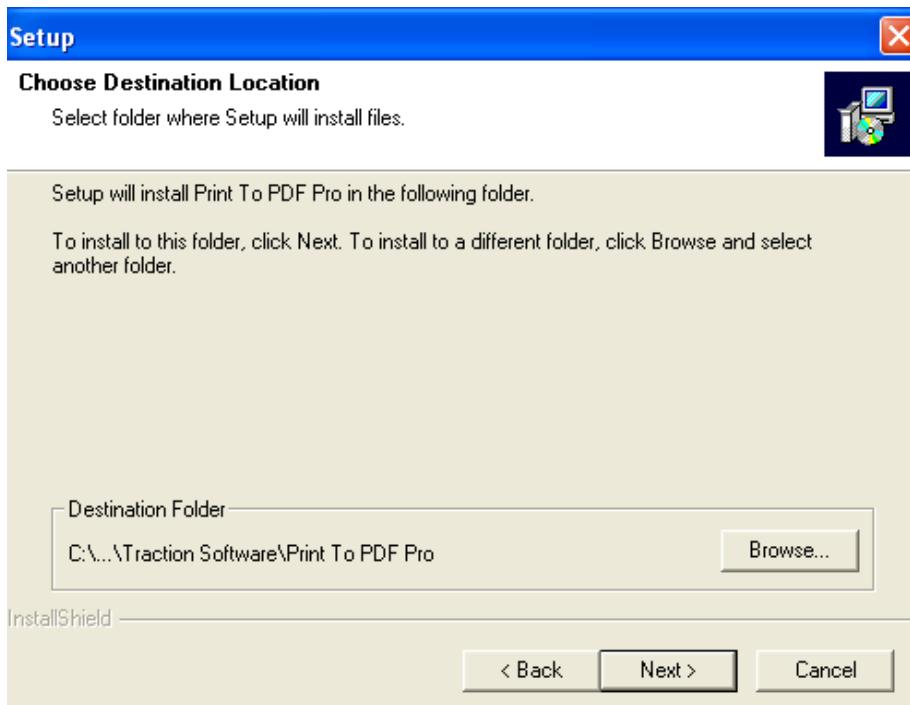


Figura A.12: Selección del directorio de instalación de *Print to PDF Pro*

4. Tras haber elegido el directorio de instalación se procede a la instalación del programa y, cuando ésta concluya, se da la opción al usuario la opción a registrar el producto, introduciendo su nombre, el nombre de la compañía donde trabaja y la dirección de correo electrónico.

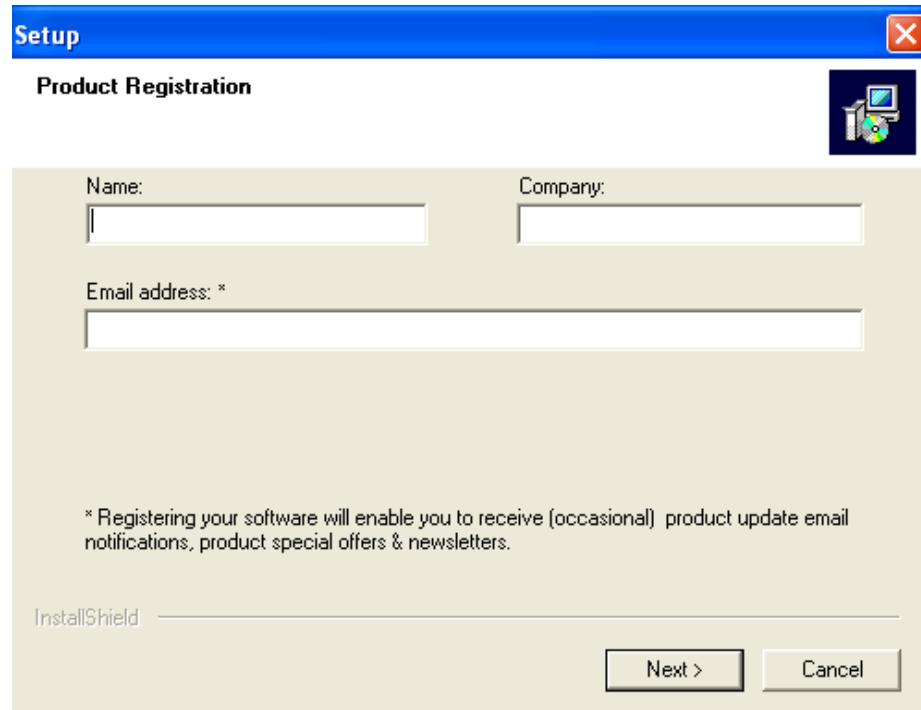


Figura A.13: Registro de *Print to PDF Pro*, de cumplimiento opcional.

5. Al finalizar el asistente de instalación se da la opción al usuario de imprimir un fichero de prueba en formato *.pdf* tras cerrar el programa de instalación. En caso de que se deje marcada la casilla, como se muestra a continuación, se procede a la creación del fichero:

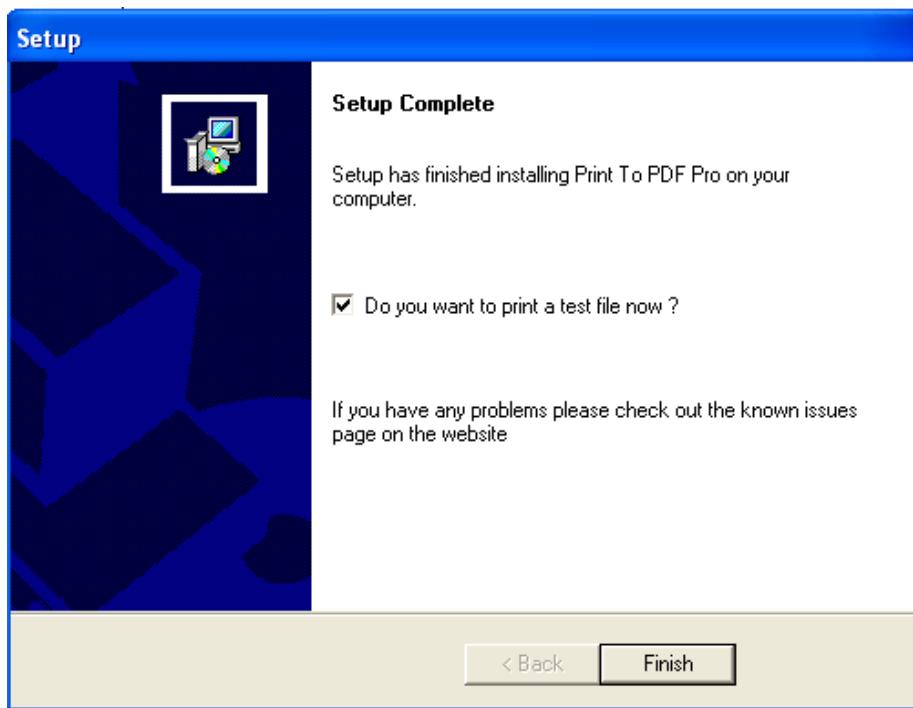


Figura A.14: Finalización del asistente de instalación de *Print to PDF Pro*

Cuando dicho fichero se haya terminado de crear, mostrará esta ventana emergente donde se solicita al usuario la visualización del fichero de prueba. En caso de seleccionar *Yes*, se mostrará el contenido de dicho fichero de prueba:

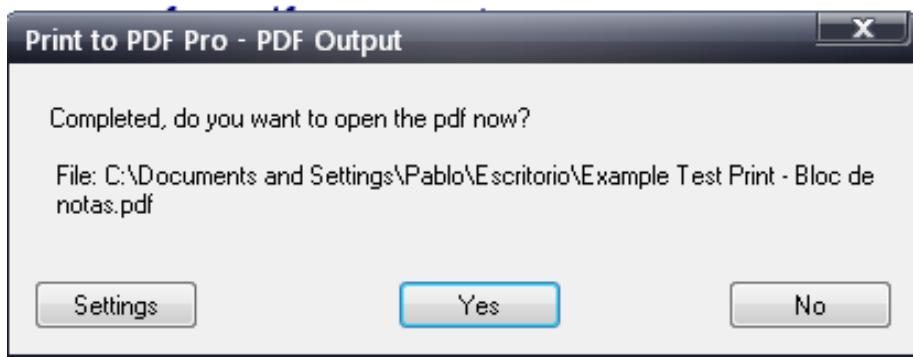


Figura A.15: Ventana emergente donde se pregunta si se desea visualizar el fichero recién generado

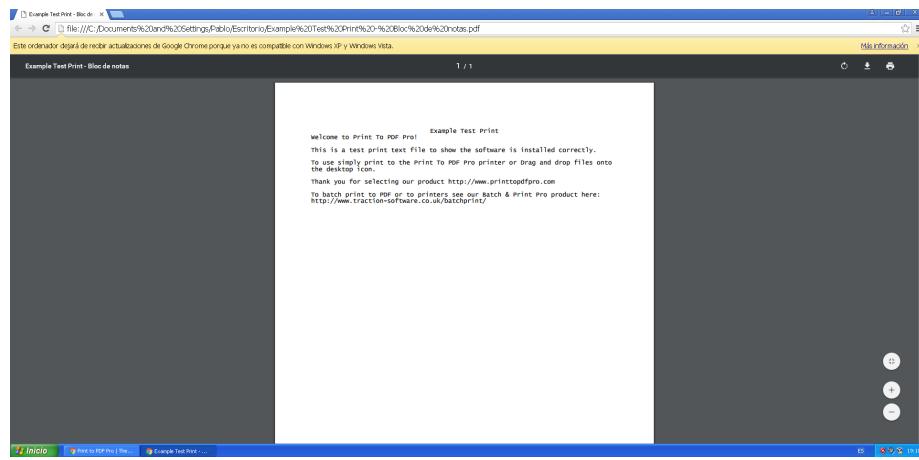


Figura A.16: Fichero de prueba de *Print to PDF Pro*

Por tanto, tras haber instalado este programa y haber comprobado que *Print to PDF Pro* es la impresora predefinida del sistema operativo, se procede a la impresión a un fichero .pdf del Informe Final, el cual se obtiene presionando el botón de la impresora de la parte inferior de la ventana de **Informe Final**

tbl_datos_centro

| | |
|-----------------------------------|--|
| E-MAIL | |
| OTROS DATOS DE INTERÉS | Terapias para niños y orientación a padres |
| CONSULTOR/A EXTERNO/A | Miguel Gómez Gentil |
| ORGANIZACIÓN A LA QUE PERTENECE | Fundación Miradas (Burgos) |
| DIRECTOR/A DE LA ORGANIZACIÓN | |
| PROFESIONAL DE ATENCIÓN DIRECTA | |
| FAMILIAR | |
| OTROS PARTICIPANTES | |
| CONSTITUCIÓN DEL EQUIPO EVALUADOR | 08/09/2022 |
| VISITA AL CENTRO | 14/11/2022 |
| PRIMERA SESIÓN DE VALORACIÓN | 16/01/2023 |
| SEGUNDA SESIÓN DE VALORACIÓN | 14/04/2023 |
| TERCERA SESIÓN DE VALORACIÓN | 14/07/2023 |
| CUARTA SESIÓN DE VALORACIÓN | 16/10/2023 |

Observaciones:

En este centro se tienen realizan terapias que estimulan el desarrollo cerebral de los niños. Dichas terapias constan de programas tanto fisiológicos y sensoriales como físicos e intelectuales. Para diseñar de forma individualizada los programas que cada niño necesita, el niño y su familia realizan una visita de dos días al centro. El primer día es dedicado a la evaluación, audiometría e historial del menor por la mañana y la realización de la revisión médica y el resumen de todo lo realizado durante el día por la tarde. Al día siguiente se realiza una conferencia de formación y se presenta el programa fisiológico y sensorial por la mañana y por la tarde se presentan los programas físico e intelectual. Dichas visitas son realizadas cada 6 meses como seguimiento de la evolución del niño.

Informe Final:

La dinámica del centro hacia los niños con Trastorno del Espectro Autista, al tratarse de visitas de dos días, cubren al completo todas las necesidades que tiene el niño a tratar, desde las fisiológicas y sensoriales hasta las físicas e intelectuales. Al tener dos días seguidos de visita, de los cuales se dedican bastantes horas a todo el proceso de evaluación, se realizan de manera personalizada los programas que el niño necesita y las pautas que deben seguir los padres y/o tutores del menor en todos los aspectos a tratar. Todos los aspectos de los indicadores están bien cubiertos con creces, a seguir así.

Guardar Cancelar

OTEA, José Luis Cuesta Gómez, jlcgomez@ubu.es

Figura A.17: Informe final con el botón de la impresora seleccionado

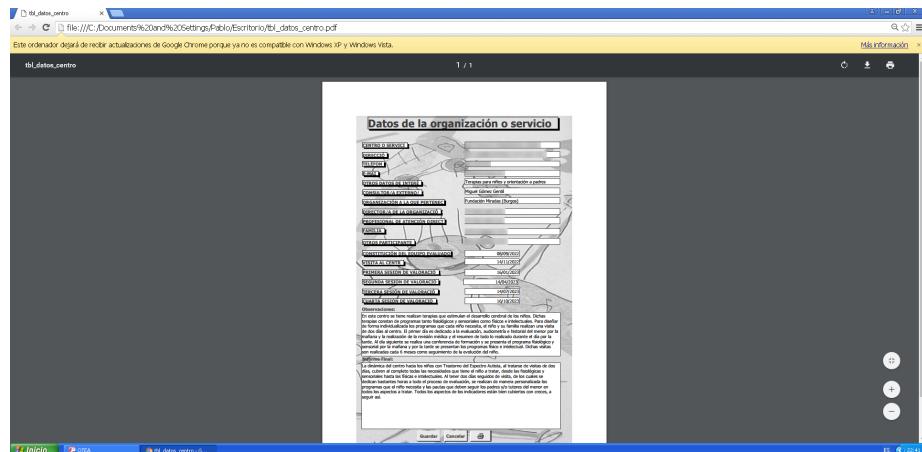


Figura A.18: Informe en PDF generado por el programa en *Microsoft Access*

Requisitos de uso del programa

Para poder ejecutar este programa en un equipo moderno, se ha tenido que instalar una máquina virtual de *Oracle VirtualBox* cuyo sistema operativo sea *Windows XP*, todo ello debido a la antigüedad de dicho programa. También se ha tenido que instalar una versión antigua del paquete de aplicaciones de ofimática *Microsoft Office*, en concreto se ha instalado *Microsoft Office 2007*, ya que tiene un mejor soporte para la aplicación de *OTEA*.

Diferencias entre *Azure* y las aplicaciones de bases de datos locales

Las diferencias que tiene *Azure* con respecto a las aplicaciones de gestión de bases de datos a nivel local, como es el caso de *Microsoft Access* o *OpenOffice Database*, son las siguientes:

- En las aplicaciones de bases de datos locales es preciso utilizar un ordenador con un fichero *.accdb* en *Microsoft Access* o un fichero *.odb* en *OpenOffice Database*, el cual aloje todos los registros de los indicadores y sus respectivas incidencias, cuya difusión depende de la cantidad de personas que tengan ese fichero. En cambio, con Microsoft Azure, la difusión es más sencilla puesto que no se necesita un fichero en cada uno de los dispositivos, ya que al alojarse los registros de los datos y de las correspondientes incidencias en la nube, permite de mejor manera la implementación en diferentes dispositivos, aparte de que solo el administrador tiene acceso a la base de datos.
- En una aplicación que se apoya en *Azure* es necesaria una conexión a internet para poder cargar los datos correctamente al servidor, al igual que para realizar todas las demás operaciones en las que esté involucrada la base de datos, por lo tanto si estuviese caído el servidor donde se ha implementado Azure, no se tendría el comportamiento esperado en la aplicación. En cambio, en el caso de la base de datos en *Microsoft Access* o en *OpenOffice Database*, como el comportamiento de la base de datos depende de que esté alojado su correspondiente fichero *.accdb* o *.odb* en una cantidad determinada de dispositivos, no se tendría esta problemática, salvo en el hipotético caso en que ninguno de los dispositivos que cuenten con el fichero de la base de datos se encuentre operativo.
- En el caso de implementar una base de datos en *Azure*, se puede implementar de mejor manera en una aplicación como la de la Fundación

Miradas, puesto que se espera que dicha aplicación reciba y envíe transacciones del cliente, que es el encargado de la Fundación Miradas que evalúa el correcto cumplimiento de los indicadores y de sus respectivas incidencias o el representante de la asociación de ayuda a la discapacidad que está siendo evaluada que comprueba los diferentes resultados realizados de las diferentes test, hacia el servidor que se encarga de alojar cada uno de los datos de las asociaciones evaluadas y de cada uno de los mencionados diagnósticos, proporcionando inmediatez y automatización en el proceso de muestra de resultados para todos los interesados mencionados con anterioridad. En cambio, esta tarea es más tediosa en la implementación original de la aplicación en Access, puesto que los resultados obtenidos sólo se mostrarían de forma inmediata en el equipo que corre dicha aplicación, por lo que sería necesario transportar el informe resultante manualmente, ya sea mediante un dispositivo de almacenamiento físico externo o por correo electrónico.

- *Azure*, al estar enfocada a alojar grandes cantidades de datos por parte de empresas, no dispone de una versión gratuita de forma permanente, sino que sólo unos cuantos servicios son gratuitos, mientras que hay ciertos que también son gratuitos, pero únicamente durante doce meses, mientras que otros se obtienen mediante las diferentes suscripciones de las que dispone *Azure*. En cambio, sucede lo contrario con las aplicaciones de bases de datos a nivel local, como *Microsoft Access*, cuya versión gratuita se puede encontrar en OneDrive como aplicación web, como con *OpenOffice Database*, el cual forma parte del paquete de ofimática de libre distribución *OpenOffice*. En el caso del paquete de *Microsoft Office*, donde se incluye *Microsoft Access*, también es un software de pago el cual dispone de licencias desde un mes hasta los doce.
- En *Azure* se garantiza la seguridad de los datos proporcionados gracias al cifrado de los datos en reposo, el cual se produce en tres niveles:
 - **A nivel de almacenamiento en el servidor** el servicio *Azure Storage*, el cual se encarga del almacenamiento de los datos, realiza un encriptado del servicio *SSE*, concretamente los datos se cifran y descifran de forma transparente mediante el cifrado *AES* de 256 bits, uno de los cifrados de bloques más sólidos que hay disponibles, y son compatibles con *FIPS 140-2*.
 - **A nivel de cliente**, la correspondiente biblioteca de *Azure Blob Storage* usa *AES* para cifrar los datos del usuario. Hay dos

versiones de cifrados de cliente disponibles en la biblioteca de cliente:

- La versión 2 utiliza el modo *Galois/Contador (GCM)* con *AES*.
- La versión 1 utiliza el modo *Cipher Block Chain (CBC)* con *AES*.
- **A nivel de disco duro del sistema operativo**, permite cifrar los discos del sistema operativo y los discos de datos usados por una máquina virtual *IaaS*. Dicho proceso se encarga de realizarlo *Azure Disk Encryption*, el cual ayuda a custodiar y proteger los datos con el objetivo de cumplir los compromisos de cumplimiento y seguridad. Usa la característica *DM-Crypt* de Linux para proporcionar cifrado de volumen tanto a los discos de datos como a los del sistema operativo de máquinas virtuales (VM) de *Azure* y se integra con *Azure Key Vault* para ayudarle a controlar y administrar las claves y los secretos del cifrado de disco.

En cambio, con las bases de datos locales, la seguridad depende de quien disponga el fichero correspondiente y del correcto uso que tenga del mismo.

A.5. Planificación temporal

| Actividad | Período | Tiempo |
|---|-------------------------|-----------|
| Búsqueda de Trabajos de Final de Grado de Referencia | 26/10/2022 - 16/11/2022 | 10 horas |
| Pruebas y detección de mejoras de la aplicación en Access | 15/11/2022 - 17/01/2023 | 28 horas |
| Formación en Azure | 9/6/2023 - 6/7/2023 | 50 horas |
| Análisis del entorno | 25/01/2023 - 10/6/2023 | 40 horas |
| Diseño de la aplicación | 25/01/2023 - 10/6/2023 | 90 horas |
| Pruebas de la aplicación | 10/6/2023 - 6/7/2023 | 20 horas |
| Actualización de GitHub | 12/12/2022 - 6/7/2023 | 2 horas |
| Desarrollo de la memoria y de los anexos | 26/10/2022 - 6/7/2023 | 300 horas |

Tabla A.1: Tiempo invertido en cada actividad

A.6. Estudio de viabilidad

Viabilidad económica

En cuanto a la viabilidad económica de este proyecto, cabe resaltar que esta se basa única y exclusivamente en los costes de mantenimiento y escalado del servidor. Podemos comprobar esa información en el propio portal de Azure, para ser más precisos, en el apartado de información general de la suscripción que se tiene activa por parte del usuario:

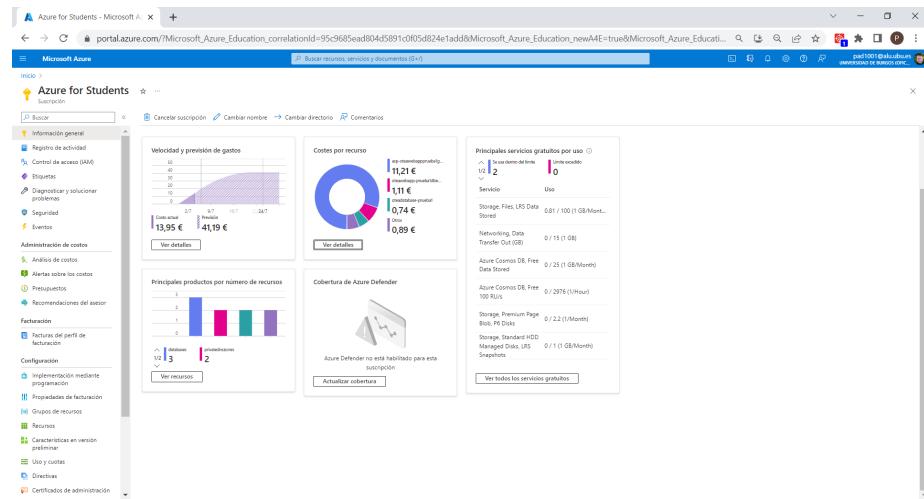


Figura A.19: Información general de la suscripción en Azure

Como se puede comprobar en esta pantalla, el análisis del coste puede medirse mediante diferentes métricas:

- **Velocidad y previsión de gastos:** Se trata de un gráfico acumulativo con los gastos totales que se han tenido en la suscripción. En la figura anterior aparecen costes muy bajos, ya que se está utilizando la base de datos más básica, la de la versión de prueba del servicio web, que se ha ido manteniendo con los 200\$ de crédito gratuito disponibles (*Véase apartado de técnicas y herramientas de la memoria*).
- **Coste por recurso:** Es un gráfico circular el cual muestra el porcentaje de gasto total para cada uno de los recursos utilizados.

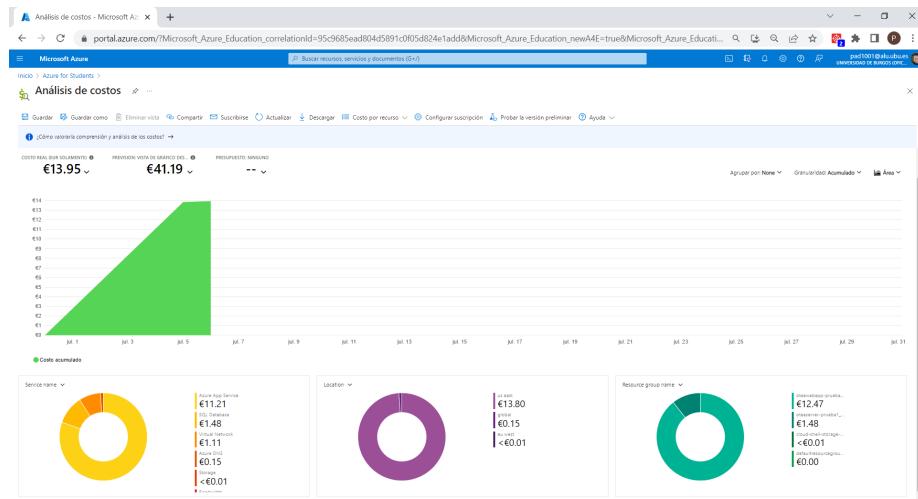


Figura A.20: Velocidad y previsión de gastos junto con los costes por cada recurso

Como se ha podido comprobar, la previsión de costes ayuda a conocer la tendencia del gasto a partir del gasto acumulado total, mientras que los recursos del servidor de la base de datos y la propia base de datos son los que más consumen. Estos datos son de una versión básica, pero con mayor escalabilidad y prestaciones, el gasto se amplía y a su vez se mantienen esos porcentajes.

Cabe resaltar también que Azure dispone de diferentes herramientas las cuales se utilizan para realizar buenas prácticas en estudios de viabilidad a nivel económico. La propia suscripción da la posibilidad de programar alertas si se supera un cierto nivel de gasto, al igual que también dar la posibilidad de generar presupuestos que ayuden a manejar dichos costes. Todas estas herramientas pertenecen al servicio de *Cost Management* de Azure, disponibles para cualquier tipo de suscripción.

Viabilidad legal

En cuanto a la viabilidad legal, hay que tener en cuenta que la base de datos trabaja con datos catalogados como muy sensibles, como es el caso de números telefónicos o contraseñas. Todas las organizaciones que tratan con grandes volúmenes de datos están sujetas a la ley en materia de protección de datos de cada país, siendo el caso de España la conocida Ley Orgánica de Protección de Datos.

La LOPD es una ley que tiene como objetivo adaptarse al *Reglamento*

Oficial de Protección de Datos de la Unión Europea, en la cual se pretende garantizar que la libre circulación de los datos esté protegida. Dicha ley exige que se acepte que la empresa, en este caso la *Fundación Miradas*, esté obligada a solicitar el permiso a los usuarios para que los datos de usuarios y organizaciones para poder ser tratados con fines estadísticos, dentro de lo marcado por la LOPD o las leyes equivalentes en otros países en esta materia:



Miguel

Gomez Gentil

mgg0174@ubu.es

.....

687878787

Usuario de la Fundación Miradas



Autismo Gamonal



Acepto que la Fundación Miradas guarde registro
sobre mis datos de acuerdo con la Ley Orgánica
de Protección de Datos

REGISTRARSE

En esta captura, cuyos datos no son verdaderos, al tener activada la casilla de aceptación, da la posibilidad de registrarse al usuario sin ningún problema.

Apéndice B

Especificación de Requisitos

B.1. Introducción

En este apéndice se desarrollan los aspectos relacionados con la obtención de los requisitos, tanto funcionales como no funcionales, y su posterior especificación mediante los casos de uso. Cada requisito representa una acción o actividad esperada por parte de la aplicación para un correcto funcionamiento de la misma, siendo relacionado a posteriori con un caso de uso. Al tener todos los casos de uso, se relacionan posteriormente con los actores correspondientes.

También se desarrolla en este apéndice, los diferentes tipos de actores o usuarios que están involucrados en la utilización de la aplicación, los cuales son relacionados a posteriori con los diferentes casos de uso de la aplicación, con el fin de conocer quién es el encargado de realizar cada actividad relacionada con cada caso de uso.

B.2. Objetivos generales

El análisis de requisitos, actores y casos de uso para la aplicación tiene como finalidad realizar un estudio sobre qué actividades se espera que realice la aplicación, quién tiene los permisos necesarios para realizarlas y qué resultados se esperan de su realización.

En cuanto a la obtención de requisitos, se tienen que tener en cuenta tanto los requisitos funcionales, los cuales indican los objetivos que debería marcar la propia aplicación, como los requisitos no funcionales, que se encargan

de marcar las limitaciones que tiene la aplicación. Estos requisitos deben utilizarse posteriormente para la creación de los diferentes casos de uso que abarca la aplicación, con el objetivo de tener que cubrir todo lo esperable dentro del funcionamiento de la misma.

Los requisitos utilizados en este proyecto aparecen en el apartado **Catálogo de requisitos** de este apéndice.

En cuanto a los casos de uso posteriores, se tienen que identificar los actores que están involucrados en el uso de la aplicación, siendo en el caso de esta aplicación usuarios repartidos en cuatro tipos diferentes, teniendo cada uno de ellos diferentes funciones y permisos, siendo éstos últimos representados mediante los diferentes casos de uso dentro de la aplicación:

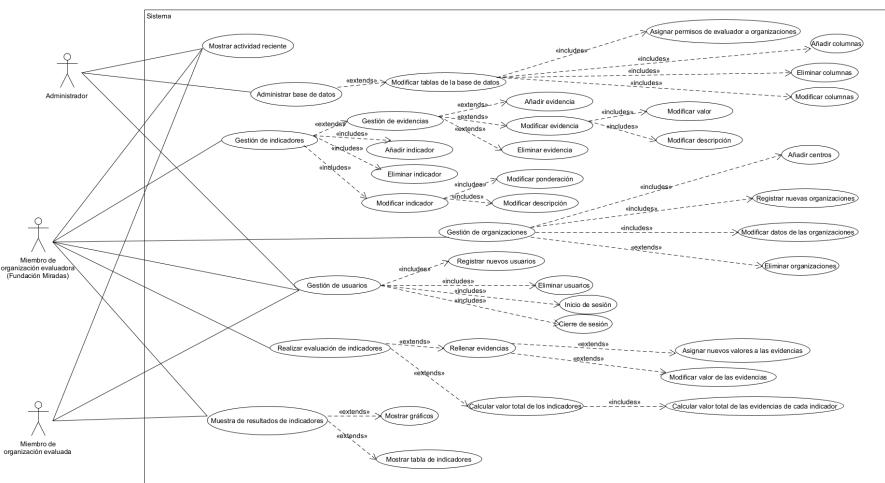


Figura B.1: Diagrama de casos de uso de la aplicación

1 Como se puede comprobar en este diagrama, se distinguen cuatro tipos distintos de usuario con diferentes permisos dentro de la aplicación:

- **Administrador:** Es el usuario encargado de gestionar todas las operaciones relacionadas con las organizaciones y con los usuarios almacenados en la base de datos. En cuanto a las organizaciones, este usuario se encarga de dar de alta o de baja a las organizaciones que así lo deseen, además de proporcionar los permisos de evaluador a las organizaciones evaluadoras y modificar datos de las mismas. En cuanto a los usuarios, las operaciones a realizar son las mismas que en el caso de las organizaciones, con la excepción de dar permisos de evaluación, al ser éste un requisito exclusivo de las organizaciones.

- **Miembro de la organización evaluadora:** Es un usuario cuya organización a la que pertenece posee permisos de evaluación sobre otras organizaciones que no los tienen, llenando los valores de las evidencias y calculando el valor total de cada indicador a partir de las mismas, también calculando el valor total obtenido mediante el valor total de cada indicador. También tiene permisos para modificar dichos indicadores durante la evaluación. Además de eso puede observar los resultados de cada evaluación de indicadores, ya sea mediante tablas o mediante gráficos.
- **Miembro de la organización evaluada:** Es un usuario cuya organización a la que pertenece posee no permisos de evaluación, por lo tanto, dicho usuario pertenece a una organización que está a disposición de ser evaluada. Además de recibir la evaluación por parte de la organización evaluadora, también puede observar los resultados de las diferentes evaluaciones, ya sea mediante tablas o mediante gráficos.

La especificación de los requisitos funcionales en forma de casos de uso aparece en el apartado **Especificación de requisitos** de este apéndice.

B.3. Catálogo de requisitos

Como se ha mencionado en los apartados anteriores de este apéndice, el catálogo de requisitos resume el comportamiento que se desea obtener por parte de la aplicación, a partir de todas las actividades que se desean implementar en la aplicación.

Hay que considerar también que a medida que se va desarrollando la aplicación, se van introduciendo mejoras y a la par realizando cambios debido a que la aplicación debe adaptarse en la medida de lo posible al objetivo de poder ser utilizada por la *Fundación Miradas*, por las organizaciones a las que evalúa y por los administradores de la base de datos, por lo que durante todo el tiempo de desarrollo se ha tomado conciencia de lo que se pretende implementar, empleando para ello los siguientes requisitos funcionales:

- **RF-1:** Todos los usuarios tienen la capacidad de mostrar su actividad reciente, mostrándose en la app cuál fue la última acción que se realizó en la app.
- **RF-2:** Como usuario de la *Fundación Miradas*, se desea que la aplicación pueda realizar la evaluación de indicadores a cada una de las

organizaciones a evaluar, teniendo en cuenta el caso específico del afectado con TEA, rellenando así las evidencias que son cumplidas por la organización evaluada.

- **RF-3:** Como usuario de la *Fundación Miradas*, se desea que tras realizar el test de indicadores, se realice la suma ponderada de todas las evidencias de todos los indicadores, para poder obtener así los resultados deseados.
- **RF-4:** Tanto los usuarios de organizaciones evaluadas como los usuarios de la *Fundación Miradas* pueden mostrar los resultados de cada uno de los test de indicadores que se realizan mediante gráficos.
- **RF-5:** Tanto los usuarios de organizaciones evaluadas como los usuarios de la *Fundación Miradas* pueden mostrar los resultados de cada uno de los test de indicadores que se realizan mediante la muestra del resultado total.
- **RF-6:** Todos los usuarios tienen la capacidad de poder iniciar sesión.
- **RF-7:** Todos los usuarios tienen la capacidad de poder cerrar sesión.
- **RF-8:** Todos los usuarios tienen la capacidad de poder registrarse en la aplicación.
- **RF-9:** Todos los usuarios tienen la capacidad de poder eliminarse de la aplicación.
- **RF-10:** Los usuarios de la *Fundación Miradas* tienen la capacidad de modificar la ponderación y la descripción de los indicadores.
- **RF-11:** Los usuarios de la *Fundación Miradas* tienen la capacidad de añadir nuevos indicadores.
- **RF-12:** Los usuarios de la *Fundación Miradas* tienen la capacidad de eliminar indicadores.
- **RF-13:** Los usuarios de la *Fundación Miradas* tienen la capacidad de añadir evidencias
- **RF-14:** Los usuarios de la *Fundación Miradas* tienen la capacidad de eliminar evidencias
- **RF-15:** Los usuarios de la *Fundación Miradas* tienen la capacidad de modificar el valor y la descripción evidencias.

- **RF-16:** Los usuarios de la *Fundación Miradas* tienen la capacidad de registrar nuevas organizaciones.
- **RF-17:** Los usuarios de la *Fundación Miradas* tienen la capacidad de modificar los datos de las organizaciones.
- **RF-18:** Los usuarios de la *Fundación Miradas* tienen la capacidad de eliminar organizaciones.
- **RF-19:** Los usuarios de la *Fundación Miradas* tienen la capacidad de añadir nuevos centros de las organizaciones.
- **RF-20:** El administrador de la base de datos tiene la capacidad de añadir columnas a las tablas que conforman dicha base de datos.
- **RF-21:** El administrador de la base de datos tiene la capacidad de eliminar columnas de las tablas que conforman dicha base de datos.
- **RF-22:** El administrador de la base de datos tiene la capacidad de modificar columnas de las tablas que conforman dicha base de datos.

B.4. Especificación de requisitos

| CU-1 | Mostrar actividad reciente |
|---------------------------------|--|
| Versión | 1.0 |
| Autor | Pablo Ahíta del Barrio |
| Requisitos asociados | RF-1 |
| Descripción | Se muestra la actividad reciente del usuario |
| Precondición | Estar registrado en la aplicación y tener la sesión activa |
| Acciones | |
| Postcondición | Ninguna |
| Excepciones | Ninguna |
| Importancia | Media |

Tabla B.1: CU-1 Mostrar actividad reciente

| CU-2 Administrar base de datos | |
|--|--|
| Versión | 1.0 |
| Autor | Alumno |
| Requisitos asociados | RF-20, RF-21, RF-22 |
| Descripción | Realización de operaciones de modificación de las tablas de la base de datos |
| Precondición | Estar registrado como administrador en la app y tener la sesión activa |
| Acciones | <ol style="list-style-type: none"> 1. Añadir columnas de las tablas 2. Eliminar columnas de las tablas 3. Modificar columnas de las tablas 4. Asignar permisos de evaluador a organizaciones |
| Postcondición | <ol style="list-style-type: none"> 1. Tabla con columna añadida 2. Tabla con columna eliminada 3. Tabla con columna modificada 4. Organización transformada en evaluadora |
| Excepciones | Ninguna |
| Importancia | Alta |

Tabla B.2: CU-2 Administrar base de datos.

| CU-3 | Gestión de indicadores |
|-----------------------------|--|
| Versión | 1.0 |
| Autor | Pablo Ahíta del Barrio |
| Requisitos asociados | RF-10, RF-11, RF-12, RF-13, RF-14, RF-15 |
| Descripción | Realización de operaciones de gestión de indicadores |
| Precondición | Estar registrado como usuario de la <i>Fundación Miradas</i> en la app y tener la sesión activa |
| Acciones | <ol style="list-style-type: none"> 1. Modificar ponderación del indicador 2. Modificar descripción del indicador 3. Añadir indicador 4. Eliminar indicador 5. Añadir evidencia 6. Modificar valor de la evidencia 7. Modificar descripción de la evidencia 8. Eliminar evidencia |
| Postcondición | <ol style="list-style-type: none"> 1. Indicador con ponderación cambiada 2. Indicador con descripción modificada 3. Nuevo indicador añadido 4. Indicador eliminado 5. Nueva evidencia añadida 6. Evidencia con valor modificado 7. Evidencia con descripción modificada 8. Evidencia eliminada |
| Excepciones | No se puede tener un indicador sin evidencias o no tener indicadores |
| Importancia | Alta |

Tabla B.3: CU-3 Gestión de indicadores.

| CU-4 | Gestión de organizaciones |
|-----------------------------|---|
| Versión | 1.0 |
| Autor | Pablo Ahíta del Barrio |
| Requisitos asociados | RF-16, RF-17, RF-18, RF-19 |
| Descripción | Realización de operaciones de gestión de organizaciones |
| Precondición | Estar registrado como usuario de la <i>Fundación Miradas</i> en la app y tener la sesión activa |
| Acciones | <ol style="list-style-type: none"> 1. Añadir nuevos centros de las organizaciones 2. Registrar nuevas organizaciones 3. Modificar datos de las organizaciones 4. Eliminar organizaciones |
| Postcondición | <ol style="list-style-type: none"> 1. Un centro más para la organización 2. Una organización más en la base de datos 3. Organización modificada 4. Una organización menos en la base de datos |
| Excepciones | No se pueden eliminar organizaciones evaluadoras, al tratarse de la <i>Fundación Miradas</i> . Sería responsabilidad del administrador de la base de datos añadir nuevos organismos evaluadores. |
| Importancia | Alta |

Tabla B.4: CU-4 Gestión de organizaciones.

| CU-5 | Gestión de usuarios |
|-----------------------------|--|
| Versión | 1.0 |
| Autor | Pablo Ahita del Barrio |
| Requisitos asociados | RF-6, RF-7, RF-8, RF-9 |
| Descripción | Realización de operaciones de gestión de usuarios |
| Precondición | Ninguna |
| Acciones | <ul style="list-style-type: none"> 1. Registrar nuevos usuarios 2. Eliminar usuarios 3. Iniciar sesión 4. Cerrar sesión |
| Postcondición | <ul style="list-style-type: none"> 1. Nuevo usuario registrado 2. Usuario eliminado 3. Sesión iniciada 4. Sesión cerrada |
| Excepciones | No se puede cerrar una sesión inactiva ni se puede registrar un usuario ya existente. |
| Importancia | Alta |

Tabla B.5: CU-4 Gestión de usuarios.

| CU-6 | Realizar evaluación de indicadores |
|-----------------------------|--|
| Versión | 1.0 |
| Autor | Pablo Ahíta del Barrio |
| Requisitos asociados | RF-2, RF-3 |
| Descripción | Realización de la evaluación de indicadores, llenando todas las evidencias correspondientes. Luego se calculan los resultados |
| Precondición | Estar registrado como usuario de la <i>Fundación Miradas</i> en la app, tener la sesión activa, que la organización evaluada esté registrada en la app y que el representante y director de la organización evaluada estén registrados como miembros de dicha organización en la app |
| Acciones | <ol style="list-style-type: none"> 1. Rellenar evidencias 2. Calcular valor total de la evaluación |
| Postcondición | <ol style="list-style-type: none"> 1. Evidencias llenadas 2. Valor total calculado |
| Excepciones | No se pueden eliminar organizaciones evaluadoras, al tratarse de la <i>Fundación Miradas</i> . Sería responsabilidad del administrador de la base de datos añadir nuevos organismos evaluadores. |
| Importancia | Alta |

Tabla B.6: CU-6 Realizar evaluación de indicadores.

| CU-7 | Muestra de evaluación de indicadores |
|-----------------------------|---|
| Versión | 1.0 |
| Autor | Pablo Ahita del Barrio |
| Requisitos asociados | RF-4, RF-5 |
| Descripción | Realización de la evaluación de indicadores, llenando todas las evidencias correspondientes. Luego se calculan los resultados |
| Precondición | Estar registrado como usuario de la <i>Fundación Miradas</i> o como usuario de una organización evaluada en la app y tener la sesión activa |
| Acciones | <ol style="list-style-type: none"> 1. Mostrar gráficos 2. Mostrar tablas de indicadores |
| Postcondición | Ninguna |
| Excepciones | Ninguna |
| Importancia | Media |

Tabla B.7: CU-6 Realizar evaluación de indicadores.

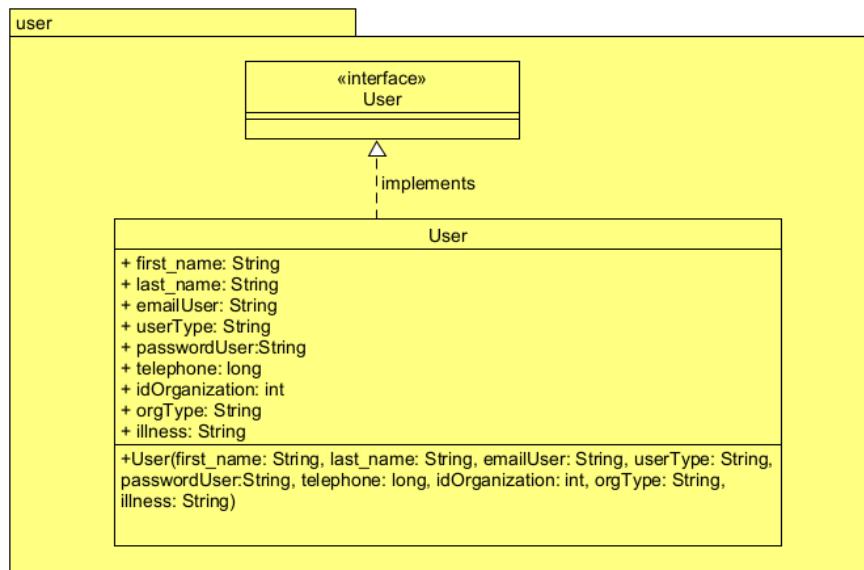
Apéndice C

Especificación de diseño

C.1. Introducción

C.2. Diseño de datos

En cuanto al diseño de los datos, se han utilizado las siguientes clases o entidades, que son idénticas tanto en la base de datos, como en el servidor, como en el cliente:

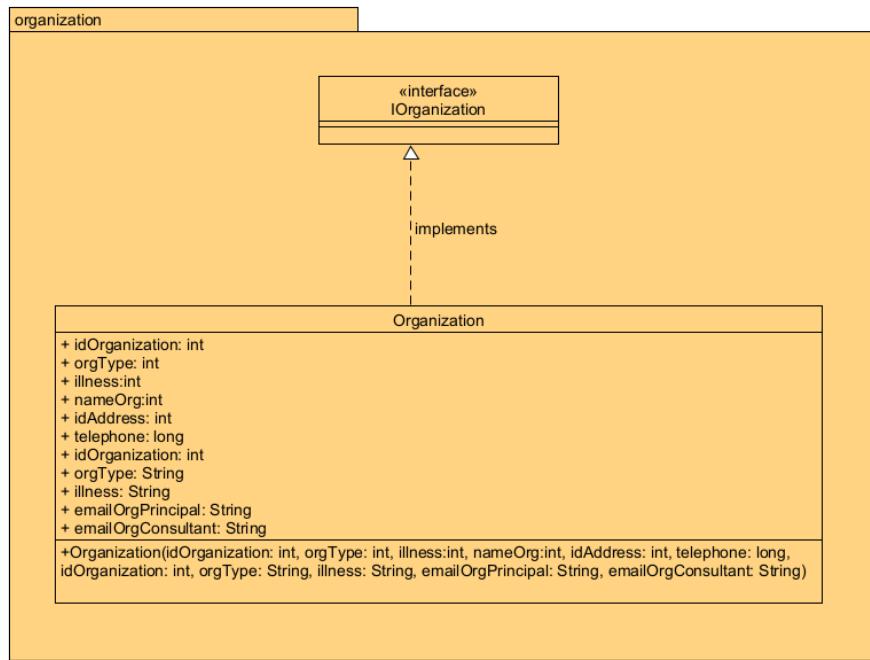
Figura C.1: Paquete `user`

- **User:** Esta entidad se encarga de almacenar información sobre los usuarios registrados en la base de datos. Dicha entidad consta de los siguientes campos:
 - **emailUser:** Almacena la dirección de correo electrónico del usuario. Es de tipo `String` o `VARCHAR(500)`
 - **userType:** Indica el tipo de usuario, que puede ser `ADMIN` o `ORGANIZATION`. Es de tipo `String` o `VARCHAR(50)`
 - **passwordUser:** Almacena la contraseña del usuario. Es de tipo `String` o `VARCHAR(500)`.
 - **first_name:** Almacena el nombre del usuario. Es de tipo `String` o `VARCHAR(500)`.
 - **last_name:** Almacena el apellido del usuario. Es de tipo `String` o `VARCHAR(500)`.
 - **telephone:** Almacena el número de teléfono del usuario. Es de tipo `long` o `BIGINT`
 - **idOrganization:** Almacena el identificador de la organización a la que pertenece el usuario. En caso de que `userType=ADMIN`, ese valor es un nulo representado tanto en cliente como en servidor como -1. Es de tipo `INT`

- **organizationType:** Indica el tipo de la organización, que puede ser ".EV ALUATED." o ".EV ALUATOR". En caso de que userType=ADMIN, ese valor es un nulo representado tanto en cliente como en servidor como ". Es de tipo String o VARCHAR(50).
 - **illness:** Almacena el tipo de enfermedad relacionada con la organización. En caso de que userType=ADMIN, ese valor es un nulo representado tanto en cliente como en servidor como ". Es de tipo String o VARCHAR(50).
 - **PRIMARY KEY:** La clave primaria de esta tabla es el campo ".emailUser", considerando que no se pueden añadir dos emails iguales.
 - **CHECK:** Define una serie de condiciones que los datos deben cumplir para ser válidos, comprobando léxicamente la clave primaria.
- **City:** Esta entidad se utiliza como carga de las ciudades españolas, utilizando para ello los siguientes campos:
- **idCity:** Es el identificador de la ciudad. Es de tipo INT.
 - **idProvince:** Es el identificador de la provincia a la que pertenece la ciudad. Es de tipo INT.
 - **idRegion:** Es el identificador de la región o comunidad autónoma a la que pertenece la ciudad. Es de tipo INT.
 - **idCountry:** Es el identificador del país al que pertenece la ciudad. Es de tipo String o VARCHAR(50)
 - **cityName:** Es el nombre de la ciudad. Es de tipo String o VARCHAR(500)
 - **PRIMARY KEY:** La clave primaria está conformada por las columnas idCity, idProvince, idRegion e idCountry.
 - **FOREIGN KEY:** Los campos idProvince, idRegion e idCountry establecen una relación directa con la entidad Province.
- **Province:** Esta entidad se utiliza como carga de las provincias españolas, utilizando para ello los siguientes campos:
- **idProvince:** Almacena el identificador de la provincia. Es de tipo INT.
 - **idRegion:** Almacena el identificador de la región a la que pertenece la provincia. Es de tipo INT.
 - **idCountry:** Almacena el identificador del país al que pertenece la provincia. Es de tipo String o VARCHAR(50)

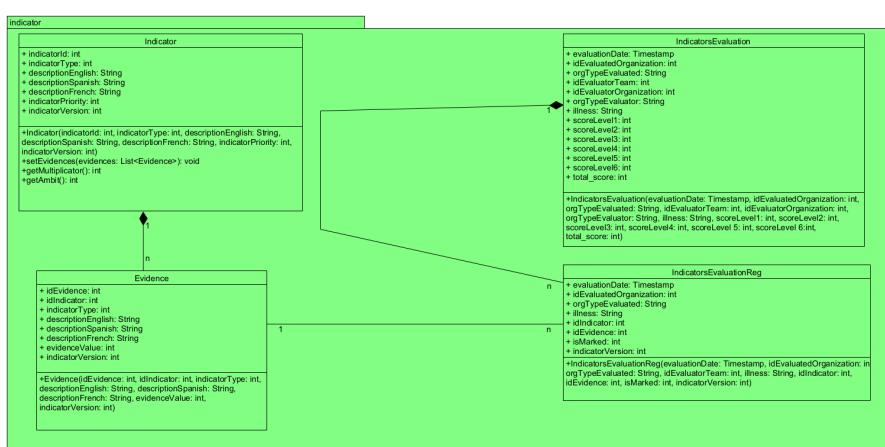
- **nameProvince:** Almacena el nombre de la provincia. Es de tipo **String** o **VARCHAR(500)**
- **PRIMARY KEY:** La clave principal de esta tabla está compuesta por los campos **idProvince**, **idRegion** e **idCountry**.
- **FOREIGN KEY:** Establece una relación con la tabla **regions**.^a través de los campos **idRegion** e **idCountry**.
- **Region:** Esta entidad se utiliza como carga de las comunidades autónomas españolas, utilizando para ello los siguientes campos:
 - **idRegion:** Almacena el identificador de la región. Es de tipo **INT**.
 - **idCountry:** Almacena el identificador del país al que pertenece la región. Es de tipo **String** o **VARCHAR(50)**.
 - **nameRegion:** Almacena el nombre de la región. Es de tipo **String** o **VARCHAR(500)**.
 - **PRIMARY KEY:** La clave principal de esta tabla está compuesta por los campos **idRegion** e **idCountry**.
 - **FOREIGN KEY:** Establece una relación con la entidad **Country** a través del campo **idCountry**.
- **Country:** Esta entidad se utiliza como carga de las ciudades españolas, utilizando para ello los siguientes campos:
 - **idCountry:** Es el identificador del país. Es de tipo **String** o **VARCHAR(500)**.
 - **nameEnglish:** Es el nombre del país en inglés. Es de tipo **(String)** o **VARCHAR(500)**.
 - **nameSpanish:** Es el nombre del país en español. Es de tipo **(String)** o **VARCHAR(500)**.
 - **nameFrench:** Es el nombre del país en inglés. Es de tipo **(String)** o **VARCHAR(500)**.
 - **PRIMARY KEY:** La clave primaria de esta entidad es el campo **idCountry**
- **Address:** Esta entidad se utiliza para almacenar información sobre las diferentes direcciones que se almacenan para organizaciones y centros, utilizando para ello los siguientes campos:
 - **idAddress:** Almacena el identificador de la dirección. Es de tipo **INT**.

- **addressName:** Almacena el nombre de la dirección.Es de tipo String o VARCHAR(500).
- **zipCode:** Almacena el código postal de la dirección.Es de tipo INT.
- **idCity:** Almacena el identificador de la ciudad.Es de tipo INT.
- **idProvince:** Almacena el identificador de la provincia.Es de tipo INT.
- **idRegion:** Almacena el identificador de la región.Es de tipo INT.
- **idCountry:** Almacena el identificador del país al que pertenece la dirección.Es de tipo String o VARCHAR(50).
- **nameCity:** Almacena el nombre de la ciudad (solo para ciudades fuera de España).Es de tipo String o VARCHAR(500).
- **nameProvince:** Almacena el nombre de la provincia (solo para provincias fuera de España).Es de tipo String o VARCHAR(500).
- **nameRegion:** Almacena el nombre de la región (solo para regiones fuera de España).Es de tipo String o VARCHAR(500).
- **PRIMARY KEY:** La clave primaria de esta entidad es el campo **idAddress**
- **FOREIGN KEY:** Establece una relación con la entidad **Country** a través del campo **idCountry**.
- **FOREIGN KEY:** Establece una relación con la entidad **City** a través de los campos **idCity**, **idProvince**, **idRegion** e **idCountry** (solo si es una ciudad española).

Figura C.2: Paquete `organization`

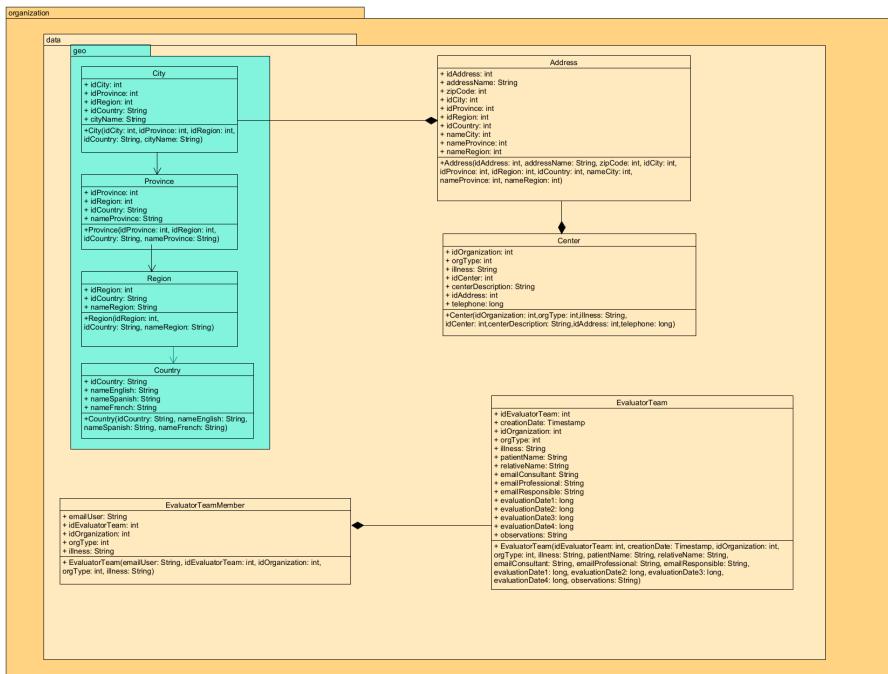
- **Organization:** Esta tabla se encarga de almacenar la información sobre las organizaciones registradas de todos los tipos, utilizando para ello los siguientes campos:
 - **IdOrganization:** Almacena el identificador de la organización. Es de tipo **INT**.
 - **orgType:** Indica el tipo de la organización. Es de tipo **String** o **VARCHAR(50)**.
 - **illness:** Almacena el tipo de enfermedad relacionada con la organización. Es de tipo **String** o **VARCHAR(50)**.
 - **nameOrg:** Almacena el nombre de la organización. Es de tipo **String** o **VARCHAR(500)**.
 - **idAddress:** Almacena el identificador de la dirección de la organización. Es de tipo **INT**.
 - **email:** Almacena la dirección de correo electrónico de la organización. Es de tipo **String** o **VARCHAR(500)**.
 - **telephone:** Almacena el número de teléfono de la organización. Es de tipo **long** o **BIGINT**.

- **information:** Almacena información sobre la organización. Es de tipo **String** o **VARCHAR(500)**.
- **emailOrgPrincipal:** Almacena la dirección de correo electrónico del usuario principal de la organización. Es de tipo **String** o **VARCHAR(500)**.
- **PRIMARY KEY:** La clave principal de esta tabla está compuesta por los campos **IdOrganization**, **orgType** e **illness**.
- **FOREIGN KEY:** Establece una relación con la entidad **Address** a través del campo **idAddress**.

Figura C.3: Paquete **indicator**

- **Indicator:** Esta tabla se encarga de almacenar información sobre los indicadores almacenados en la base de datos, utilizando para ello los siguientes campos:
 - **indicatorId:** Almacena el identificador del indicador. Es de tipo **INT**.
 - **indicatorType:** Indica el tipo de indicador. Es de tipo **String** o **VARCHAR(50)**.
 - **descriptionEnglish:** Almacena la descripción en inglés del indicador. Es de tipo **String** o **VARCHAR(5000)**.
 - **descriptionSpanish:** Almacena la descripción en español del indicador. Es de tipo **String** o **VARCHAR(5000)**.

- **descriptionFrench:** Almacena la descripción en francés del indicador. Es de tipo String o VARCHAR(5000).
 - **indicatorPriority:** Almacena la prioridad del indicador. Es de tipo INT.
 - **indicatorVersion:** Almacena la versión del indicador. Es de tipo INT.
 - **isActive:** Indica si el indicador está en uso o no. Es de tipo INT.
 - **PRIMARY KEY:** La clave principal de esta tabla está compuesta por los campos **indicatorId**, **indicatorType** e **indicatorVersion**.
- **Evidence:** Esta entidad se encarga de almacenar información sobre todas las evidencias de cada uno de los indicadores, utilizando para ello los siguientes campos:
- **idEvidence:** Almacena el identificador de la evidencia. Es de tipo INT.
 - **idIndicator:** Almacena el identificador del indicador al que pertenece la evidencia. Es de tipo INT.
 - **indicatorType:** Indica el tipo de indicador. Es de tipo String o VARCHAR(50).
 - **descriptionEnglish:** Almacena la descripción en inglés de la evidencia. Es de tipo String o VARCHAR(5000).
 - **descriptionSpanish:** Almacena la descripción en español de la evidencia. Es de tipo String o VARCHAR(5000).
 - **descriptionFrench:** Almacena la descripción en francés de la evidencia. Es de tipo String o VARCHAR(5000).
 - **evidenceValue:** Almacena el valor de la evidencia. Es de tipo INT.
 - **indicatorVersion:** Almacena la versión del indicador. Es de tipo INT.
 - **PRIMARY KEY:** La clave principal de esta tabla está compuesta por los campos **idEvidence**, **idIndicator**, **indicatorType** e **indicatorVersion**.
 - **FOREIGN KEY:** Establece una relación con la entidad **Indicator** a través de los campos **idIndicator**, **indicatorType** e **indicatorVersion**.

Figura C.4: Paquete `orgData`

- **Center:** Esta entidad se encarga de almacenar los centros de la diferentes organizaciones que tiene cada organización, utilizando para ello las siguientes columnas:
 - **IdOrganization:** Almacena el identificador de la organización. Es de tipo INT.
 - **orgType:** Indica el tipo de la organización del centro. Es de tipo String o VARCHAR(50).
 - **illness:** Almacena el tipo de enfermedad relacionada con la organización del centro. Es de tipo String o VARCHAR(50).
 - **idCenter:** Almacena el identificador del centro. Es de tipo INT.
 - **centerDescription:** Almacena la descripción del centro. Es de tipo String o VARCHAR(500).
 - **idAddress:** Almacena el identificador de la dirección del centro. Es de tipo INT.
 - **telephone:** Almacena el número de teléfono del centro. Es de tipo long o BIGINT.

- **PRIMARY KEY:** La clave principal de esta tabla está compuesta por los campos `IdOrganization`, `orgType`, `illness` e `idCenter`.
 - **FOREIGN KEY:** Establece una relación con la entidad **Organization** a través de los campos `IdOrganization`, `orgType` e `illness`.
 - **FOREIGN KEY:** Establece una relación con la tabla **Address** a través del campo `idAddress`.
- **EvaluatorTeam:** Esta entidad se encarga de almacenar información sobre los equipos evaluadores, utilizando para ello los siguientes campos:
- `idEvaluatorTeam`: Almacena el identificador del equipo evaluador. Es de tipo INT.
 - `creationDate`: Almacena la fecha de creación del equipo. Es de tipo long o BIGINT.
 - `emailConsultant`: Almacena la dirección de correo electrónico del consultor del equipo. Es de tipo String o VARCHAR(500).
 - `emailResponsible`: Almacena la dirección de correo electrónico del responsable del equipo. Es de tipo String o VARCHAR(500).
 - `emailProfessional`: Almacena la dirección de correo electrónico del profesional del equipo. Es de tipo String o VARCHAR(500).
 - `idOrganization`: Almacena el identificador de la organización a la que pertenece el equipo. Es de tipo INT.
 - `orgType`: Indica el tipo de la organización del equipo. Es de tipo String o VARCHAR(50).
 - `illness`: Almacena el tipo de enfermedad relacionada con la organización del equipo. Es de tipo String o VARCHAR(50).
 - `patientName`: Almacena el nombre del paciente evaluado por el equipo. Es de tipo String o VARCHAR(500).
 - `relativeName`: Almacena el nombre del familiar del paciente evaluado por el equipo. Es de tipo String o VARCHAR(500).
 - `evaluationDate1`: Almacena la primera fecha de evaluación. Es de tipo long o BIGINT.
 - `evaluationDate2`: Almacena la segunda fecha de evaluación. Es de tipo long o BIGINT.
 - `evaluationDate3`: Almacena la tercera fecha de evaluación. Es de tipo long o BIGINT.

- **evaluationDate4:** Almacena la cuarta fecha de evaluación. Es de tipo long o BIGINT.
 - **PRIMARY KEY:** La clave principal de esta tabla está compuesta por el campo **idEvaluatorTeam**.
 - **FOREIGN KEY:** Establece una relación con la tabla **Organization** a través de los campos **idOrganization**, **orgType** e **illness**.
- **EvaluatorTeamMember:** Esta entidad se encarga de registrar aquellos usuarios que pertenezcan a equipos evaluadores, utilizando para ello los siguientes campos:
- **emailUser:** Almacena la dirección de correo electrónico del miembro del equipo. Es de tipo String o VARCHAR(500).
 - **idEvaluatorTeam:** Almacena el identificador del equipo evaluador al que pertenece el miembro. Es de tipo INT.
 - **idEvaluatorOrganization:** Almacena el identificador de la organización evaluadora del miembro. Es de tipo INT.
 - **orgType:** Indica el tipo de la organización del miembro. Es de tipo String o VARCHAR(50).
 - **illness:** Almacena el tipo de enfermedad relacionada con la organización del miembro. Es de tipo String o VARCHAR(50).
 - **PRIMARY KEY:** La clave principal de esta tabla está compuesta por los campos **emailUser**, **idEvaluatorTeam**, **idEvaluatorOrganization**, **orgType** e **illness**.
 - **FOREIGN KEY:** Establece una relación con la entidad **User** a través del campo **emailUser**.
 - **FOREIGN KEY:** Establece una relación con la entidad **EvaluatorTeam** a través de los campos **idEvaluatorTeam**, **idOrganization**, **orgType** e **illness**.
- **IndicatorsEvaluation:** Esta entidad se encarga de almacenar la información sobre los diferentes test de indicadores, utilizando para ello los siguientes campos:
- **evaluationDate:** Almacena la fecha de evaluación de los indicadores. Es de tipo long o BIGINT.
 - **idEvaluatedOrganization:** Almacena el identificador de la organización evaluada. Es de tipo INT.

- **orgTypeEvaluated:** Indica el tipo de la organización evaluada. Es de tipo String o de tipo
 - **idEvaluatorTeam:** Almacena el identificador del equipo evaluador que realizó la evaluación. Es de tipo INT.
 - **idEvaluatorOrganization:** Almacena el identificador de la organización a la que pertenece el equipo evaluador. Es de tipo INT.
 - **orgTypeEvaluator:** Indica el tipo de la organización del equipo evaluador.
 - **illness:** Almacena el tipo de enfermedad relacionada con la organización evaluada.
 - **scoreLevel1:** Almacena el puntaje para el nivel 1 de los indicadores. Es de tipo INT.
 - **scoreLevel2:** Almacena el puntaje para el nivel 2 de los indicadores. Es de tipo INT.
 - **scoreLevel3:** Almacena el puntaje para el nivel 3 de los indicadores. Es de tipo INT.
 - **scoreLevel4:** Almacena el puntaje para el nivel 4 de los indicadores. Es de tipo INT.
 - **scoreLevel5:** Almacena el puntaje para el nivel 5 de los indicadores. Es de tipo INT.
 - **scoreLevel6:** Almacena el puntaje para el nivel 6 de los indicadores. Es de tipo INT.
 - **totalScore:** Almacena el puntaje total de la evaluación. Es de tipo INT.
 - **PRIMARY KEY:** La clave principal de esta tabla está compuesta por los campos **evaluationDate**, **idEvaluatedOrganization**, **orgTypeEvaluated** e **illness**.
 - **FOREIGN KEY:** Establece una relación con la entidad **Organization** a través de los campos **idEvaluatedOrganization**, **orgTypeEvaluated** e **illness**.
 - **FOREIGN KEY:** Establece una relación con la entidad **EvaluatorTeam** a través de los campos **idEvaluatorTeam**, **idEvaluatorOrganization**, **orgTypeEvaluator** e **illness**.
- **IndicatorsEvaluationReg:** Esta entidad se encarga de almacenar los registros de cada uno de los test de indicadores realizados, existiendo

uno para cada evidencia rellenada, utilizando para ello los siguientes campos:

- **evaluationDate**: Almacena la fecha de la evaluación de los indicadores.
- **idEvaluatedOrganization**: Almacena el identificador de la organización evaluada. Es de tipo INT.
- **orgTypeEvaluated**: Indica el tipo de la organización evaluada.
- **illness**: Almacena el tipo de enfermedad relacionada con la organización evaluada.
- **indicatorId**: Almacena el identificador del indicador evaluado. Es de tipo INT.
- **idEvidence**: Almacena el identificador de la evidencia relacionada con el indicador. Es de tipo INT.
- **isMarked**: Es un campo entero no nulo que indica si el indicador está marcado o no. Solo puede tener valores 0 o 1, lo que se verifica con una restricción CHECK. Es de tipo INT.
- **indicatorVersion**: Almacena la versión del indicador evaluado. Es de tipo INT.
- **PRIMARY KEY**: La clave principal de esta tabla está compuesta por los campos **evaluationDate**, **idEvaluatedOrganization**, **orgTypeEvaluated**, **illness**, **indicatorId**, **idEvidence** e **indicatorVersion**.
- **FOREIGN KEY**: Establece una relación con la entidad **Indicator** a través del campo **indicatorId**.
- **FOREIGN KEY**: Establece una relación con la entidad **Evidence** a través del campo **idEvidence**.
- **FOREIGN KEY**: Establece una relación con la entidad **IndicatorsEvaluation** a través de los campos **evaluationDate**, **idEvaluatedOrganization**, **orgTypeEvaluated**, **illness** e **indicatorVersion**.

C.3. Diseño procedimental

En cuanto al diseño procedimental de la aplicación, se tiene que tener en cuenta qué se espera de cada una de las características que tiene la aplicación. El objetivo fundamental de realizar este tipo de diseño es saber los pasos a seguir para realizar cada una de las funcionalidades más importantes para cada uno de los algoritmos a seguir. Para realizar el diseño procedimental

se ha establecido como base la realización de simples diagramas de flujo, que indiquen cómo tiene que realizarse paso a paso estos algoritmos. No se requieren de algoritmos bastante sofisticados para la realización de esta aplicación, por lo que se antoja bastante más sencillo conocer los pasos a seguir. En nuestro caso, tenemos varias funcionalidades diferentes:

- **Iniciar sesión:** Para el inicio de sesión se ha decidido implementar este diagrama de flujo:

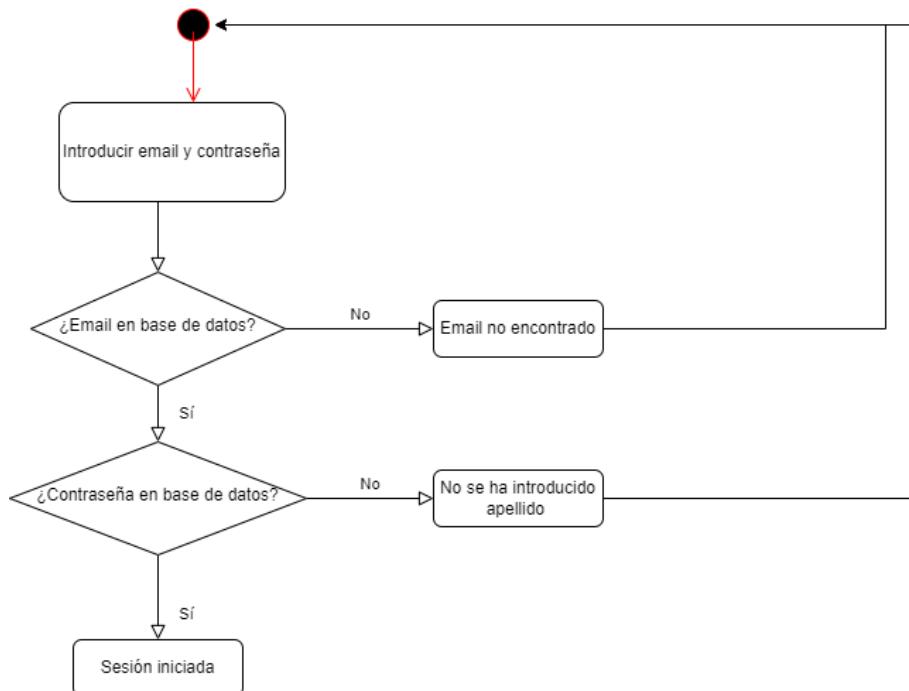


Figura C.5: Diagrama de flujo para el inicio de sesión

En este caso se inicializa con la introducción de los campos de usuario y de contraseña. Posteriormente, tras el intento de inicio de sesión, se intenta buscar el usuario con las credenciales de la base de datos. En primer lugar busca si el email está correctamente introducido, en caso de que el email no esté bien introducido, lanza el error y vuelve a empezar. En caso contrario, comprueba la contraseña. Si no es correcta, lanza el error y vuelve a empezar, y en caso contrario se consigue un inicio de sesión satisfactorio.

- **Registrar usuario:** Para el registro de usuario se ha decidido implementar este diagrama de flujo:

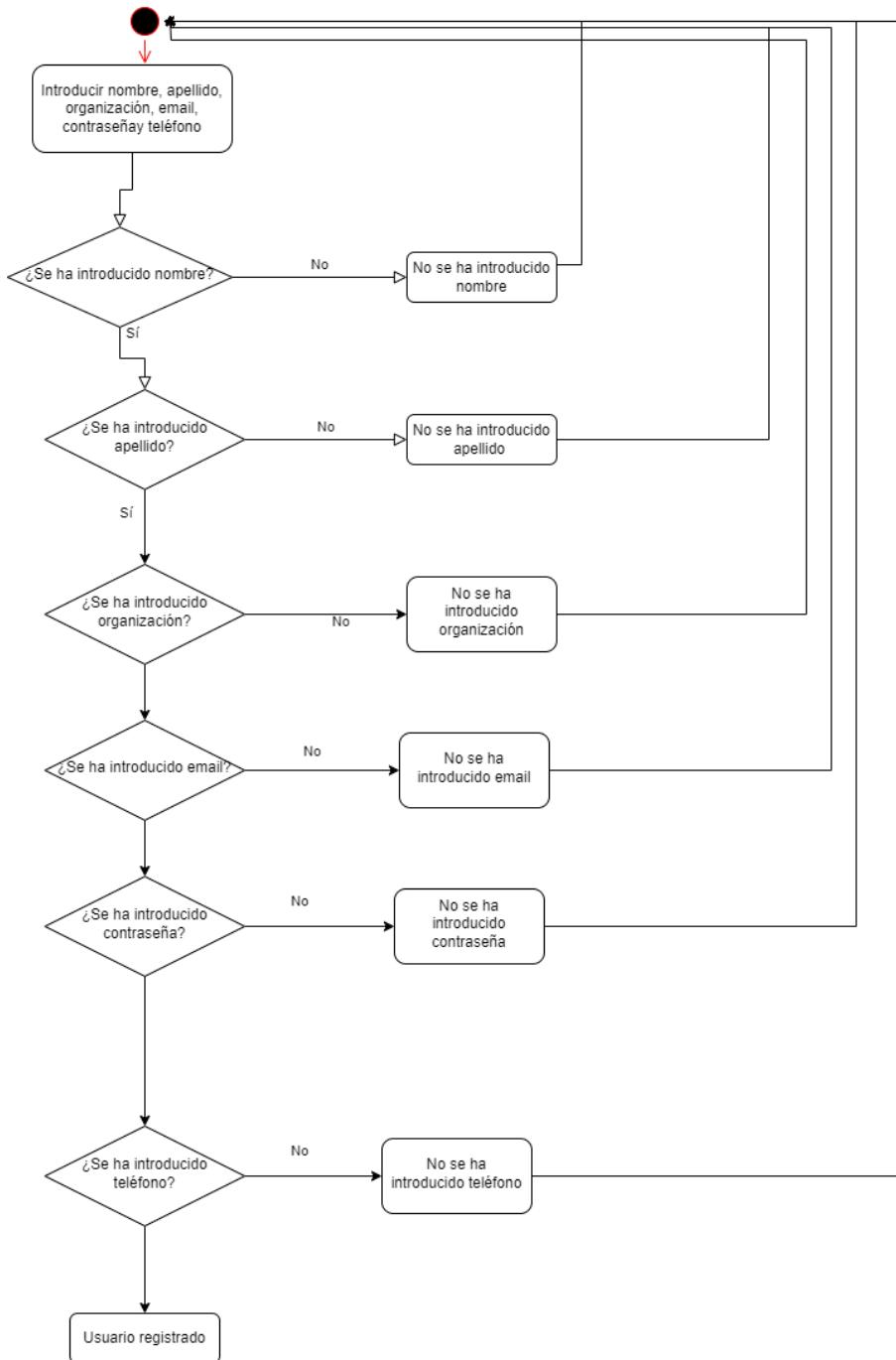


Figura C.6: Diagrama de flujo para el registro del usuario

Al igual que en el caso anterior, la dinámica es muy similar, con la diferencia que se necesitan introducir más campos de forma obligatoria.

En primer lugar se comprueba si se ha introducido el nombre del usuario. Si no se ha introducido vuelve al punto de arranque y en caso contrario pasa a comprobar el apellido. En caso de que no se haya introducido ningún apellido vuelve al punto de arranque y en caso contrario pasa a comprobar la organización. En caso de que no se haya introducido ninguna organización vuelve al punto de arranque y en caso contrario pasa a comprobar el email. Si no se ha introducido el email vuelve al punto de arranque y en caso contrario pasa a comprobar la contraseña. Si no se ha introducido la contraseña vuelve al punto de arranque y en caso contrario pasa a comprobar el teléfono. Si no se ha introducido ningún teléfono, vuelve al punto de arranque y en caso contrario se considera el registro del usuario como realizado.

- **Registrar organización:** Para el registro de la organización se cumple la misma dinámica que en la actividad anterior. Cabe resaltar que el campo de *información* no se ha introducido en este diagrama debido a que no es un campo que se considere de obligatorio llenado, más allá que para comentar algún elemento adicional. Se ha seguido el siguiente diagrama de flujo:

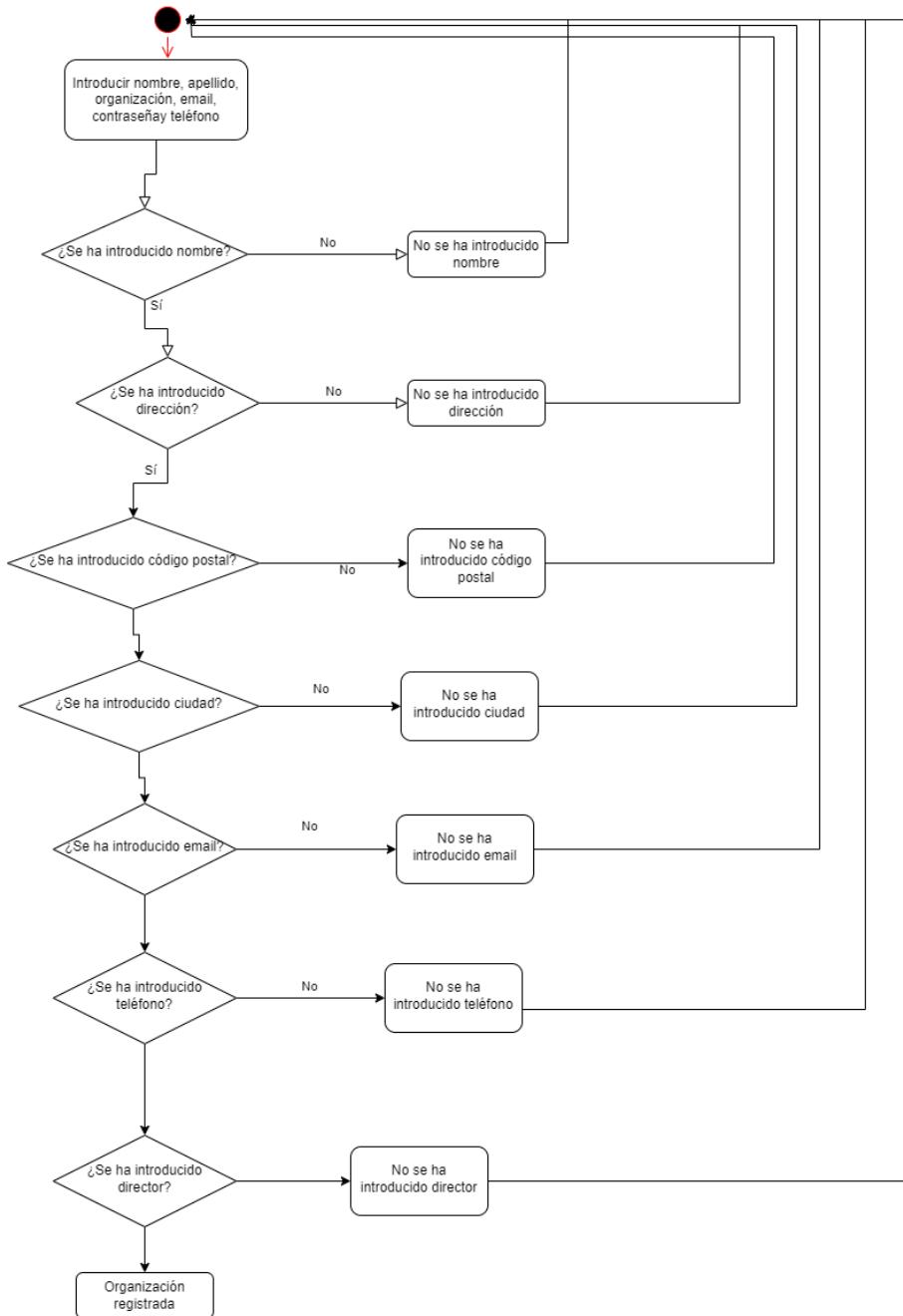


Figura C.7: Diagrama de flujo para el registro de la organización

Como se puede apreciar, en primer lugar se comprueba si se ha introducido el nombre de la organización. Si no se ha introducido vuelve al punto de arranque y en caso contrario pasa a comprobar la dirección.

En caso de que no se haya introducido ninguna dirección vuelve al punto de arranque y en caso contrario pasa a comprobar el código postal. En caso de que no se haya introducido ningún código postal vuelve al punto de arranque y en caso contrario pasa a comprobar la ciudad. Si no se ha introducido ninguna ciudad vuelve al punto de arranque y en caso contrario pasa a comprobar el email. Si no se ha introducido el email vuelve al punto de arranque y en caso contrario pasa a comprobar el teléfono. Si no se ha introducido ningún teléfono, vuelve al punto de arranque y en caso contrario se pasa a comprobar el director de la organización. Si no se ha introducido ningún director, se vuelve al punto de arranque y en caso contrario, se considera el registro de la organización como realizado.

- **Registrar equipo evaluador:** Para el registro del equipo evaluador se sigue una dinámica muy similar al del caso anterior, siguiendo para ello el siguiente diagrama de flujo:

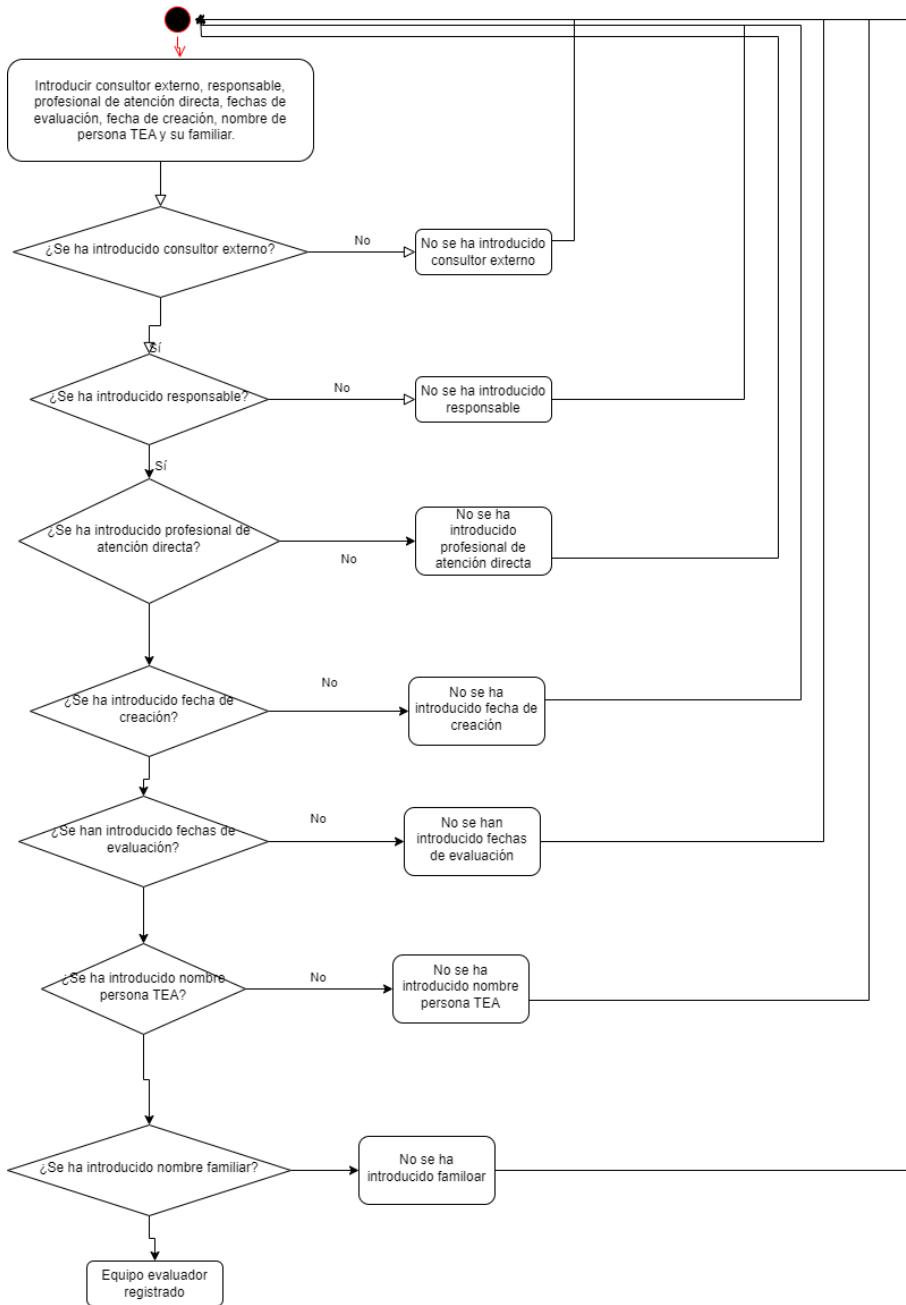


Figura C.8: Diagrama de flujo para el registro del equipo evaluador

Como se puede apreciar, tras la introducción del consultor externo, el responsable, el profesional de atención directa, las fechas de evaluación, la fecha de creación, el nombre de la persona con TEA y el nombre

de su familiar, se busca realizar la comprobación de dichos campos. En primer lugar si no se ha introducido un consultor externo vuelve al punto de arranque y en caso contrario comprueba el responsable. Si no se ha introducido ningún responsable, se vuelve al punto de arranque y en caso contrario se pasa a comprobar el profesional de atención directa. Si no se ha introducido ningún profesional de atención directa, se vuelve al punto de arranque y en caso contrario se pasan a comprobar las fechas de evaluación y de creación del equipo evaluador. Si no se ha introducido alguna de esas fechas, se vuelve al punto de arranque y en caso contrario se pasa a comprobar los nombres tanto de la persona con TEA como el de su familiar. En caso de que no se haya introducido alguno de los nombres, se vuelve al punto de arranque y en caso contrario se considera como registrado el equipo evaluador.

- **Realizar test de indicadores:** La realización del test de indicadores básica se considera como un proceso iterativo, el cual sigue el siguiente diagrama de flujo:

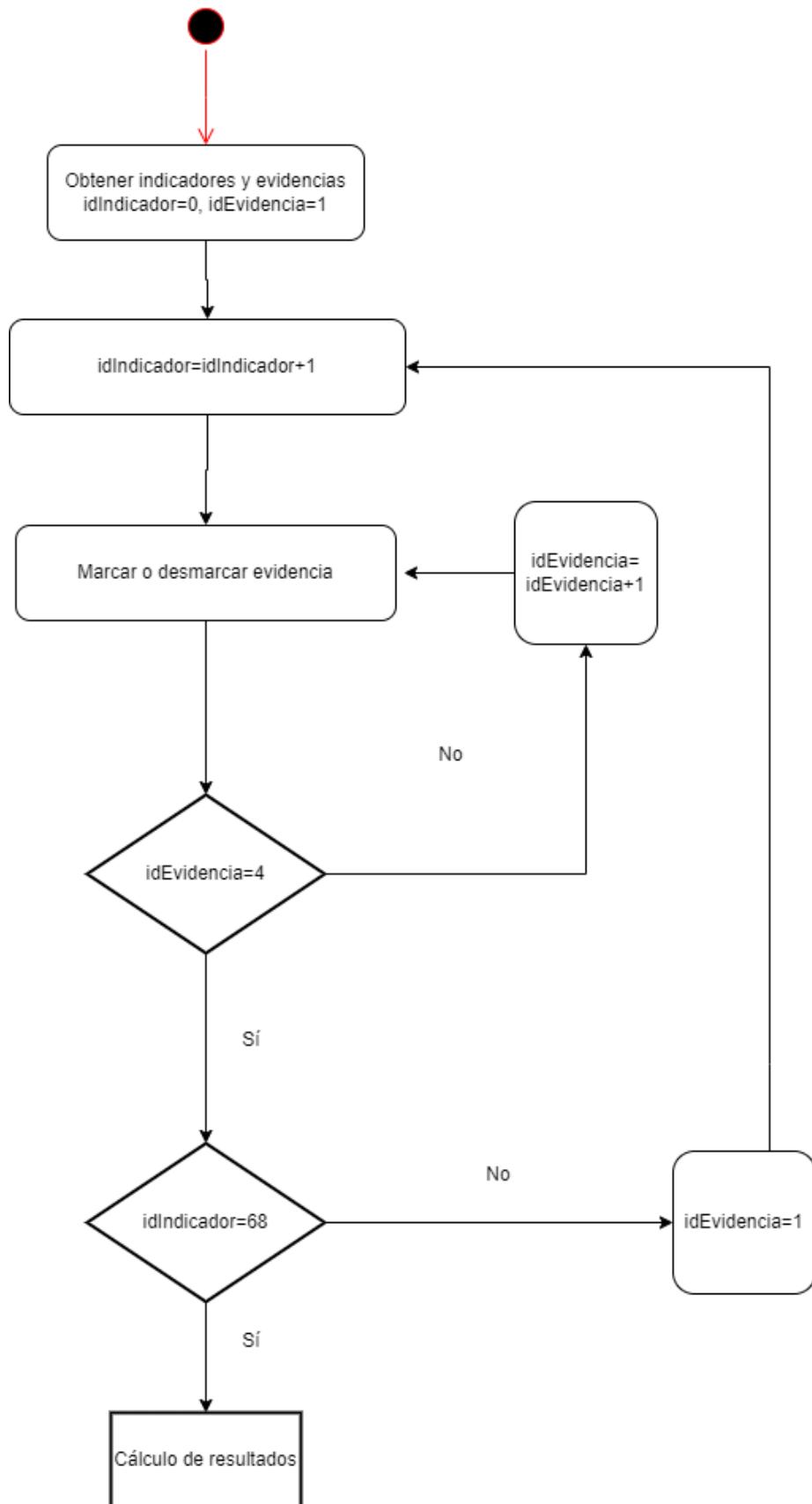


Figura C.9: Diagrama de flujo para la realización del test de indicadores

. Como se puede comprobar, tras la obtención de los indicadores y evidencias con sus respectivos contadores, ambos se van actualizando a medida que se van avanzando en los test de indicadores. Si se ha superado el número de evidencias, refresca el número del indicador y retorna al valor 1 como valor de evidencia. En caso contrario ya habríamos terminado con el test y se obtendrían los resultados.

C.4. Diseño arquitectónico

Como se ha mencionado con anterioridad, la arquitectura utilizada para el desarrollo de esta aplicación es una arquitectura de tipo *Modelo-Vista-Controlador*:

- En el lado del servidor tenemos tres tipos de clase diferentes segregadas por el paquete al que pertenecen. Dichos paquetes son:
 - **Models:** Este paquete incluye todos los modelos de las clases que son incluidos en el servidor. Ejemplo:

```

1           using Newtonsoft.Json;
2
3 namespace OTEAServer.Models
4 {
5     public class Indicator
6     {
7         public Indicator(int idIndicator,
8             ↪ string indicatorType, string
9             ↪ descriptionEnglish, string
10            ↪ descriptionSpanish, string
11            ↪ descriptionFrench, int
12            ↪ indicatorPriority, int
13            ↪ indicatorVersion) {
14             this.indicatorId = idIndicator;
15             this.indicatorType = indicatorType;
16             this.descriptionEnglish =
17                 ↪ descriptionEnglish;
18             this.descriptionSpanish =
19                 ↪ descriptionSpanish;
20             this.descriptionFrench =
21                 ↪ descriptionFrench;
22             this.indicatorPriority =
23                 ↪ indicatorPriority;

```

```

14         this.indicatorVersion =
15             ↪ indicatorVersion;
16     }
17
18     [JsonProperty("indicatorId")]
19     public int indicatorId { get; set; }
20
21     [JsonProperty("indicatorType")]
22     public string indicatorType { get; set;
23             ↪ }
24
25     [JsonProperty("descriptionEnglish")]
26     public string descriptionEnglish { get;
27             ↪ set; }
28
29     [JsonProperty("descriptionSpanish")]
30     public string descriptionSpanish { get;
31             ↪ set; }
32
33     [JsonProperty("descriptionFrench")]
34     public string descriptionFrench { get;
35             ↪ set; }
36
37     [JsonProperty("indicatorPriority")]
38     public int indicatorPriority { get; set
39             ↪ ; }
}

```

Como se comprueba en el siguiente ejemplo, todos los campos del modelo de la clase `Indicator` tienen la propiedad de *Newtonsoft.Json* denominada `JsonProperty`, la cual se utiliza para la serialización de los datos de este tipo para enviarlos en formato JSON.

- **Controllers:** Este paquete incluye todas las clases que estructuran los endpoints de la web app, utilizándose para realizar las operaciones en el servidor. Ejemplo:

```
1     using Microsoft.AspNetCore.Http;
```

```
2  using Microsoft.AspNetCore.Mvc;
3  using OTEAServer.Models;
4  using OTEAServer.Services;
5
6  namespace OTEAServer.Controllers
7  {
8      [ApiController]
9      [Route("Indicators")]
10     public class IndicatorsController :
11         ControllerBase
12     {
13         private readonly ILogger<
14             IndicatorsController> _logger;
15         private readonly IndicatorsService
16             _indicatorsService;
17
18         public IndicatorsController(ILogger<
19             IndicatorsController> logger,
20             IndicatorsService
21             indicatorsService)
22         {
23             _logger = logger;
24             _indicatorsService =
25                 indicatorsService;
26         }
27
28         // GET all action
29         [HttpGet]
30         public IActionResult GetAll()
31         {
32             var indicators = _indicatorsService
33                 .GetAll();
34             return Ok(indicators);
35         }
36
37         // GET all by INDICATORTYPE action
38         [HttpGet("all::indicatorType={"
39             indicatorType}")]
40         public IActionResult GetAllByType(
41             string indicatorType)
42         {
```

```
34             var indicators = _indicatorsService
35                 ↪ .GetAllByType(indicatorType);
36             return Ok(indicators);
37         }
38
39         // GET by ID AND INDICATOR TYPE action
40
41         [HttpGet("get::idIndicator={idIndicator
42             ↪ }:indicatorType={indicatorType}:
43             ↪ indicatorVersion={
44                 ↪ indicatorVersion}")]
45         public ActionResult<Indicator> Get(int
46             ↪ idIndicator, string indicatorType
47             ↪ , int indicatorVersion)
48         {
49             var indicator = _indicatorsService.
50                 ↪ Get(idIndicator, indicatorType
51                 ↪ , indicatorVersion);
52
53             if (indicator == null)
54                 ↪ return NotFound();
55
56             return indicator;
57         }
58
59         // POST action
60         [HttpPost]
61         public IActionResult Create([FromBody]
62             ↪ Indicator indicator)
63         {
64             _indicatorsService.Add(indicator.
65                 ↪ indicatorId, indicator.
66                 ↪ indicatorType, indicator.
67                 ↪ descriptionEnglish, indicator.
68                 ↪ .descriptionSpanish,
69                 ↪ indicator.descriptionFrench,
70                 ↪ indicator.indicatorPriority,
71                 ↪ indicator.indicatorVersion);
72             return CreatedAtAction(nameof(Get),
73                 ↪ new { id = indicator.
```

```
59             ↪ indicatorId , type=indicator .
60             ↪ indicatorType }, indicator);
61     }
62
63     // PUT action
64     [HttpPost("upd::idIndicator={idIndicator
65         ↪ }:indicatorType={indicatorType}:
66         ↪ indicatorVersion={
67         ↪ indicatorVersion}")]
68
69     public IActionResult Update(int
70         ↪ idIndicator , string indicatorType
71         ↪ , int indicatorVersion , [FromBody]
72         ↪ ] Indicator indicator)
73     {
74
75         // This code will update the mesa
76         ↪ and return a result
77
78         if (idIndicator != indicator.
79             ↪ indicatorId || indicatorType
80             ↪ != indicator.indicatorType ||

81             ↪ indicatorVersion !=

82             ↪ indicator.indicatorVersion)
83
84             ↪ return BadRequest();
85
86
87         var existingIndicator =
88             ↪ _indicatorsService.Get(
89                 ↪ idIndicator , indicatorType ,
90                 ↪ indicatorVersion-1);
91
92         if (existingIndicator is null)
93
94             ↪ return NotFound();
95
96
97         _indicatorsService.Update(
98             ↪ idIndicator , indicatorType ,
99             ↪ indicatorVersion , indicator);
100
101
102             ↪ return Ok(indicator);
103     }
104
105
106     // DELETE action
107     [HttpDelete("del::idIndicator={
108         ↪ idIndicator}:indicatorType={
109         ↪ indicatorType}:indicatorVersion={
110         ↪ indicatorVersion}")]
```

```

80         public IActionResult Delete(int
81             ↪ idIndicator, string indicatorType
82             ↪ , int indicatorVersion)
83     {
84         // This code will delete the mesa
85         ↪ and return a result
86         var indicator = _indicatorsService.
87             ↪ Get(idIndicator,
88             ↪ indicatorType,
89             ↪ indicatorVersion);
90
91         if (indicator is null)
92             ↪ return NotFound();
93
94         _indicatorsService.Delete(
95             ↪ idIndicator, indicatorType,
96             ↪ indicatorVersion);
97
98         return NoContent();
99     }
100 }
```

Como se puede comprobar en este controlador, todos los métodos vienen acompañados por `HttpGet`, `HttpPost`, `HttpPut` o `HttpDelete`, acompañados del endpoint correspondiente establecido. En el caso del `HttpPut`, siempre viene acompañado su parámetro por un `FromBody`, el cual indica que el objeto procede de un formato JSON. También cabe resaltar que antes de colocar el `public class`, se coloca con anterioridad `ApiController` y posteriormente `Route("Indicators")`, que establecen el endpoint base de la API.

- **Services:** Este paquete incluye todas las clases en las que se apoyan los controladores para poder realizar las operaciones en la base de datos. Ejemplo:

```

1         using OTEAServer.Models;
2 using System.Data.SqlClient;
3
4 namespace OTEAServer.Services
5 {
6     public class IndicatorsService
7     {
```

```

8     private static IConfiguration
9         ↪ _configuration;
10    public IndicatorsService(IConfiguration
11        ↪ configuration)
12    {
13        _configuration = configuration;
14    }
15
16    public List<Indicator> GetAll() {
17        List<Indicator> indicatorsList =
18            ↪ new List<Indicator>();
19
20        string connectionString =
21            ↪ _configuration.
22            ↪ GetConnectionString(
23            ↪ "DefaultConnection");
24        string query = "SELECT * FROM
25            ↪ INDICATORS WHERE ISACTIVE=1
26            ↪ ORDER BY INDICATORID";
27
28        using (SqlConnection connection =
29            ↪ new SqlConnection(
30            ↪ connectionString))
31        {
32            connection.Open();
33
34            using (SqlCommand command = new
35                ↪ SqlCommand(query,
36                ↪ connection))
37            {
38                using (SqlDataReader reader
39                    ↪ = command.
40                    ↪ ExecuteReader())
41                {
42                    while (reader.Read())
43                    {
44                        indicatorsList.Add(
45                            ↪ new Indicator
46                            ↪ (reader.
47                            ↪ GetInt32(0),
48                            ↪ reader.
49                            ↪ GetString(1),
50                            ↪ reader.
51                            ↪ GetString(2));
52                    }
53                }
54            }
55        }
56    }
57
58    public void Add(Indicator indicator)
59    {
60        _configuration.GetConnectionString("DefaultConnection");
61        using (SqlConnection connection =
62            ↪ new SqlConnection(
63            ↪ _configuration.
64            ↪ GetConnectionString("DefaultConnection")))
65        {
66            connection.Open();
67
68            using (SqlCommand command = new
69                ↪ SqlCommand("INSERT INTO INDICATORS
70                    ↪ (NAME, DESCRIPTION, ISACTIVE) VALUES
71                    ↪ (@NAME, @DESCRIPTION, @ISACTIVE)", connection))
72            {
73                command.Parameters.AddWithValue("@NAME",
74                    indicator.Name);
75                command.Parameters.AddWithValue("@DESCRIPTION",
76                    indicator.Description);
77                command.Parameters.AddWithValue("@ISACTIVE",
78                    indicator.IsActive);
79
80                command.ExecuteNonQuery();
81            }
82        }
83    }
84
85    public void Update(Indicator indicator)
86    {
87        _configuration.GetConnectionString("DefaultConnection");
88        using (SqlConnection connection =
89            ↪ new SqlConnection(
90            ↪ _configuration.
91            ↪ GetConnectionString("DefaultConnection")))
92        {
93            connection.Open();
94
95            using (SqlCommand command = new
96                ↪ SqlCommand("UPDATE INDICATORS SET
97                    ↪ NAME = @NAME, DESCRIPTION = @DESCRIPTION,
98                    ↪ ISACTIVE = @ISACTIVE WHERE INDICATORID = @INDICATORID",
99                    ↪ connection))
100           {
101               command.Parameters.AddWithValue("@NAME",
102                   indicator.Name);
103               command.Parameters.AddWithValue("@DESCRIPTION",
104                   indicator.Description);
105               command.Parameters.AddWithValue("@ISACTIVE",
106                   indicator.IsActive);
107               command.Parameters.AddWithValue("@INDICATORID",
108                   indicator.IndicatorId);
109
110               command.ExecuteNonQuery();
111           }
112       }
113   }
114 }
```



```

51         command.Parameters.
52             ↪ AddWithValue(
53                 ↪ @INDICATORTYPE",
54                 ↪ indicatorType);
55             using (SqlDataReader reader
56                 = command.
57                     ↪ ExecuteReader())
58             {
59                 while (reader.Read())
60                 {
61                     indicatorsList.Add(
62                         ↪ new Indicator
63                         ↪ (reader.
64                             ↪ GetInt32(0),
65                             ↪ indicatorType
66                             , reader.
67                             ↪ GetString(2),
68                             ↪ reader.
69                             ↪ GetString(3),
70                             ↪ reader.
71                             ↪ GetString(4),
72                             ↪ reader.
73                             ↪GetInt32(5),
74                             ↪ reader.
75                             ↪GetInt32(6)))
76                         ;
77                 }
78             }
79         }
80     }
81     return indicatorsList;
82 }
83
84     public Indicator? Get(int idIndicator,
85             ↪ string indicatorType, int
86             ↪ indicatorVersion)
87     {
88         string connectionString =
89             ↪ _configuration.
90             ↪ GetConnectionString(
91                 ↪ DefaultConnection");
92         string query = "SELECT * FROM
93             ↪ INDICATORS WHERE indicatorId="

```

```
    ↵ @IDINDICATOR AND
    ↵ indicatorType=@INDICATORTYPE
    ↵ AND indicatorVersion=
    ↵ @INDICATORVERSION ORDER BY
    ↵ INDICATORID";  
68  
69         using (SqlConnection connection =
    ↵     new SqlConnection(
    ↵         connectionString))
70     {
71         connection.Open();
72
73         using (SqlCommand command = new
    ↵             SqlCommand(query,
    ↵             connection))
74     {
75         command.Parameters.
    ↵             AddWithValue(
    ↵                 "@IDINDICATOR",
    ↵                 idIndicator);
76         command.Parameters.
    ↵             AddWithValue(
    ↵                 "@INDICATORTYPE",
    ↵                 indicatorType);
77         command.Parameters.
    ↵             AddWithValue(
    ↵                 "@INDICATORVERSION",
    ↵                 indicatorVersion);
78         using (SqlDataReader reader
    ↵             = command.
    ↵             ExecuteReader())
79     {
80         if (reader.Read())
81     {
82             return new
    ↵                 Indicator(
    ↵                     idIndicator,
    ↵                     indicatorType
    ↵                     , reader.
    ↵                     GetString(2),
    ↵                     reader.
    ↵                     GetString(3),
    ↵                     reader.
```

```
83             ↪ GetString(4),
84             ↪
85             ↪ indicatorVersion
86             ↪ , reader.
87             ↪ GetInt32(6));
88         }
89     }
90     return null;
91 }
92
93     public void Add(int idIndicator, string
94         ↪ indicatorType, string
95         ↪ descriptionEnglish, string
96         ↪ descriptionSpanish, string
97         ↪ descriptionFrench, int
98         ↪ indicatorPriority, int
99         ↪ indicatorVersion)
100    {
101        string connectionString =
102            ↪ _configuration.
103            ↪ GetConnectionString(
104            ↪ DefaultConnection");
105
106        using (SqlConnection connection =
107            ↪ new SqlConnection(
108            ↪ connectionString))
109        {
110            connection.Open();
111
112            // Crea el command SQL
113            string sql = "INSERT INTO
114                ↪ INDICATORS (INDICATORID,
115                ↪ INDICATORTYPE,
116                ↪ DESCRIPTIONENGLISH ,
117                ↪ DESCRIPTIONSPANISH ,
118                ↪ DESCRIPTIONFRENCH ,
119                ↪ INDICATORPRIORITY ,
120                ↪ INDICATORVERSION,ISACTIVE
121                ↪ ) VALUES (@IDINDICATOR ,
122                ↪ @INDICATORTYPE ,
123                ↪ @DESCRIPTIONENGLISH ,
```

```
    ↪ @DESCRIPTIONSPANISH ,
    ↪ @DESCRIPTIONFRENCH ,
    ↪ @INDICATORPRIORITY ,
    ↪ @INDICATORVERSION ,1) ";
100    using (SqlCommand command = new
    ↪ SqlCommand(sql ,
    ↪ connection))
101    {
102        command.Parameters .
    ↪ AddWithValue("
    ↪ @IDINDICATOR",
    ↪ idIndicator);
103        command.Parameters .
    ↪ AddWithValue("
    ↪ @INDICATORTYPE",
    ↪ indicatorType);
104        command.Parameters .
    ↪ AddWithValue("
    ↪ @DESCRIPTIONENGLISH",
    ↪ descriptionEnglish);
105        command.Parameters .
    ↪ AddWithValue("
    ↪ @DESCRIPTIONSPANISH",
    ↪ descriptionSpanish);
106        command.Parameters .
    ↪ AddWithValue("
    ↪ @DESCRIPTIONFRENCH",
    ↪ descriptionFrench);
107        command.Parameters .
    ↪ AddWithValue("
    ↪ @INDICATORPRIORITY",
    ↪ indicatorPriority);
108        command.Parameters .
    ↪ AddWithValue("
    ↪ @INDICATORVERSION",
    ↪ indicatorVersion);
109        command.ExecuteNonQuery();
110    }
111}
112connection.Close();
113}
114}
115}
```

```

116
117
118     public void Delete(int idIndicator,
119                         ↪ string indicatorType, int
120                         ↪ indicatorVersion)
121     {
122         if (Get(idIndicator, indicatorType,
123                         ↪ indicatorVersion) != null)
124         {
125             string connectionString =
126                 ↪ _configuration.
127                 ↪ GetConnectionString(
128                         ↪ "DefaultConnection");
129
130             using (SqlConnection connection
131                         ↪ = new SqlConnection(
132                         ↪ connectionString))
133             {
134                 connection.Open();
135
136                 // Crea el command SQL
137                 string sql = "DELETE FROM
138                         ↪ INDICATORS WHERE
139                         ↪ indicatorId=
140                         ↪ @IDINDICATOR AND
141                         ↪ indicatorType=
142                         ↪ @INDICATORTYPE AND
143                         ↪ indicatorVersion=
144                         ↪ @INDICATORVERSION";
145
146                 using (SqlCommand command =
147                         ↪ new SqlCommand(sql,
148                         ↪ connection))
149                 {
150                     command.Parameters.
151                         ↪ AddWithValue("
152                         ↪ @IDINDICATOR",
153                         ↪ idIndicator);
154                     command.Parameters.
155                         ↪ AddWithValue("
156                         ↪ @INDICATORTYPE",
157                         ↪ indicatorType);
158                     command.Parameters.
159                         ↪ AddWithValue("

```

```
    ↵ @INDICATORVERSION
    ↵ ",
    ↵ indicatorVersion)
    ↵ ;
135
136             command.ExecuteNonQuery
    ↵ ();
137         }
138         connection.Close();
139     }
140 }
141 }
142
143     public void Update(int idIndicator,
    ↵ string indicatorType, int
    ↵ indicatorVersion, Indicator
    ↵ indicator)
144 {
145     if (indicator != null && Get(
        ↵ idIndicator, indicatorType,
        ↵ indicatorVersion) != null &&
        ↵ idIndicator==indicator.
        ↵ indicatorId && indicatorType
        ↵ ==indicator.indicatorType &&
        ↵ indicatorVersion==indicator.
        ↵ indicatorVersion-1)
146 {
147
148     Add(indicator.indicatorId,
        ↵ indicator.indicatorType,
        ↵ indicator.
        ↵ descriptionEnglish,
        ↵ indicator.
        ↵ descriptionSpanish,
        ↵ indicator.
        ↵ descriptionFrench,
        ↵ indicator.
        ↵ indicatorPriority,
        ↵ indicator.
        ↵ indicatorVersion);
149
150     string connectionString =
    ↵ _configuration.
```

```

151           ↪ GetConnectionString(""
152           ↪ DefaultConnection");
153
154           using (SqlConnection connection
155               ↪ = new SqlConnection(
156               ↪ connectionString))
157           {
158               connection.Open();
159
160
161               string sql = "UPDATE USERS
162                   ↪ SET ISACTIVE=0 WHERE
163                   ↪ INDICATORID=
164                   ↪ @IDINDICATOR AND
165                   ↪ INDICATORTYPE=
166                   ↪ @INDICATORTYPE AND
167                   ↪ INDICATORVERSION=
168                   ↪ @INDICATORVERSION";
169               using (SqlCommand command =
170                   ↪ new SqlCommand(sql,
171                   ↪ connection))
172               {
173
174                   command.Parameters.
175                       ↪ AddWithValue(""
176                       ↪ @IDINDICATOR",
177                       ↪ indicator.
178                       ↪ indicatorId);
179                   command.Parameters.
180                       ↪ AddWithValue(""
181                       ↪ @INDICATORTYPE",
182                       ↪ indicator.
183                       ↪ indicatorType);
184                   command.Parameters.
185                       ↪ AddWithValue(""
186                       ↪ @INDICATORVERSION
187                       ↪ ", indicator.
188                       ↪ indicatorVersion
189                       ↪ -1);
190
191                   command.ExecuteNonQuery
192                       ↪ ();
193               }

```

```

167
168         connection.Close();
169     }
170   }
171 }
172 }
173 }
```

En este caso se puede comprobar las diferentes operaciones que realiza el servicio de Indicadores. Todas ellas siguen una misma estructura. En primer lugar se tiene que llamar al `connectionString` procedente del fichero de configuración, más adelante construir la consulta parametrizada en SQL mediante `SqlCommand`, que se construye con la consulta parametrizada y con la conexión. Por último, utiliza el `reader` en las operaciones de GET.

- En el lado del cliente también tenemos tres tipos de clase diferentes, pero llamadas de diferente forma:
 - **Callers:** Son los métodos que se encargan de realizar de forma asíncrona las peticiones del servidor y a su vez procesar la respuesta del cliente, todo ello de forma asíncrona mediante `AsyncTask`. Dichos Callers sigue el patrón de diseño Singleton, ya que no es necesario contar con múltiples instancias de la clase, pudiendo crear una sola para cada uno de estos. Cuando se tienen múltiples instancias repartidas en el tiempo de ejecución, los tiempos de respuesta son muy lentos, por lo que utilizar este patrón de diseño es muy importante para poder acelerar dicho proceso. Ejemplo:

```

1  public class IndicatorsCaller {
2
3    private static IndicatorsApi api;
4    private static IndicatorsCaller instance;
5
6    private IndicatorsCaller(){
7      api=ConnectionClient.getInstance() .
8        ↪ getRetrofit().create(
9          ↪ IndicatorsApi.class);
10
11
10  public static IndicatorsCaller getInstance
11    ↪ () {
11      if(instance==null){
```

```

12         synchronized (IndicatorsCaller.
13             ↪ class){
14                 if(instance==null){
15                     instance=new
16                         ↪ IndicatorsCaller();
17                 }
18             }
19         }
20
21
22     public static List<Indicator>
23         ↪ obtainIndicators(String illness){
24             Call<List<Indicator>> call=api.
25                 ↪ GetAllByType(illness);
26             AsyncTask<Void, Void, List<Indicator>>
27                 ↪ asyncTask = new AsyncTask<Void,
28                     ↪ Void, List<Indicator>>() {
29                         List<Indicator> resultList= null;
30                         @Override
31                         protected List<Indicator>
32                             ↪ doInBackground(Void... voids)
33                             ↪ {
34                                 try {
35                                     Response<List<Indicator>>
36                                         ↪ response = call.
37                                         ↪ execute();
38                                         if (response.isSuccessful()
39                                             ↪ ) {
40                                             return response.body();
41                                         } else {
42                                             throw new IOException(
43                                                 ↪ Error: " +
44                                                 ↪ response.code() +
45                                                 " " + response.
46                                                 ↪ message());
47                                         }
48                                         } catch (IOException e) {
49                                             throw new RuntimeException(
50                                                 ↪ e);
51                                         }
52                                     }
53                                 }
54     }
```

```
39         @Override
40         protected void onPostExecute(List<
41             ↪ Indicator> indicatorList) {
42             resultList=indicatorList;
43         }
44     };
45     AsyncTask.execute();
46     try {
47         List<Indicator> list=asyncTask.get
48             ↪ ();
49         return list;
50     } catch (Exception e) {
51         Log.d("ERROR", e.toString());
52     }
53     return null;
54 }
55
56     public static Indicator Update(int
57         ↪ idIndicator, String indicatorType,
58         ↪ int indicatorVersion, Indicator
59         ↪ indicator){
60         Call<Indicator> call=api.Update(
61             ↪ idIndicator, indicatorType,
62             ↪ indicatorVersion, indicator);
63         AsyncTask<Void, Void, Indicator>
64             ↪ asyncTask = new AsyncTask<Void,
65             ↪ Void, Indicator>() {
66             Indicator resultIndicator= null;
67             @Override
68             protected Indicator doInBackground(
69                 ↪ Void... voids) {
70                 try {
71                     Response<Indicator>
72                         ↪ response = call.
73                         ↪ execute();
74                     if (response.isSuccessful())
75                         ↪ ) {
76                         return response.body();
77                     } else {
78                         throw new IOException(
79                             ↪ Error: " +
```

```

    ↵     response.code() +
    ↵     " " + response.
    ↵     message());
68     }
69 } catch (IOException e) {
70     throw new RuntimeException(
71         ↵     e);
72 }
73 @Override
74 protected void onPostExecute(
75     ↵     Indicator indicator) {
76     resultIndicator=indicator;
77 }
78 };
79 AsyncTask.execute();
80 try {
81     Indicator i=asyncTask.get();
82     return new Indicator(i.
83         ↵     getIdIndicator(),i.
84         ↵     getIndicatorType(),i.
85         ↵     getDescriptionEnglish(),i.
86         ↵     getDescriptionSpanish(),i.
87         ↵     getDescriptionFrench(),i.
88         ↵     getPriority(),i.
89         ↵     getIndicatorVersion());
90 } catch (Exception e) {
91     Log.d("ERROR", e.toString());
92 }
93     return null;
94 }

89 public static Indicator Create(Indicator
90     ↵     indicator){
91     Call<Indicator> call=api.Create(
92         ↵     indicator);
93     AsyncTask<Void, Void, Indicator>
94         ↵     asyncTask = new AsyncTask<Void,
95         ↵     Void, Indicator>() {
96         Indicator resultIndicator= null;
97         @Override

```

```
94         protected Indicator doInBackground(
95             ↪ Void... voids) {
96             try {
97                 Response<Indicator>
98                     ↪ response = call.
99                     ↪ execute();
100                if (response.isSuccessful()
101                    ↪ ) {
102                    return response.body();
103                } else {
104                    throw new IOException(
105                        ↪ Error: " +
106                        ↪ response.code() +
107                        ↪ " " + response.
108                        ↪ message());
109                }
110            }
111        }
112        @Override
113        protected void onPostExecute(
114            ↪ Indicator indicator) {
115            resultIndicator=indicator;
116        }
117    };
118    AsyncTask.execute();
119    try {
120        Indicator i=asyncTask.get();
121        return new Indicator(i.
122            ↪ getIdIndicator(),i.
123            ↪ getIndicatorType(),i.
124            ↪ getDescriptionEnglish(),i.
125            ↪ getDescriptionSpanish(),i.
126            ↪ getDescriptionFrench(),i.
127            ↪ getPriority(),i.
128            ↪ getIndicatorVersion());
129    } catch (Exception e) {
130        Log.d("ERROR", e.toString());
131    }
132    return null;
```

```

120     }
121
122     public static Indicator Get(int idIndicator
123         ↪ , String indicatorType, int
124         ↪ indicatorVersion){
125         Call<Indicator> call=api.Get(
126             ↪ idIndicator, indicatorType,
127             ↪ indicatorVersion);
128         AsyncTask<Void, Void, Indicator>
129             ↪ asyncTask = new AsyncTask<Void,
130             ↪ Void, Indicator>() {
131                 Indicator resultIndicator= null;
132                 @Override
133                 protected Indicator doInBackground(
134                     ↪ Void... voids) {
135                     try {
136                         Response<Indicator>
137                             ↪ response = call.
138                             ↪ execute();
139                         if (response.isSuccessful())
140                             ↪ ) {
141                             return response.body();
142                         } else {
143                             throw new IOException(""
144                                 ↪ Error: " +
145                                 ↪ response.code() +
146                                 ↪ " " + response.
147                                 ↪ message());
148                         }
149                     } catch (IOException e) {
150                         throw new RuntimeException(
151                             ↪ e);
152                     }
153                 }
154                 @Override
155                 protected void onPostExecute(
156                     ↪ Indicator indicator) {
157                         resultIndicator=indicator;
158                     }
159                 };
160                 asyncTask.execute();
161                 try {

```

```
147         Indicator i=asyncTask.get();
148         return new Indicator(i.
149             ↪ getIdIndicator(),i.
149             ↪ getIndicatorType(),i.
149             ↪ getDescriptionEnglish(),i.
149             ↪ getDescriptionSpanish(),i.
149             ↪ getDescriptionFrench(),i.
149             ↪ getPriority(),i.
149             ↪ getIndicatorVersion());
150     } catch (Exception e) {
151         Log.d("ERROR", e.toString());
152     }
153     return null;
154 }
155 public static List<Indicator> GetAll(){
156     Call<List<Indicator>> call=api.GetAll()
156         ↪ ;
157     AsyncTask<Void, Void, List<Indicator>>
157         ↪ asyncTask = new AsyncTask<Void,
157             ↪ Void, List<Indicator>>() {
158         List<Indicator> resultList= null;
159         @Override
159         protected List<Indicator>
159             ↪ doInBackground(Void... voids)
159             ↪ {
160             try {
161                 Response<List<Indicator>>
161                     ↪ response = call.
161                     ↪ execute();
162                 if (response.isSuccessful())
162                     ↪ ) {
163                     return response.body();
164                 } else {
165                     throw new IOException(
165                         ↪ Error: " +
165                         ↪ response.code() +
165                         ↪ " " + response.
165                         ↪ message());
166                 }
167             } catch (IOException e) {
168                 throw new RuntimeException(
168                     ↪ e);
169             }
```

```

170         }
171     @Override
172     protected void onPostExecute(List<
173         ↪ Indicator> indicatorList) {
174         resultList=indicatorList;
175     }
176 }
177 AsyncTask.execute();
178 try {
179     List<Indicator> aux=asyncTask.get()
180     ↪ ;
181     List<Indicator> list=new LinkedList
182     ↪ <>();
183     for(Indicator i:aux){
184         list.add(new Indicator(i.
185             ↪ getIdIndicator(),i.
186             ↪ getIndicatorType(),i.
187             ↪ getDescriptionEnglish(),i
188             ↪ .getDescriptionSpanish(),
189             ↪ i.getDescriptionFrench(),
190             ↪ i.getPriority(),i.
191             ↪ getIndicatorVersion()));
192     }
193     return list;
194 } catch (Exception e) {
195     Log.d("ERROR", e.toString());
196 }
197 return null;
198 }
199 public static Indicator Delete(int
200     ↪ idIndicator, String indicatorType,
201     ↪ int indicatorVersion){
202     Call<Indicator> call=api.Get(
203         ↪ idIndicator,indicatorType,
204         ↪ indicatorVersion);
205     AsyncTask<Void, Void, Indicator>
206         ↪ asyncTask = new AsyncTask<Void,
207         ↪ Void, Indicator>() {
208         Indicator resultIndicator= null;
209         @Override
210         protected Indicator doInBackground(
211             ↪ Void... voids) {

```

```

196             try {
197                 Response<Indicator>
198                     ↪ response = call.
199                     ↪ execute();
200
201                     if (response.isSuccessful())
202                         ↪ ) {
203                             return response.body();
204
205                     } else {
206                         throw new IOException(
207                             ↪ Error: " +
208                             ↪ response.code() +
209                             ↪ " " + response.
210                             ↪ message());
211
212                     }
213                     catch (IOException e) {
214                         throw new RuntimeException(
215                             ↪ e);
216
217                     }
218
219                     @Override
220                     protected void onPostExecute(
221                         ↪ Indicator indicator) {
222                         resultIndicator=indicator;
223
224                     }
225
226                     };
227                     AsyncTask.execute();
228
229                     try {
230                         return AsyncTask.get();
231                     } catch (Exception e) {
232                         Log.d("ERROR", e.toString());
233
234                     }
235
236                     return null;
237
238             }
239         }
240     }
241 }
```

Como se puede comprobar, todos los métodos utilizan en primer lugar `AsyncTask` para poder comunicarse con la API, que contiene el endpoint referenciado en el servidor, cuya URL base se encuentra en la clase `ConnectionClient`, la cual utiliza también el patrón Singleton por el mismo motivo que los callers. `ASyncTask` se encarga de llamar al método de la API para luego procesar

la respuesta. El método siempre devuelve `asyncTask.get()`, el cual devuelve la respuesta de forma bloqueante.

- **Api:** Las API en Java están definidas como interfaces. Las llamadas a los endpoint se realizan mediante *Retrofit*, como se ha mencionado en la memoria. Ejemplo:

```

1      public interface IndicatorsApi {
2          @PUT("Indicators/upd::"
3              ↪ idIndicator={idIndicator
4                  ↪ }:indicatorType={
5                      ↪ indicatorType}:
6                          ↪ indicatorVersion={
7                              ↪ indicatorVersion})")
8          Call<Indicator> Update(@Path(""
9              ↪ idIndicator") int
10             ↪ idIndicator, @Path(""
11                 ↪ indicatorType") String
12                 ↪ indicatorType, @Path(""
13                     ↪ indicatorVersion") int
14                     ↪ indicatorVersion, @Body
15                         ↪ Indicator indicator);
16
17          @POST("Indicators")
18          Call<Indicator> Create(@Body
19              ↪ Indicator indicator);
20
21          @GET("Indicators/get::"
22              ↪ idIndicator={idIndicator
23                  ↪ }:indicatorType={
24                      ↪ indicatorType}:
25                          ↪ indicatorVersion={
26                              ↪ indicatorVersion})")
27          Call<Indicator> Get(@Path(""
28              ↪ idIndicator") int
29              ↪ idIndicator, @Path(""
30                  ↪ indicatorType") String
31                  ↪ indicatorType, @Path(""
32                      ↪ indicatorVersion") int
33                      ↪ indicatorVersion);
34
35          @GET("Indicators")
36          Call<List<Indicator>> GetAll();
37
38          @GET("Indicators/all::"
39              ↪ indicatorType={
40                  ↪ indicatorType})")

```

```

11      Call<List<Indicator>>
12          ↪ GetAllByType(@Path(
13              ↪ "indicatorType") String
14              ↪ indicatorType);
15
16      @DELETE("Indicators/del::")
17          ↪ idIndicator={idIndicator
18              ↪ }: indicatorType={
19                  ↪ indicatorType};
20          ↪ indicatorVersion={
21              ↪ indicatorVersion})
22
23      Call<Indicator> Delete(@Path(""
24          ↪ "idIndicator") int
25          ↪ idIndicator, @Path(""
26              ↪ "indicatorType") String
27              ↪ indicatorType, @Path(""
28                  ↪ "indicatorVersion") int
29                      ↪ indicatorVersion);
30
31  }
```

Como se puede apreciar, todos los métodos vienen con la notación GET, POST, PUT o DELETE acompañada de su endpoint correspondiente. Si se desean añadir datos a los endpoint se tiene que utilizar Path, mientras que para los POST se tiene que utilizar BODY para la serialización en JSON

- **Modelos:** Se trata de las propias clases de Java. Ejemplo:

```

1      public class Indicator implements
2          ↪ Serializable {
3
4          @SerializedName(""
5              ↪ "indicatorId")
6          public int indicatorId;
7
8          @SerializedName(""
9              ↪ "indicatorType")
10         public String indicatorType
11             ↪ ;
12
13         @SerializedName(""
14             ↪ "descriptionEnglish")
15         public String
16             ↪ descriptionEnglish;
17         @SerializedName(""
18             ↪ "descriptionSpanish")
```

```
12     public String  
13         ↪ descriptionSpanish;  
14     @SerializedName("  
15         ↪ descriptionFrench")  
16     public String  
17         ↪ descriptionFrench;  
18  
19     @SerializedName("  
20         ↪ indicatorPriority")  
21     public int  
22         ↪ indicatorPriority;  
23  
24     @SerializedName("  
25         ↪ indicatorVersion")  
26     public int indicatorVersion  
27         ↪ ;  
28  
29     public Indicator(int  
30         ↪ indicatorId, String  
31         ↪ indicatorType, String  
32         ↪ descriptionEnglish,  
33         ↪ String  
34         ↪ descriptionSpanish,  
35         ↪ String  
36         ↪ descriptionFrench,  
37         ↪ int indicatorPriority  
38         ↪ , int  
39         ↪ indicatorVersion) {  
40             setIdIndicator(  
41                 ↪ indicatorId);  
42             setIndicatorType(  
43                 ↪ indicatorType);  
44             setDescriptionEnglish(  
45                 ↪ descriptionEnglish  
46                 ↪ );  
47             setDescriptionSpanish(  
48                 ↪ descriptionSpanish  
49                 ↪ );  
50             setDescriptionFrench(  
51                 ↪ descriptionFrench  
52                 ↪ );  
53         }
```

```

28         setPriority(
29             ↪ indicatorPriority
30             ↪ );
31         setIndicatorVersion(
32             ↪ indicatorVersion)
33             ↪ ;
34     }
35 }
```

Como se puede apreciar, todos los campos de la clase vienen acompañados por `SerializedName` acompañado del nombre que recibe el cliente de los JSON. El modelo implementa serializable para poder pasar los objetos entre las propias actividades de la aplicación.

Por lo tanto, la comunicación de todos los casos entre cliente y servidor es la siguiente:

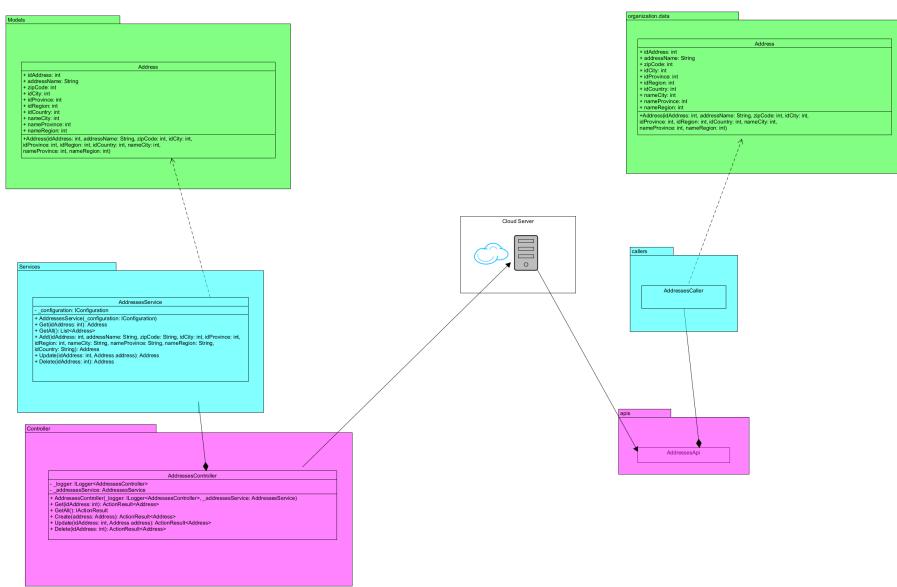


Figura C.10: Esquema de comunicación entre el cliente (lado derecho) y el servidor (lado izquierdo), para la clase `Address`

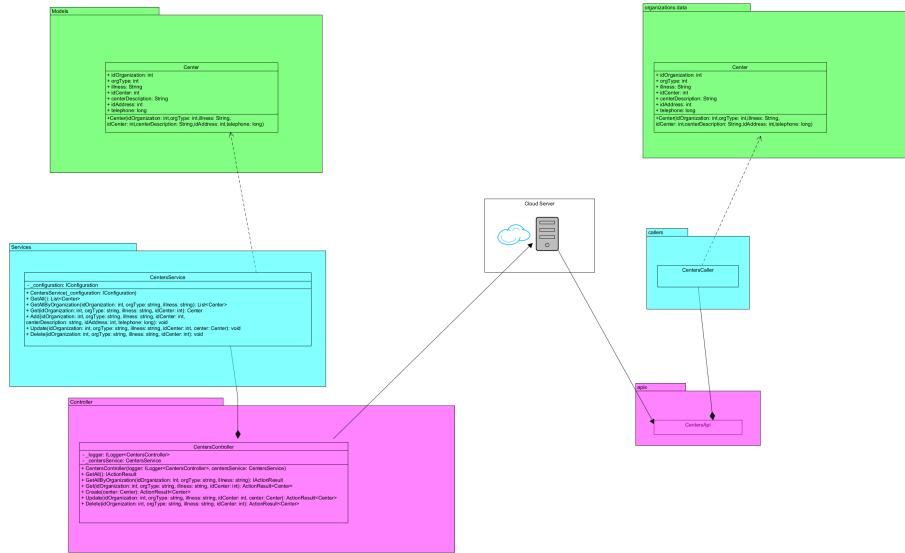


Figura C.11: Esquema de comunicación entre el cliente (lado derecho) y el servidor (lado izquierdo), para la clase Center

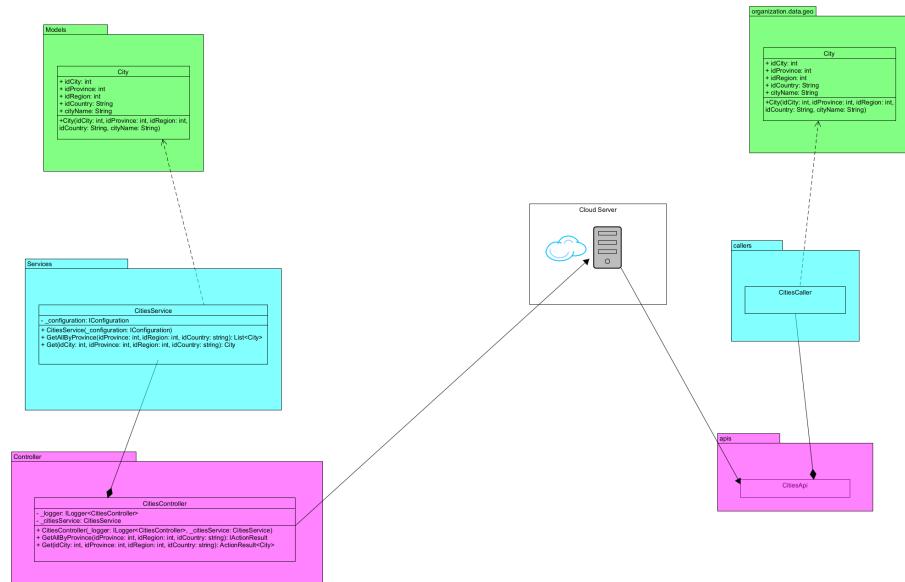


Figura C.12: Esquema de comunicación entre el cliente (lado derecho) y el servidor (lado izquierdo), para la clase City

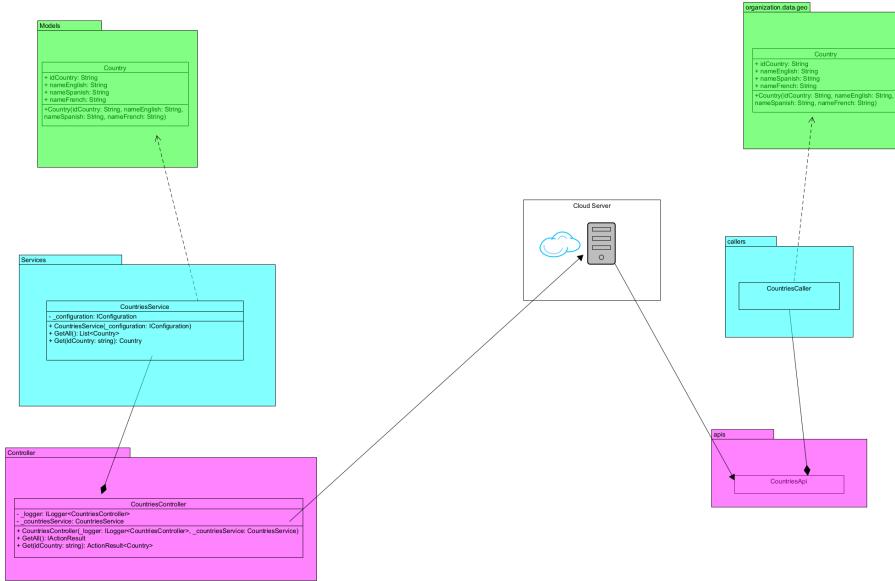


Figura C.13: Esquema de comunicación entre el cliente (lado derecho) y el servidor (lado izquierdo), para la clase Country

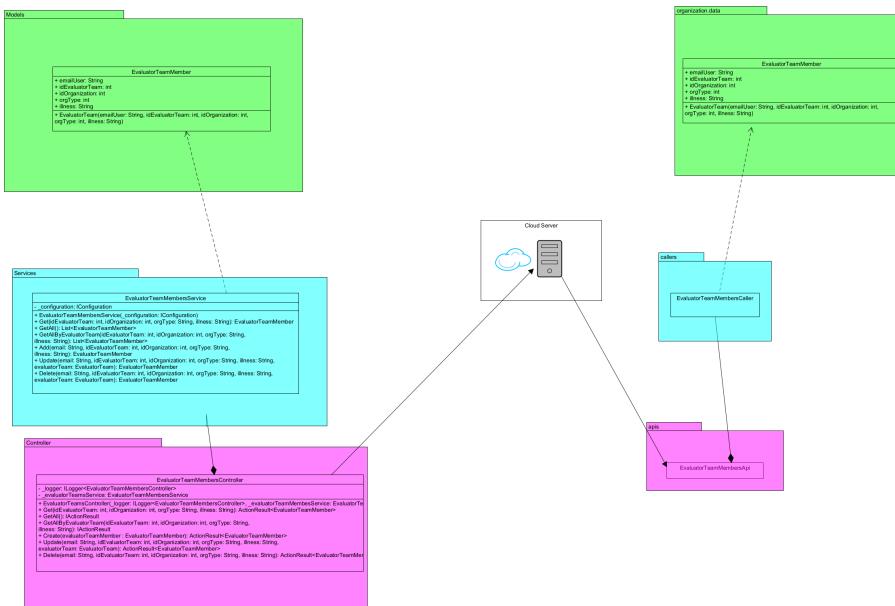


Figura C.14: Esquema de comunicación entre el cliente (lado derecho) y el servidor (lado izquierdo), para la clase EvaluatorTeamMember

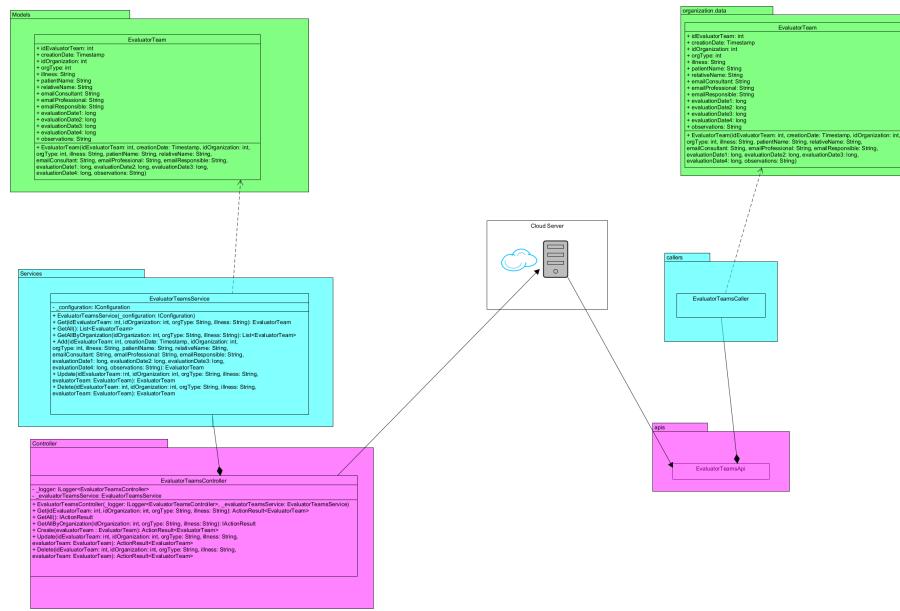


Figura C.15: Esquema de comunicación entre el cliente (lado derecho) y el servidor (lado izquierdo), para la clase `EvaluatorTeam`

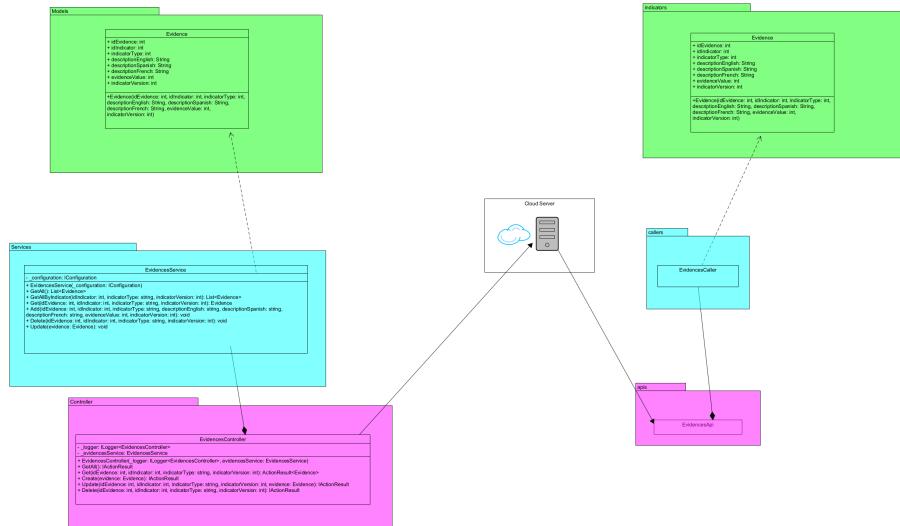


Figura C.16: Esquema de comunicación entre el cliente (lado derecho) y el servidor (lado izquierdo), para la clase `Evidence`

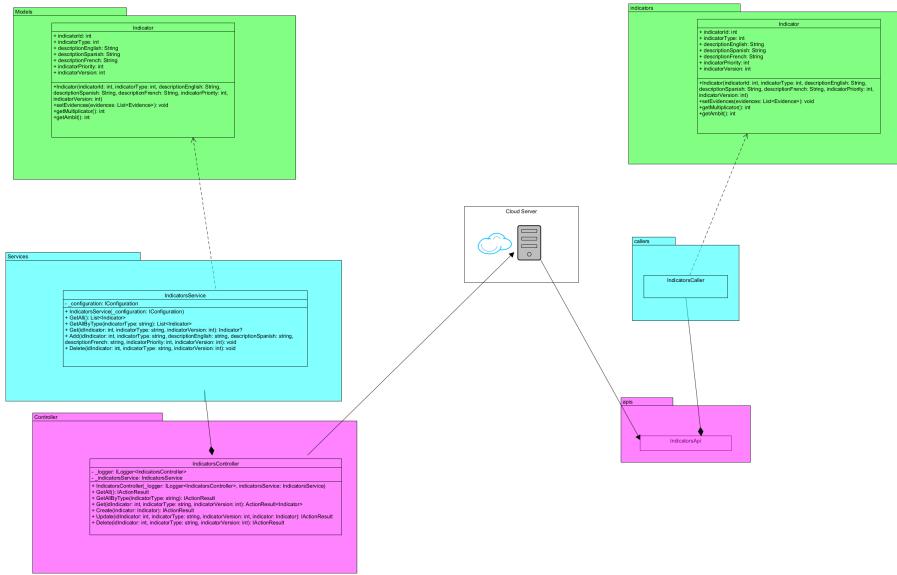


Figura C.17: Esquema de comunicación entre el cliente (lado derecho) y el servidor (lado izquierdo), para la clase Indicator

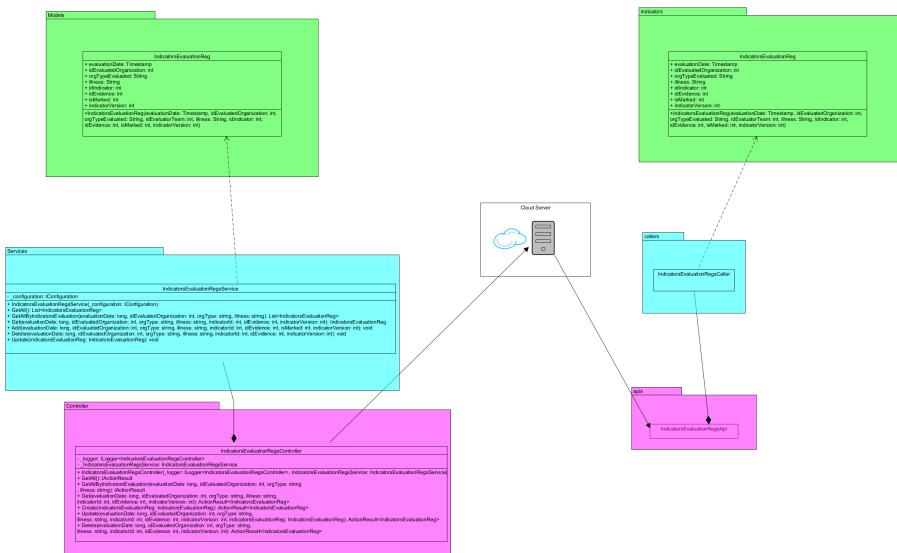


Figura C.18: Esquema de comunicación entre el cliente (lado derecho) y el servidor (lado izquierdo), para la clase IndicatorsEvaluationReg

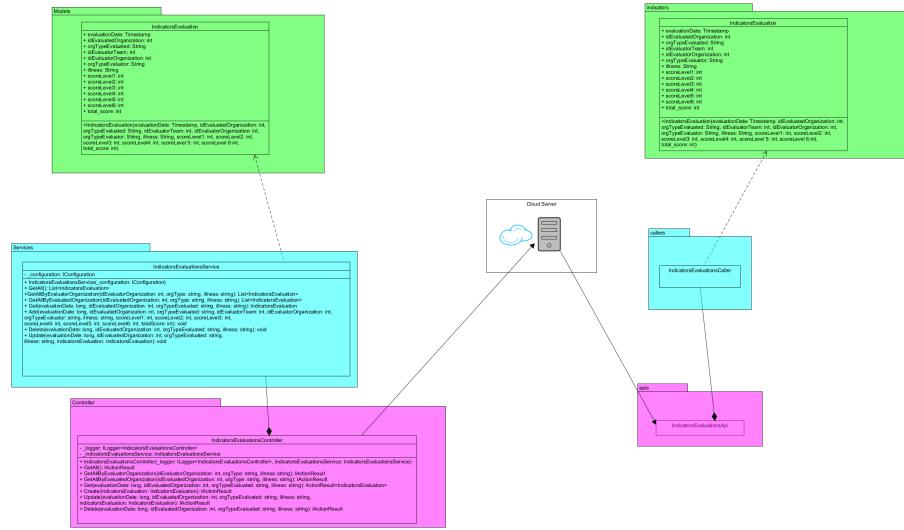


Figura C.19: Esquema de comunicación entre el cliente (lado derecho) y el servidor (lado izquierdo), para la clase IndicatorsEvaluation

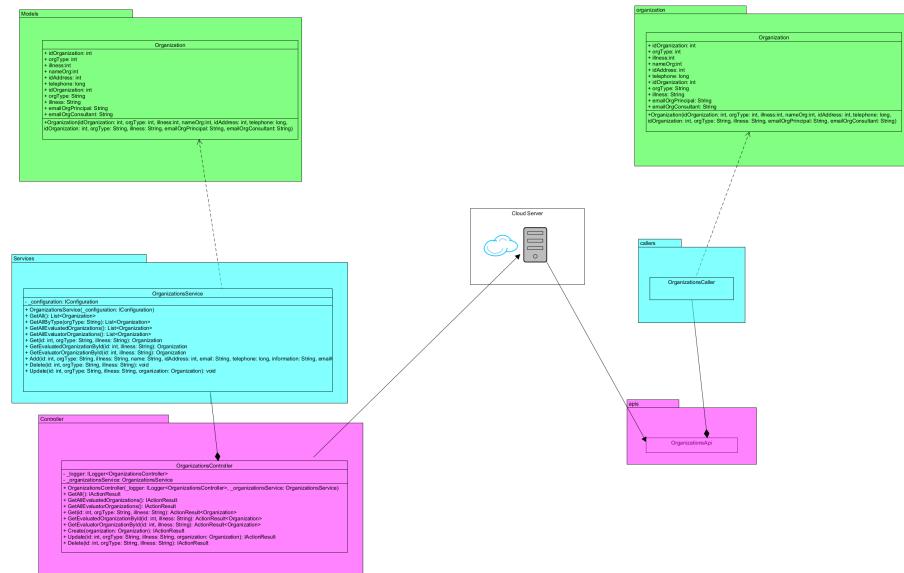


Figura C.20: Esquema de comunicación entre el cliente (lado derecho) y el servidor (lado izquierdo), para la clase Organization

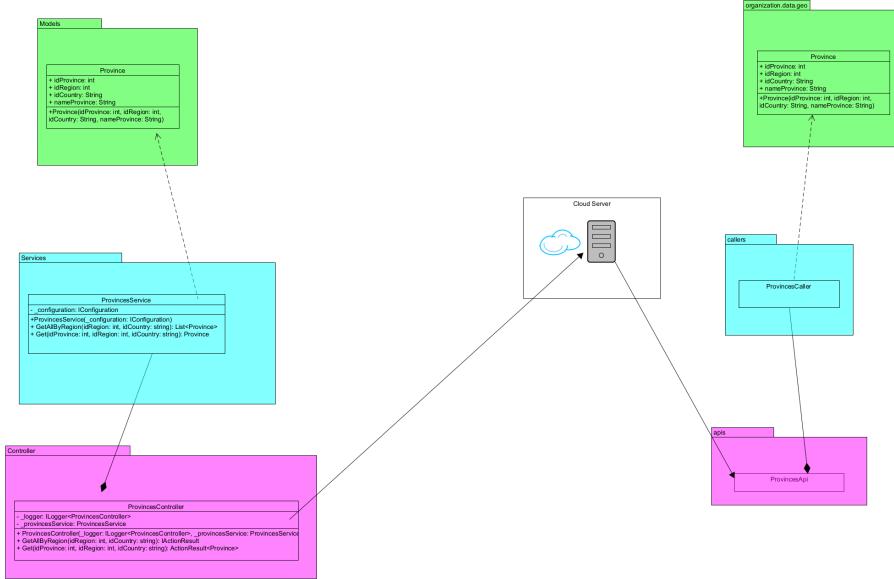


Figura C.21: Esquema de comunicación entre el cliente (lado derecho) y el servidor (lado izquierdo), para la clase Province

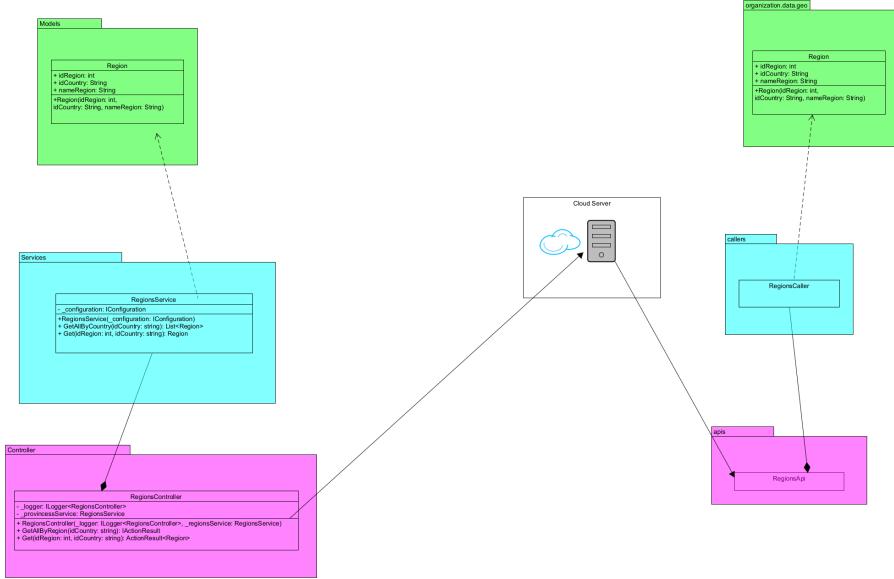


Figura C.22: Esquema de comunicación entre el cliente (lado derecho) y el servidor (lado izquierdo), para la clase Region

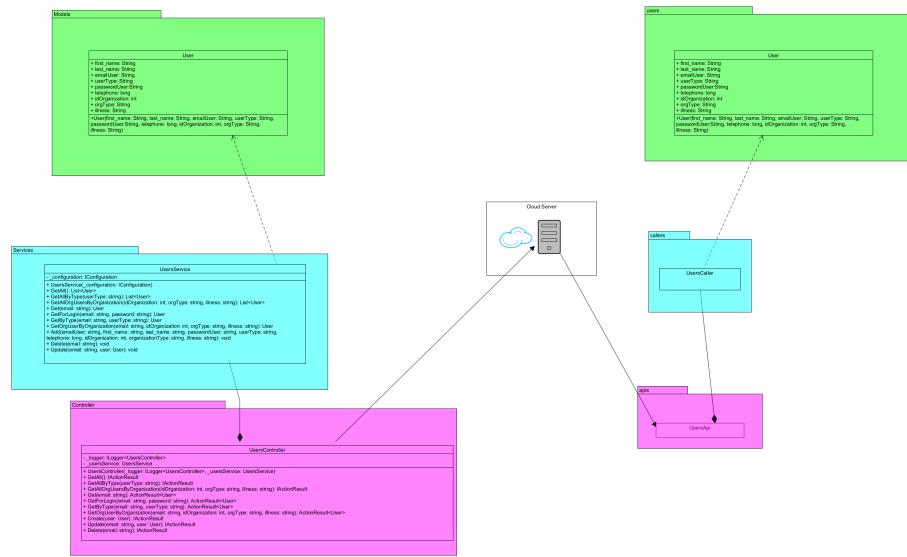


Figura C.23: Esquema de comunicación entre el cliente (lado derecho) y el servidor (lado izquierdo), para la clase User

Apéndice D

Documentación técnica de programación

D.1. Introducción

D.2. Estructura de directorios

La estructura de directorios del proyecto consta de las siguientes carpetas (teniendo en cuenta que el directorio raíz es *tfg2223*):

- *app/build/outputs/apk/debug*: Lugar donde se ubica la apk en el directorio.
- *app/main/java/gui/adapters*: Directorio que alberga las vistas adaptativas de los spinner o desplegables adaptados a cada una de las clases que lo requieren.
- *app/main/java/gui/data*: Clases auxiliares que ayudan a manejar la actividad de inicio de sesión
- *app/main/java/gui/mainMenu/admin*: Directorio que alberga la lógica detrás del menú principal del administrador
- *app/main/java/gui/mainMenu/evaluated*: Directorio que alberga la lógica detrás del menú principal del usuario de organización evaluada.
- *app/main/java/gui/mainMenu/evaluator*: Directorio que alberga la lógica detrás del menú principal del usuario de la *Fundación Miradas*.

- *connection/src/main/java/cli*: Directorio donde se albergan los modelos de las diferentes entidades manejadas por la aplicación.
- *connection/src/main/java/otea/connection*: Directorio donde se albergan los callers y los api del cliente
- *connection/src/main/java/misc*: Directorio donde se ubican clases auxiliares para formatear datos o comprobar los mismos.
- *oteaserver*: Directorio donde se encuentra el código del servidor.
- *videos*: Directorio donde se encuentran los vídeos del usuario.

D.3. Manual del programador

D.4. Compilación, instalación y ejecución del proyecto

Android Studio

En el lado del cliente cabe resaltar que se ha utilizado *Android Studio*, cuya descripción aparece en el apartado de Técnicas y Herramientas de la memoria de este proyecto. En este caso, la instalación consta de los siguientes pasos:

1. En la página de descarga de la aplicación se selecciona el botón **Android Studio Electric Eel** para descargar el instalador de Android Studio:

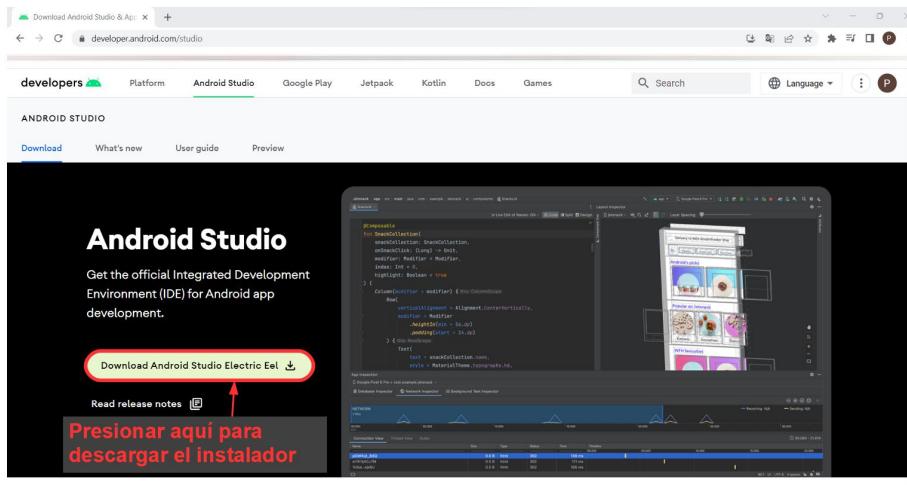


Figura D.1: Página de descarga del instalador de Android Studio

2. Posteriormente, se tienen que aceptar los términos y condiciones marcando la opción *I have read and agree with the above terms and conditions*, para presionar posteriormente el botón **Download Android Studio Electric Eel | 2022.1.1 for Windows**.



Figura D.2: Sección de términos y condiciones anterior a la descarga del instalador de Android Studio

3. Posteriormente se obtiene el instalador:

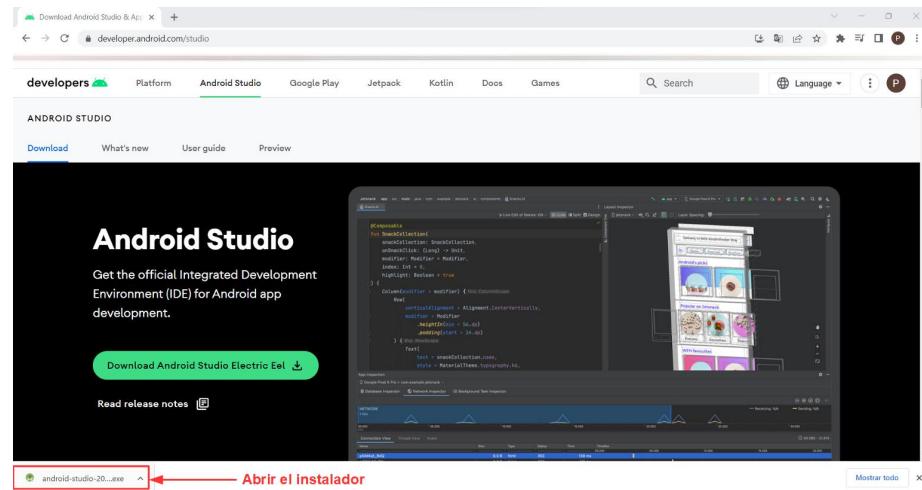


Figura D.3: El fichero de instalación ya ha sido descargado

4. Ya abierto el asistente de instalación, se tienen que seguir los pasos que nos piden:
 - a) En primer lugar se muestra el mensaje de bienvenida del asistente de instalación:

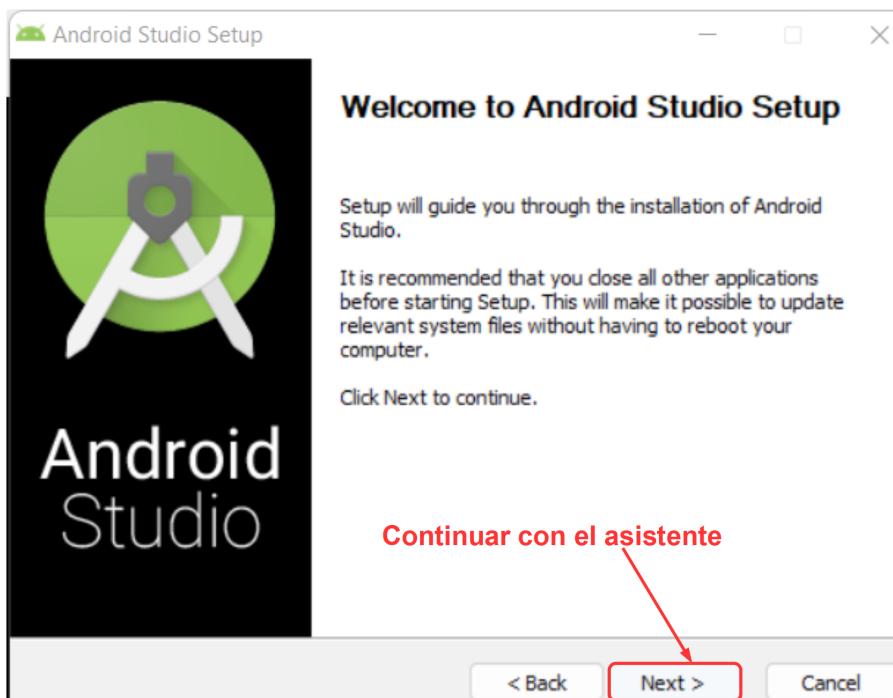


Figura D.4: Mensaje de bienvenida del instalador de Android Studio

- b) Posteriormente se seleccionan los componentes a instalar. Se seleccionan tanto *Android Studio*, que es la propia IDE de Google para el desarrollo de aplicaciones para Android, como *Android Virtual Device*, que se utiliza de forma integrada en Android Studio para la simulación del funcionamiento de las aplicaciones para diferentes aplicaciones:

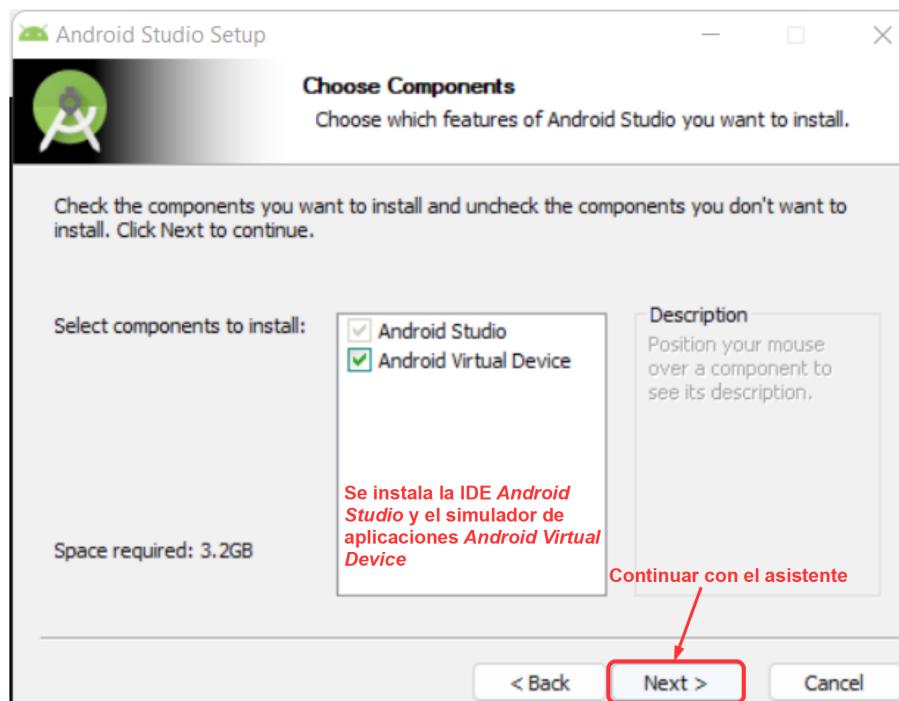


Figura D.5: Selección de componentes a instalar en el instalador de Android Studio

- c) Más adelante se establece un directorio para la instalación del programa. Se puede elegir libremente dicho directorio, pero en este caso se va a escoger el directorio por defecto:

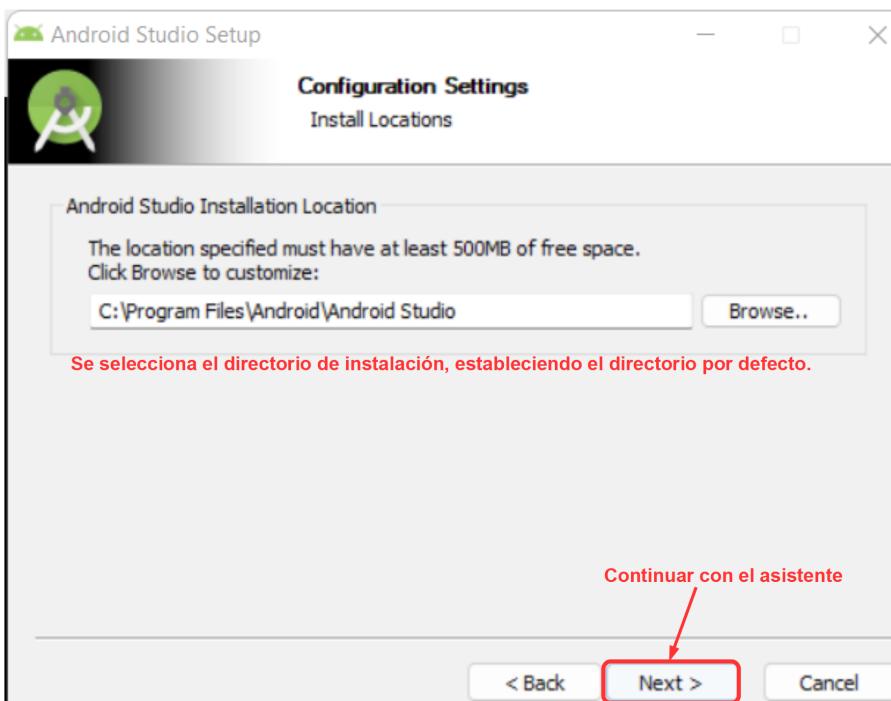


Figura D.6: Selección del directorio de instalación

- d) Además del directorio de instalación se puede establecer un directorio donde guardar el acceso directo al programa en el menú de inicio. No es un paso obligatorio, pero se establecerán las opciones por defecto:

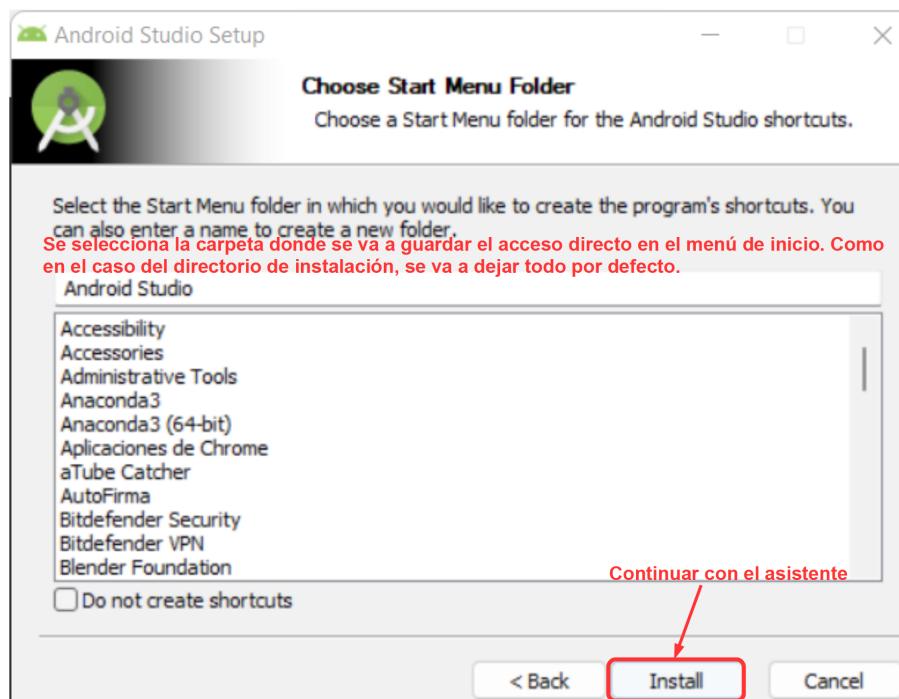


Figura D.7: Selección del directorio de creación de accesos directos en el menú de inicio

- e) Cuando se han seguido los pasos anteriores, se procede con la instalación:

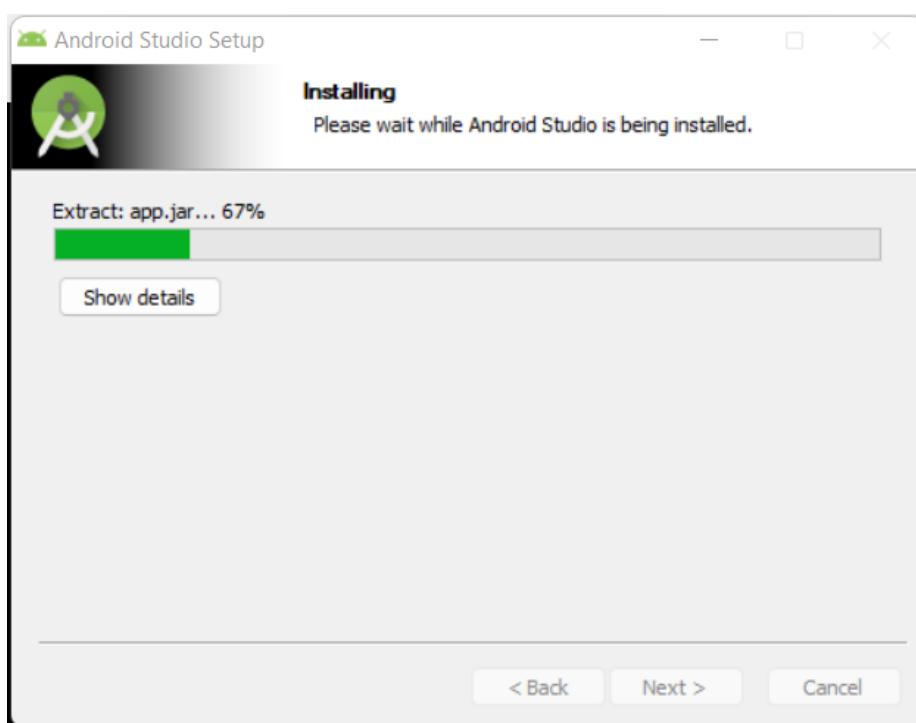


Figura D.8: Inicio de instalación de Android Studio

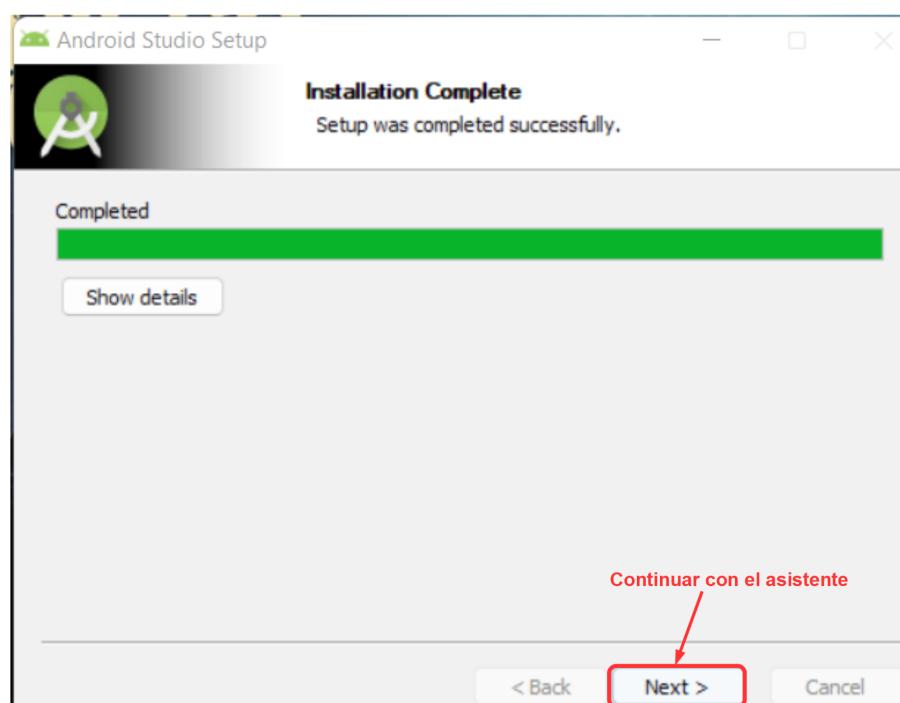


Figura D.9: Finalización de la instalación de Android Studio

- f) Por último se marca la opción de iniciar Android Studio para proceder así con la configuración del mismo:

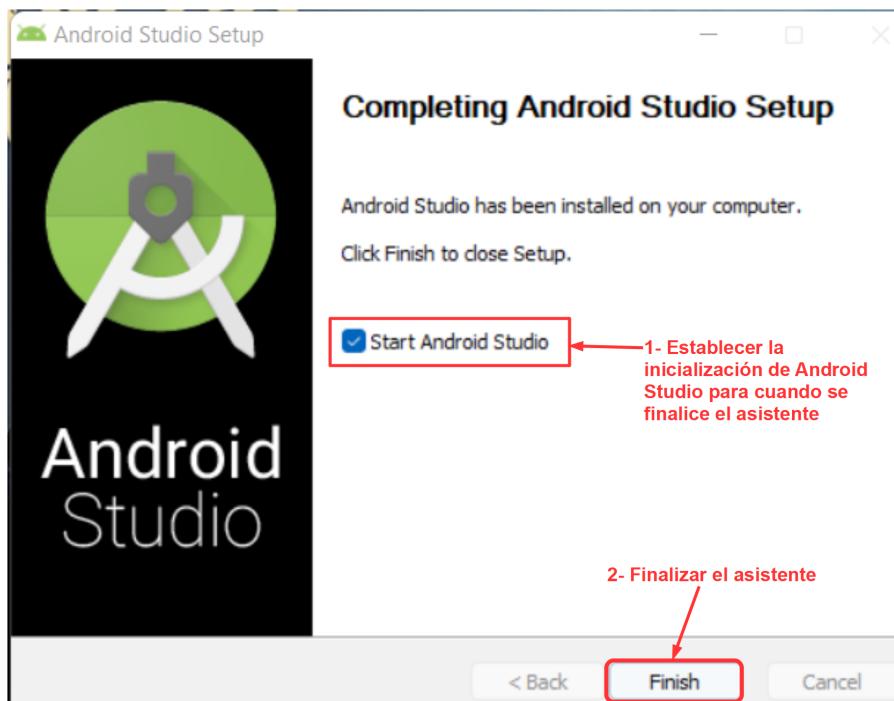


Figura D.10: Mensaje de finalización de la instalación

Tras haber finalizado con la instalación de *Android Studio*, se procede con su configuración inicial. En cuanto a la configuración inicial de *Android Studio* y de la aplicación, los pasos a seguir son los siguientes:

1. En primer lugar, *Android Studio* da la opción de importar la configuración de otro directorio, pero como es una instalación nueva, no se precisa de importar configuración alguna:
2. Posteriormente se muestra un mensaje para recolectar información sobre *Android Studio* y sus diferentes herramientas, el cual se va a rechazar:
- 3.

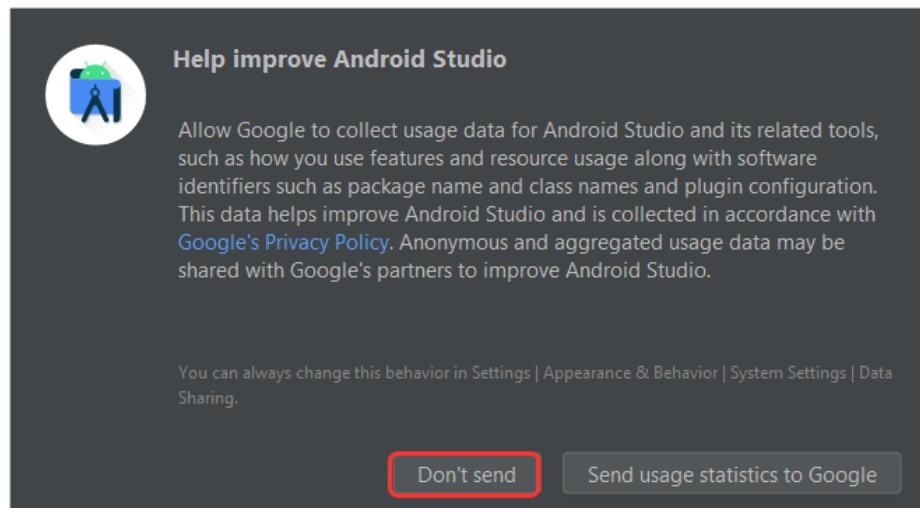


Figura D.11: Ventana emergente de importación de configuración

4. A continuación aparece el asistente de configuración de Android Studio, el cual aparece únicamente en la primera ejecución del programa. Para ir avanzando con el asistente se presiona el botón **Next**, el cual lleva al usuario al siguiente paso de la configuración:

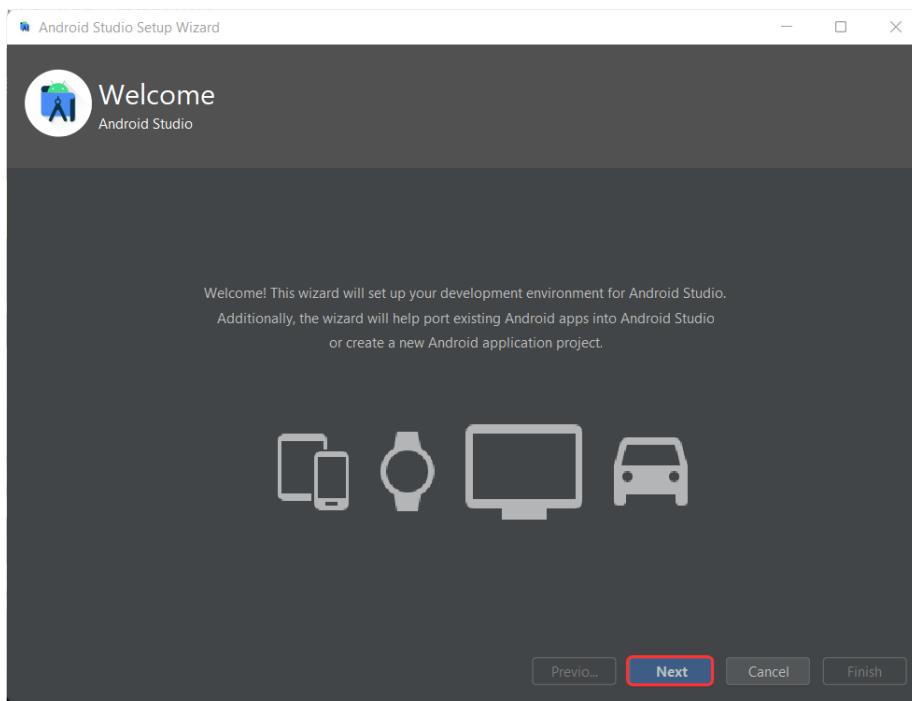


Figura D.12: Mensaje de bienvenida del asistente de configuración inicial de Android Studio

5. Como tipo de instalación, Android Studio proporciona dos diferentes opciones, **la opción estándar**, la cual instala los componentes predeterminados, y **la opción personalizada**, la cual permite al usuario seleccionar la configuración y los componentes a instalar:

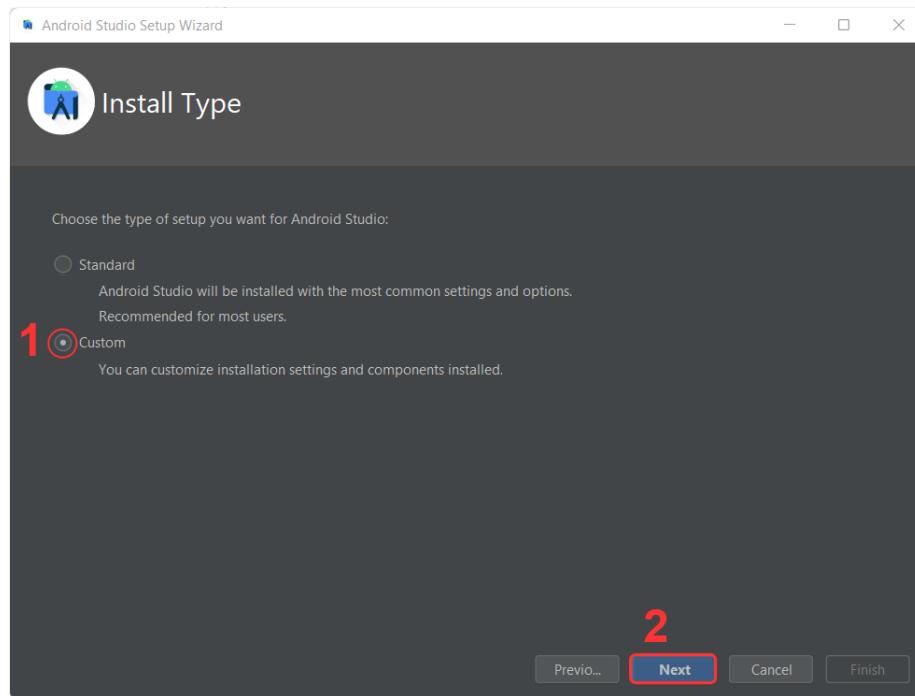


Figura D.13: Selección del tipo de instalación de los componentes de Android Studio

6. En cuanto a la selección del directorio del JDK, Android Studio mostrará el JDK instalado más reciente. También se puede personalizar si se dispone de más de un JDK en el equipo.

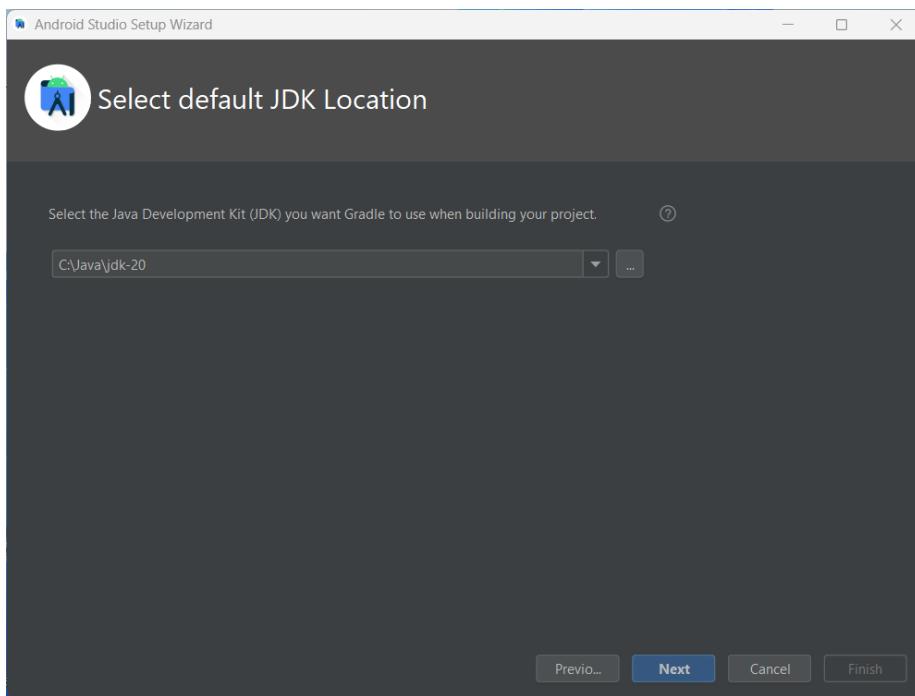


Figura D.14: Selección del directorio predeterminado del JDK

7. A continuación se selecciona el diseño de la interfaz, el cual es irrelevante para el correcto funcionamiento de la herramienta:

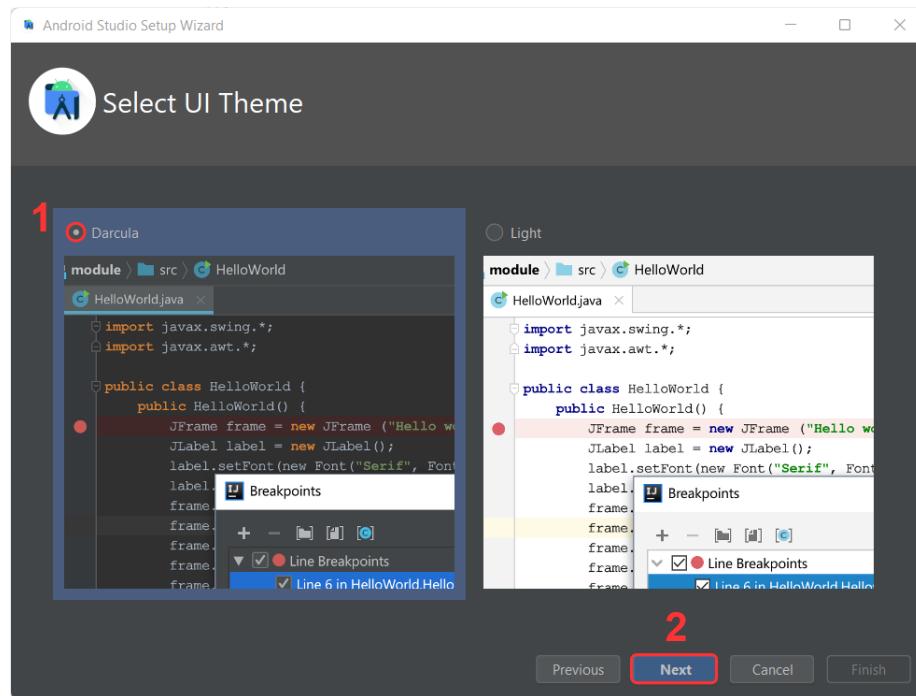


Figura D.15: Selección del diseño de la interfaz de Android Studio

8. Posteriormente se seleccionan los componentes SDK (*Software Development Kit*) a instalar, los cuales permiten desarrollar las aplicaciones para Android. En este caso se seleccionarán los componentes de la captura posterior:

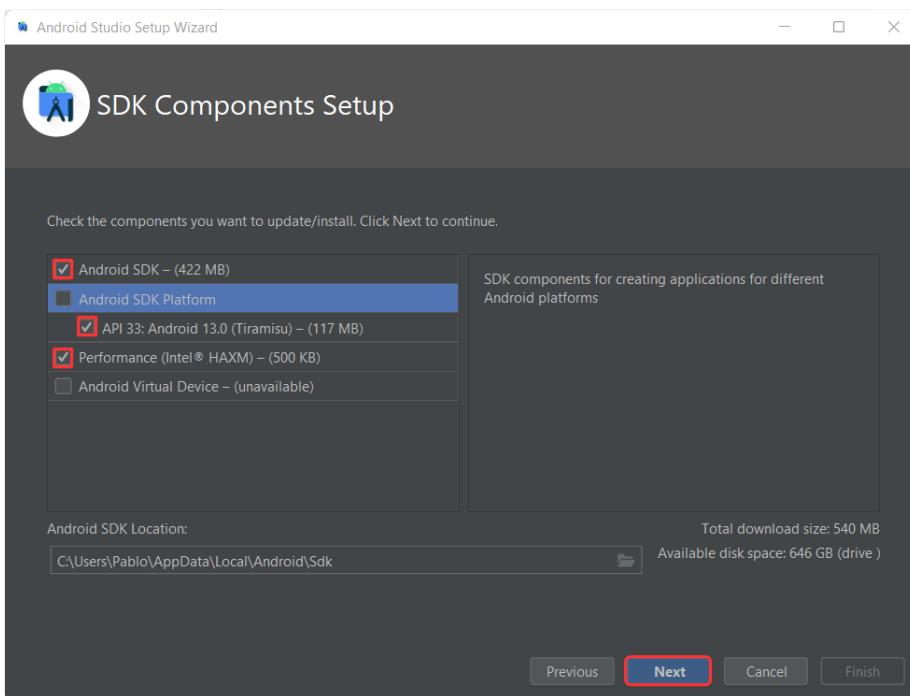


Figura D.16: Selección de componentes SDK a instalar en Android Studio

Como se puede comprobar, en el momento de haberse realizado la captura anterior, el componente Android Virtual Device no está disponible. Aun así, podemos proseguir con la instalación presionando **OK**:

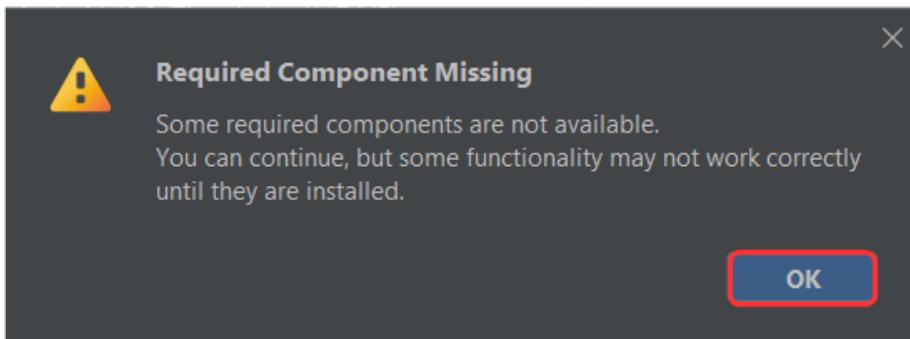


Figura D.17: Advertencia por falta de componentes a instalar

9. Posteriormente se configura la memoria RAM máxima que utilizará el emulador de aplicaciones. En este caso dejamos la opción recomendada de 2GB de RAM:

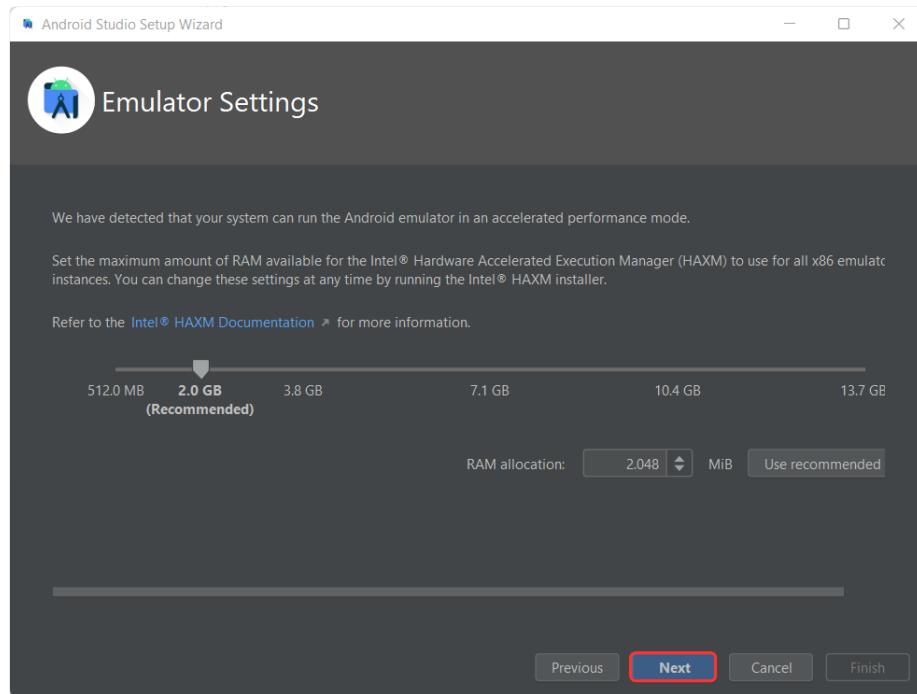


Figura D.18: Configuración de la memoria RAM utilizada por el emulador de Android

10. Más adelante se comprueban todos los componentes a instalar y a configurar durante la configuración inicial de Android Studio:

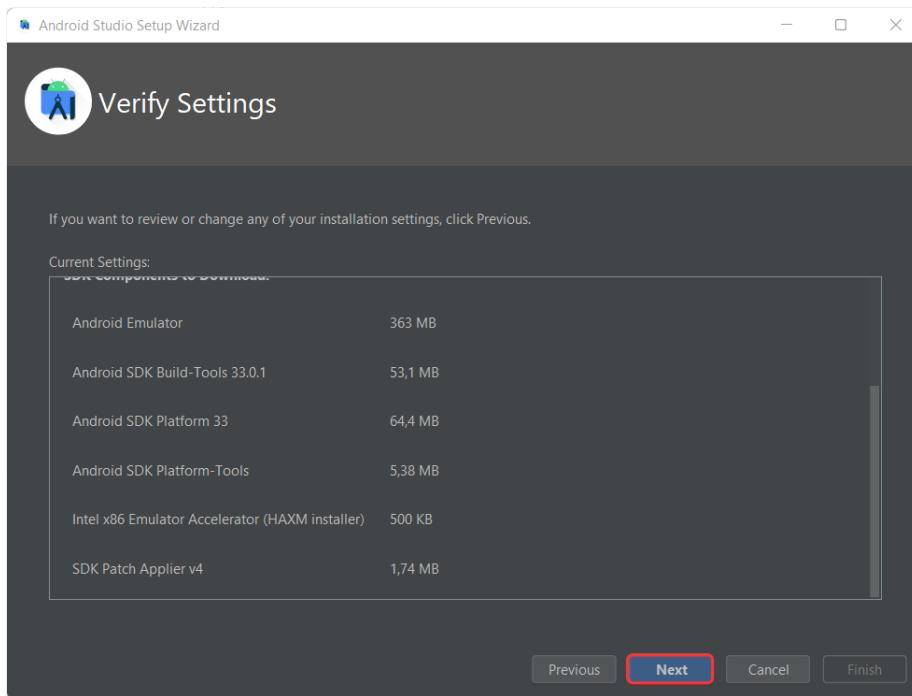


Figura D.19: Verificación de componentes a instalar en Android Studio

11. Antes de instalar los componentes, se tienen que aceptar la licencia del software tanto del *paquete SDK* como del *acelerador de emulación de Intel*:

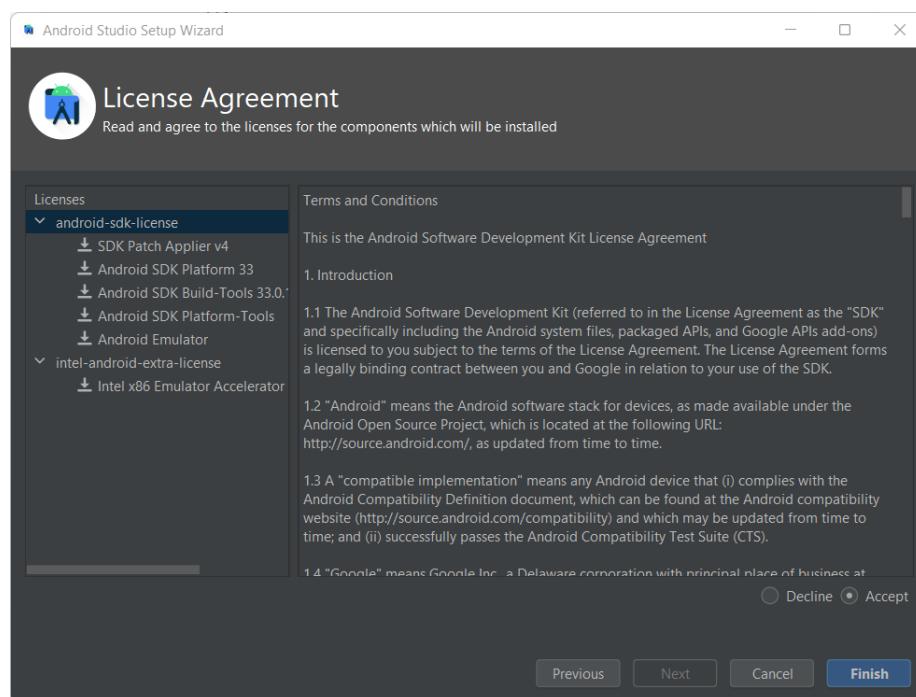


Figura D.20: Licencia de software del paquete SDK

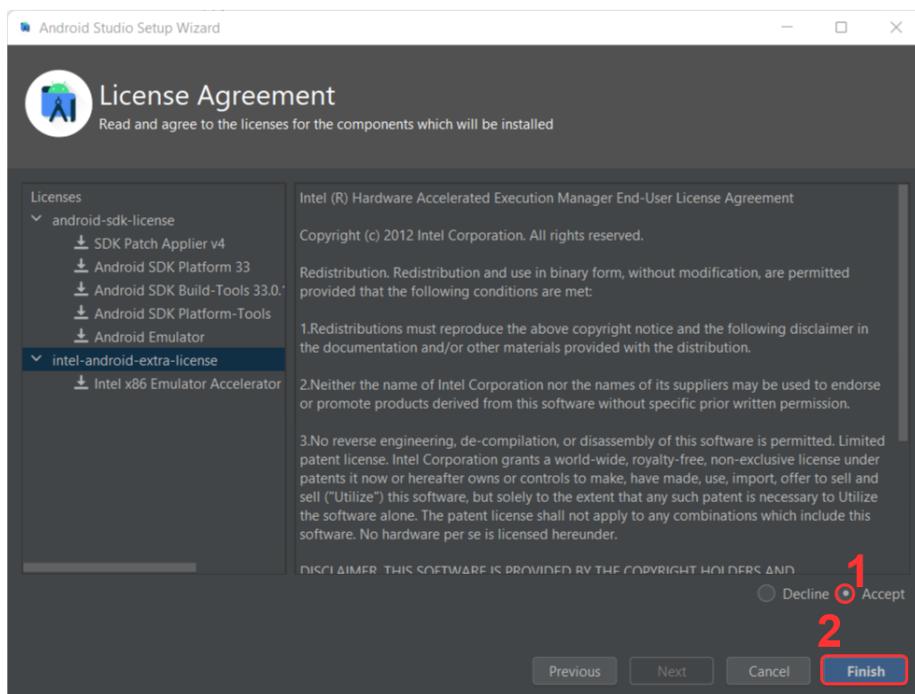


Figura D.21: Licencia de software del acelerador de emulación de Intel

12. Por último se procede con la instalación de los componentes mencionados con anterioridad y se finaliza con el asistente de configuración inicial presionando el botón **Finish**:

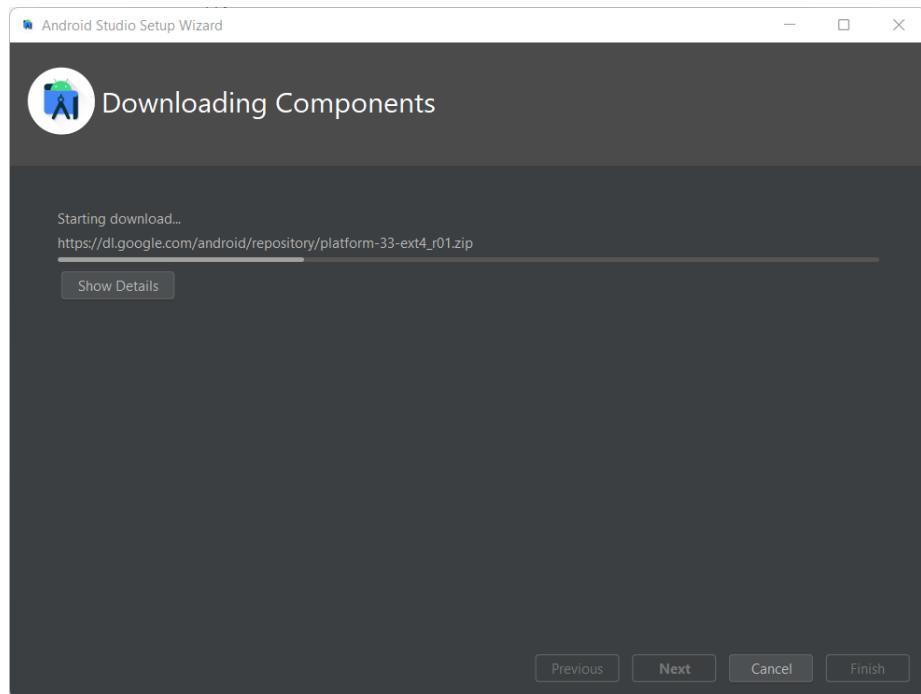


Figura D.22: Instalación de los componentes en la configuración inicial

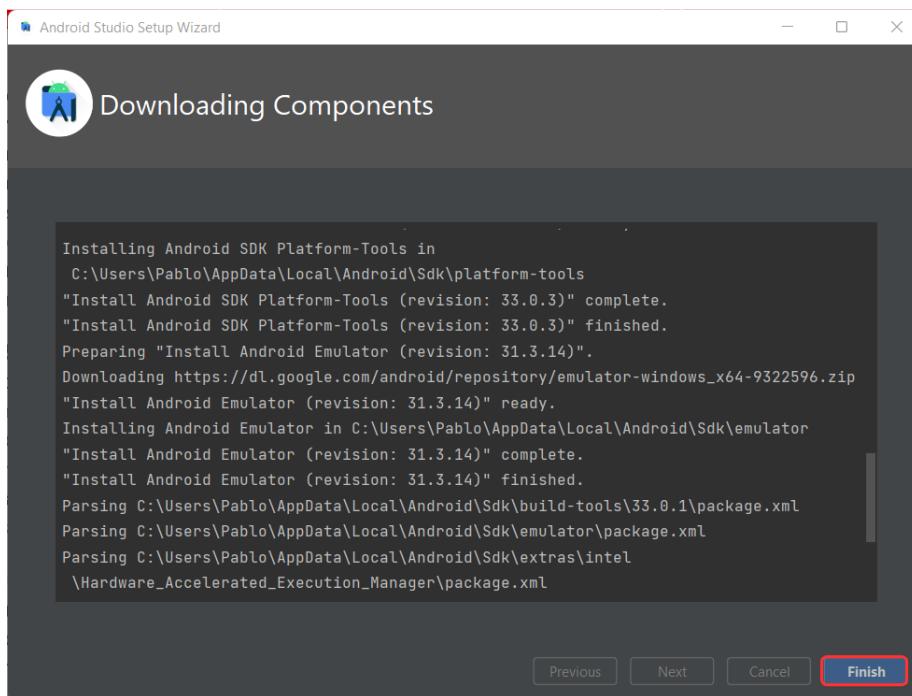


Figura D.23: Finalización del asistente de instalación de componentes

Tras haber finalizado con la instalación y con la configuración inicial de Android Studio, se tiene que iniciar un proyecto inicial donde se va a desarrollar la aplicación. Los pasos a seguir son los siguientes:

1. En primer lugar en la pantalla de bienvenida se presiona sobre el botón **New Project** para crear un proyecto de aplicación nuevo:

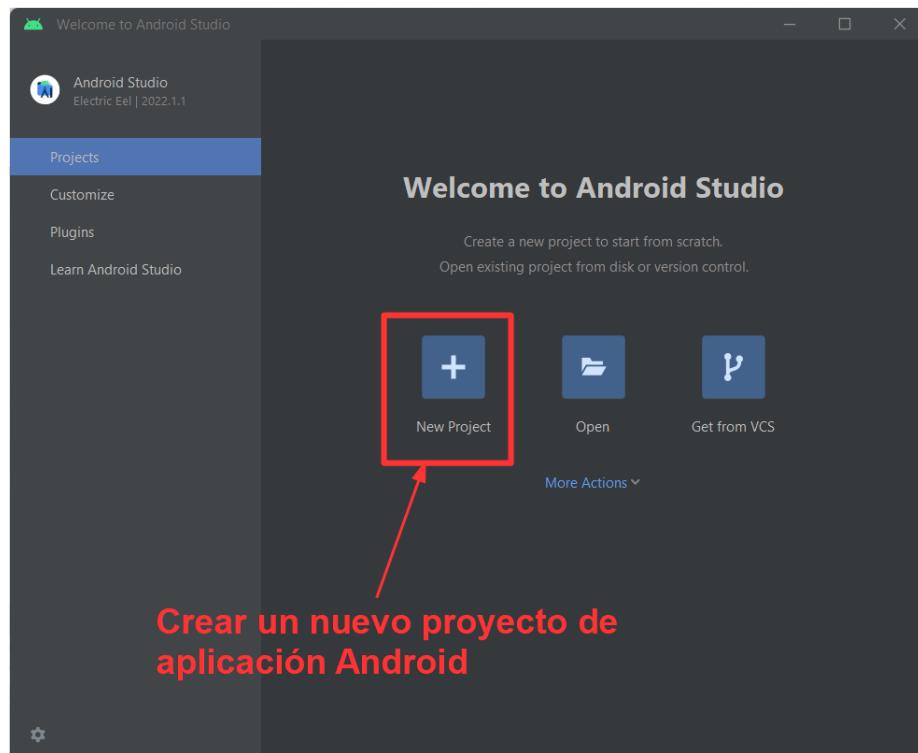


Figura D.24: Pantalla de selección de proyecto

2. Posteriormente se selecciona el diseño de la actividad principal de la aplicación. Para simplificar la estructura inicial del proyecto se selecciona la actividad vacía o *Empty Activity*. Al tener la actividad ya seleccionada se presiona el botón **Next** para continuar con la configuración de la actividad:

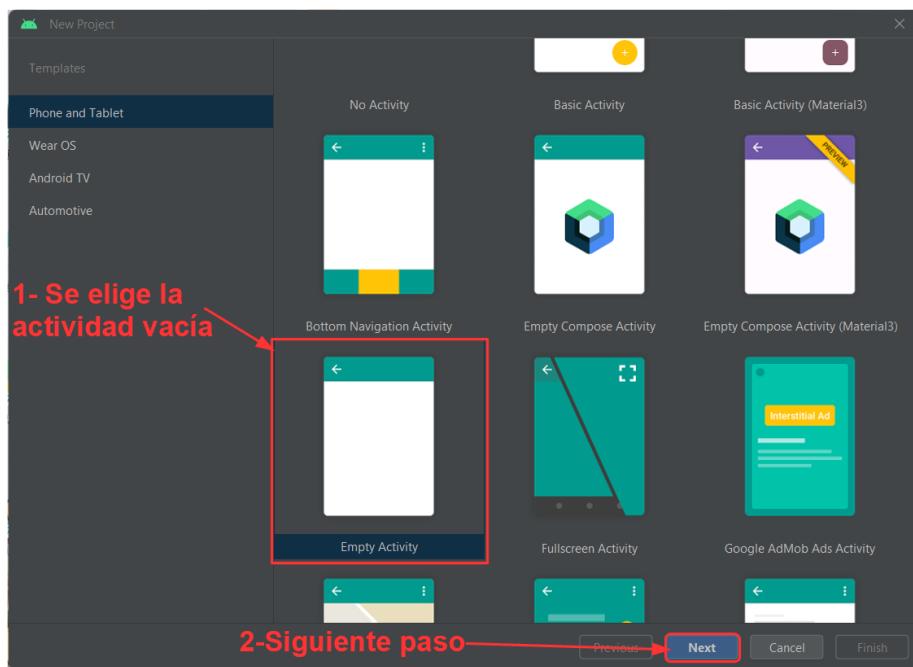


Figura D.25: Pantalla de selección de actividad principal

3. Ya en la pantalla de configuración de la actividad, se puede modificar el nombre de la misma, el nombre del paquete a la que pertenece, el directorio donde se va a guardar, el lenguaje de programación en la que va a ser programada y la versión mínima de Android en la que puede ser ejecutada:

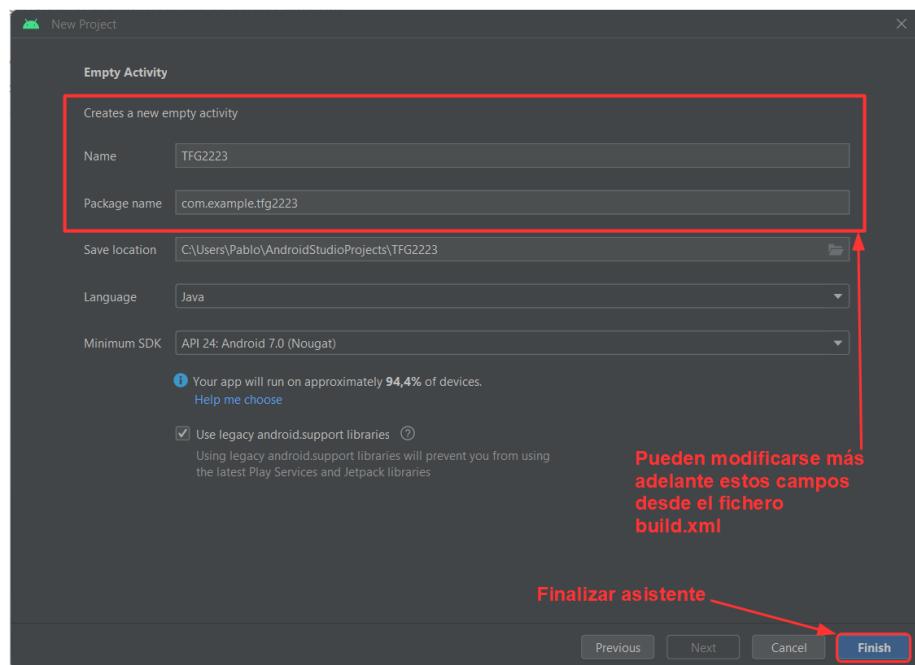


Figura D.26: Pantalla de configuración de la actividad

Cuando ya se ha configurado la actividad, se instalará el soporte JDK para la emulación de la misma:

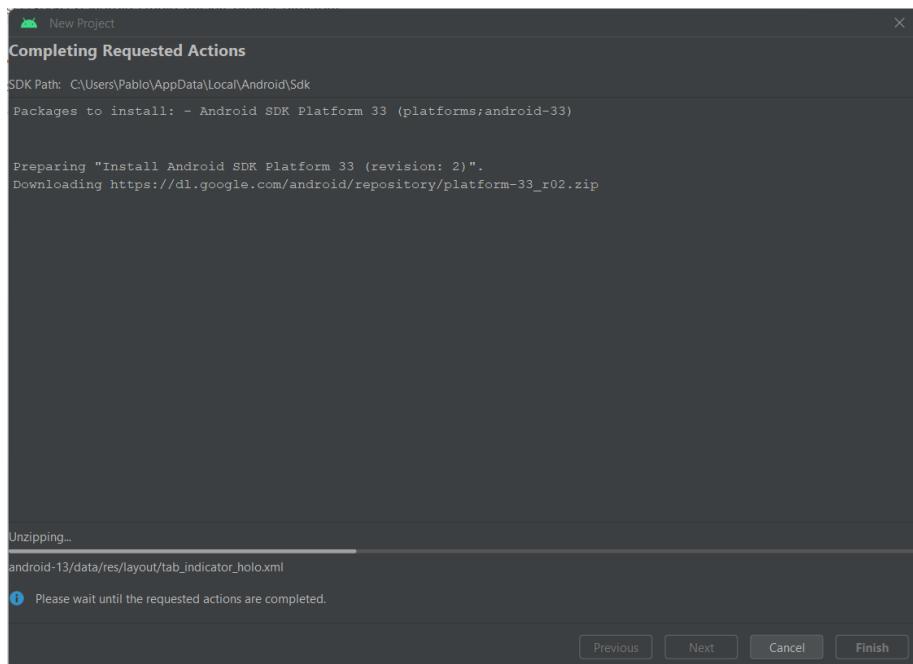


Figura D.27: Desarrollo del proceso de instalación del soporte JDK

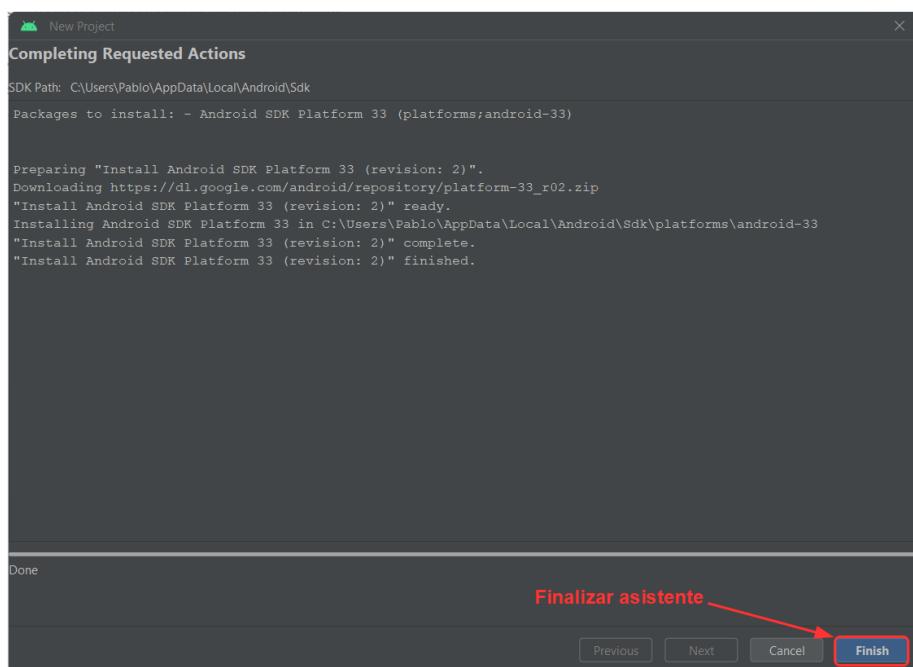


Figura D.28: Finalización del proceso de instalación del soporte JDK

Cuando ya se haya terminado con el asistente de creación del proyecto, ya nos mostrará el código Java de esta actividad vacía que acabamos de crear, además de que en la parte izquierda de la pantalla encontramos todas las rutas creadas dentro del directorio del proyecto.

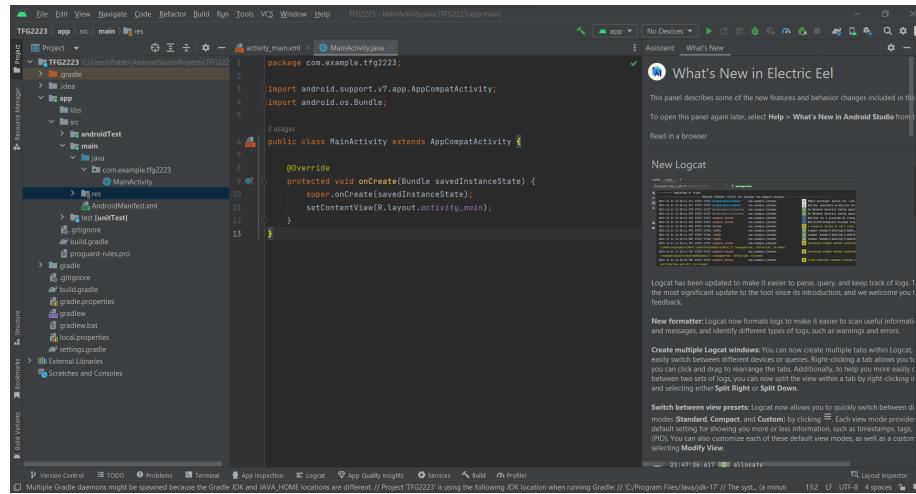


Figura D.29: Interfaz de Android Studio con el proyecto Android recién creado

4. Tras haberse creado el proyecto, se va a proceder a añadir una cuenta de GitHub para publicar los progresos que vayan realizándose durante el desarrollo de la aplicación. Para ello en primer lugar se va a **File → Settings:**

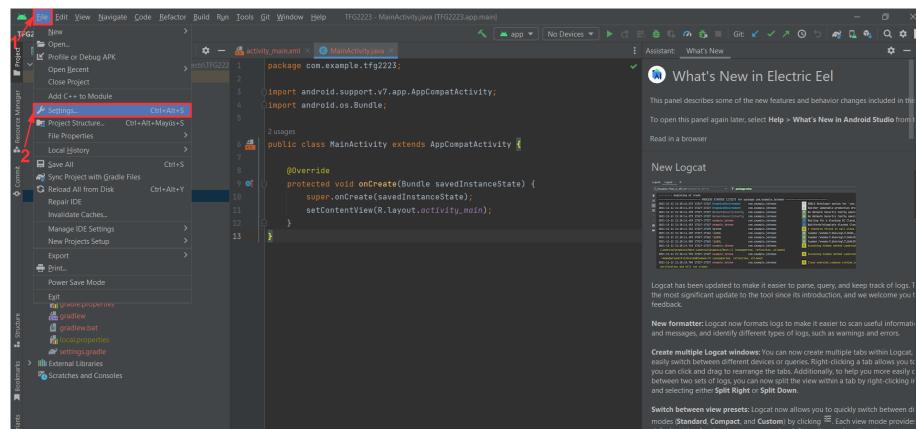


Figura D.30: Acceso a las opciones de Android Studio

5. Posteriormente se tiene que acceder a **Version Control → GitHub**

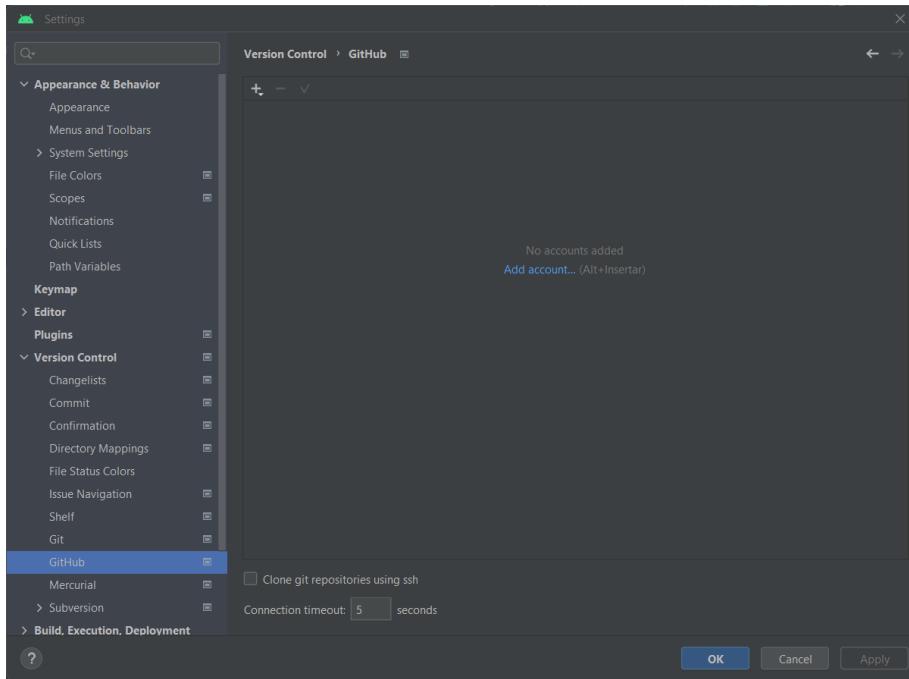


Figura D.31: Pantalla de control de cuentas de GitHub

6. Más adelante se accede a la configuración de GitHub, concretamente en **Settings → Developer Settings → Personal access tokens → Tokens (classic)**, para ir a la página de muestra de tokens de usuario **Generate new token → Generate new token (classic)**:

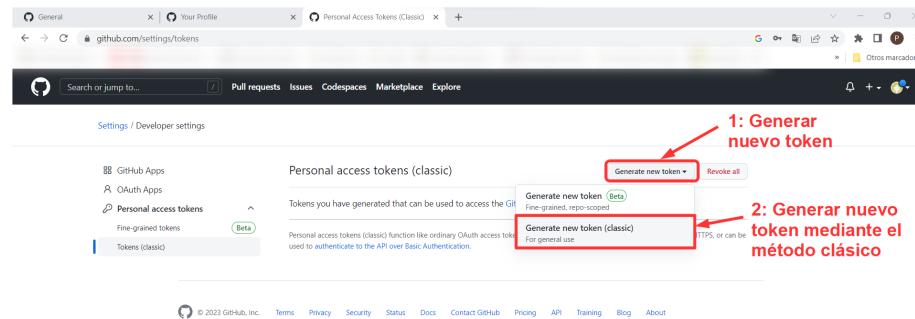


Figura D.32: Pantalla de tokens de usuario antes de ir a la pantalla de generación de tokens

7. En cuanto a la generación del token, se menciona en una nota para qué queremos utilizar ese token, la fecha de expiración del mismo (la cual se aconseja que sea de duración hasta junio, fecha de la defensa) y se seleccionan todos los permisos de uso del token:

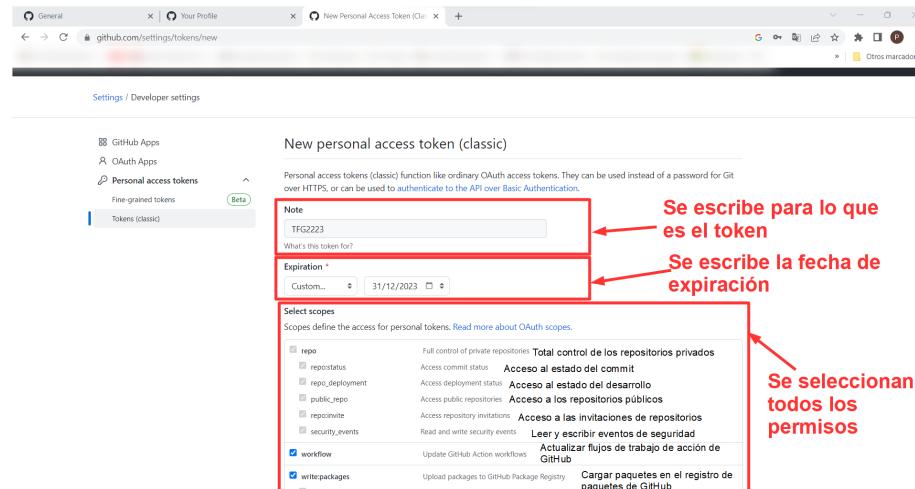


Figura D.33: Pantalla de generación de tokens con los permisos de usuario

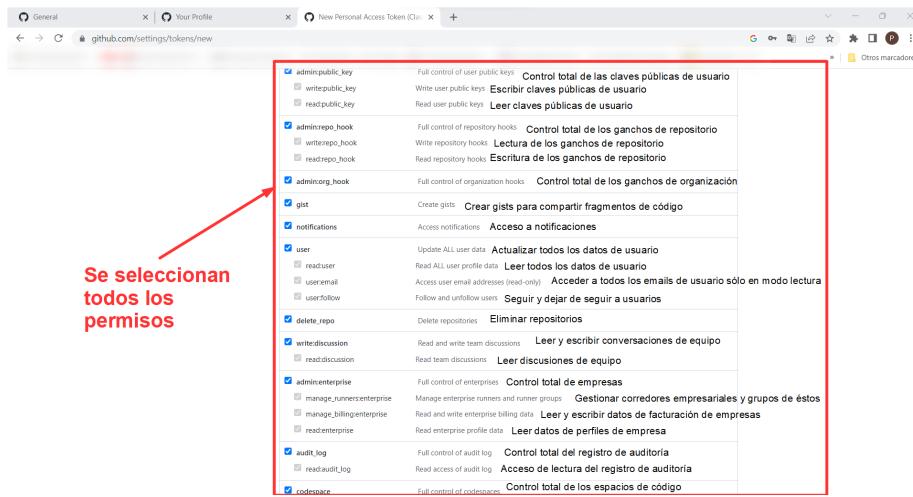


Figura D.34: Siguientes permisos de usuario

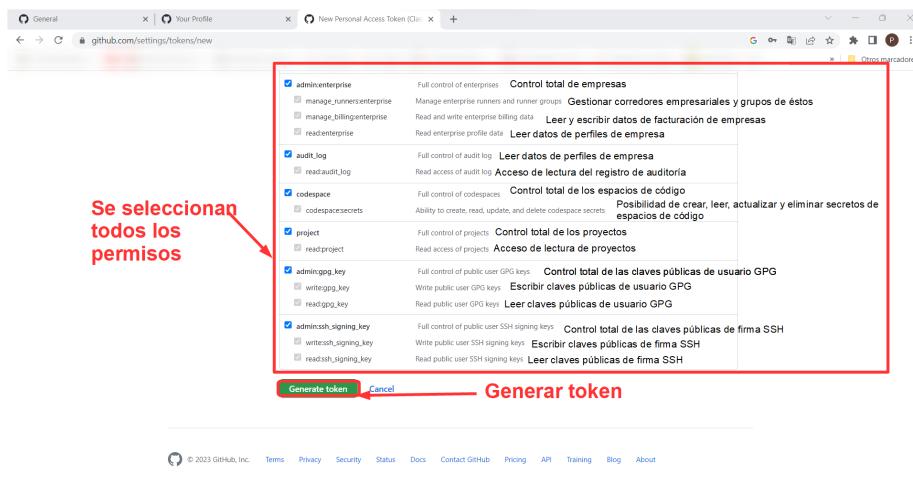


Figura D.35: Últimos permisos de usuario

8. Con el token recién generado, se copia para poder añadirlo a Android Studio:

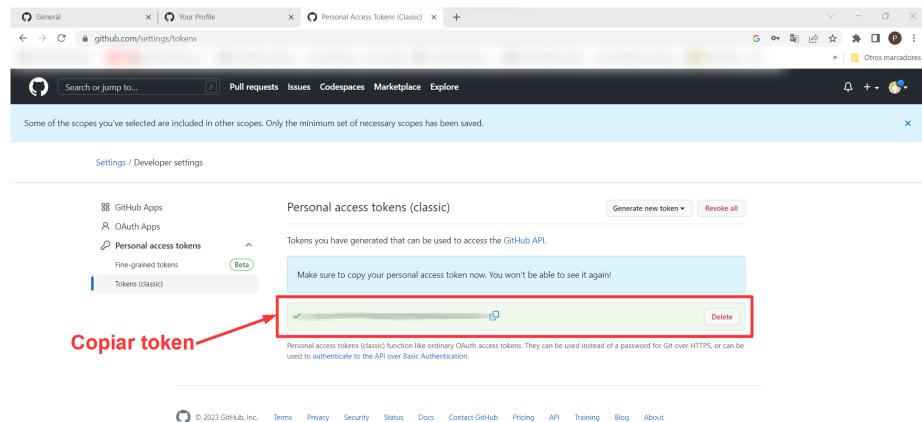


Figura D.36: Pantalla de tokens de usuario con el token de usuario generado

En la pantalla de control de cuentas de GitHub en Android Studio, se presiona en **Add account** y se añade el token

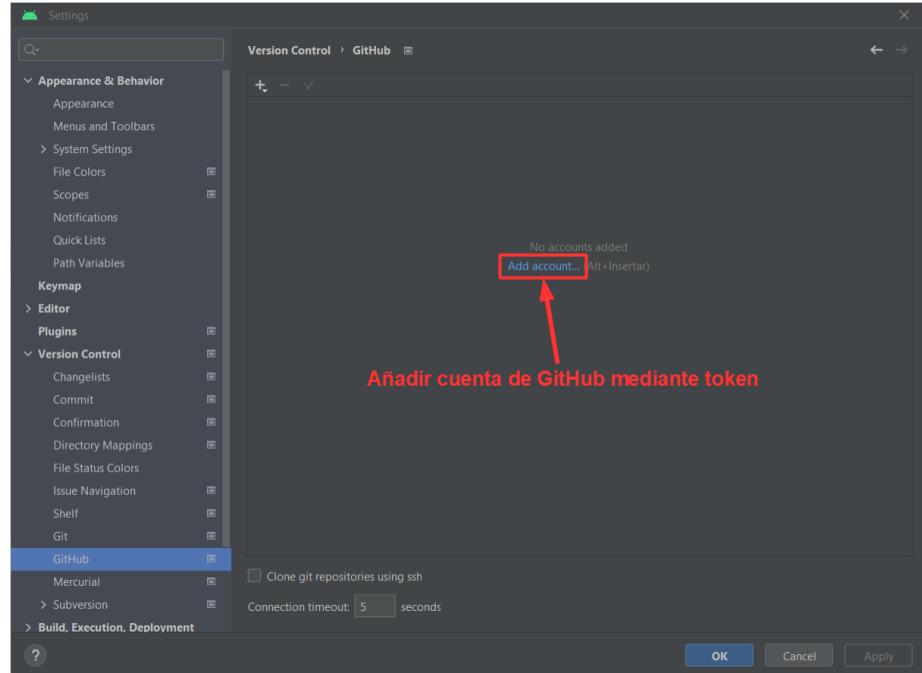


Figura D.37: Pantalla de control de cuentas de GitHub lista para añadir usuario

Para confirmar la agregación del token, se presiona sobre el botón **Add Account**:

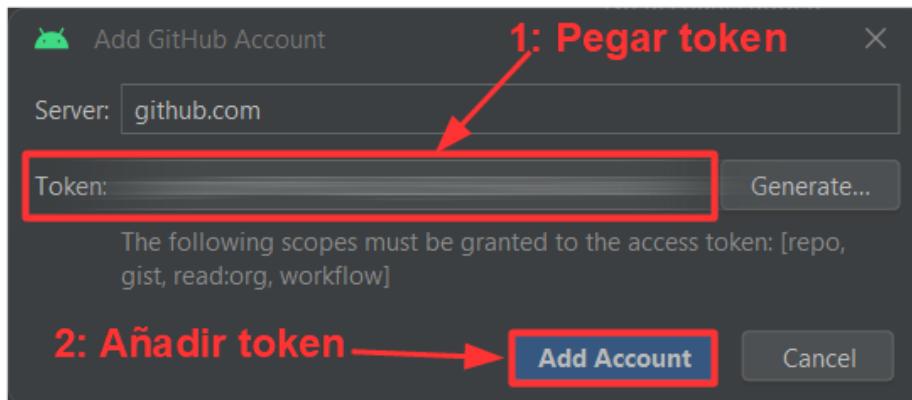


Figura D.38: Añadir token de usuario de GitHub en Android Studio

Con el token ya añadido, se presiona el botón **Ok**:

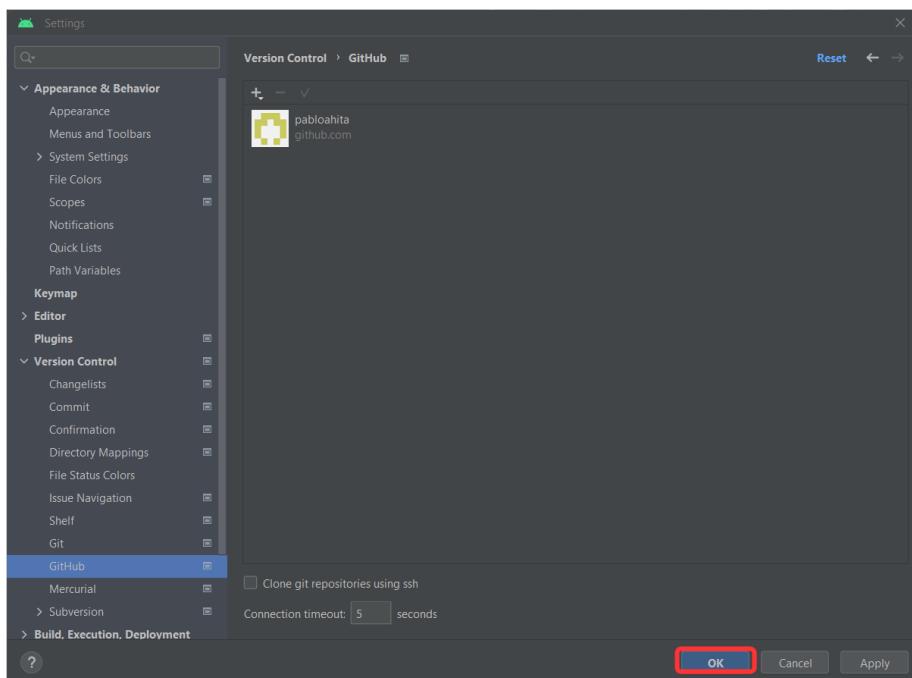


Figura D.39: Cuenta de GitHub recién añadida

9. Tras haber añadido satisfactoriamente la cuenta de GitHub a Android Studio, se va a proceder a añadir un dispositivo virtual. Para ello

se accede al menú desplegable con los dispositivos y posteriormente al administrador de dispositivos, es decir, **No Device** → **Device Manager**:

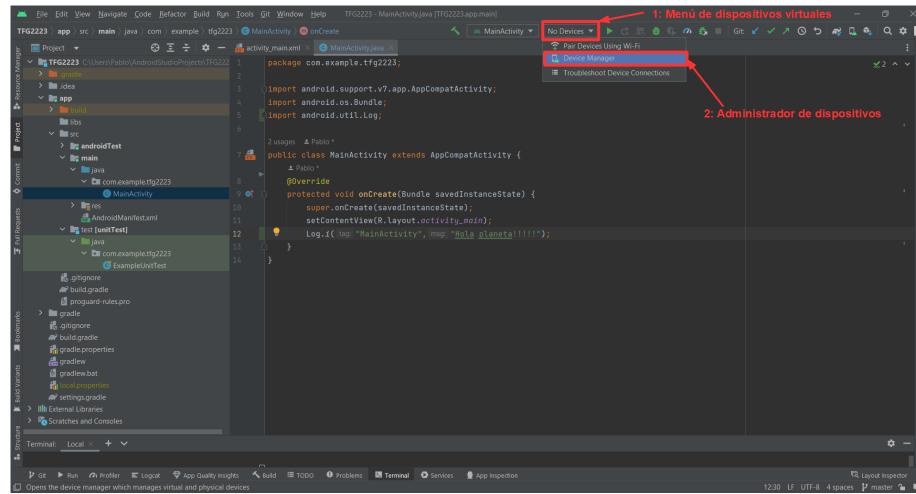


Figura D.40: Acceso al administrador de dispositivo mediante el menú desplegable de dispositivos

10. Posteriormente sale en la parte derecha de la pantalla el administrador de dispositivos (**Device Manager**), el cual muestra todos los dispositivos virtuales que se utilizan durante las simulaciones de funcionamiento. Para añadir un dispositivo virtual basta con presionar el botón **Create Device**:

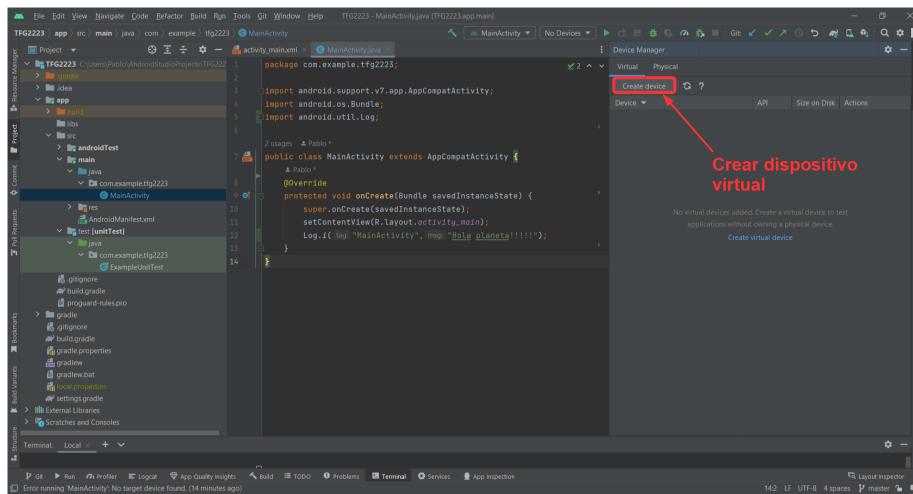


Figura D.41: Administrador de dispositivos

11. En cuanto a la configuración del nuevo dispositivo virtual, existen diferentes opciones para elegir, pudiendo simular aplicaciones para smartphones, tablets, dispositivos wearable, escritorio, smart TV y Android Auto. Como primer dispositivo se va a añadir el smartphone *Pixel 6 Pro* de Google. Posteriormente se presiona el botón **Next** para seguir con el siguiente paso:



Figura D.42: Selección del nuevo dispositivo virtual

12. Posteriormente se elige el sistema operativo de este dispositivo virtual. Para ello se va a seleccionar dicho sistema operativo para descargarlo para el dispositivo y posteriormente se presiona sobre **Next**:

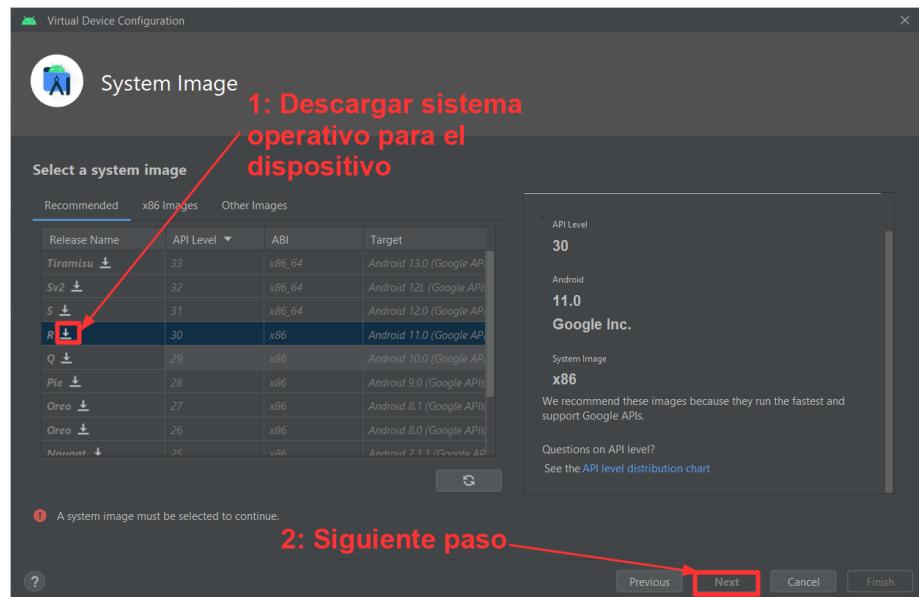


Figura D.43: Selección de sistema operativo para el nuevo dispositivo virtual

13. Tras haber seleccionado la configuración tanto de hardware como de software se procede con la instalación tanto del dispositivo como de su sistema operativo. Al finalizar se presiona el botón **Finish**:

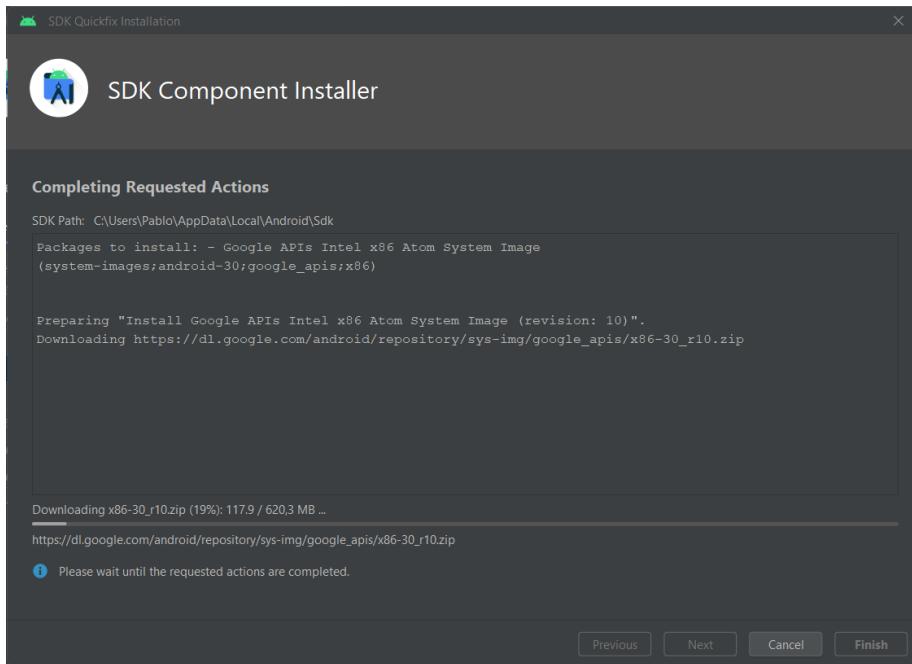


Figura D.44: Proceso de instalación del sistema operativo y de configuración del dispositivo virtual



Figura D.45: Finalización del proceso de instalación del sistema operativo y de configuración del dispositivo virtual

14. Para finalizar se puede comprobar la configuración que se ha establecido para el dispositivo nuevo, además de poder cambiar la orientación inicial del mismo. Cuando se quiera finalizar con la creación del dispositivo virtual, se presiona el botón **Finish**:



Figura D.46: Finalización del asistente de creación del dispositivo y comprobación de la configuración

Al tener la cuenta de GitHub agregada en Android Studio, se pueden realizar todas las operaciones de git desde la propia interfaz. Por lo tanto, los pasos a seguir para hacer un commit y publicarlo en GitHub son los siguientes:

1. En primer lugar hay que desplazarse a la barra de herramientas superior y ahí acceder a la ruta **Git** → **Commit**. Alternativamente se puede acceder a la configuración del nuevo commit mediante el comando *Ctrl+K*:

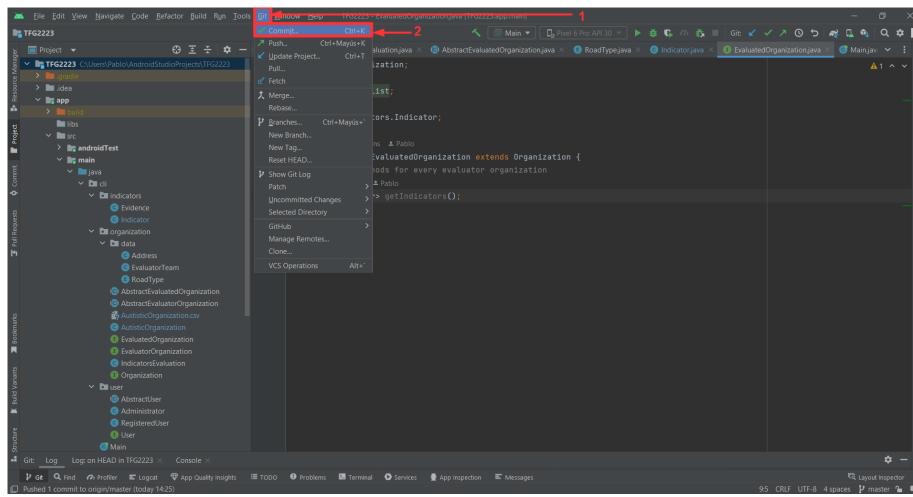


Figura D.47: Acceso a la configuración del nuevo commit

2. Posteriormente se seleccionan los ficheros que hayan tenido cambios, se escribe el mensaje de commit y se presiona en el botón ***Commit and Push:***

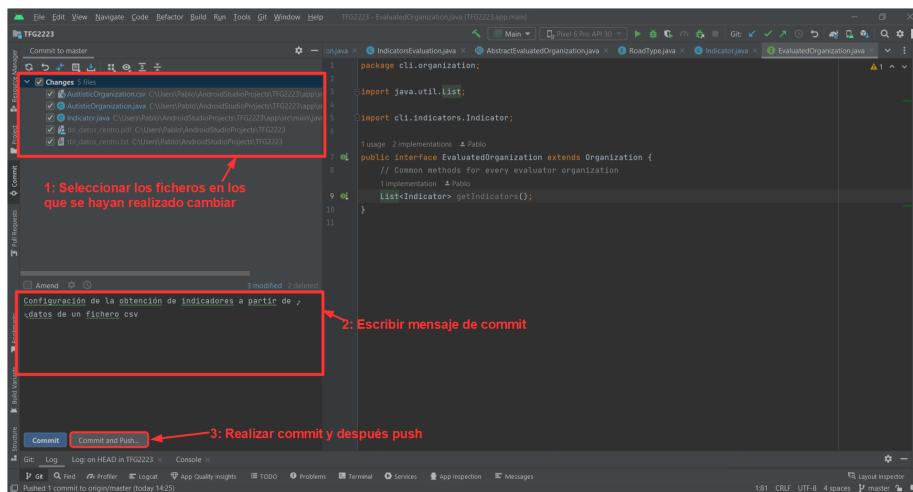


Figura D.48: Configuración del nuevo commit

Tras presionar ese botón analizará los posibles fallos y warnings que tenga el código de la aplicación. En la siguiente captura se han detectado cuatro warnings, dos en la clase **Indicator** y dos en la clase **AutisticOrganization**. Se puede optar por solucionar esos warnings y reiniciar la comprobación anteriormente mencionada, o por realizar las

operaciones de commit y push mediante el botón ***Commit Anyway and Push:***

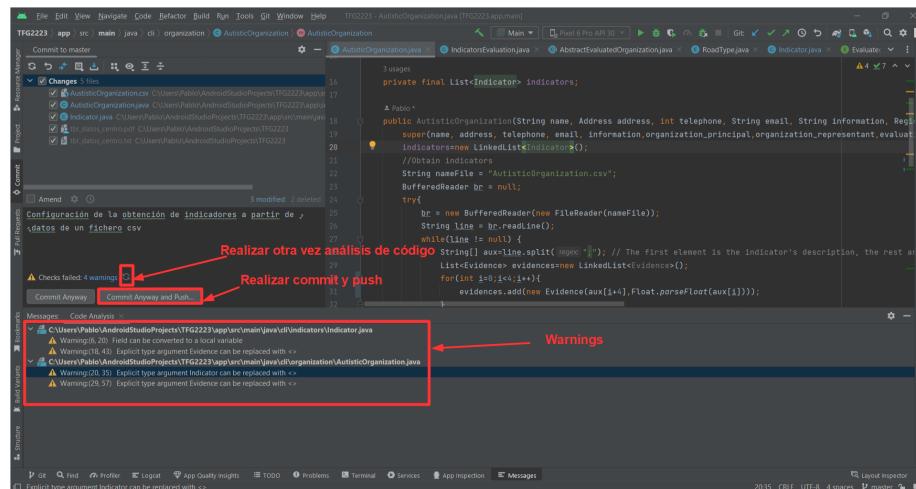


Figura D.49: Soluciones a warnings durante el proceso de commit

3. Tras haber solucionado todos los warnings, se presiona el botón ***Push*** para publicar los cambios realizados en GitHub:

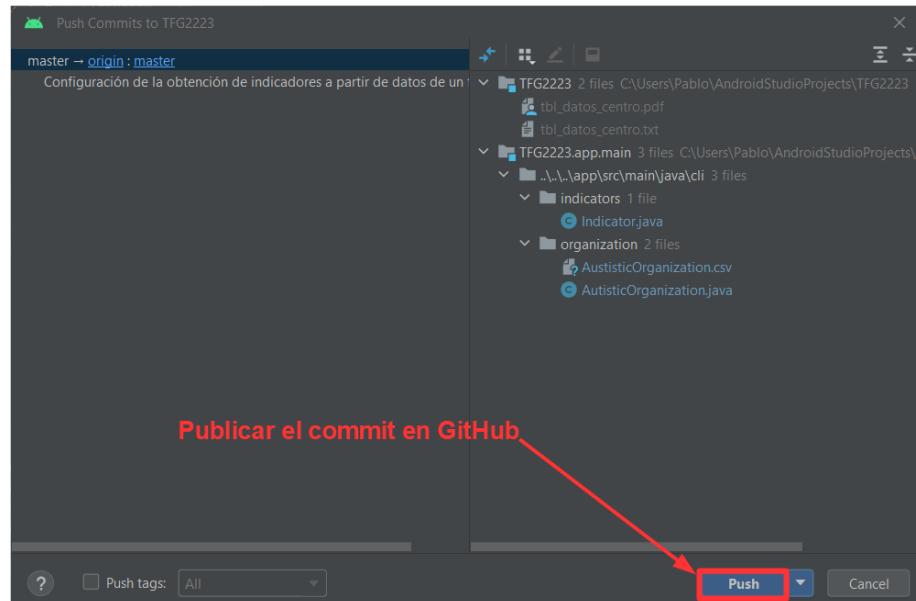


Figura D.50: Menú de publicación de cambios realizados

Al acabar con todo este proceso sale este mensaje:

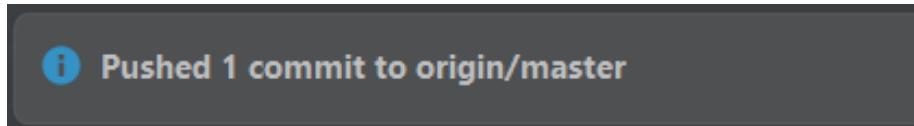


Figura D.51: Commit y push realizados satisfactoriamente

Azure Portal

Para crear un webservice con una base de datos en *Microsoft Azure*, se deben seguir los siguientes pasos:

1. En primer lugar se tiene que acceder al portal de *Azure* e iniciar sesión con la cuenta ya creada. Ya con la sesión iniciada, en la sección *Servicios de Azure*, se selecciona la opción ***App Services***, posteriormente en ***Crear*** y por último en ***Aplicación web + base de datos***, como se muestra en la siguiente captura:

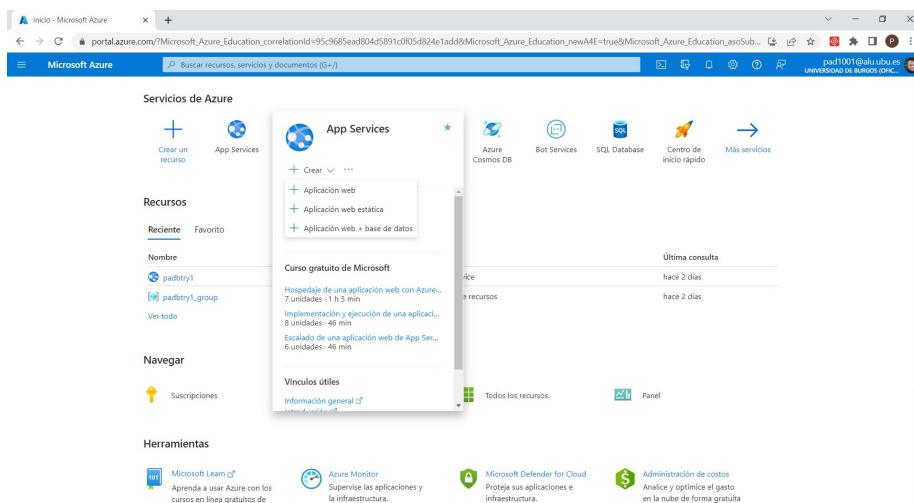


Figura D.52: Acceso a la creación de la web app desde el menú principal del portal de *Azure*

2. Posteriormente, se procede a la configuración de la aplicación base y de la base de datos. Como se puede comprobar en los ***Detalles del proyecto***, la ***suscripción elegida*** debe dejarse por defecto ya que

se trata de la suscripción asociada a la cuenta de *Azure*, el **grupo de recursos** debe ser nuevo y la **región** puede ser cualquiera de las disponibles. En cuanto a los **detalles de la aplicación web**, el **nombre** puede ser cualquiera que se encuentre disponible en ese momento y la **pila del entorno en tiempo de ejecución** debe ser *.NET 6* ya que el servidor está programado en C# utilizando *ASP.NET* (en caso de utilizarse Java, hay que seleccionar la versión en la que se compila el servicio web como *JAX-RS*).

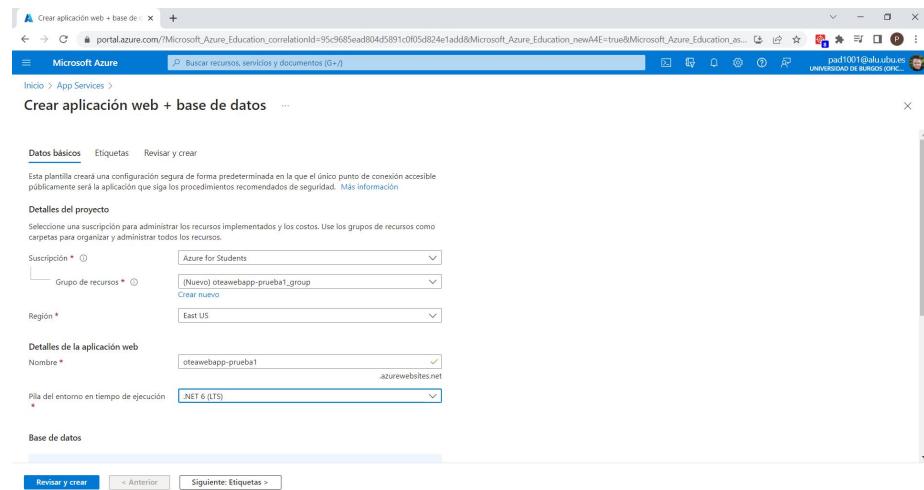


Figura D.53: Configuración de los detalles del proyecto y de la aplicación web del nuevo *App Service* con *Azure SQL*

En cuanto a la **base de datos**, el **motor** a elegir es *SQLAzure*, debido a su compatibilidad e integración con *Transact-SQL*, la adaptación de *Microsoft* del lenguaje SQL. El **nombre del servidor de la base de datos** y el **nombre de la base de datos** son personalizados bajo disponibilidad, recomendando utilizar los sufijos **server** y **database** respectivamente para poder relacionarlos con facilidad con la web app. Como no se va a agregar **Azure Cache for Redis** y el **plan de hospedaje** va a ser estándar, se presiona en *Revisar y crear* para comprobar antes de crear todo si la configuración es la deseada para lo que se le va a utilizar.

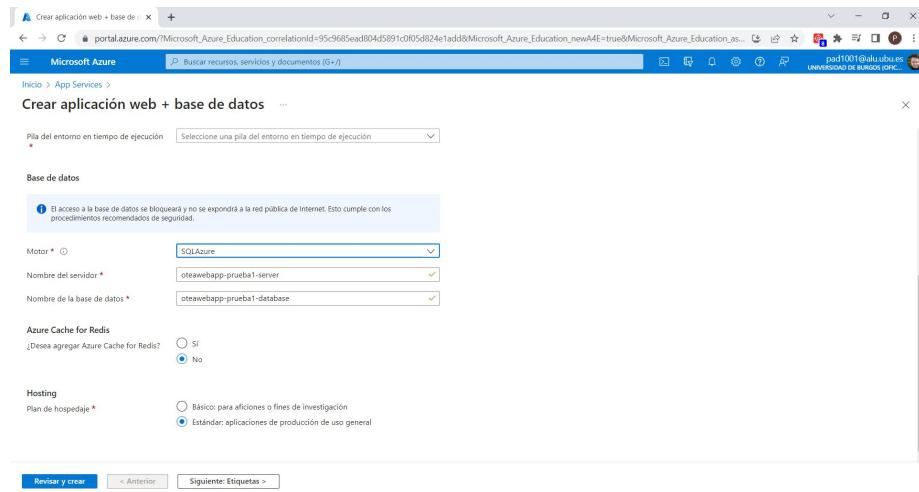


Figura D.54: Configuración de la base de datos del nuevo *App Service* con *Azure SQL*

3. Tras esperar a la validación de los cambios, se comprueba que todo esté configurado según las necesidades que se dispongan y posteriormente se presiona el botón *Crear*.

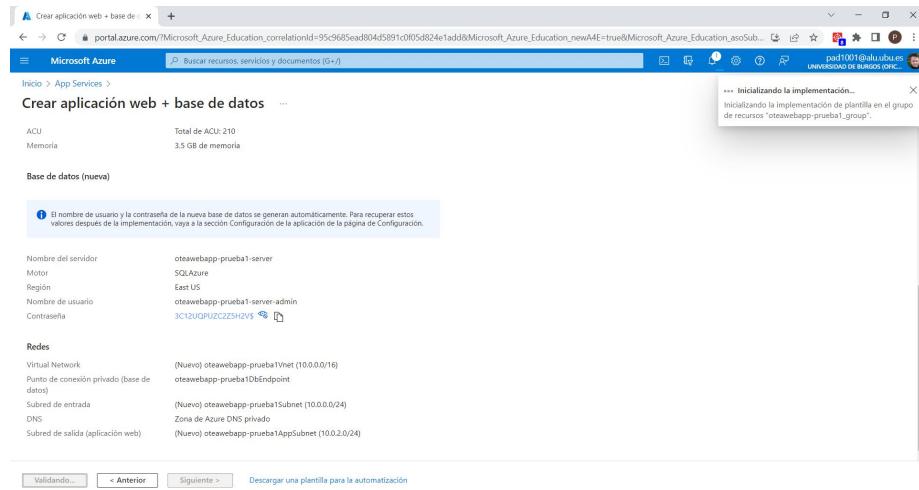


Figura D.55: La web app ya ha empezado a crearse

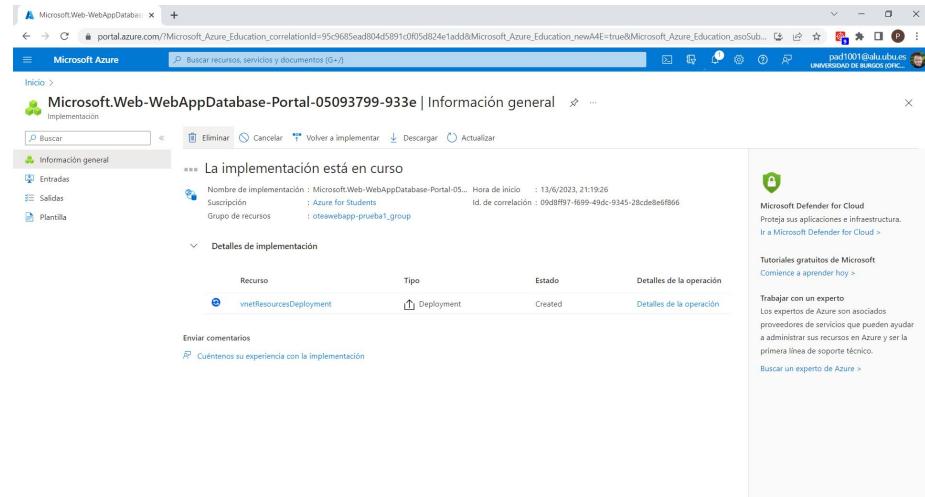


Figura D.56: Implementación en curso

Visual Studio 2022

Para poder instalar *Visual Studio 2022 Community*, hay que seguir los siguientes pasos:

1. En primer lugar se tiene que acceder a la página oficial de descarga de Microsoft y elegir la versión gratuita *Community 2022*. Cuando se ha elegido esa opción, se pasa a la descarga:

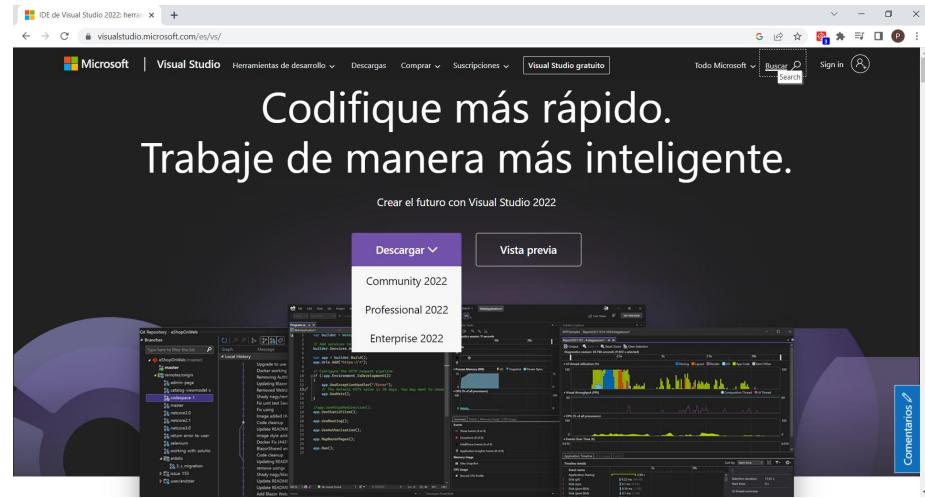


Figura D.57: Página oficial de descarga de *Visual Studio 2022*

2. Posteriormente se tiene que preparar el propio instalador, el cual bajará todas las características necesarias para el mismo

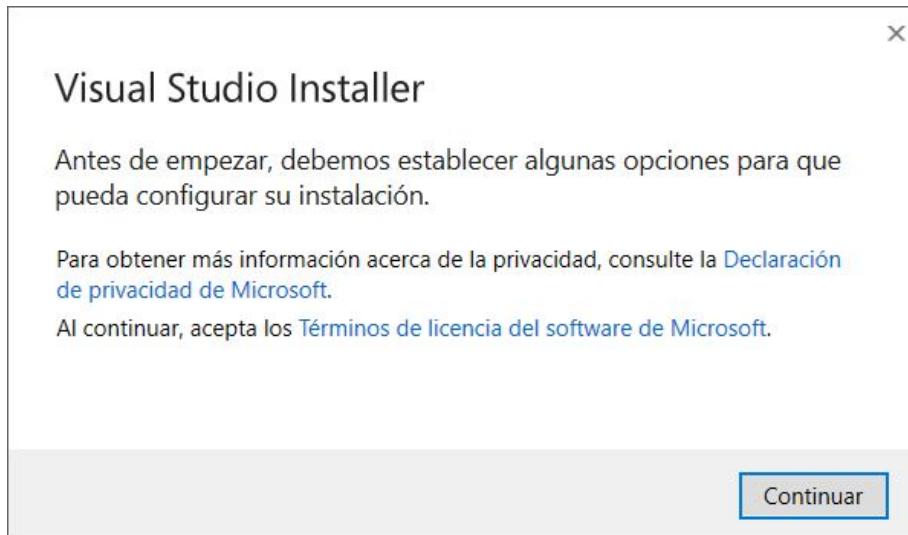


Figura D.58: Antes de empezar hay que preparar el instalador

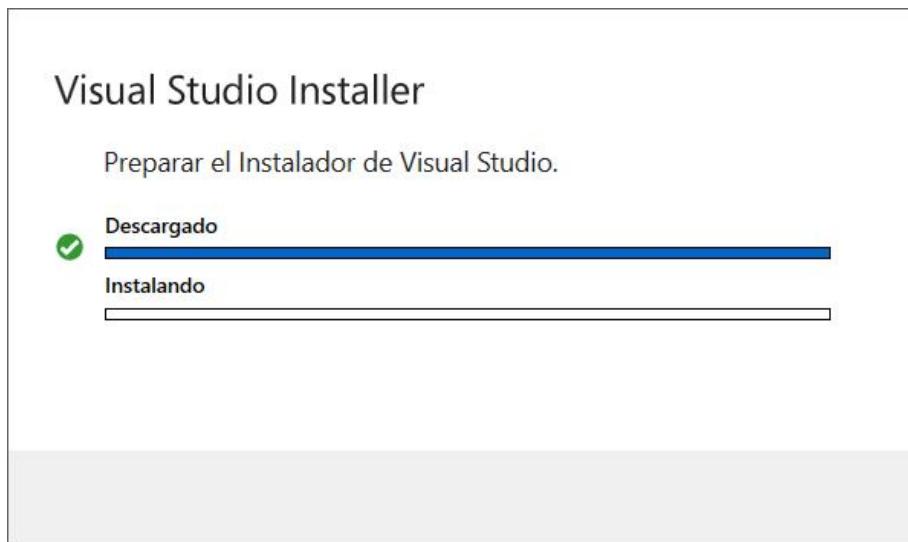


Figura D.59: Preparando el instalador...

3. Posteriormente se tienen que elegir las cargas de trabajo necesarias para nuestro caso. Aquí marcamos las opciones *Desarrollo de ASP.NET y web* y *Desarrollo de Azure*:

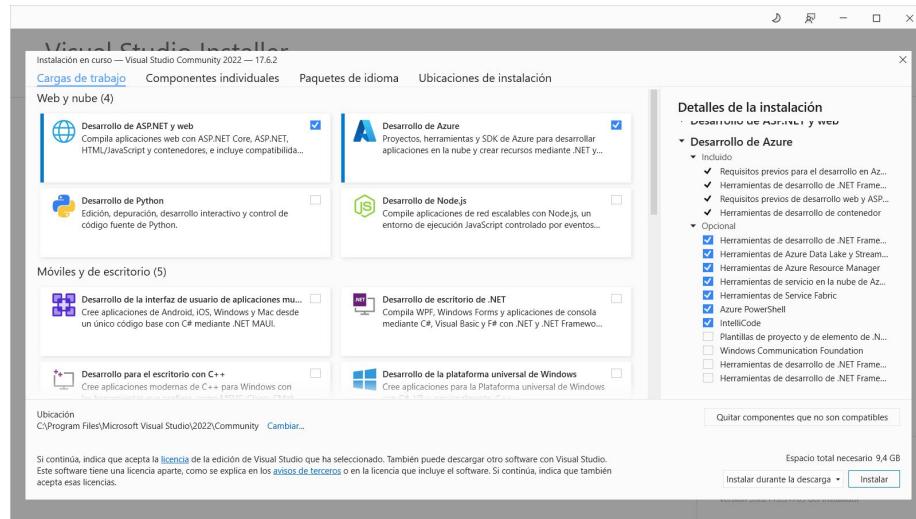


Figura D.60: Selección de las cargas de trabajo

4.

5. Posteriormente hay que esperar a que se instale todo:

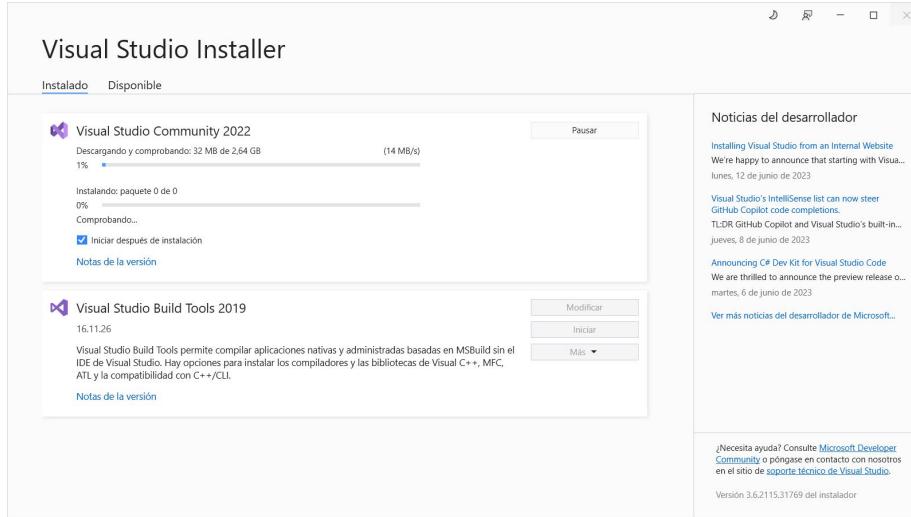


Figura D.61: Instalación en marcha

Cuando ya se ha terminado de configurar todo, se tiene que pasar a configurar el entorno correctamente:

1. En primer lugar se elegir una de las cuatro opciones existentes como tareas iniciales, en este caso se va a abrir un proyecto o una solución:

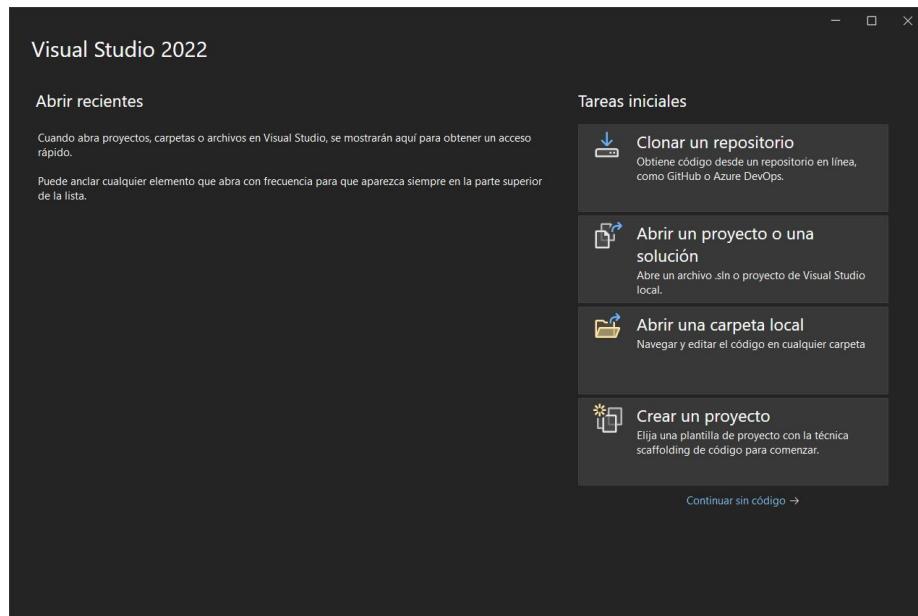


Figura D.62: Tareas iniciales

2. Posteriormente en el menú de selección de plantillas, tenemos que seleccionar *Aplicación web de ASP.NET Core*

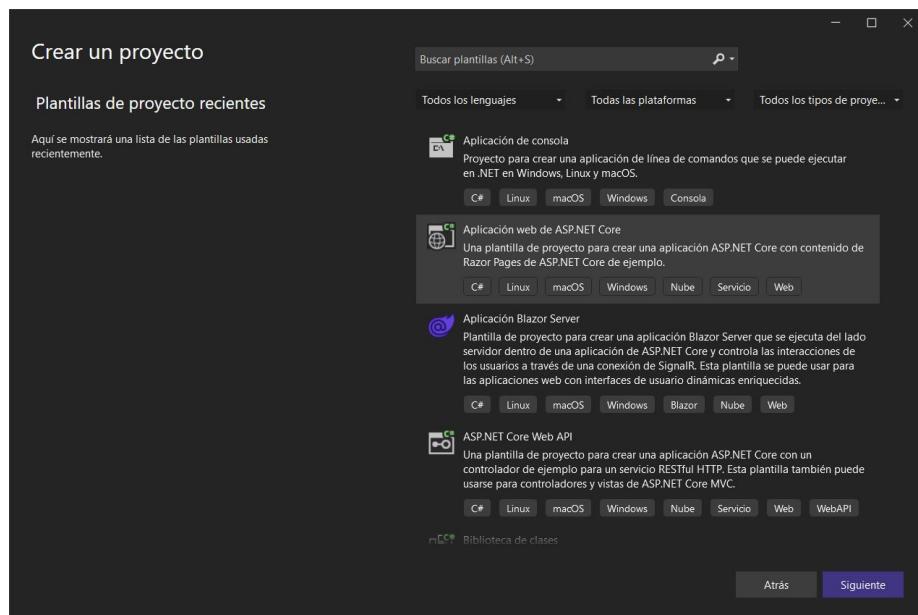


Figura D.63: Selección de plantilla

3. Más adelante se tiene que configurar el proyecto, eligiendo el nombre y el lugar en el que se va a guardar:

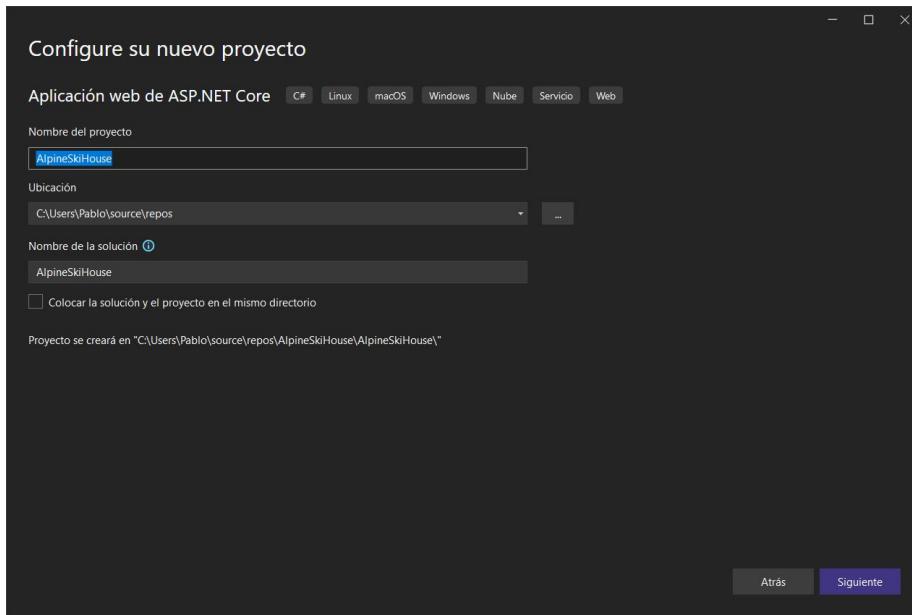


Figura D.64: Configuración de proyecto

4. Posteriormente se selecciona el framework .NET 6.0:

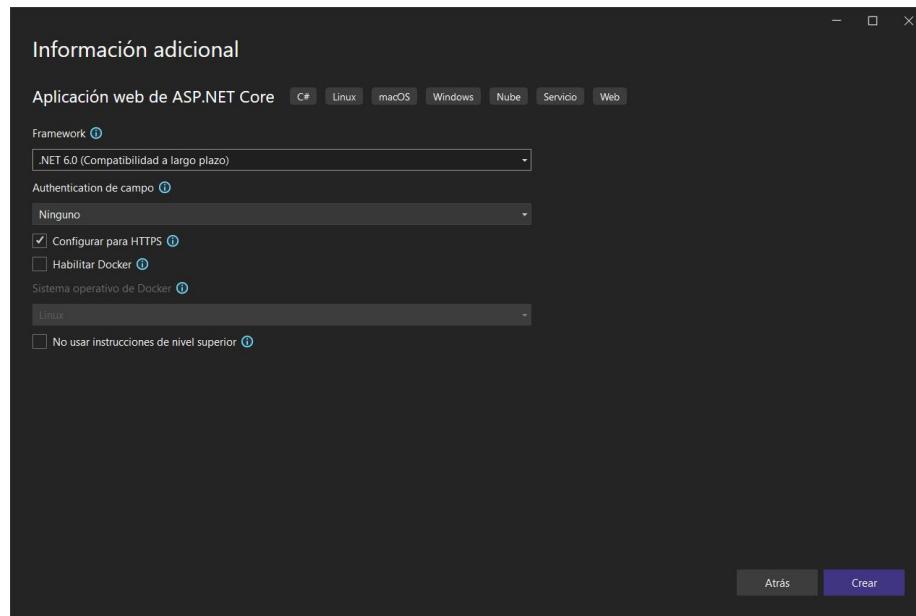


Figura D.65: Selección de framework

Por último, para actualizar los cambios en el servidor de Azure, se tienen que seguir los siguientes pasos:

1. En primer lugar hay que acceder al *Explorador de soluciones* para luego hacer click derecho sobre el nombre del proyecto. Más adelante le damos a *Publicar*

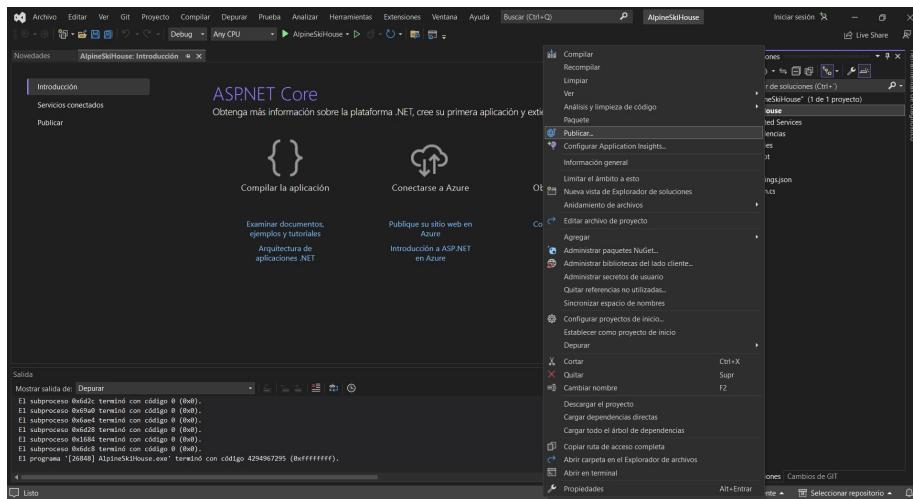


Figura D.66: Ir a publicar

- Como no se tiene ningún perfil de publicación, se va a proceder a hacer clic en *Agregar perfil de publicación*:

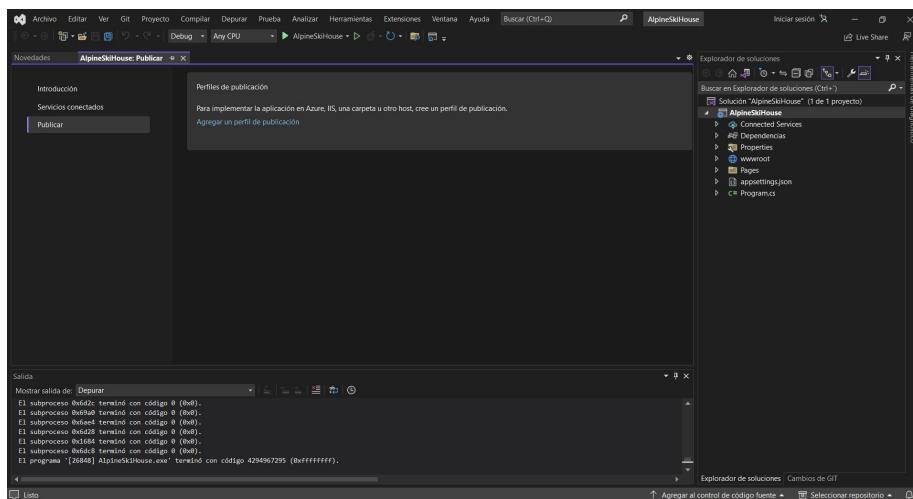


Figura D.67: Agregar perfil de publicación

- Posteriormente se selecciona Azure:

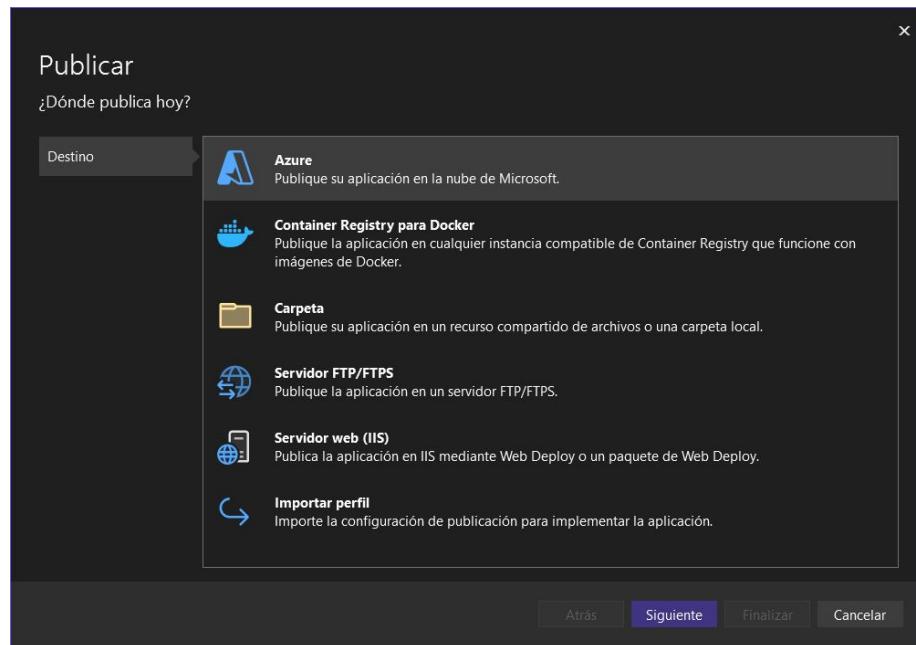


Figura D.68: Selección de publicación en Azure

4. Posteriormente se selecciona la web app de *Azure App Service*:
- 5.

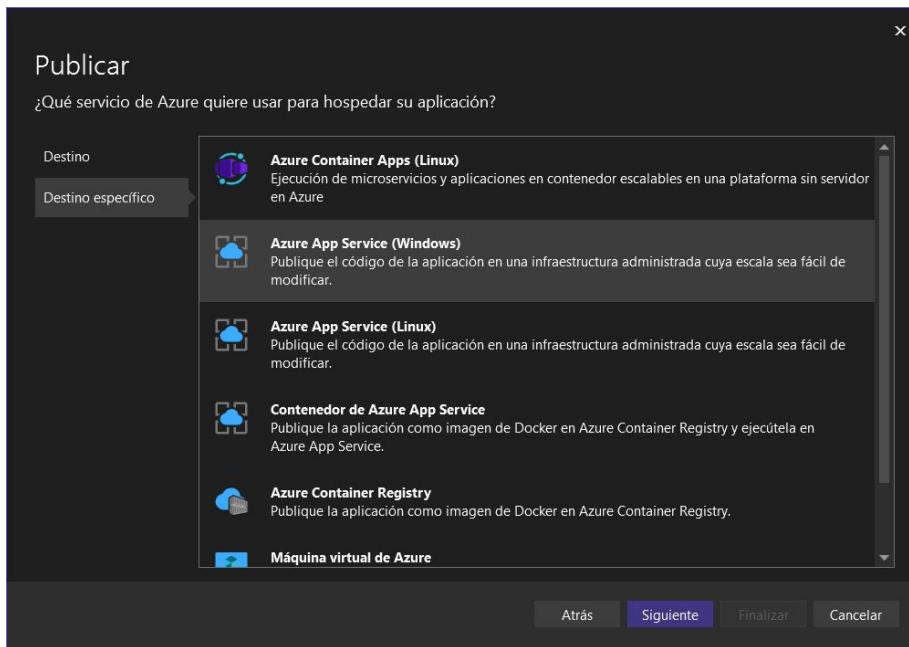


Figura D.69:

6. Tras haber iniciado sesión, se elige el servicio web donde se desea implementar la aplicación

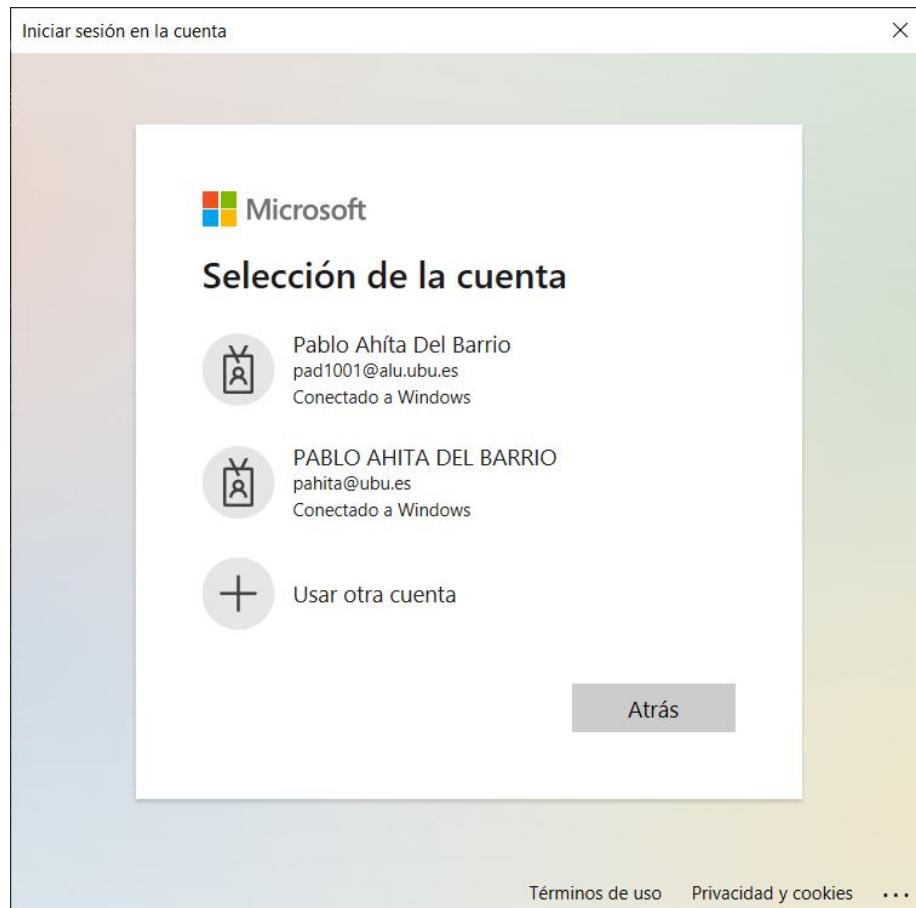


Figura D.70: Selección de web service

7. Cuando ya esté todo, ya se puede empezar a trabajar

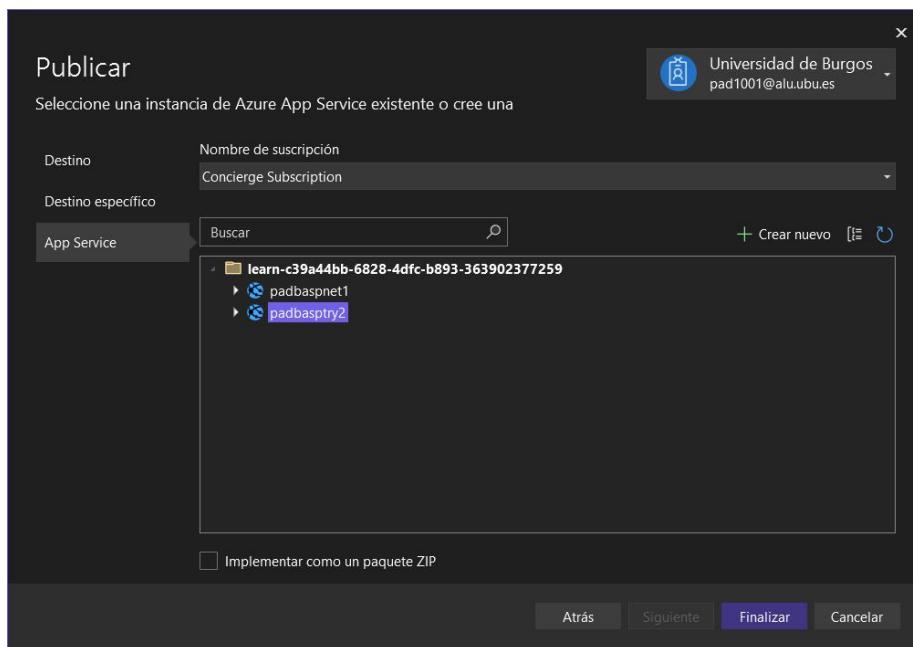


Figura D.71: Despliegue completado

Apéndice E

Documentación de usuario

E.1. Requisitos de usuarios

Para poder utilizar esta aplicación es necesario contar con cualquier móvil o tablet que sea mínimo del *API 33* de Android, ya que el dispositivo virtual en el que se ha probado tiene este sistema operativo. En cuanto a otros aspectos, no es una aplicación que requiera de otros recursos especiales puesto que es una aplicación bastante sencilla en cuanto a consumo de recursos.

E.2. Instalación

Para poder instalar la APK correctamente, hay que descargarla del repositorio de GitHub del proyecto final de grado, encontrándose en el directorio *app/build/outputs/apk/debug* del mismo, todo ello desde un dispositivo compatible. Antes de proceder a la instalación hay que activar los permisos de instalación necesarios para poder instalarlo, debido a que no se encuentra por ahora en *Google Play*. Cuando ya se han activado los permisos, se podrá instalar como ordena el dispositivo.

E.3. Manual del usuario

(Véanse vídeos de la carpeta de vídeos mencionada en el manual de programador)

Bibliografía

[1]

- [2] Google. Android studio, 2023. [Página de descarga del instalador de Android Studio].
- [3] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.
- [4] Wikipedia. Latex — wikipedia, la enciclopedia libre, 2015. [Internet; descargado 30-septiembre-2015].