PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

PROTOTIPO DE HERRAMIENTA DE APOYO PARA ACTIVIDADES DE ORDENAMIENTO DE TARJETAS

MICHAEL THOMAS JARA MOLINA

INFORME FINAL DEL PROYECTO
PARA OPTAR AL TÍTULO PROFESIONAL DE
INGENIERO DE EJECUCIÓN EN INFORMÁTICA

JULIO 2015

PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

PROTOTIPO DE HERRAMIENTA DE APOYO PARA ACTIVIDADES DE ORDENAMIENTO DE TARJETAS

MICHAEL THOMAS JARA MOLINA

PROFESOR GUÍA: CRISTIAN RUSU

INFORME FINAL DEL PROYECTO
PARA OPTAR AL TÍTULO PROFESIONAL DE
INGENIERO DE EJECUCIÓN EN INFORMÁTICA

JULIO 2015

Dedicado a todos los que estuvieron presentes durante este periodo compartiendo y formando experiencias.

En especial a mi familia y amigos.

Resumen

La organización de los contenidos que aparecen en las vistas de un sistema – el cómo, qué y/o dónde – es relevante para la experiencia de usuario y para la usabilidad del mismo. El Card Sorting es un método, fiable y barato, para encontrar patrones sobre la forma en que los usuarios esperan encontrar el contenido o funcionalidades dentro de un sistema.

Este informe documenta el análisis, diseño y construcción de un prototipo de herramienta web que permita realizar actividades de ordenamiento de tarjetas en pantallas táctiles de escritorio. El prototipo también permite visualizar e imprimir las pruebas de ordenamiento de tarjetas terminadas, y además, resume los datos en un archivo de hojas de cálculo.

Luego de la construcción y pruebas del sistema, se identifica un problema en la anidación de más de dos niveles y se entrega una propuesta de modificación a la notación para resumir los datos de la prueba.

Palabras Clave: Ordenamiento de tarjetas, usabilidad, herramienta web, pantalla táctil.

Abstract

The contents organization which appear in the system's view – the how to, what and/or where – is relevant for the users experience and the usability itself. The Card Sorting is a method, reliable and inexpensive, to find patterns about the way which the users waiting to find the content or functionality inside the system.

This report documents the analysis, design and construction of a prototype web tool for realize card sorting activities in a desktop touch screen. The prototype also allows visualize and print the completed test, and furthermore, summarizes data into spreadsheet.

After to building and testing of the system test, it is identificate a problem in more two nested levels and it is proposes an amendment in notation to summary data test.

Keywords: Card Sorting, usability, web tool, touch screen.

Índice

Índice de tablas

Tabla de imágenes

1 Introducción

El presente informe documenta el desarrollo de un prototipo de aplicación web para el apoyo a las actividades de ordenamiento de tarjetas (del inglés Card Sorting) que sea compatible con la infraestructura del laboratorio de usabilidad e implemente la metodología de resumen de jerarquía propuesta en un trabajo previo [1] realizado por un miembro del grupo de usabilidad UseCV de esta escuela.

El prototipo ayuda a crear, realizar y resumir una prueba de ordenamiento de tarjetas. El evaluador mantiene una serie de proyectos y para cada proyecto genera estudios. Cada estudio tiene una lista de tarjetas definidas por el evaluador que se ponen a disposición del usuario que realiza una prueba de ordenamiento en ese estudio. El usuario también puede crear tarjetas durante la prueba si lo estima necesario.

Las tarjetas pueden ser agrupadas con diferentes niveles de anidación. El nivel máximo de niveles es definido por evaluador y puede ir desde cero hasta infinito. Los grupos y tarjetas creadas por el usuario son identificados mediante el color de estos.

Los equipos donde será implementado el prototipo son computadores de escritorio que además de los periféricos de entrada mouse y teclado, poseen una pantalla táctil. Estas pantallas tienen un software propietario para controlar la respuesta a los eventos táctiles.

Los problemas de este software es que al ser propietario no se tienen derechos para modificarlo y no cuenta con una API para aprovechar sus beneficios. Para solucionar esto, el prototipo utiliza librerías de código abierto que cumplen la misma función, y permite al prototipo ser independiente de la maquina pudiendo ser ejecutado en cualquier dispositivo con pantalla táctil bajo ciertas restricciones.

La metodología propuesta en [1] consiste en resumir los datos sobre la jerarquía con que fueron ordenados los grupos de tarjetas en una “hoja de cálculo semi automatizada”.

Define una plantilla para las hojas de cálculo donde identifica cada tarjeta con un número único y una serie de columnas para representar por medio de referencias la jerarquía de los grupos y las tarjetas que lo componen. Para rellenar la plantilla los datos son recabados por el evaluador a través del análisis visual del estado final de la prueba.

En [1] luego de volcar los datos a la plantilla, se definen una serie de pasos para obtener una representación jerárquica promedio para las pruebas del estudio, sin embargo, aquel procedimiento esta fuera del alcance del actual proyecto.

El informe está estructurado con la definición de objetivos y presentación del marco referencial. Por medio de un estudio de factibilidad se analiza la viabilidad del proyecto. Luego se definen la metodología de desarrollo de software y la arquitectura utilizada en la construcción del prototipo para finalizar con la implementación de las funcionalidades. En las hojas finales se encuentran las conclusiones de este desarrollo y se deja una sugerencia para mejorar el prototipo y la síntesis de resultados.

2 Definición de los objetivos del proyecto

2.1 Objetivo General

Desarrollar un prototipo de herramienta para el apoyo de actividades de ordenamiento de tarjetas en dispositivos táctiles.

2.2 Objetivos Específicos

1. Proporcionar una plataforma para gestionar pruebas de ordenamiento de tarjetas.
2. Exportar los resultados de las pruebas de ordenamiento de tarjetas en un archivo de Hojas de Cálculo.
3. Validar el prototipo de herramienta, el método y la plantilla que resume los datos de la pruebas de ordenamiento de tarjetas.
4. Ofrecer una base para futuros proyectos mediante la reutilización de código, interoperabilidad y escalabilidad.

3 Marco referencial

3.1 Interacción persona-computador

La Interacción Persona-Computador (del inglés *Human Computer Interaction* - HCI) es una disciplina que estudia el diseño, implementación y evaluación de sistemas informáticos interactivos, conjuntamente con los fenómenos más relevantes que los rodean. [2].

El objetivo de HCI está en estudiar el universo de formas en que uno o muchos humanos, a los que llamaremos usuarios, interactúan con software mediante dispositivos de interacción convencional y/o no convencional, en ambientes provechosos o perjudiciales para su uso.

Dado que HCI estudia la comunicación entre un ser humano y una máquina, se basa y fundamenta en conocimientos sobre ambas partes. Por el lado de la máquina, las técnicas de gráficos por ordenador, sistemas operativos, lenguajes de programación y entornos de desarrollo son relevantes. En el lado humano, son relevantes la teoría de la comunicación, las disciplinas de diseño gráfico e industrial, la lingüística, las ciencias sociales, la psicología cognitiva y el rendimiento humano. Y, por supuesto, también son importantes los métodos de ingeniería y diseño [3].

3.2 Ingeniería de la usabilidad

La Usabilidad según la norma ISO 9241 es la medida en la que un producto se puede usar por determinados usuarios para conseguir objetivos específicos con efectividad, eficiencia y satisfacción en un contexto de uso especificado [4].

La Ingeniería de la usabilidad, atiende los aspectos de la usabilidad de manera ingenieril. Metodológicamente se estudia el grado de efectividad, eficiencia y satisfacción con que los usuarios utilizan un producto en condiciones específicas, y propone alternativas o mejoras para elevar el grado de usabilidad de dicho producto.

La ingeniería de la usabilidad entrega métodos y herramientas para la evaluación de la usabilidad de productos. Para medir el grado de usabilidad de un producto están establecidos cinco atributos de usabilidad: facilidad de aprendizaje, eficiencia, recuerdo en el tiempo, tasa de errores, satisfacción subjetiva [5].

3.3 Ordenamiento de Tarjetas

Del inglés Cardsorting, el Ordenamiento de Tarjetas es un método para descubrir la estructura latente en una lista desordenada de enunciados o ideas [6]. Es utilizado para ayudar en el diseño y/o evaluar la arquitectura de información de un sitio [7].

El método consiste en realizar pruebas donde él o los participantes deben organizar tarjetas con conceptos en grupos y categorías jerarquizadas. Los resultados de cada prueba son combinados y analizados con diferentes técnicas para revelar patrones.

La técnica de ordenamiento de tarjetas puede realizarse de manera abierta o cerrada. Se denominan abiertas cuando los usuarios pueden modificar el juego de tarjetas con el que se trabaja, por ejemplo cambiándole el rótulo a un grupo de tarjetas. En cambio, son cerradas cuando las tarjetas, sus rótulos y categorías no pueden alterarse por los usuarios.

Utilizando esta técnica es posible reducir significativamente los problemas de usabilidad relacionados con una incorrecta arquitectura de información. La participación de usuarios reales es fundamental y es el principio que asegura un mejor resultado final.

3.4 Aplicaciones web

Una aplicación web es cualquier aplicación que es accedida por una red como internet o una intranet. En general, el término también se utiliza para designar aquellos programas informáticos que son ejecutados en el entorno del navegador web.

Las ventajas de este tipo de aplicaciones son múltiples entre las que destacan la facilidad distribución y de mantenimiento, las aplicaciones son escalables y entrega una alta interoperabilidad.

Los diferentes módulos que componen un sistema se agrupan, según su función, en *capas lógicas*. Esta separación permite abstraer la lógica interna del módulo interactuando con las demás capas solo por medio de la API que estas ponen a disposición [8].

3.5 Tecnología de Pantalla Táctil.

Una pantalla táctil es un periférico de entrada y salida. Recibe señales de entrada mediante el toque directo sobre su superficie y muestra datos de salida del dispositivo [9].

Las pantallas táctiles pueden implementar diferentes tecnologías pero las más utilizadas son las: Resistivas y Capacitivas.

3.5.1 Pantallas con tecnología Resistiva.

Una pantalla de este tipo posee dos capas separadas entre sí por pequeños espaciadores. La capa más externa es flexible, mientras que la interna es rígida. Ambas capas tienen un revestimiento conductivo por uno de sus lados. Cada lado conductivo está a un voltaje diferente, se posicionan de frente dejando en el medio de ambos los espaciadores [9].

Cuando uno toca la pantalla con suficiente fuerza, la capa flexible exterior se hunde hasta hacer contacto con la interna. Como los lados que entran en contacto están a voltajes diferentes, se establece una corriente entre ambas capas, lo cual se interpreta como una instrucción.

Estas pantallas son mucho más durables porque son más gruesas, además de que son más baratas, no se ven afectadas por el polvo y suelen ser mucho más precisas. Sin embargo, tienen menos brillo y reflejan mucho la luz solar.

3.5.2 Pantallas con tecnología Capacitiva.

Están conformadas por un aislante, como por ejemplo vidrio, cubierto por un conductor transparente, generalmente óxido de indio dopado con estaño (o también conocido como ITO, del inglés *Indium Tin Oxide)* [10] que se mantiene constantemente a un voltaje determinado [9].

Cuando un usuario toca este tipo de pantallas, la electricidad presente en el cuerpo humano distorsiona el campo electrostático del recubrimiento de ITO. Esta distorsión se puede medir como un cambio de capacitancia (cantidad de energía eléctrica). La posición del cambio es enviada al controlador quien la traduce en funciones para la máquina.

Estas pantallas son mejores en cuanto a calidad de imagen y respuesta multitáctil, pero son más caras y necesitan punteros especiales.

4 Estudio de Factibilidad.

El estudio de factibilidad se realiza para evaluar la viabilidad de proyecto y se divide en 4 partes que analizan distintos ámbitos: técnico, económica, legal y operativo. En este análisis pueden aparecer restricciones de cualquier ámbito que podrían transformarse en requerimientos no funcionales del prototipo de sistema.

4.1 Factibilidad técnica.

La factibilidad técnica consiste en establecer la necesidad, disponibilidad y capacidad de hardware y software para el desarrollo del prototipo.

4.1.1 Hardware.

La implementación del sistema requiere del siguiente hardware con acceso a internet:

- Un dispositivo táctil que ejecute un navegador web con motor JavaScript para realizar la prueba de ordenamiento de tarjetas. No es necesario que soporte multitáctil.
- Un dispositivo que pueda ejecutar un navegador web para acceder al sistema y capaz de descargar, almacenar y visualizar la hoja de cálculo.
- Las dimensiones mínimas de la pantalla de los dispositivos será 7,5 pulgadas, con resolución de 1024 x 720 píxeles.

Para asegurar la fluidez y precisión de la interacción del usuario con el sistema se recomienda utilizar dispositivos con RAM de al menos 2 GB. Como requerimiento mínimo el dispositivo debe contar con una unidad de procesamiento de doble núcleo y de frecuencia 1.7 GHz.

En el laboratorio de usabilidad de la Escuela de Ingeniería en Informática de la Pontifica Universidad de Valparaíso dispone de equipos que cumplen con los requerimientos mínimos para la ejecución del prototipo de sistema. En total son 4 equipos marca Dell que conforman el hardware donde se validará e implementará el sistema, sus características son:

- 2 Equipos modelo Vostro 360, procesador de dos núcleos Intel Core i5 [11], 4 GB de RAM y 250 GB de capacidad de almacenamiento en disco duro. Posee pantalla táctil de 23 pulgadas full HD.
- 2 Equipos modelo Optiplex 390 con pantalla LCD de 21 pulgadas, procesador de dos núcleos Intel Core i3 [12], 4 GB de RAM y 160 GB de capacidad de almacenamiento en disco duro.

4.1.2 Software.

En algunos casos los dispositivos táctiles tienen paquetes de desarrollo específicos y/o de pago, pero los ordenadores donde se probará y validará el prototipo desarrollado no cuentan con estos paquetes. Los gestos táctiles son gestionados por un software específico del

ordenador, esté software presenta problemas de compatibilidad con la aplicación además de hacerla dependiente de este hardware por lo que es desactivado.

Para asegurar la interoperabilidad y escalabilidad del sistema se desarrolla el prototipo como una aplicación web. De esta manera se tiene un amplio catálogo de tecnologías y bibliotecas disponibles para resolver los problemas del desarrollo.

El sistema utiliza una biblioteca que mapea los gestos táctiles de la pantalla a eventos del mouse [13] lo que permite al prototipo de herramienta ser compatible con casi cualquier dispositivo con pantalla táctil.

Dado que existe la disponibilidad de hardware y software necesario el sistema es técnicamente factible de implementar.

4.2 Factibilidad económica

Desde el punto de vista presupuestario la elección de las herramientas estuvo orientada hacia las de distribución libre y bajo licencia GNU GPL (General Public License) que garantiza la libertad de usar, compartir y modificar el software [14]. Estas herramientas son ampliamente utilizadas en diversos proyectos y validadas a diario por la comunidad de usuarios por lo que se mantienen muy estables.

La tabla 4.1 que se muestra a continuación, contiene los insumos, su disponibilidad y us valores.

Insumo	Disponible	Valor
Servidor web	Apache, Nginx, IIS, Cherokee, Tomcat, Glassfish.	Gratis
Sistema de gestión de base de datos (SGDB)	MySQL, PostgreSQL	Gratis
IDE	Netbeans, Eclipse, PhpDesigner, Komodo, Aptana.	Gratis
Software compatible con Hojas de Calculo	LibreOffice, OpenOffice, GoogleDrive, Microsoft Office	Gratis (Excepto Microsoft Office)

Tabla 4.1 Valor de los insumos necesarios para el desarrollo.

Por esto no existe costo alguno generado por las herramientas software (IDE, bibliotecas, sistema operativo, software en general) ni por el pago de derechos de autor o derechos de uso de los algoritmos utilizados. Para aumentar la compatibilidad del archivo se opta por el formato y extensión de Microsoft Excel 2003 XLS que puede ser interpretado por diferentes programas sin la necesidad de utilizar el programa Microsoft Excel de la suite Microsoft Office.

Para la programación, diseño de diagramas, redacción de informe, implementación y pruebas de los prototipos se hará uso de los equipos disponibles en las instalaciones del laboratorio de computación de la escuela de ingeniería informática, este computador de uso libre cumple con las características mínimas definidas en la factibilidad técnica.

Los usuarios para tomar las pruebas serán usuarios voluntarios de la escuela o estudiantes de la PUCV lo que no conlleva un gasto de dinero.

Según lo anterior descrito, el desarrollo del prototipo de herramienta es económicamente viable.

4.3 Factibilidad operativa

La factibilidad de operación del sistema aumenta porque funciona mediante interacción táctil y/o interacción mediante el ratón. Así, para completar una tarea, un usuario puede realizar algunas acciones con el ratón y otras con el dedo en cualquier momento, sin tener que configurar algo. A demás, esto permite que el sistema sea utilizado en diversos dispositivos sin tener que depender de la pantalla táctil o del ratón respectivamente, ya que con solo un método de entrada se puede realizar todas las acciones en las vistas.

El usuario no debería tardar en acostumbrarse a la interacción táctil simple (con un punto de presión) porque es la más cercana a la interacción natural (tocar, tomar, accionar) además de que en la actualidad este tipo de interacción ya está presente muchos objetos electrónicos (celulares inteligentes, tabletas, computadores, etc.).

Antes de cada prueba, se recomienda realizar una capacitación previa para asegurar que el usuario sea capaz de utilizar el sistema para que durante la prueba de ordenamiento de tarjetas la atención del usuario solo se centre en la problemática presentada y no perjudique su desempeño.

Los evaluadores deben ser capacitados para utilizar el sistema (tal como con cualquier sistema) que al facilitar las tarea de creación de una prueba y la toma de esta a diferentes usuarios no debería resistirse a la implementación dado los beneficios que otorga para realizar su cometido.

Según el análisis el sistema puede estar operativo luego de su implementación.

4.4 Factibilidad legal

Para asegurar que el funcionamiento del sistema y todo el proceso de desarrollo se encuentra dentro del margen legal establecido en chile se realizó un estudio de factibilidad.

El primer aspecto que puede generar conflicto son las herramientas utilizadas en el desarrollo pero todas estas son de distribución gratuita bajo licencia GPL [14]. Al distribuirse bajo esta licencia toda copia descargada de internet es totalmente legal y no infringe la ley Nº 17.336 sobre Propiedad Intelectual.

La ley Nº 19.039 sobre propiedad industrial y la ley Nº 17.336 sobre Propiedad Intelectual no se infringen dado que los algoritmos utilizados en el diseño de la solución son de propiedad de su(s) creador(es). La utilización de estas soluciones no está restringida para uso académico ni comercial y en ningún momento el desarrollador pretende adjudicarse la autoría de algún algoritmo si se usa íntegramente, cualquier modificación a estos algoritmos se expresará de forma clara y siempre citando la fuente y archivo de origen.

La ley sobre delitos informáticos no tiene relevancia para este análisis ya que penaliza la infiltración o adulteración de software ajeno. Terminado este análisis se comprueba que el proyecto no infringe ninguna ley y es factible legalmente.

5 Modelo de procesos

Un proceso es un conjunto de actividades que conducen a la creación de un producto. Los procesos fundamentales en el ciclo de vida de cualquier producto son: especificación, diseño, implementación, verificación y evolución.

Un modelo de proceso es una descripción simplificada de un proceso que representa una visión de ese proceso desde actividades a realizar hasta roles involucrados. En ingeniería de software los modelos de procesos son utilizados para definir marcos de trabajo del proceso que pueden ser extendidos y adaptados para crear procesos más específicos de ingeniería de software [15].

5.1 Modelo de procesos de desarrollo evolutivo

Este enfoque entrelaza las actividades de especificación, desarrollo y validación. Un sistema inicial se desarrolla rápidamente a partir de especificaciones abstractas que se refina basándose en las peticiones del cliente para producir un sistema que satisfaga sus necesidades.

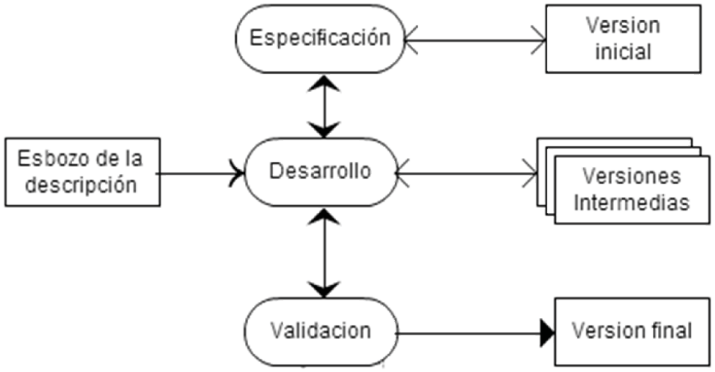


Imagen 5.1 Modelo evolutivo

Se utiliza este modelo cuando no se posee una descripción fija y completa de los requerimientos del sistema. Este es un modelo flexible a los cambios de requerimientos, la especificación del software se desarrolla de forma creciente y junto con el producto. Si los sistemas se desarrollan rápidamente, no es rentable producir documentos que reflejen cada versión del sistema (ver imagen 5.1).

A menudo, los modelos de proceso se mezclan para adaptarse a las necesidades del desarrollo del software dando lugar a nuevos modelos. En [15] se identifican dos tipos de desarrollo evolutivo:

i. **Desarrollo Exploratorio:** El objetivo del proceso es trabajar con el cliente para explorar sus requerimientos y entregar un sistema final. El desarrollo empieza con las partes del sistema que se comprenden mejor y evoluciona agregando nuevos atributos propuestos por el cliente.
ii. **Prototipos Desechables:** El Proceso de desarrollo evolutivo es comprender los requerimientos del cliente y entonces desarrollar una definición mejorada de los

requerimientos para el sistema. El prototipo se centra en experimentar con los requerimientos del cliente que no se comprenden del todo.

Si bien al comienzo del desarrollo del proyecto la funcionalidad principal estaba clara los demás requerimientos del sistema estaban vagamente definidos razón por la que se opta por utilizar un modelo de procesos evolutivo de *"desarrollo exploratorio basado en prototipos escalables",* un método que mezcla ambos tipos de desarrollo evolutivo antes descritos. Cada prototipo es desarrollado con un enfoque exploratorio y evoluciona conjuntamente con la especificación del sistema.

Dado el acotado plazo para completar el proyecto – análisis, diseño, construcción y validación en alrededor de 4 meses – se adopta parte de la filosofía de las metodologías agiles que promueven una reducción de documentación para mantener el enfoque en lo entregable. Para el diseño y análisis se utilizaron Casos de Uso junto con una descripción literal de los actores, comportamiento y alcance del sistema; y el Modelo Relacional de la base de datos, modelo que fue refinado durante la etapa de construcción.

Las metodologías agiles permiten a los equipos de desarrollo centrase en el software mismo en vez de en su diseño y documentación. En [15] se explica que "si los sistemas se desarrollan rápidamente, no es rentable producir documentos que reflejen cada versión del sistema" además de que "estos métodos (ágiles) son los más idóneos para el desarrollo de sistemas de tamaño medio y para el desarrollo de productos para ordenadores personales. No son adecuados para el desarrollo de sistemas a gran escala y donde puedan haber complejas interacción con otros sistemas software o hardware".

La Programación Extrema (del inglés eXtreme Programming o XP) es posiblemente el método ágil de desarrollo de software más conocido y ampliamente utilizado. Se adoptan algunas de sus buenas prácticas las que se resumen en la tabla 5.1 (en [15] se encuentra la lista completa de buenas prácticas de XP).

Principio	**Descripción**
Diseño sencillo	Sólo se lleva a cabo el diseño necesario para cumplir los requerimientos actuales.
Entregas pequeñas	El mínimo conjunto útil de funcionalidad que proporcione valor de negocio se desarrolla primero. Las entregas del sistema son frecuentes y añaden funcionalidad a la primera entrega.
Refactorización	Se refactoriza el código tan pronto como encuentren posibles mejoras, esto mantiene el código sencillo y mantenible.
Integración continua	En cuanto se termina una tarea (unidad básica de una funcionalidad) se integra al código y se hacen pruebas de unidad.

Tabla 5.1 Prácticas utilizadas de la programación extrema

6 Paradigma de programación estructurado.

La programación estructurada es un método disciplinado de escribir programas que sean claros, que se demuestre que sean correctos y fáciles de modificar, combina esquemas sencillos para construir sistemas amplios y complejos pero de fácil entendimiento. Al utilizar la programación estructurada, es mucho más sencillo entender la codificación del programa.

Para la solución de un problema en particular, se inicia considerando las funciones que tiene que cumplir el programa en general y después se va desmembrando estas funciones en subfunciones más pequeñas (es decir, sigue un enfoque Top-Down para abarcar el problema general y modulara al máximo la implementación de las funciones).

Si se ha utilizado adecuadamente la programación estructurada, la integración de módulos para extender la aplicación debe ser relativamente sencillo y de presentar algún problema, será rápidamente detectable para su corrección [16].

La elección del paradigma de programación estuvo supeditada por las tecnologías utilizadas en la implementación de los requerimientos principales. El sistema no es desarrolla bajo ningún Framework por lo que no está atado a las restricciones impuestas por este; el lenguaje PHP (encargado de generar el contenido de las vistas) no cuenta con una orientación a objetos estandarizada[1]y; JavaScript junto con su biblioteca jQuery (encargado de gestionar la interacción de usuario) son lenguajes funcionales e interpretados.

Si bien, el diseño y programación de algunas bibliotecas pudo haber sido con otro paradigma, el sistema se sirve de estas por medio de sus API lo que no genera conflicto entre el paradigma del sistema y las bibliotecas.

[1] Durante el periodo de desarrollo, año 2013.

7 Arquitectura del sistema.

El sistema funciona como una aplicación web. La arquitectura se diseñó como un modelo de 3 capas: Capa de presentación, capa de negocios y capa de datos (ver imagen 7.1). En cada una de ellas están implicadas distintas tecnologías, lenguajes y librerías. Cada capa realiza una tarea específica de manera independiente y transparente para las demás capas.

Imagen 7.1: Arquitectura de 3 capas, tecnologías y elementos en cada una.

7.1 Capa de Presentación

Corresponde a la visualización del contenido y captura de datos de parte del usuario, en este nivel se encuentra el navegador web. Para esta capa se analizaron las características que tiene la nueva especificación de HTML en su versión 5 junto con CSS3 y JavaScript que en conjunto forman la base para cualquier aplicación web. A continuación un resumen de lo utilizado en el desarrollo en la capa de presentación.

7.1.1.1 HTML– HyperText Markup Language

HTML es un leguaje publicación estándar utilizado por la W3C. Funciona mediante el marcado de un conjunto de elementos mediante tags. La especificación de HTML se puede encontrar en [17]

HTML5 corresponde a la recomendación de nueva versión de la especificación del lenguaje (a la fecha de publicación aún está en revisión [18]). Implementa nuevos tags para audio y video e implementa otras funciones muy potentes como por ejemplo geolocalizacion. HTML5 como estándar implementa tags para definir la estructura y el contenido del elemento (*section, header, article, aside, footer,...)* que construyen la página.

7.1.1.2 CSS - Cascading Style Sheets

CSS permite a adjuntar estilos (por ejemplo tamaño y color) para documentos estructurados como HTML. Separa el estilo de presentación del documento del contenido y

organización de este. Actualmente CSS está en la versión 3, la cual se está desarrollando por módulos a partir de la especificación 2.2 [19].

CSS3 es mucho más potente y versátil. Agrega nuevos selectores que simplifica la aplicación de características a ciertos elementos dentro de un conjunto, selectores como *nth-child(n)* que permite elegir un nodo n dentro de un conjunto o *[atributo$=cadena]* que me permite selecciona los atributos cuyo valor termine con el texto '*cadena',* además, entrega desde opciones de sombreado y redondeado, hasta funciones avanzadas de movimiento y transformación.

7.1.1.3 JS - JavaScript

Lenguaje de scripting lado del navegador, se define como basado en prototipos, imperativo, débilmente tipado y dinámico [20]. Cada navegador utiliza un motor de JavaScript propio por lo que la lógica que define el archivo JavaScript se ejecuta en el cliente. JS permite generar páginas webs dinámicas por medio de la manipulación del DOM (Document Object Model).

AJAX (sigla de *Asynchronous JavaScript And XML*) es una técnica que permite comunicarse con el servidor web de manera asíncrona y en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, mejorando la interactividad, velocidad y usabilidad en las aplicaciones.

7.2 Capa de negocios

Aquí es donde se definen las condiciones que deben cumplir los datos para ser guardados en la base de datos, se procesan las consultas que se harán a la base de datos y se formatean las respuestas antes de pasar a la capa de presentación. Aquí también se vela por la seguridad de los datos impidiendo el acceso a usuarios no autorizados protegiendo la integridad de los datos guardados en la base de datos.

Se utiliza como servidor web al servidor HTTP Apache Versión 2.4.4. Este servidor recibe las peticiones de la capa de presentación, las procesa y las envía al navegador.

7.2.1.1 PHP - PHP Hypertext Pre-processor

PHP es un lenguaje de preprocesador de texto que trabaja a nivel de servidor y maneja la lógica del sistema en la capa de negocios. Para utilizarlo se debe instalar el módulo de Apache *mod_php* y configurar la directiva del núcleo de php '*short_open_tag*' como activa.

PHP es el encargado de la lógica en el servidor, es decir, recibe los mensajes desde la aplicación y entrega una respuesta, realiza procesos lógicos a los datos, se conecta y recupera información desde la base de datos y también genera las paginas HTML en forma dinámica en el servidor que luego son enviadas al cliente. Soporta orientación a objetos es versátil y uno de los lenguajes más utilizado en Internet.

7.3 Capa de Datos

La capa de datos se encarga de guardar todos los datos generados por la aplicación para una posterior consulta.

Los datos son guardados en una Base de Datos Relacional administrada por MySQL 5.6, un Sistema de Gestión de Base de Datos (SGBD) confiable, estable y de distribución libre con el motor de almacenamiento InnoDB. Su característica principal es que soporta transacciones de tipo ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad) y bloqueo de registros e integridad referencial.

Un SGBD es un conjunto de programas que permiten el almacenamiento, modificación y extracción de la información en una base de datos, además de proporcionar herramientas para añadir, borrar, modificar y analizar los datos. MySQL es el segundo[2] SGDB más utilizado en el mundo después de Oracle.

Las consultas a la base de datos son realizadas utilizando el lenguaje de MySQL, una versión propia de SQL que no difiere en estructura ni semántica pero agrega funciones personalizadas y acotadas al funcionamiento en este SGBD.

7.4 Comunicación entre capas.

La comunicación entre la Capa de Presentación (Navegador Cliente) y Capa de Negocios (Servidor Web) se realiza por el protocolo HTTP utilizando métodos POST y GET que solo permiten el paso de mensajes en texto plano. Los datos son manejados en estructuras complejas que agrupan más de una variable lo que hace difícil transformarlas en una cadena de texto.

Para enviar mensajes con más de una variable se utiliza un lenguaje de marcado. Un lenguaje de marcado es básicamente una serie reglas de estructura y etiquetas que comparten ambas partes para codificar un mensaje. Existen muchos lenguajes de marcados, HTML es uno de ellos, la tecnología asociada a estos lenguajes más conocida es XML (*Extensible Markup Language)*.

En Capa de Presentación JavaScript es quien envía y recibe mensajes desde o hacia el Servidor Web y PHP quien procesa los mensajes. Se descarta el uso de XML porque JavaScript tiene lenguaje de marcado propio de acrónimo JSON descrito en el estándar ECMA-404 [21] y anteriormente era un subconjunto del estándar ECMA-262 publicado en diciembre de 1999.

JSON (*JavaScript Object Notation*) es un formato ligero para el intercambio de datos. Es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML [22]. La simplicidad de JSON ha dado lugar a la generalización de su uso, especialmente como alternativa a XML en AJAX (en Anexo A se ejemplifica su notación y uso). El formato de JSON es ampliamente reconocido por una gran variedad de lenguajes como ActionScript, C++, C#, ColdFusion, Common Lisp, Delphi, Java, Objective-C, Objective CAML, Perl, PHP, Python, Rebol, Ruby, Lua, entre otros.

[2] Según el ranking publicado en http://db-engines.com/en/ranking

7.5 Herramientas para el desarrollo del proyecto.

De las opciones disponibles para montar el ambiente web se optó por el paquete de programas Wampserver 2.4 para Windows – en Linux se puede utilizar XAMPP 1.8.2 – que contiene el servidor Apache 2.4 con el módulo de PHP 5.5, el sistema de gestión de base de datos MySQL 5.6 junto al administrador de base de datos web PHPMyAdmin 4.1.14.

Para codificar se utilizaron 2 software de código abierto: Netbeans 7.3 como IDE y SublimeText2 como editor de texto; ambos fueron elegidos por su estabilidad, experiencia de uso previa y la posibilidad de extender sus funcionalidades por medio de módulos instalables desde el mismo software.

La "Herramienta para Desarrolladores" que proporciona Google Chrome se utiliza para depurar la aplicación, éste permite ejecutar – por medio de una consola – y revisar los script JavaScript y también visualizar y modificar en tiempo real el DOM de la página y su código fuente, entre otras funcionalidades.

En la capa de presentación se eligieron los framework Bootstrap 3 y jQuery 1.10. Bootstrap es un proyecto desarrollado por Twitter para la creación de interfaces de usuario cuyas principales características son que genera vistas utilizando el patrón de diseño web adaptable (del inglés Responsive Web Design), su sintaxis y diseño orientado a objetos y que proporciona una gran gama de componentes reutilizables que facilitan la construcción.

La biblioteca JavaScript jQuery es un proyecto colaborativo de distribución libre y código abierto que proporciona funciones para el manejo del DOM y un API para controlar llamadas AJAX. Se utilizaron tres librerías para solucionar problemas específicos del diseño de interacción:

i. **jQuery-UI 1.10**: librería desarrollada por la fundación jQuery. Implementa funciones para interacciones de arrastrar y ordenar elementos de la interfaz utilizando el ratón.

ii. **nestedSortable 2.0**: librería que extiende las funcionalidades de jquery-ui para el manejo de listas anidadas desarrollado por Manuele J Sarfatti hasta la versión 2.0.

iii. **touch-punch**: librería desarrollado por David Furfero que habilita el uso de eventos táctiles en interfaces que implementen jQuery-UI.

En la capa de negocio se utilizaron dos librerías escritas en PHP para implementar parte de las funcionalidades: Ez_sql desarrollada por Justin Vincent, administra la conexión con la capa de datos y aporta seguridad al prevenir la inyección SQL ya que entrega un API para ejecutar consultas con una sintaxis simplificada y mapea los resultados a colecciones de objetos; y PHPExcel 1.7.9 librería de código abierto que provee un conjunto de clases y métodos que permiten leer y escribir hojas de cálculo en diferentes formatos.

8 Desarrollo del prototipo de sistema.

En los siguientes puntos se detallan las tareas que han acaecido para dar lugar al prototipo del software. Comienza con el planteamiento de requerimientos, en segundo lugar se muestra el análisis mediante casos de uso, después, se explica el modelo de datos representado por un diagrama relacional seguido por la definición de la arquitectura del sistema, para culminar con la implementación de los prototipos evolutivos que formarán el prototipo final de sistema.

8.1 Requerimientos.

A continuación se listan los requerimientos funcionales y no funcionales (o restricciones) identificados para el desarrollo del prototipo de sistema.

8.1.1 Requerimientos Funcionales.

✓ El participante o usuario de la prueba generará grupos de tarjetas en columnas (indistintamente llamadas "categorías") dispuestas en un lienzo según su criterio.
✓ El participante podrá crear nuevas tarjetas con conceptos definidos por él durante la prueba.
✓ Las tarjetas creadas por el participante deben diferenciarse visualmente de las tarjetas creadas por el evaluador para la prueba en cuestión.
✓ El sistema debe permitir arrastrar una tarjeta dentro de una categoría, y desde una categoría a otra.
✓ Se puede generar un subgrupo (o subcategoría) de tarjetas arrastrando una tarjeta dentro de otra.
✓ Las tarjetas pueden cambiar de posición, ya sea subir y bajar dentro de la lista de tarjetas de su (sub) categoría.
✓ La tarjeta puede ser eliminada (o movida) de una categoría o grupo sin afectar el orden actual.
✓ El evaluador definirá un la cantidad de anidación máxima que puede tener una tarjeta (si es 0 significa que no puede generar subgrupos).
✓ Un subgrupo debe tener las mismas funciones que una tarjeta, o sea puede ser ordenada y arrastrada como un solo bloque.
✓ Un subgrupo de tarjetas debe diferenciarse visualmente de las demás tarjeta agrupadas en la columna.

✓ El evaluador del estudio debe poder entregar las instrucciones para que la actividad se lleve a cabo correctamente. Las instrucciones de la aplicación están en una plantilla que el evaluador de la prueba puede editar.
✓ El evaluador tendrá un portafolio de Proyectos. Cada Proyecto agrupa uno o más estudios.
✓ Los Proyectos y Estudios pueden ser creados, editados y eliminados.
✓ El evaluador puede editar el nombre de un estudio y/o proyecto.
✓ El Evaluador puede modificar la lista de tarjetas creadas para una prueba. Las acciones son: Eliminar, Agregar y Editar Tarjetas. No puede eliminar las tarjetas que ya son parte de alguna prueba realizada.
✓ El sistema debe poder visualizar el estado final de la prueba de ordenamiento.
✓ El sistema entregará la opción de imprimir el estado final de una prueba cuando es visualizada.
✓ La impresión debe diferenciar las tarjetas creadas por el evaluador y usuario, además debe mantener la estructura de las categorías y subgrupos.
✓ El sistema generará automáticamente un reporte consolidado del resultado de todos los ordenamientos pertenecientes a un estudio, en formato Excel y que será descargado automáticamente al computador del evaluador.
✓ El reporte debe llevar como nombre el Nombre del proyecto y el Nombre del estudio.
✓ Cada prueba realizada en un estudio se resumirá en una hoja diferente del libro Excel.
✓ Proporcionar un panel de administración tipo "súper-usuario" para la gestión de evaluadores.

8.1.2 Requerimientos no funcionales.

✓ El sistema debe contar con conexión a internet al momento de guardar los datos del ordenamiento.
✓ La aplicación funciona en los navegadores webs: Mozilla Firefox 22 o Google Chrome 24 o en versiones superiores.
✓ El navegador web debe tener la ejecución de JavaScript activada.

✓	No se permite el uso de caracteres especiales ni espacios en los campos de entrada de datos. Caracteres /&%$)(¿ no están permitidos, al igual que los tildes. Solo se admite el uso de letras, números y guiones.
✓	La interacción touch debe realizarse solo con un dedo (simple touch). No se reconoce gesto multitouch.
✓	La interacción del usuario con el sistema debe ser con mouse y/o touch sin diferencias en funcionalidad.

8.2 Casos de uso.

Los diagramas de casos de uso sirven para especificar la comunicación y el comportamiento de un sistema mediante su interacción con los usuarios y/u otros sistemas. Permite definir las funcionalidades generales del sistema, la interacción entre estas y con los actores, y los límites del sistema así como la relación con su entorno. Los casos de uso se complementan con descripciones cortas de las características, funcionalidades y salida esperada para el software, módulo o incremento que pueden reemplazar a los documentos de especificación funcional [23].

En las siguientes páginas se presenta el caso de uso general del sistema y luego la especificación de cada caso de uso acompañado de su descripción.

La imagen 8.1 corresponde a la visión general del sistema donde se representan las funciones que entrega la aplicación y quién interactúa con cada una de ellas. Se definen dos actores principales: el *participante*, quien desarrolla las pruebas de ordenamiento de tarjetas en la aplicación; y el *evaluador*, quien prepara las pruebas y revisa los resultados.

El actor *Administrador* es un 'súper-usuario' que hereda las características del actor *evaluador*, esto significa que además de trabajar como evaluador (preparando y tomando pruebas), puede gestionar los evaluadores existentes, agregando o eliminando usuarios del sistema.

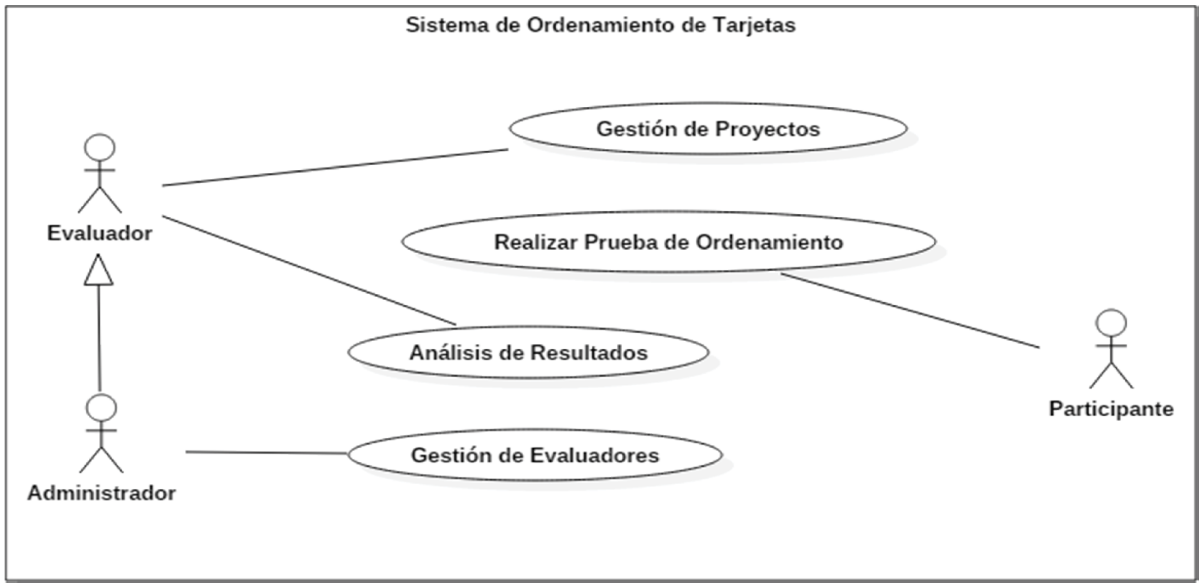


Imagen 8.1 Caso de uso general

La aplicación permite al evaluador gestionar – crear, editar y eliminar - sus proyectos; cada proyecto agrupa un conjunto de estudios. El evaluador debe gestionar – crear, editar, eliminar – los estudios pertenecientes a sus proyectos. Al crear un estudio, el evaluador debe definir el nombre del estudio y la lista de tarjetas predeterminadas para el estudio. Estas tarjetas estarán disponibles para todas las pruebas de ordenamiento que se realicen en dicho estudio. En la imagen 8.2 se representa el caso de uso *"Gestionar estudios"*.

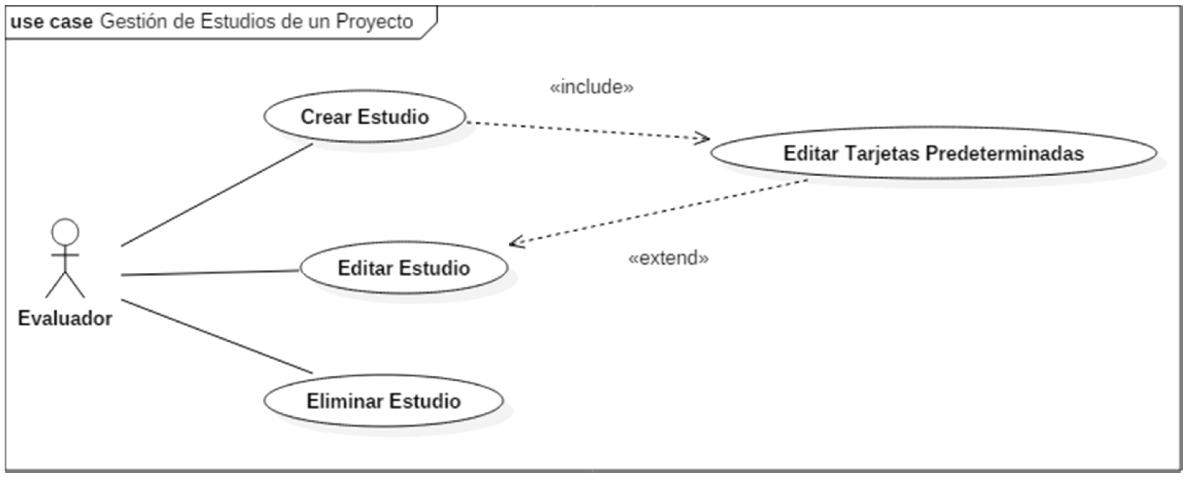


Imagen 8.2 Caso de uso: Gestión de estudios

El caso de uso *"Realizar prueba de ordenamiento"* (especificado en la imagen 8.3) es el único que interactúa al sistema con el participante de las pruebas. Es el participante quien genera los datos que serán recopilados por el sistema para que el evaluador pueda analizarlos.

El usuario participante de una prueba de ordenamiento de tarjetas puede crear nuevas tarjetas si lo estima necesario (que estarán disponibles solo en la prueba en que son creadas); puede agrupar las tarjetas en diferentes grupos, creando, eliminando y/o modificando los

existentes; y, cuando la prueba termina, guardar el estado final de los grupos sin que pueda modificarlos posteriormente.

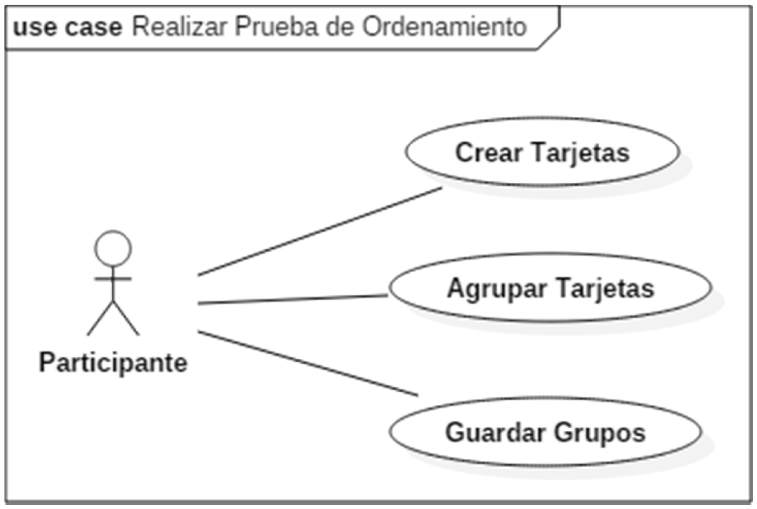


Imagen 8.3 Caso de uso: Realizar prueba de ordenamiento.

La imagen 8.4 que se muestra a continuación corresponde a la especificación del caso de uso de *"Análisis de Resultados"*, en este caso de uso el evaluador consultará al sistema por el resultado de las pruebas terminadas en algún estudio para analizar los resultados. El sistema entrega la opción de visualizar una prueba terminada (representando el orden y jerarquía que tenían las tarjetas al momento que el usuario finalizó su prueba) o exportar una síntesis de todas las pruebas de un estudio según la notación propuesta en [1].

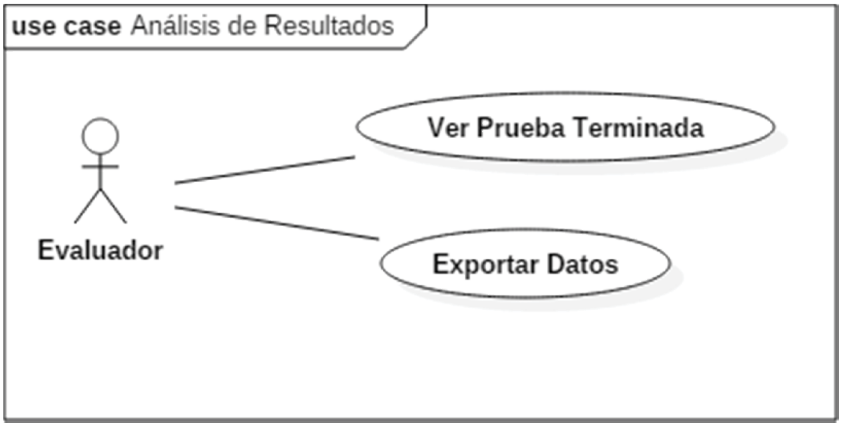


Imagen 8.4 Caso de uso: Análisis de Resultados.

8.3 Modelo de base de datos.

Del análisis realizado en el punto anterior se realiza el modelo de datos representado en un Diagrama Relacional. Este modelo permite representar la información del mundo real por medio de *tablas* unidas a través de *relaciones* que pueden ser representadas usando *tuplas* de datos pertenecientes a diferentes tablas.

El desarrollo del modelo de datos del prototipo de sistema comienza con una representación inicial que busca ser lo más completa posible y que se refina durante las primeras fases del desarrollo del sistema.

Se establecieron estándares para el manejo de los datos: las claves foráneas son nombradas con el prefijo 'fk_'; los campos que posean nombres compuestos se utiliza un guión bajo ('_') para separarlos (por ejemplo *numero_categoria* de la tabla *categoria*, ver imagen 8.5), y; todos los nombres de elementos (campos, tablas, restricciones, etc.) de la base de datos serán escritos en minúsculas.

La imagen 8.5 corresponde al diagrama relacional final de la base de datos. Está compuesto por 7 tablas donde *evaluador* y *tarjeta* son las principales. Un evaluador tiene *proyectos* a cargo, cada proyecto está compuesto por *estudios* que definen una lista de *tarjetas* para realizar pruebas de *ordenamiento*. Cada registro de *ordenamiento* se diferencia de otro según su *estampa,* este campo corresponde a la hora en que el usuario participante inicia por primera vez su prueba de ordenamiento de tarjetas de un estudio.

La tabla *categoria* representa a los grupos principales de una prueba y la tabla *subcategoria* representa los subgrupos que se forman en dentro de un grupo principal. En los siguientes puntos de este documento se utiliza indistintamente los conceptos "categoría" para referir a "grupos principales", y "subcategoría" para referir a subgrupos.

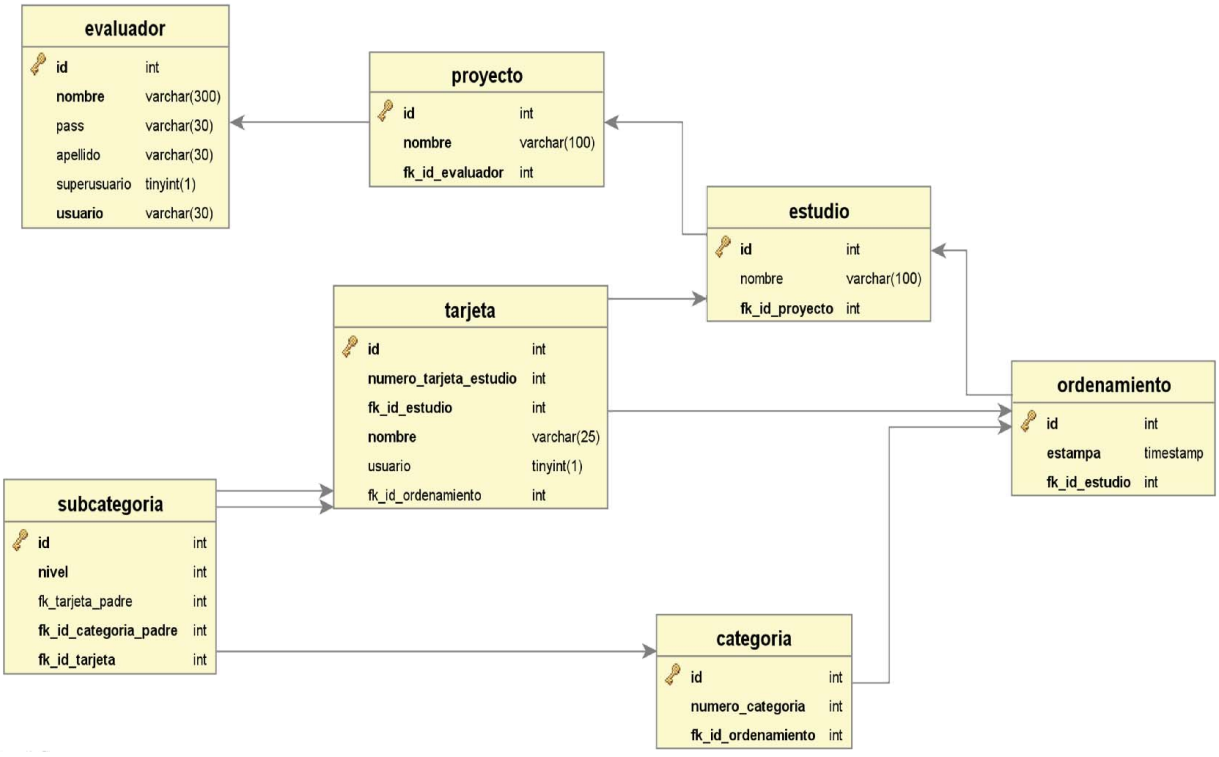


Imagen 8.5 Diagrama Relacional de la base de datos.

La categoría (o grupo principal) mediante el campo *numero_categoria* indica la posición que usa esa categoría en el eje horizontal – de izquierda a derecha – de la mesa de ordenamiento y la distingue de las demás categorías de un mismo ordenamiento. Cada registro

de *subcategoria* representa una tarjeta presente en el grupo principal, con el campo *nivel* se identifica la profundidad de anidación dentro del grupo principal. La posición en el eje vertical – desde arriba hacia abajo – de una tarjeta dentro de un grupo principal se obtiene según el orden en que se encuentran guardados los registros en la base de datos.

Usando el campo *nivel* y *fk_tarjeta_padre* se puede diferenciar un subgrupo de tarjetas de otro subgrupo que pertenece a la misma categoría debido a que todos los registros de *subcategoria* que pertenecen a al mismo subgrupo tienen el mismo valor en los campos antes mencionados.

La tabla *tarjeta* contiene todas las tarjetas de todos los estudios. Una tarjeta tiene un número correlativo para el estudio al que pertenece y se guarda en el campo *numero_tarjeta_estudio*.

Una tarjeta creada por un usuario durante el desarrollo de una prueba tiene el campo *usuario* activado y *fk_id_ordenamiento* tendrá el valor de la clave primaria de la prueba en que fue creada (registro en la tabla *ordenamiento*). El valor que tendrá este registro y todas las demás tarjetas de usuario en *numero_tarjeta_estudio* será mayor que – y correlativo desde – un número máximo de tarjetas por defecto definido previamente.

La plantilla utilizada en [1] para resumir los datos de una prueba de ordenamiento de tarjetas puntualiza 5 columnas que podemos vincular a los campos de las tablas en la base de datos. La relación es la siguiente:

Columna en [1]	**Descripción**	**Tabla**	**Campo**
Father's Order	Posición del padre en relación a las categorías principales	categoria	*numero_categoria*
Sub Order	Posición del ítem dentro de una sub categoría	subcategoria	*nivel*
Order	Posición del ítem dentro de la categoría	subcategoria	Calculada según registros de la subcategoria
Father's Id	Numero identificador de la tarjeta padre de la categoría	subcategoria	*fk_tarjeta_padre*
Sub Father's Id	Numero identificador de la tarjeta padre de la sub categoría	subcategoria	Calculada según registros de la subcategoria

Tabla 8.1 Relación de elementos de la plantilla con campos de la base de datos.

8.4 Plan de pruebas

Se realizaron pruebas informales durante la programación de las funcionalidades de cada módulo, además de pruebas de integración y de sistema. No se establece un plan de pruebas formal ni se realizan pruebas automatizadas.

Cuando el módulo es completado se efectúan pruebas de caja negra. Los errores encontrados en durante las pruebas realizadas en este proceso son corregidos de inmediato y se repiten las pruebas. Una vez verificado el comportamiento esperado el módulo es agregado al sistema y se realizan pruebas de integración y sistema.

Las pruebas de caja negra y de integración fueron realizadas por el desarrollador, las pruebas de sistema se realizaron con una inspección de usuario guiada. De la prueba de sistema se genera una lista de incidencias, una vez todas estas son corregidas, se repite la prueba de sistema por un usuario para comprobar que los cambios no hayan tenido repercusiones en otras partes del sistema.

8.5 Entregas de software.

Al inicio del proceso no fueron definidos los entregables debido al modelo de procesos evolutivo exploratorio. El sistema es construido a medida que los requerimientos eran definidos y evaluados por medio de prototipos.

Las vistas principales del sistema son dos. La primera se despliega una vez el usuario inicia sesión (ver imagen 8.6) donde encuentra las instrucciones de uso del sistema y su portafolio de proyectos. Los proyectos están ordenados en pestañas, cada pestaña tiene la lista de estudios que pertenecen al proyecto, y además, cada proyecto posee una barra para realizar acciones sobre el mismo.

La segunda vista corresponde a la toma de una prueba de usabilidad (ver imagen 8.7). Contiene como título el nombre del estudio para el que se realiza la prueba, a la derecha del titulo hay con una barra de acciones. Debajo de estas, hay dos secciones, una contiene las tarjetas disponibles para agrupar; la otra seccion, de mayor tamaño, se utiliza para crear los grupos de tarjetas.

Las entregas de software realizadas durante la construcción se pueden dividir en cinco:

1. Prototipo de interacción táctil y anidación en grupos.
2. Prototipo de creación de tarjeta, coloreado y persistencia de datos.
3. Implementación de creación de proyectos y de estudios.
4. Implementación de visualización en navegador e impresión de prueba.
5. Implementación de exportar resultados.

En las siguientes páginas se detalla la implementación de las funcionalidades de cada prototipo.

Imagen 8.6: Vista inicial del sistema una vez iniciada la sesión.

Imagen 8.7: Vista de una prueba de ordenamiento de tarjetas.

8.5.1 Prototipo de interacción táctil y anidación en grupos.

El análisis de la tarea de agrupar tarjetas dio como resultado la división de ésta en 3 tareas más pequeñas: arrastrar, ordenar y colorear. En esta sección se explica la implementación de las dos primeras tareas, arrastrar y ordenar, la tarea de colorear esta explicada en detalle en la sección 8.5.2.

8.5.1.1 Implementación de la interacción táctil y objetos móviles.

Los objetos móviles del área de trabajo obtienen esa propiedad al utilizar sobre estos la función *draggable()* de jQueryUI [24] (para entender cómo aplicar una función a una colección de datos con jQuery ver Anexo A). Utilizando sólo jQuery la interacción táctil se iniciaba con mucho retardo, era poco sensible y poco precisa.

Como solución se integró el plugin jQueryUI Touch Punch que habilita el uso de los eventos táctil en sitios que usan jQueryUI, mapeando los eventos del mouse a eventos táctiles de forma transparente para el programador, pudiendo utilizar las funciones jQueryUI sin preocuparse de los problemas de interacción entre el dispositivo y la aplicación.

Para utilizar jQuery UI Touch Punch en la vista, se debe primero incluir jQueryUI y después el plugin Touch Punch. A continuación se encuentra el código utilizado para incluir estas librerías en una página HTML y la última línea otorga el comportamiento "arrastrar y ordenar" a todos los *objetos*.

```
<script src="http://code.jquery.com/jquery-1.9.1.min.js"/>
<script src="http://code.jquery.com/ui/1.10.2/jquery-ui.min.js"/>
<script src="jquery.ui.touch-punch.min.js"/>
<script>$('#objeto').draggable();</script>
```

8.5.1.2 Implementación de la agrupación de tarjetas.

Durante el ordenamiento de tarjetas el usuario crea grupos de tarjetas según su criterio, a efecto de la lógica del sistema estos grupos se les llama *categoría* y están compuestos por 0 o más subgrupos (o subcategorías), que a su vez están compuestas por una o más tarjetas.

La categoría es básicamente una lista de elementos que se pueden ordenar dentro de la misma y recibe elementos desde otras listas. Entre categorías se identifican por su id numérico que aumenta a medida que se crean más categorías en la prueba

El botón *'Nueva Categoría'* presente en la barra de acciones de la prueba (ver imagen 8.7) ejecuta una función JavaScript que inserta en el DOM una lista y le asigna el comportamiento de deseado. Con este botón el usuario crea una nueva categoría vacía con un mensaje por defecto *"Arrastre abajo"* que indica al usuario la acción a seguir para utilizarla.

Imagen 8.8: Nueva categoría vacía

El mensaje por defecto de la categoría es un elemento *'li'* identificado con la clase *tarjeta-default* que debe ser eliminado cuando la lista recibe una tarjeta (ver imagen 8.10) y reaparecer cuando la categoría está vacía (ver imagen 8.8).

La tarea de ordenar una lista la resuelve jQuery UI con la función *sortable()* [24] que permite ordenar elementos arrastrándolos con el mouse, esta función usa el núcleo de *draggable()* por lo que las opciones y métodos de esta también están disponibles, además, tiene opciones y métodos para configurar la interacción entre los elementos.

Para gestionar el mensaje por defecto de una categoría vacía se recurrió a los eventos de la interacción *sortable*: *receive* y *remove.* Cuando una categoría recibe un elemento – evento receive – revisa si tiene elementos que posean la clase *.tarjeta-default* y lo elimina. Al eliminar un elemento de una categoría – evento remove – revisa si es necesario agregar la tarjeta por defecto y si fuese así, se agrega.

De los métodos del API *sortable*, los utilizados fueron:

i. **serialize**: Crea un String concatenando los id de los elementos de la lista ordenable. El id del elemento debe tener el formato "nombre1_valor1" y la función separa el id en otro String de la forma "nombre1[]=valor1&nombren[]=valorn"
ii. **toArray:** Crea un vector de String con los id de cada elemento separados por coma (,).

En las pruebas realizadas con jQuery UI *draggable* o *sortable* no se logró tener varios niveles de anidación para crear subgrupos en un grupo de tarjetas, utilizar el concepto que nombra la tarjeta y conocer su posición con certeza. Para solucionar esto se utilizó la librería *nestedSortable*.

8.5.1.3 Implementación de la anidación multinivel.

La librería nestedSortable para jQuery UI extiende las funcionalidades por defecto de *sortable()* para crear listas con varios niveles de anidación.

La librería requiere que los elementos de la lista contengan dos hijos:

1. Un elemento contenedor (<*div*>).
2. Una lista que siga el mismo formato (opcional).

El elemento de la lista (<*li*>) representa a la tarjeta y el contenedor tiene el concepto de esta, la lista representa otro nivel de anidación para formar grupos cuyas tarjetas dependen de la jerárquicamente del concepto superior (definido en el *div*).

Las tarjetas y grupos formados en este nuevo nivel de anidación son llamados subcategoría o subgrupo. Las tarjetas presentes en los subgrupos se dice que tienen un nivel de anidación – o profundidad – superior que el elemento que los contiene y que dependen de este.

Sin embargo, a nivel de datos una categoría está compuesta solo por subcategorías que se diferencian entre sí por su id y se agrupan según nivel de anidación y tarjeta padre (ver punto 8.3 para más información sobre el modelo de datos).

El siguiente código HTML corresponde a una categoría y su lista de las tarjetas en un ordenamiento cualquiera. La tarjeta '*menta'* contiene una lista generando así una subcategoría.

Para utilizar nestedSortable debemos seleccionar la colección de datos y aplicar la función. Se utiliza un objeto para configurar el comportamiento de los objetos e un objeto está compuesto por las opciones de *sortable* más las opciones de *nestedSortable* que se deseen ajustar. Para más información sobre cómo aplicar una función a una colección de datos con jQuery ver Anexo A, en el Anexo B hay un ejemplo de configuración del comportamiento de nestedSortable.

La imagen 8.9 es un ejemplo de un grupo organizado por un usuario, corresponde a una categoría con 3 tarjetas en nivel 0, dos de las cuales tienen subcategorías anidadas. Las tarjetas que dependen de otra categoría son de color verde.

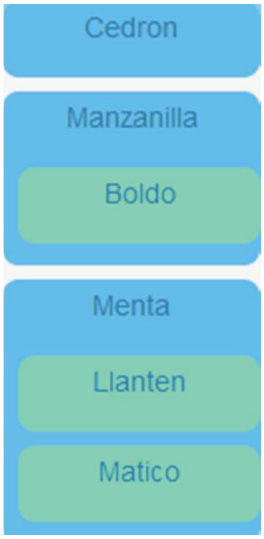


Imagen 8.9: Categoría ordenada por un usuario con subniveles de anidación.

El código HTML que genera la imagen anterior es el siguiente:

```
<ul class="span2 categoria ui-sortable" id="1">
        <li class="tarjeta" id="Cedron_157">
            <div>Cedron</div>
        </li>
        <li class="tarjeta" id="Manzanilla_153">
            <div>Manzanilla</div>
            <ul>
                <li class="tarjeta subTarjetaColor" id="Boldo_156">
                    <div>Boldo</div>
                </li>
            </ul>
        </li>
        <li class="tarjeta" id="Menta_152">
```

```
            <div>Menta</div>
            <ul>
                <li class="tarjeta subTarjetaColor">
                    <div>Llanten</div>
                </li>
                <li class="tarjeta subTarjetaColor" id="Matico_151" >
                    <div>Matico</div>
                </li>
            </ul>
        </li>
</ul>
```

8.5.1.4 Funcionamiento y configuración de la anidación en subniveles.

La interacción comienza al presionar – el botón del ratón o la pantalla – sobre un elemento de la interfaz al cual se haya asignado previamente el comportamiento utilizando nestedSortable. En el desarrollo de la prueba, los elementos HTML que poseen este comportamiento son los ítems – <*li*> – de una lista.

Mientras la tarjeta permanezca presionada puede ser arrastrada, ordenada y anidada. La ubicación final de la tarjeta está guiada por un recuadro que muestra cual será el orden de los elementos si se coloca en ese lugar (ver imagen 8.11). Finalmente, la interacción termina cuando el botón es soltado y el elemento es liberado en la ubicación asignada (ver imagen 8.12).

El proceso con el que nestedSortable maneja la interacción y la anidación de elementos se describe con la siguiente serie de pasos:

1. Copia el elemento *li* a arrastrar y lo añade al final de los existentes, éste tiene la clase *ui-sortable-helper* y los estilos para dar el efecto de flotar.
2. Oculta el elemento original ajustando su propiedad '*display=none'*.
3. Coloca un nuevo elemento *'<li class="placeholder">
'* debajo del original a modo de marcador de posición. La clase *placeholder* se definió en el atributo del mismo nombre del objeto de opciones.
4. Cuando se detecta el evento "*mouseup"*, se elimina la copia del elemento que está siendo arrastrada
5. El elemento original se mueve hacia donde está el marcador de posición.
6. Se elimina el marcador de posición.

Para anidar una tarjeta a otra y crear así un subnivel de anidación (entiéndase subcategoría o subgrupo), se debe ubicar el marcador de posición debajo del elemento que será la cabeza del subnivel y luego, el puntero debe desplazarse ligeramente en sentido diagonal superior derecha para que el marcador de posición se mueva dentro de la tarjeta. Para reordenar una tarjeta se debe presionar sobre ella y desplazarla hacia cualquier otra posición.

En la siguiente secuencia de imágenes se muestra el proceso de anidación de una tarjeta a otra para crear una subcategoría. En un principio la categoría solo posee la tarjeta *'Manzanilla'* de nivel de anidación 0, a esta tarjeta a modo de subcategoría se le añadirá la tarjeta *'Boldo'* que estará al nivel de anidación 1.

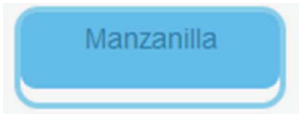


Imagen 8.10 Categoría con una subcategoría de nivel 0

Imagen 8.11 Agregando tarjeta como subcategoría de otra

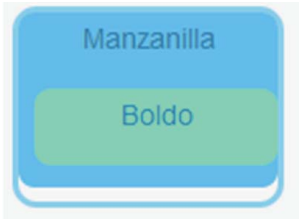


Imagen 8.12 Tarjeta anidada como subcategoría de otra.

Los objetos con los que se conectarán los elementos se establecen en el atributo *'connectWith'* del objeto de configuración. Este atributo recibe como parámetro un selector de elementos que define con quien se puede conectar. Es una relación de un solo sentido, si se desea conectar en ambos sentidos, ambos elementos deben configurarse para esto [25].

La profundidad de la anidación es establecida por el atributo *'maxLevels'*, si su valor es 0 la profundidad es ilimitada. En caso que no se pueda añadir un elemento como subnivel a otra se ejecuta la animación de regreso a su posición.

El comportamiento agregado por nestedSortable a una lista se hereda a todos los nodos hijos, esto permite mover todo un subnivel y/o los ítems de manera individual.

8.5.2 Prototipo de creación de tarjeta, coloreado y persistencia de datos.

Una vez logrado la interacción y anidación en subniveles de tarjetas se desarrolla un prototipo para solucionar el problema de diferenciación entre niveles de anidación, el almacenamiento de datos y la creación de tarjetas por parte del usuario durante el desarrollo de una prueba.

8.5.2.1 Implementación del coloreado de subgrupos

En este punto las tarjetas se agrupan en categorías, pueden tener muchos subniveles de anidación dentro de una misma categoría y los subniveles pueden moverse por completo de una lista a otra.

Se utilizaron dos colores para diferenciar los niveles de anidación y con esto diferenciar también los grupos dependientes de una tarjeta. En la imagen 8.13 se aprecian los beneficios que entrega esta medida al compararlo con otra categoría con el mismo orden jerárquico pero sin colorear.

En el ordenamiento coloreado se puede identificar el grupo dependiente del concepto *Menta: Manzanilla* y *llantén;* y de *RosaMosqueta: Matico* y *Cedrón*. La tarjeta con el concepto *Manzanilla* es padre de *Bailahuen* y este a su vez es padre de *Boldo*. *Bailahuen* está en el nivel de anidación 2 y Boldo en el nivel 3 mientras que *Manzanilla, Llantén, Matico y Cedron* están en el nivel de anidación 1.

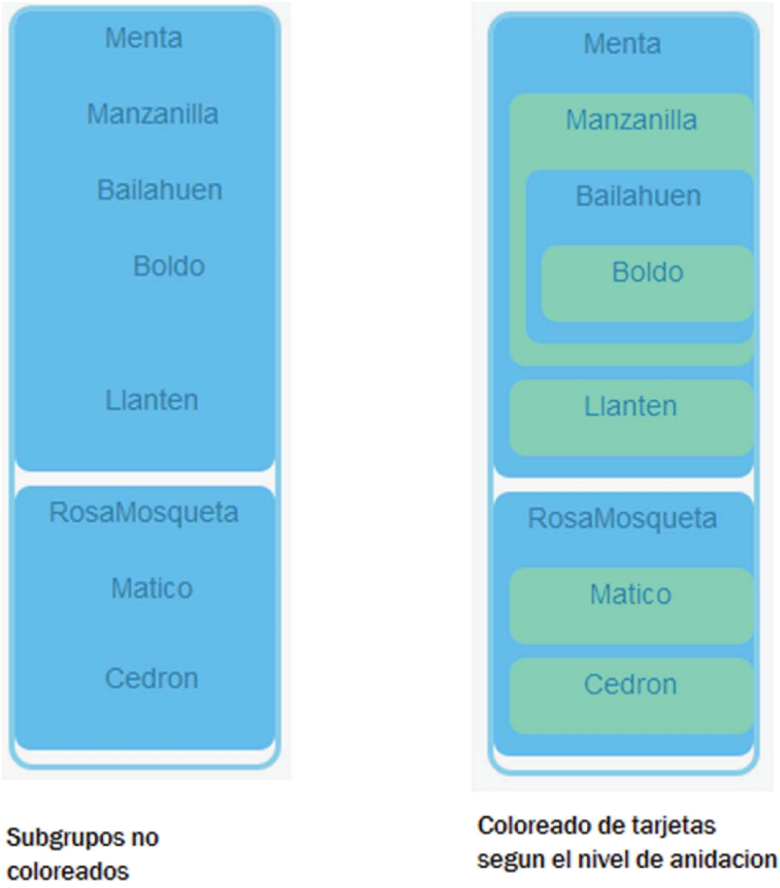


Imagen 8.13: Patrón de colores en la anidación de tarjetas según la profundidad.

Cada vez que el orden de las tarjetas cambia el color de estas debe ser actualizado para que corresponda con su nivel de anidación. Para pintar las tarjetas, del API de jQueryUI, utilizamos los eventos *receive* cuando las tarjetas son ordenadas entre categorías y *stop* para tarjetas ordenadas dentro de una misma categoría. La función que define el comportamiento de

estos puede recibir como parámetro un evento tipo *event* y un objeto *ui*, de este objeto (que se describe en [25]) se utiliza el atributo *item* que es un objeto con el que se representa el elemento del DOM que está siendo arrastrado.

Cuando hay un subgrupo dentro de otro subgrupo los colores se alternan (Azul-verde-Azul) para identificar la jerarquía de anidación. Se crean clases que definen el color de la tarjeta: clase *tarjeta* es color azul, clase *subTarjetaColor* para el color verde y *tarjeta-usuario* para las tarjetas grises.

La clase *tarjeta* es la clase por defecto en el elemento que representa a la tarjeta física. Cuando una tarjeta debe ser pintada azul, se elimina la clase *subTarjetaColor* de ella y se revisa que los elementos anidados a esta tarjeta posean el color que corresponde a su nivel de anidación actual. Las tarjetas con color gris no modifican su color.

El siguiente algoritmo expresado en pseudocódigo explica la lógica en el cambio de color de las tarjetas al ser ordenadas. El algoritmo se implementó usando JavaScript y jQuery.

```
Procedimiento PintarTarjetas (tarjetaMovida)
     Var elementoPadre <- tarjetaMovida.padre;
     Text actual;
     Text padre;
Inicio
     si (elementoPadre = "categoría") entonces
          tarjetaMovida.pintarTarjeta("azul")
     sino
          si (tarjetaMovida.color = "verde") entonces
               actual <- "verde"
          sino
               actual <- "azul"
          fin_si

          si (elementoPadre.color = "verde") entonces
               padre <- "verde"
          sino
               padre <- "azul"
          fin_si

          si (actual = padre) entonces
               cambiarColor(tarjetaMovida)
          fin_si
     fin_si
```

```
    /*Realiza el procedimiento de pintar a todas las tarjetas presentes en
    todos los subniveles (si tiene)*/
    PintarTarjetasSubNiveles(tarjetaMovida)
Fin
```

La lógica comienza revisando la posición a la que fue movida la tarjeta actual y si el color que tiene corresponde al nuevo nivel de anidación. Luego revisa todos los nodos hijos de la tarjeta y cambia el color de este si corresponde.

8.5.2.2 Implementación de la creación de tarjetas por usuarios.

El usuario puede crear una o más tarjetas durante el desarrollo de una prueba si lo considera necesario. La tarjeta creada por el usuario tiene las mismas propiedades que las definidas por el evaluador (arrastrable, ordenable y poseer subniveles) y está asociada a la prueba de ordenamiento en que se creó por lo que no se mostrará a los usuarios que realicen otras pruebas en el mismo estudio.

La tarjeta creada por el usuario (desde ahora *tarjeta usuario*) se diferencia del resto de tarjetas de una prueba por ser de color gris claro; las tarjetas creadas por el evaluador (desde ahora *tarjeta evaluador*) son de color azul y cuando corresponde a una subcategoría son de color verde claro.

Para crear una tarjeta se utiliza el botón *"Crear Tarjeta"* que está bajo la lista de tarjetas por defecto de la prueba (ver imagen 8.14), este botón activa un cuadro de dialogo *modal* para ingresar el concepto de la nueva tarjeta (ver imagen 8.15).

Imagen 8.14 Lista de tarjetas para ordenar.

El cuadro de dialogo es un componente proporcionado por Bootstrap. La declaración del botón *"Crear Tarjeta!"* incluye atributos de configuración para el comportamiento del cuadro.

Imagen 8.15 Lista de tarjetas para ordenar

Al presionar el botón *"Agregar!"* se solicita la creación del registro de la nueva tarjeta usuario con el concepto introducido. También modifica el DOM para colocar la tarjeta nueva al inicio de la lista de tarjetas tal como se muestra en la imagen 8.14 con la tarjeta *'Ojos'*. En el anexo C está la función JavaScript que ejecuta la lógica descrita.

La solicitud para crear el nuevo registro es atendida en el servidor por scripts en PHP. Los archivos PHP utilizan el API de *ez_sql* para conectarse y realizar consultas a la base de datos MySQL. Se instancia un objeto de tipo *conexión*, luego cada consulta actualizará el estado del atributo *$db* con los datos de la respuesta a dicha consulta. En el Anexo E hay una explicación detallada sobre el uso y composición de un objeto *ez_sql*. El código de las funciones con la lógica para PHP responder las peticiones Ajax se encuentra en el Anexo D.

Una vez la funcion *agregarTarjeta()* se ejecuta por completo la *tarjeta usuario* es creada en la base de datos, agregada al DOM, mantiene las propiedades de nestedSortable y se diferencia de las demás por su color gris.

La tarjeta usuario es asociada con la prueba en que fue creada y agregada al registro total de tarjetas de un estudio. No está disponible para los demás ordenamientos, otro usuario en otro ordenamiento puede crear una tarjeta con el mismo concepto pero en la base de datos se inserta en un nuevo registro que asocia esa tarjeta a ese ordenamiento.

Las nuevas tarjetas creadas por el usuario comenzaran su numeración posterior al máximo establecido, cada nueva tarjeta usuario tendrá el valor de *numero_tarjeta_estudio* superior a la última tarjeta usuario independiente de la prueba en la que fue creada.

La tabla 8.1es un ejemplo de los datos guardados de cada tarjeta y su interpretación es la siguiente: todas las tarjetas están asociadas al estudio 34; solo la tarjeta con el concepto 'Melisa' fue creada por el evaluador porque el valor de la columna *'Usuario'* es 0 y '*Ordenamiento'* tiene valor NULL; las demás tarjetas también están asociadas al estudio 34 pero fueron creadas en diferentes pruebas (115, 116, 117), y su *numero_tarjeta_estudio* es correlativo entre ellas respecto al estudio pero superior al máximo establecido para todo el sistema (250 tarjetas).

Id	Nº Tarjeta en estudio	Estudio	Nombre	Usuario	Ordenamiento
1	11	34	Melisa	0	NULL
n	251	34	Cultura	1	115
n	252	34	Sociedad	1	116
n	253	34	Comunidad	1	116
n	254	34	Comunidad	1	117

Tabla 8.2 Correlatividad de tarjeta usuario creadas en diferentes pruebas.

8.5.2.3 Implementación del guardado de un ordenamiento en la BD.

La prueba se puede cancelar, pero si guardó una vez o si el usuario creó una tarjeta la prueba quedará registrada en la base de datos de todas formas, aunque se perderán los cambios realizados luego de la última de estas acciones. Si guarda un estado y luego presiona cancelar, los cambios realizados entre el guardado y el momento que cancela no son guardados en la base de datos, se pierden.

La orden de guardado es gatillada al presionar el botón '*Guardar'* y/o '*Terminar'* de la barra de acciones, '*Guardar'* guarda el estado actual del ordenamiento y permite continuar con este; el segundo, solicita la confirmación de la acción, guardar los datos y sale del sistema por lo que el ordenamiento no podrá ser modificado.

Al presionar cualquiera de los botones se ejecuta una función JavaScript que elimina los datos ya guardados del ordenamiento actual (si existen) y escribe los datos actualizados. La función realiza una llamada AJAX de escritura por cada categoría en la prueba.

Para formatear los datos de cada categoría se utiliza el método *'toArray'* de *nestedSortable* [26] de la siguiente forma:

$('#CategoriaId').nestedSortable('toArray', {startDepthCount: 0}).

Esto genera un vector de objetos que representa la jerarquía de anidación de la colección a la que se aplica la función. El formato del objeto que conforma el Array está descrito en el Anexo G. El grupo que se muestra en la imagen 8.16 cuyo código HTML se muestra a continuación genera el vector de objetos que se muestra más abajo.

Imagen 8.16: Tarjeta con un nivel de anidación.

```
<ul class="categoria" id="1">
     <li class="tarjeta" id="RosaMosqueta_154">
          <div>RosaMosqueta</div>
          <ul>
               <li class="tarjeta subTarjetaColor" id="Manzanilla_153">
                    <div>Manzanilla</div>
               </li>
               <li class="tarjeta subTarjetaColor" id="Melisa_161">
                    <div>Melisa</div>
```

```
            </li>
        </ul>
    </li>
</ul>
```

El vector de objetos que representa la jerarquía obtenida al aplicar la función sobre el subgrupo de la imagen 5.17 es:

Posición 0	Posición 1	Posición 2	Posición 3
0: Object depth: 0 item_id: null left: 1 parent_id: "none" right: 8	1: Object depth: 1 item_id: "154" left: 2 parent_id: null right: 7	2: Object depth: 2 item_id: "153" left: 3 parent_id: "154" right: 4	3: Object depth: 2 item_id: "161" left: 5 parent_id: "154" right: 6

Tabla 8.3 Vector de objetos obtenido al aplicar la función "toArray".

En el vector anterior, la posición 0 corresponde a un objeto generado por defecto. Los objetos 2 y 3 se pueden decir hermanos porque tienen el mismo *parent_id.* La posición 1 tiene el *item_id* que es referenciado por los elementos 2 y 3 en el *parent_id*, lo que deduce que es el padre de estos, y éste a su vez no tiene padre.

Para enviar el vector de jerarquía generado al servidor PHP se utiliza la función JavaScript *JSON.stringify()* [29] que convierte el vector a formato JSON[3] y permite enviarlo mediante la función $.ajax() al servidor. El servidor responde a la solicitud y realiza los siguientes pasos:

1. El servidor recupera los datos pasados en la solicitud AJAX usando la función *json_decode()* de PHP [30].
2. Establece conexión con la base de datos usando ezSQL.
3. Crea el registro en categoría a través del objeto ezSQL *conexión*.
4. Inserta los registros de cada tarjeta en la tabla subcategoría usando ezSQL (para revisar los datos que representan la revisar el punto 8.3 Modelo de base de datos).

Esto se repite para cada categoría en el ordenamiento. Las funciones principales que contienen la lógica para gestionar los datos recibidos por el servidor se explican en el Anexo H.

Los datos de cada ordenamiento posteriormente son sintetizados con el formato definido en [1]. Se sintetizan en hojas de cálculo y agrupan en un archivo Excel que se descarga al

[3] Ver Anexo F para una descripción del formato JSON.

computador del evaluador, también la aplicación entrega la opción de visualizar en el navegador el estado final de un ordenamiento en específico.

8.5.3 Implementación de creación de proyectos y de estudios.

Inicialmente el usuario no tendrá proyectos y por ende tampoco estudios que gestionar. Para crear un proyecto, debe entrar a la administración de proyectos mediante el botón *'Gestión Proyectos'* que direcciona al usuario a la vista para este propósito. La imagen 8.17 corresponde a esa vista.

En la administración de proyectos podrá eliminar proyectos (y todos los estudios asociados a él), cambiar de nombre a un proyecto existente o crear uno nuevo.

Imagen 8.17: Gestión de proyectos

Al crear un nuevo proyecto el sistema consulta si desea crear un estudio para aquel proyecto, aunque si no crea un estudio en ese momento puede crearlo más tarde.

Para crear un estudio, el usuario debe definir una cantidad inicial de tarjetas y aparecerá el formulario que se ve en la imagen 8.18. El usuario define un nombre para el estudio y los conceptos de las tarjetas. Los conceptos para las tarjetas no pueden llevar caracteres especiales ni espacios en blanco. Los conceptos pueden repetirse.

Si el usuario necesita más campos para conceptos aparecerán de forma automática en el formulario al presionar la tecla '*enter*' en el último campo visible o el botón azul '[+]' de la parte inferior.

La lista de tarjetas se puede editar al presionar *"Editar estudio"*, sin embargo, las tarjetas que ya fueron agrupadas en alguna prueba no podrán cambiar su contenido. Para finalizar el proceso de creación de un estudio el usuario debe presionar el botón verde *"Guardar y crear!"*.

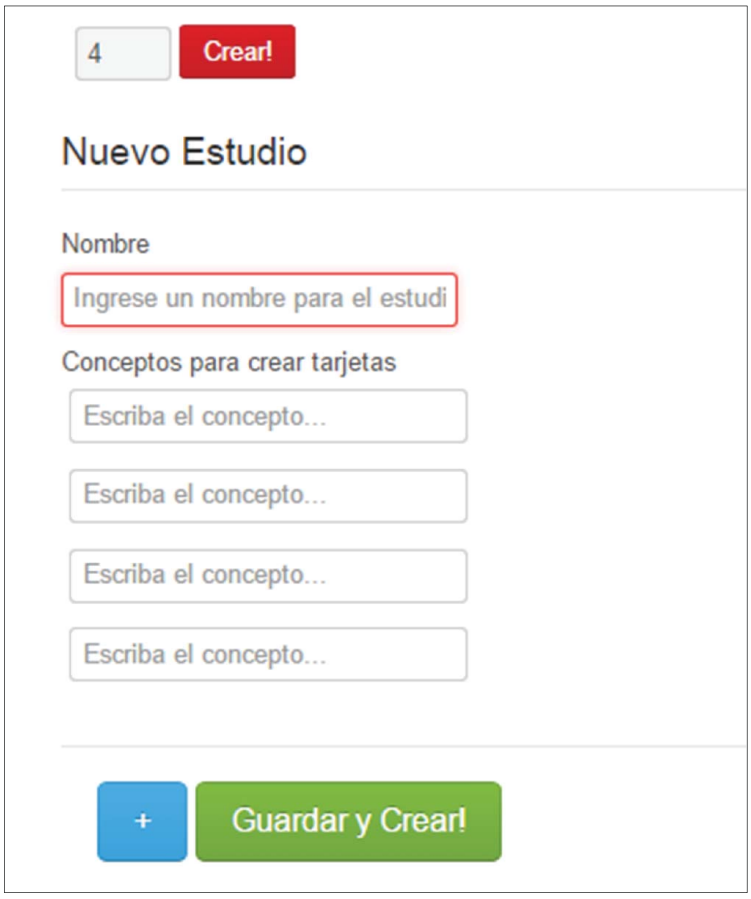


Imagen 8.18: Creación de un estudio y sus tarjetas por defecto.

8.5.4 Implementación de visualización en navegador e impresión de prueba.

El sistema entrega la opción de visualizar cualquier prueba guardada. Al visualizar una prueba solo se cargan las tarjetas (por defecto y creadas por el usuario) que fueron agrupadas, las que no fueron agrupadas se omiten en este proceso. La hoja para imprimir también omite esas tarjetas.

8.5.4.1 Implementación de la visualización de una prueba terminada.

Presionando el botón '*Pruebas Terminadas*' en la barra de acciones de un estudio (ver Imagen 8.19) se genera una lista con las pruebas guardadas en aquel estudio, ordenadas según la fecha y hora en que fue guardada de manera ascendente.

Mediante el botón '*Estado Final'* se carga la vista de la prueba con las tarjetas y los grupos formados por el participante. La prueba representada no puede ser modificada.

La imagen 8.20 representa el ultimo estado guardado de un ordenamiento en particular, en este, el usuario creó una tarjeta con el concepto '*personal'* para agrupar 3 nombres, en la primera categoría agrupo los impuestos y por ultimo agrupo los concepto débilmente relacionados en la tercera categoría.

Imagen 8.19 Opciones de un estudio.

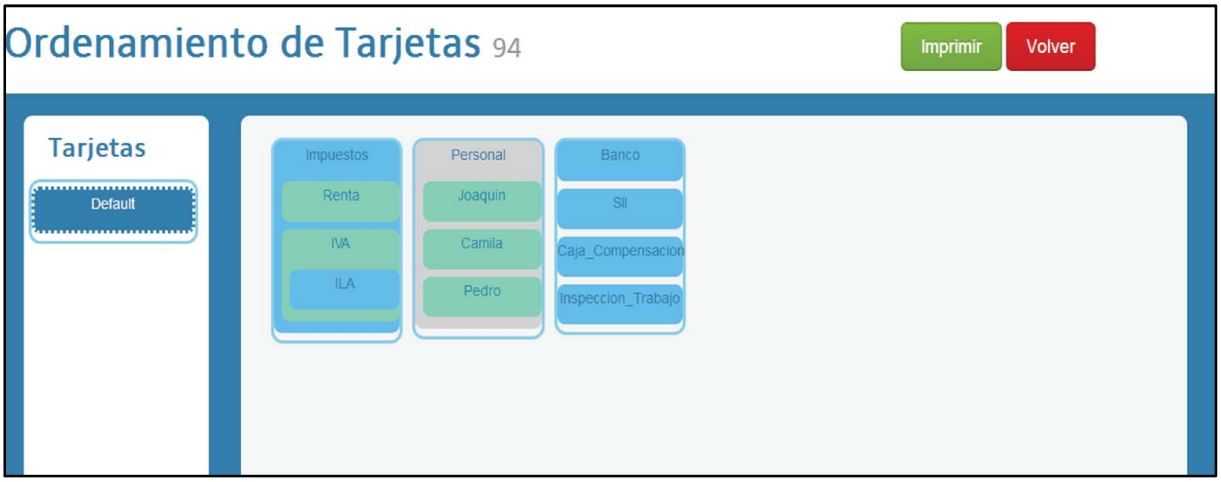


Imagen 8.20: Visualización del estado final de un ordenamiento.

Las funciones PHP para dibujar el estado final de una prueba se encuentran explicadas en detalle en el Anexo I.

En resumen, se recupera de la tabla categoría los registros que pertenecen al ordenamiento, estos registros representa al total de categorías y su posición ordinal respecto a las demás.

Por cada registro *categoría* se consulta la tabla *subcategoría*, cada registro subcategoría tiene los datos para identificar la tarjeta, el grupo al que pertenece y su posición respecto a los demás grupos dentro de la misma.

La función *crearSubCategoria()* traza el orden de los grupos en la categoría y crea las tarjetas. Cuando el código ha sido generado, el archivo *'terminado.js'* – incluido en la cabecera de la página – agrega la clase correspondiente a los subniveles para mantener el esquema de colores; *'terminado.js'* utiliza las funciones descritas en el punto 5.4.5 Implementación del coloreado de subgrupos.

8.5.4.2 Implementación de impresión de resultados.

En el punto anterior se expuso como se implementó de la visualización del resultado de una prueba con sus grupos y tarjetas. El estado final puede ser impreso desde la vista presionando el botón *'Imprimir'* (ver imagen 5.22) que ejecuta *window.print()*, función JavaScript que imprime el contenido de la ventana actual utilizando el gestor por defecto del navegador.

CSS Media Type es una característica de CSS que especifica como un documento es presentado en diferentes medios. Ciertas propiedades son diseñadas para medios específicos, sin embargo, hojas de estilo para diferentes medios pueden compartir una propiedad, pero poseen diferentes valores para esa propiedad [31].

A la vista de prueba terminada (punto anterior) se agregan dos hojas de estilos diferentes: *ordenamiento.css* que contiene los estilos a aplicar cuando el documento es visualizado por pantalla; y, *version_imprimir.css* con el formato para la hoja a imprimir.

```
<link rel="stylesheet" href="css/version_imprimir.css" media="print">
<link rel="stylesheet" href="css/ordenamiento.css" media="screen">
```

Con el atributo *media* se define el medio al que aplicar cada estilo: *screen* para la salida por pantalla y *print* para la salida por impresora. En *version_imprimir*.css por medio de el atributo *size* de la regla *@page* se establece la orientación horizontal de la página generada para imprimir. El gestor de impresiones configurará por defecto esta orientación.

La hoja de impresión debe mantener la estructura del ordenamiento. Las categorías, subcategorías y tarjeta usuario se deben diferenciar de forma clara similar a la vista reconstruida.

La imagen 8.22 muestra por pantalla el estado final de un ordenamiento, al presionar el botón *Imprimir* se genera la hoja de impresión utilizando los estilos definidos para este medio. La hoja para imprimir se puede ver en la imagen 8.21.

En la hoja de impresión las tarjetas se alinean a la izquierda en una columna de borde celeste que representa la categoría y los subniveles aparecen con sangría. Cuando un subnivel aumenta en profundidad la sangría también lo hace.

Finalmente, el concepto de la tarjeta creada por el usuario se destaca en color rojo mientras que las creadas por el evaluador se imprimen en color negro.

Imagen 8.21: Ejemplo de formato de impresión de un ordenamiento.

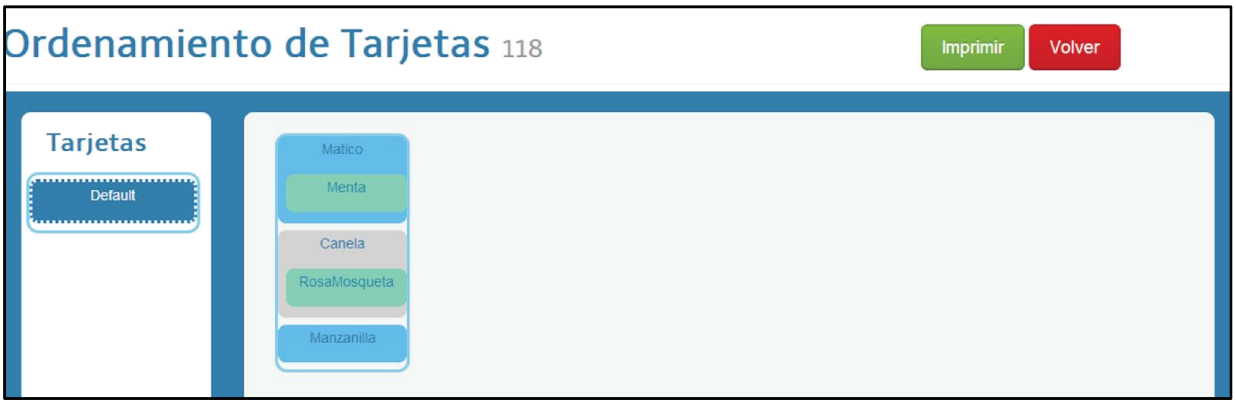


Imagen 8.22: Visualización de un ordenamiento finalizado.

8.5.5 Implementación de exportar resultados.

Para la síntesis de los resultados se utiliza el formato descrito en [1] que propone una metodología para analizar los datos usando un conjunto de hojas de datos semi-automatizadas.

El evaluador solicita la descarga del archivo presionando el botón '*Exportar Resultados*' de la barra de acciones de un estudio (ver imagen 8.19) que mediante AJAX solicita al servidor la descarga del archivo.

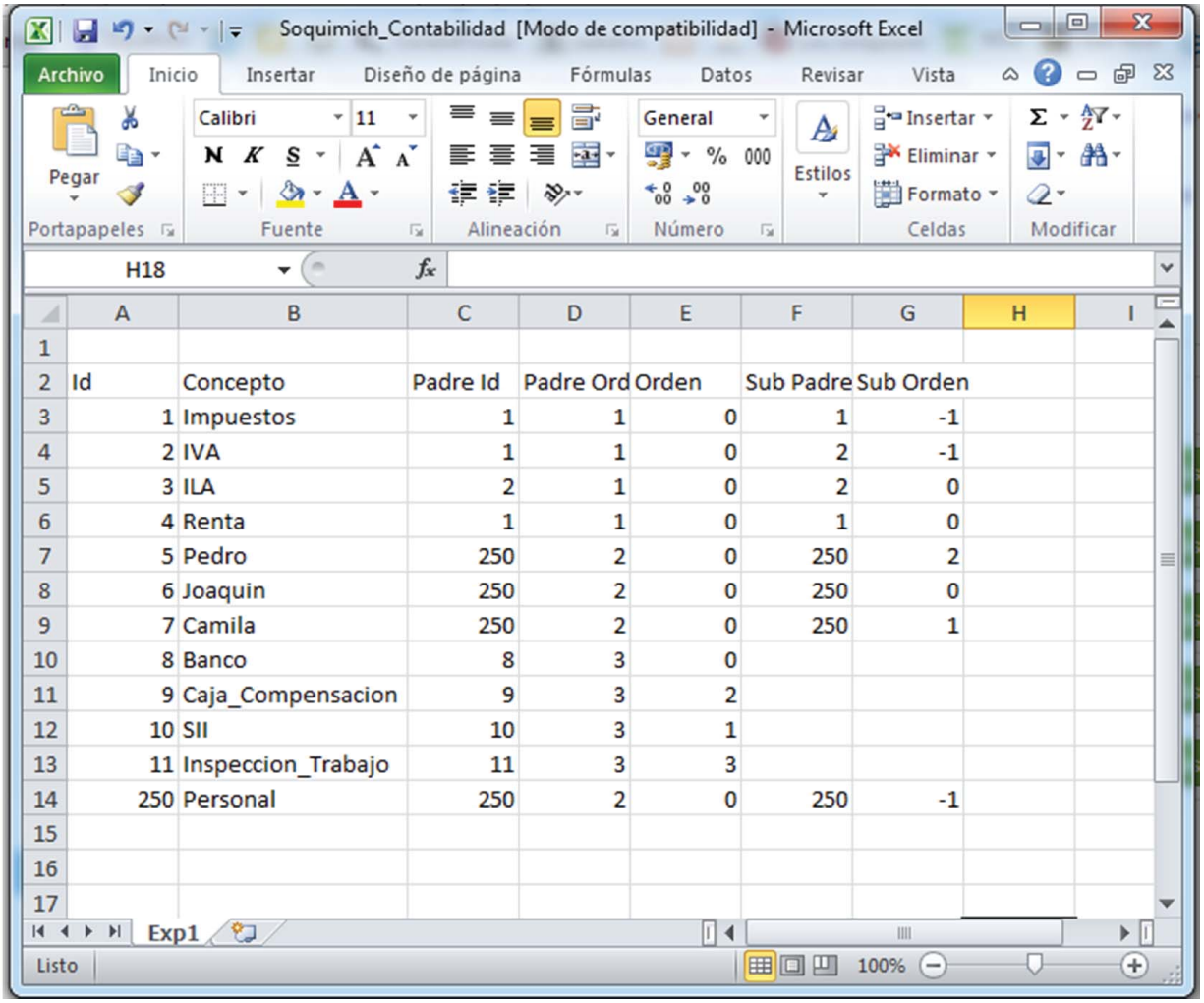
Empleando la biblioteca PHPExcel [32] se genera un objeto que representa las hojas de cálculo. Luego de cargar los datos al objeto, se solicita la generación y descarga del archivo. La librería PHPExcel provee de una "fábrica" de archivos con la clase *PHPExcel_IOFactory* que permite crear archivos en varios formatos, en este caso se utiliza el formato Excel5 de Microsoft Office.

La función que ejecuta el servidor PHP para generar y descargar el archivo se muestra a continuación. Recibe un objeto PHPExcel y dos cadenas de texto, crea el archivo, lo nombra y establece los parámetros de trasferencia de datos y gatilla la descarga.

```
function descargar($xls, $nombreProyecto, $nombreEstudio) {
     //Define el tipo de archivo a transferir
     header('Content-Type: application/vnd.ms-excel');
     //Nombra al archivo con el formato nombreProyecto_nombreEstudio
     header('Content-Disposition:
          attachment;filename="'.$nombreProyecto.'_'.$nombreEstudio.'"');
     header('Cache-Control: max-age=0');
     //Usa $xls para crear un objeto con el formato Excel5 (Excel 97-2003)
     $objWriter=PHPExcel_IOFactory::createWriter($xls,'Excel5');
     //Escribe en el flujo de salida del servidor.
     $objWriter->save('php://output');
     exit;
}
```

El archivo descargado está conformado por hojas de cálculo correspondientes a cada prueba de ordenamiento de tarjetas realizada en ese estudio. Cada prueba genera una hoja nombrada como *'Exp**n**'* dentro del libro, donde *n* representa el número ordinal de la prueba dentro del estudio.

Como resultado de esta operación, se obtiene un libro Excel con todas las pruebas realizadas en un estudio específico. Cada prueba esta resumida en una hoja diferente del libro utilizando la notación propuesta en [1]. La imagen 8.23 corresponde al resumen de los datos de la prueba de ordenamiento de tarjetas que se muestra en la imagen 8.22.



	A	B	C	D	E	F	G	H	I
1									
2	Id	Concepto	Padre Id	Padre Ord	Orden	Sub Padre	Sub Orden		
3	1	Impuestos	1	1	0	1	-1		
4	2	IVA	1	1	0	2	-1		
5	3	ILA	2	1	0	2	0		
6	4	Renta	1	1	0	1	0		
7	5	Pedro	250	2	0	250	2		
8	6	Joaquin	250	2	0	250	0		
9	7	Camila	250	2	0	250	1		
10	8	Banco	8	3	0				
11	9	Caja_Compensacion	9	3	2				
12	10	SII	10	3	1				
13	11	Inspeccion_Trabajo	11	3	3				
14	250	Personal	250	2	0	250	-1		
15									
16									
17									

Imagen 8.23 Ejemplo de una prueba de ordenamiento resumida en una hoja Excel.

En los párrafos siguientes se explica la lógica y funcionamiento de los algoritmos creados para cargar los datos de las pruebas realizadas al objeto PHPExcel.

Se necesitan dos conjuntos de datos: las tarjetas creadas en el estudio por el evaluador y el usuario, y los resultados de cada prueba que fueron almacenados en las tablas *'categoria'* y *'subcategoria'*. Para obtenerlos se consulta a la base de datos utilizando el del estudio en cuestión. A continuación se detalla el algoritmo de la lógica:

1. Agrega las librerías necesarias y entabla la conexión con la base de datos.
2. Si existe el estudio, crea dos vectores: uno con la lista de tarjetas del estudio y otro con la lista de pruebas (registros de la tabla ordenamiento). La lista de tarjetas incluye las creadas por defecto por el evaluador más todas las creadas por los usuarios durante el desarrollo de las pruebas de ese estudio.
3. Si hay datos, crea el objeto PHPExcel, fija los metadatos de este y establece la hoja activa del libro.
4. Escribe los datos por defecto de la hoja: la lista de tarjeta y los encabezados de las columnas. (Corresponden a los valores de la primera columna de la tabla 5.1, más información sobre el formato de la hoja de cálculo en [1]). Para el id de la tarjeta en la hoja de cálculo se utiliza el valor del campo *numero_tarjeta_estudio.*
5. Escribe los resultados de la prueba. Se obtienen los registros de la tabla *subcategoria* relacionados a una prueba de ordenamiento del estudio. Para cada registro de *tarjeta* se rellenan los campos según los datos guardados en la tabla *subcategoría.*
    a. **Padre id**: Tiene el valor del campo *numero_tarjeta_estudio* de la tarjeta accedida mediante el campo *fk_tarjeta_padre* del registro de *subcategoría.* Si no tiene padre, se coloca el valor de *numero_tarjeta_estudio* de la misma tarjeta.
    b. **Padre orden**: Tiene el valor del campo *numero_categoria* de la categoría que contiene la tarjeta.
    c. **Orden**: Calculado según la cantidad de registros de la categoría y la profundidad de anidación de la tarjeta o su padre.
    d. **Sub padre**: Corresponde al id de la tarjeta – en la hoja de cálculo – que contiene a la tarjeta actual, si la tarjeta no depende (no es subgrupo) de otra se deja en blanco. Si la tarjeta es generadora de un subgrupo se rellena con el id de la tarjeta.
    e. **Sub orden**: Si la tarjeta tiene subgrupos, se coloca un -1 en esta celda. Si la tarjeta es parte de un subgrupo tiene el valor del orden de la misma relativo al subgrupo que pertenece (orden en subgrupo). Si la tarjeta no es parte de un subgrupo la celda no tiene valor.
6. Crea una nueva hoja y la establece como hoja activa.
7. Repetir puntos 4, 5, 6 mientras existan pruebas.
8. Nombrar y descargar el archivo generado.

La implementación de esta funcionalidad conlleva cierta complejidad. En el Anexo J se muestran las funciones con una mayor carga lógica, estas se encuentran comentadas en su totalidad para facilitar su comprensión.

9 Conclusiones

9.1 Conclusiones personales

La adaptabilidad al cambio es un punto a tener en cuenta durante todo el desarrollo del proyecto, no obstante, el equipo de trabajo debe ser previsor y no reaccionario.

Una planificación acertada es la base para evitar retrasos en las entregas y desacuerdos que conllevan consecuencias negativas en los recursos destinados para el proyecto. Requerimientos claros limitan la variabilidad del sistema en construcción.

Realizar este proyecto desde sus inicios significó ser parte de todo el ciclo de vida del software, desde el levantamiento de requerimientos hasta las pruebas e implementación. Que todo esté a cargo de una sola persona no es recomendable – y más aún si no existe experiencia previa en algún área del proceso –, muchas veces se cae en la desmotivación al sentir que es una montaña demasiado grande para escalarla sin equipo.

Una parte importante para asegurar que un sistema sea adaptable y mantenible está en su código fuente. La metodología XP propone la práctica de *refactoring* para realizar mejoras continuas en el código que permitan una mejor comprensión pero sin agregar funcionalidades. El *refactoring* es más bien una mejora estética. La ventaja de un código limpio y entendible es incalculable en tiempo y dinero.

Utilizar un framework ordena la producción de código y obliga a la modularización de los componentes. Los costos de usar un framework pueden ser elevados al principio debido al tiempo de aprendizaje necesario para utilizarlo, pero los beneficios están muy por encima de los costos y el sistema será más robusto que si fuese construido sin estos. Las reglas que impone un framework ayudan a la mantenibilidad, escalabilidad y seguridad del sistema producido.

Establecer modelos propios – para nombrar archivos, carpetas, variables, etc. – también es una buena práctica. Los comentarios en el código fuente es una de las mejores formas de documentación, estos deben ser precisos y concisos para comprender fácilmente la lógica tras las líneas.

9.2 Conclusiones técnicas

Como resultado del desarrollo de este proyecto se obtuvo un prototipo funcional de sistema para gestionar las pruebas de usabilidad y ayudar a los evaluadores a obtener información concisa y precisa de los resultados de estas pruebas. Las funcionalidades del sistema fueron validadas mediante pruebas con usuarios.

Durante las pruebas se identificaron problemas en la metodología de síntesis de datos, específicamente en la jerarquía de las tarjetas cuando están agrupadas en un nivel de profundidad igual o superior a 3.

El problema consiste en que si una tarjeta es parte de una subcategoría de nivel de anidación 3, la tarjeta generadora de la subcategoría (tarjeta padre) pierde la posición que tiene dentro de su subgrupo ya que para identificar que esta tarjeta genera una subcategoría, la columna *sub order* de la hoja de cálculo se completa con valor *'-1'*. Por ejemplo usando la hoja de cálculo de la imagen 8.23 (que sintetiza la jerarquía de la prueba de la imagen 8.22) no se puede conocer la posición de la tarjeta IVA dentro de su subcategoría.

A demás de lo anterior, se considera que los nombres de las columnas en la hoja de cálculo son poco claros y no se pueden relacionar con la jerarquía que tiene la tarjeta en el orden de los grupos.

Para solucionar esto se propone un cambio en nomenclatura debido a la insuficiencia para representar varios niveles de anidación. La hoja de síntesis de resultados estará compuesta por 9 columnas, como se muestra en la tabla 9.1:

1. **Id:** Identificador de la tarjeta en el estudio.
2. **Concepto:** Nombre de la tarjeta.
3. **Profundidad:** Nivel de profundidad de anidación de la tarjeta en la categoría. Inicia en el nivel 0. Un nivel de profundidad par significa que la tarjeta tiene color azul, un nivel impar significa un color verde.
4. **Es padre:** Identifica si la tarjeta es generadora de una subcategoría, es decir, si tiene tarjetas que dependen de ella.
5. **Tarjeta padre:** Referencia a la tarjeta generadora del subgrupo por su identificador dentro del estudio. Cuando la tarjeta está en el nivel de profundidad 0, Tarjeta Padre tiene el valor del id de sí misma indicando que ella misma es su padre. Cuando está a un nivel superior tiene el identificador de la tarjeta que genera el grupo al que pertenece la tarjeta actual (por ejemplo la tarjeta *melisa* en la tabla 9.1).
6. **Categoría:** Numero de la categoría a la que pertenece la tarjeta. Comienza en 1 desde izquierda a derecha en la prueba de ordenamiento.
7. **Posición:** Posición de una tarjeta dentro de su categoría. Comienza en 0 desde arriba hacia abajo. Solo cuentan las tarjetas en nivel 0 para sumar posiciones. Todas las tarjetas que pertenezcan a un subgrupo tendrán el valor de la posición de la tarjeta padre de nivel 0 de ese subgrupo.
8. **Padre subgrupo:** Valido solo para tarjetas que están a un nivel de profundidad mayor o igual a 1. Identifica al padre del subgrupo al que pertenece la tarjeta. Adicionalmente, si la tarjeta actual es padre de un subgrupo tiene su misma id en esta columna.
9. **Posición subnivel:** Anota la posición que tiene esta tarjeta respecto al subgrupo al que pertenece. Solo valido para tarjetas de nivel 1 o superior. si no pertenece a ningún subgrupo se deja en blanco (por ejemplo *RosaMosqueta* en tabla 9.1). Comienza en 0.

La tabla a continuación corresponde a la síntesis de los datos sobre la jerarquía de los grupos de la imagen 9.1.

Las tarjetas *Manzanilla* y *Anis* tienen el mismo valor en casi todas sus columnas, se diferencian porque la tarjeta padre de estas son diferentes, de *Anis* el padre es *Manzanilla* y de

esta el padre es *Matico*, para resolver cualquier duda existe la columna profundidad y la columna es padre que ayuda para diferenciar las tarjetas que tienen un subgrupo a cuestas.

Id	Concepto	Profundidad anidación	Es padre	Tarjeta Padre	Categoria	Posicion	Padre Subgrupo	Posicion Subnivel
1	Matico	0	Si	1	1	1	1	
2	Menta	1	Si	1	1	1	2	2
3	Manzanill	1	Si	1	1	1	3	1
4	RosaMosqueta	0	Si	4	2	1	4	
5	Bailahuen	1	No	4	2	1	4	1
6	Boldo	0	No	6	2	0		
7	Cedron	1	No	1	1	1	1	0
8	Anis	2	No	3	1	1	3	1
9	Llanten	0	No	9	1	0		
10	Ruda	2	No	3	1	1	3	0
11	Melisa	2	No	2	1	1	2	0
12	Deutrofila	0	No	12	1	2		
267	Potosi	1	No	4	2	1	4	0

Tabla 9.1 Nueva notación propuesta para la síntesis de resultados

Imagen 9.1 Grupos para representar con la nueva notación.

10 Bibliografía

[1]. **Inostroza, Rodolfo.** *Card Sorting: Analyzing hierarchy of items in a menu.* Valparaíso : Pontificia Universidad Catolica de Valparaíso, 2012.

[2]. **Hewett, et al.** ACM SIGCHI Curricula for Human-Computer Interaction. *CHAPTER 2: Human-Computer Interaction.* [En línea] 29 de 07 de 2009. http://old.sigchi.org/cdg/cdg2.html#2_1.

[3]. **ACM SIGCHI.** ACM SIGCHI Curricula for Human-Computer Interaction. *CHAPTER 2: Human-Computer Interaction.* [En línea] http://old.sigchi.org/cdg/cdg2.html#U3.

[4]. **UsabilityNet.** UsabilityNet: International Standards. *International standards for HCI and usability.* [En línea] http://www.usabilitynet.org/tools/r_international.htm#9241-1x.

[5]. **Ferré, Xavier.** Universidad Politécnica de Madrid. *Principios Básicos de Usabilidad para Ingenieros Software.* [En línea] http://lucio.ls.fi.upm.es/miembros/xavier/papers/usabilidad.pdf.

[6]. **UsabilityNet.** UsabilityNet. *Card sorting.* [En línea] http://www.usabilitynet.org/tools/cardsorting.htm.

[7]. **Digital Communications Division of U.S. Department of Health and Human Services.** Usability.gov. *Card Sorting.* [En línea] http://www.usability.gov/how-to-and-tools/methods/card-sorting.html.

[8]. **Jablonski, et al.** Guide to Web Application and Platform Architectures. Berlín : Springer.

[9]. **Wikipedia Org.** Touchscreen. [En línea] https://en.wikipedia.org/wiki/Touchscreen.

[10]. —. Indium tin oxide. [En línea] https://en.wikipedia.org/wiki/Indium_tin_oxide.

[11]. **Intel Corporation.** ARK. *Intel Core i5*. [En línea] http://ark.intel.com/products/84984/Intel-Core-i5-5250U-Processor-3M-Cache-up-2_70-GHz.

[12]. —. ARK. *Intel Core i3.* [En línea] http://ark.intel.com/products/46472/Intel-Core-i3-530-Processor-4M-Cache-2_93-GHz.

[13]. **Furfero, David.** jQuery UI Touch Punch. *Touch Event Support for jQuery UI.* [En línea] 2011. http://touchpunch.furf.com.

[14]. **Free Software Foundation, Inc.** GNU General Public License. [En línea] http://www.gnu.org/licenses/gpl-3.0.en.html.

[15]. **Sommerville, Ian.** *Software Engineering 07 Edition (Traducción).* Madrid : Pearson Educación S.A., 2005. 84-7829-074-5.

[16]. **Instituto Politécnico Nacional, Mexico.** Programación estructurada. [En línea] http://www.sites.upiicsa.ipn.mx/polilibros/portal/polilibros/p_terminados/PolilibroFC/Unidad_III/Unidad%20III_8.htm.

[17]. **World Wide Web Consortium.** HTML 4.01 Specification. *HTML 4.01 Specification.* [En línea] 24 de 12 de 1999. http://www.w3.org/TR/html401/.

[18]. —. HTML5. *A vocabulary and associated APIs for HTML and XHTML.* [En línea] 28 de 10 de 2014. http://www.w3.org/TR/2014/REC-html5-20141028/.

[19]. Cascading Style Sheets (CSS) Snapshot 2010. *3. Cascading Style Sheets Definition.* [En línea] 12 de 05 de 2011. http://www.w3.org/TR/CSS/#css3.

[20]. **Ecma International.** Standard ECMA-262. *ECMAScript Language Specification.* [En línea] http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf.

[21]. —. Standard ECMA-404. *The JSON Data Interchange Format.* [En línea] Octubre de 2013. http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf.

[22]. **Wikipedia.** JSON - Wikipedia, la enciclopedia libre. *JSON.* [En línea] http://es.wikipedia.org/wiki/JSON. JSON.

[23]. **Joskowicz, José.** Reglas y Prácticas en eXtreme Programming. *Reglas y Prácticas en eXtreme Programming.* [En línea] 10 de 02 de 2008. http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf.

[24]. **The jQuery Foundation.** jQuery User Interface. *API Documentation - Interactions.* [En línea] http://api.jqueryui.com/category/interactions/.

[25]. —. jQuery UI API Documentation. *Sortable Widget.* [En línea] http://api.jqueryui.com/sortable/.

[26]. **Sarfatti, Manuele J.** mjsarfatti/nestedSortable · Github. *nestedSortable jQuery plugin.* [En línea] https://github.com/mjsarfatti/nestedSortable/blob/master/README.md.

[27]. **Teland, carlosgctes, lfottaviano, teoli.** Mozilla Developer Network. *JSON.stringify().* [En línea] https://developer.mozilla.org/docs/Web/JavaScript/Ref/Objetos_globales/JSON/stringify.

[28]. **The PHP Group.** PHP: json_decode. *Manual.* [En línea] http://php.net/manual/en/function.json-decode.php.

[29]. **World Wide Web Consortium.** Media Types. [En línea] http://www.w3.org/TR/CSS2/media.html.

[30]. **The PhpExcel Team.** PhpExcel. [En línea] https://phpexcel.codeplex.com/.

[31]. **World Wide Web Consortium.** Assigning property values, Cascading, and Inheritance. *!important rules.* [En línea] 07 de 06 de 2011. http://www.w3.org/TR/CSS2/cascade.html#important-rules.

[32]. **The jQuery Foundation.** jQuery.ajax(). *jQuery API Documentation.* [En línea] http://api.jquery.com/jQuery.ajax/.

Anexos

Anexo A: Como aplicar una función a una colección de datos con jQuery

Luego de incluir la librería en la página, para usar jQuery toda sentencia comenzará con *jQuery* o su equivalente abreviado *$*. jQuery se vale de selectores CSS para elegir a que elementos se aplicará la funcion.

Un selector sirve para elegir uno o más elementos del DOM que cumplen con cierta característica, en la especificación de CSS 2.1 se define un selector como: "*las reglas de coincidencia de patrones determinan cuales reglas de estilo se aplican a los elementos de la estructura del documento. Estos patrones, llamados selectores, pueden ir desde los nombre de elementos simples a patrones contextuales complejos. Si todas las las condiciones en el patron son verdaderas para cierto elemento, el selector equivale al elemento*".

jQuery adopta los selectores desde CSS y agrega selectores propios ofreciendo un completo set de herramientas para utilizar en la selección y filtrado de un conjunto de elementos del documento. A este conjunto de elementos se le conoce como **colección**.

Para aplicar las funciones existen dos formas, sin selectores y con selectores. A continuación hay dos sentencias equivalentes

```
/* Las dos sentencias son equivalentes, ambas aplican la función sobre un
conjunto de datos definidos por "selector" */
jQuery("selector").funcion();
$("selector").funcion();


/* Funciones sin selector */
jQuery.funcion();
$.funcion();
```

jQuery es potente, flexible y versátil. Existen distintas formas de obtener el mismo resultado. Por ejemplo, para obtener todos los elementos que tipo checkbox que están seleccionados dentro de un formulario utilizamos la siguiente funcion:

```
$("#formulario>input").filter(":checkbox").filter(":checked");
```

La siguiente línea es equivalente a la anterior, solo que usa otras funciones para obtener el resultado:

```
$("#formulario").children("[type=checkbox]:checked");
```

Ambas lineas funcionan bajo la misma lógica: selecciona los nodos *input* hijos del elemento con *id=formulario*, a esa colección se aplica un filtro para seleccionar todos los nodos del tipo *checkbox* y luego vuelve a filtrar ese resultado seleccionando solo los que tienen el atributo *checked*.

Anexo B: Uso y configuración de nestedSortable

A continuación un extracto de 'ordenamiento.js'[4], el código ejecuta la función nestedSortable con ciertas opciones en una colección de datos para que estos puedan ser ordenados:

```
function ordenarTarjetas(){
     $(".categoria").nestedSortable({
          //Define con qué tipo de elementos se puede conectar
          connectWith: ".span2:not(.tarjeta-default)",
          //Define el tipo de lista, puede ser ul u ol
          listType: "ul",
          //Especifica que elemento tendrá la propiedad sortable
          items: "li:not(.tarjeta-default)",
          //El elemento será clonado y su clon arrastrado
          helper:  'clone',
          //Transparencia que tendrá el elemento cuando flota
          opacity: .6,
          //Tiempo que demora la animación de regreso del elemento
          revert: 250,
          //Distancia en pixeles hacia la izquierda o derecha el elemento
          //debe moverse para anidar
          tabSize: 15,
          //La tolerancia de solapamiento está dada por el puntero
          tolerance: 'pointer',
          //Elemento que contiene el concepto
          toleranceElement: '> div:not(.tarjeta-default)',
          //Indica el elemento que gatilla el evento de arrastrar
          handle: 'div',
          //Estilo aplicado al elemento de destino para identificar donde
          //será añadido el elemento que está siendo arrastrado.
          placeholder: 'placeholder',
          //Define el nivel máximo de anidacion. 0 = niveles ilimitados.
          maxLevels: opciones.MAX_NIVELES_ANIDACION(),
          //Define los limites por donde puede ser arrastrado un elemento.
          containment: "#mesaTrabajo"
       }).disableSelection();
};
```

[4] Archivo que contiene parte del código JavaScript utilizado en una prueba de ordenamiento

Anexo C: Crear una tarjeta nueva durante una prueba.

La función javascript *agregarTarjeta()* es gatillada al presionar el botón "*Agregar Tarjeta!"* de la lista de tarjetas, genera las peticiones al servidor y ejecuta cierta lógica basado en la respuesta recibida.

En *agregarTarjeta()* se utiliza jQuery para hacer 2 peticiones AJAX hacia el servidor PHP: una mediante *$.post()* agrega la tarjeta creada por el usuario a la base de datos, la otra utiliza *$.ajax()* para contar las tarjetas presentes en un estudio.

El servidor PHP atiende las peticiones Ajax y ejecuta la lógica contenida en los archivos accedidos por la URL. Los archivos pueden "retornar" un valor usando el constructor de lenguaje *echo* que es captado por la función de devolución de llamada (*callback function*) de los métodos *$.post()* o *$.ajax()* según corresponda. A continuación se muestra un fragmento del código javascript para agregar una tarjeta durante una prueba de ordenamiento.

```
function agregarTarjeta(idEstudio){
     //Verifica que el valor sea valido
     var nombre=$("#nuevoNombre").val();
     var cantidadTarjetasEstudio = 0;
     //Calcula el numero_tarjeta_estudio para la nueva tarjeta
     //Ejecuta la lógica contenida en la url mediante ajax
     $.ajax({
          async: false,
          url:'lib/contarTarjetasEstudio.php',
          type: 'POST',
          data:{
               'idEstudio': idEstudio //fk para la tabla 'ordenamiento'
               },
          //Tuvo éxito la petición, 'resp' contiene el número de tarjetas
          success:function(resp){
                 if(!isNaN(resp) && resp.length>0){
                    cantidadTarjetasEstudio=parseInt(resp) +
                            parseInt(opciones.MAX_TARJETAS_POR_ESTUDIO());
                 }else{
                    //Control de errores
                    console.debug(resp);
                 }
          }
     }); //Fin AJAX
     //Si el ordenamiento no ha sido guardado, se crea uno para asociar la
     //tarjeta creada por el usuario.
```

```
     if(!idOrdenamiento && cantidadTarjetasEstudio){
          crearOrdenamiento(idEstudio);
     }

//Crea el registro de la nueva tarjeta y la agrega al DOM para ordenarla
     $.post('./lib/crearTarjetaUsuario.php',{
            "idEstudio": idEstudio,
            "nombre": nombre,
            "numeroTarjetasEstudio": cantidadTarjetasEstudio,
            "idOrdenamiento": idOrdenamiento
          },
          //Retorna el id de la tarjeta recién creada
          function(data){
            //Transforma la respuesta a entero
            var resp = parseInt(data);
            if(!isNaN(resp)){
              var li = crearTarjetaLi(resp,nombre);
             //Revisa si existen elementos y agrega nueva tarjeta al DOM
              var $elementosTabla = $("#listaTarjetas").find("li")
                                          .not(".tarjeta-default");
              if($elementosTabla.length > 0){
                   //La tarjeta es colocada primera en la lista
                   $elementosTabla.first().before(li);
              }else{
                   //Agrego al 'ul' porque no hay 'li'
                   $("#listaTarjetas").html(li);
              }
            }else{ //No se pudo transformar, hubo un error.
              alert(data);
            }
          }//Fin function
        );//Fin POST
}
```

(Continua en la siguiente página)

```
//Crea el código HTML de una nueva tarjeta usando los parámetros.
function crearTarjetaLi(idTarjeta, nombre){
    var clase ='tarjeta tarjeta-usuario';
    var idLi = nombre+'_'+idTarjeta;
    var tarjeta = '<li class="'+clase+'" id="'+idLi+'">'
                        +'<div>'+nombre+'</div></li>';
    return tarjeta;
};
```

Anexo D: Archivos involucrados en las peticiones Ajax de creación de tarjetas.

Archivo *crearTarjetaUsuario.php*, entabla la conexión con la base de datos, realiza la consulta utilizando los parámetros enviados solicitado vía POST y retorna el id de la tarjeta creada. En caso de error redirige a la página de errores.

```
if ($_POST) {
    //Establece conexion
    include_once '../lib/conexion/conexion.class.php';
    $db = new conexion();
    if( !$db->conectar()){
          error('Imposible Conectar con la base de datos');
     }

     //Obtengo los datos transferidos mediante POST
     $idEstudio = $_POST['idEstudio'];
     $nombre = $_POST['nombre'];
     $idOrdenamientoActual= $_POST['idOrdenamiento'];
     $numeroTarjetasEstudio = $_POST['numeroTarjetasEstudio'];

/*   ====      Construccion de la consulta ====== */
$query = 'insert into tarjeta '.
(fk_id_estudio,numero_tarjeta_estudio,nombre,usuario,fk_id_ordenamiento)'.'
values ('.$idEstudio.', '.$numeroTarjetasEstudio.',"'.$nombre
              .'",true,'.$idOrdenamientoActual.')';
/*   =======       Fin consulta  ===============  */

     //Realiza la consulta utilizando la API de ez_sql_mysql
     $db->db->query($query);
     //No hubo errores. Devuelvo el id de la fila afectada
     if (!$db->db->last_error && $db->db->rows_affected) {
              echo $db->db->insert_id;
     } else {
          error('No se insertó la tarjeta');
     }
    $db->desconectar();
}
```

Archivo *contarTarjetasEstudio.php* retorna la cantidad de tarjetas que tiene un estudio identificado por el *idEstudio* enviado mediante POST.

```
if($_POST){
     require_once '../lib/conexion/conexion.class.php';
     $db = new conexion();
     if($db->conectar()){
          $idEstudio = $_POST['idEstudio'];
          $query='select count(*) from tarjeta where'
                    .' fk_id_estudio='.$idEstudio;
          echo $db->db->get_var($query);
     }
}
```

Anexo E: Objeto ez_sql, atributos y uso.

La clase proporciona los métodos para realizar las consultas a la base de datos y en sus atributos mantiene datos de control tales como registro de errores, de consultas, valores útiles en el depurado y un conjunto de opciones. Se debe cargar el controlador ezSQL específico para la base de datos utilizada y el núcleo de la librería para crear el objeto:

```
//Nucleo
include_once "ez_sql_core.php";
//Incluye el componente especifico en este caso mySQL.
include_once "ez_sql_mysql.php";
```

Ahora se crear el objeto utilizando el constructor del componente incluido y con los datos de la conexión como parámetros.

```
$db = new ezSQL_mysql('db_user','db_password','db_name','db_host');
```

El objeto es creado y usando el método *quick_connect()* inicia la conexión al servidor y selecciona la base de datos a utilizar. La totalidad de funciones y variables están disponibles en la documentación del proyecto, a continuación se detallan las utilizadas en este proyecto:

```
/* == Funciones == */
$db->get_results() : Obtiene multiples registros de una base de datos.
$db->get_row()     : Coge un registro de la base de datos.
$db->get_col()     : Coge toda la columna de campos de una base de datos.
$db->get_var()     : Saca un campo de un registro de una base de datos.
$db->query()       : Envia una consulta a la base de datos, si tiene
                     respuesta, la retorna y actualiza el estado del objeto.
$db->debug()       : Imprime la ultima consulta sql y los resultados de
                     esta, si tuvo.
$db->vardump()     : Imprime el contenido y estructura de cualquier
                     variable.
/* == Variables == */
$db->num_rows      : Número de registros retornados en la última consulta
                     exitosa.
$db->insert_id     : ID generado en el campo AUTO_INCREMENT para el último
                     INSERT.
$db->rows_affected : Número de registros afectados por la ultima operacion
                     INSERT, UPDATE O DELETE
$db->last_error    : Guarda el ultimo error producido por una consulta.
```

La conexión se mantiene hasta que se desconecta con *$db->disconnect()* o hasta que el objeto se destruye. El objeto actualiza su estado solo cuando una consulta lo hace, por ejemplo, si la primera consulta genera un error, este se guarda en *last_error* y aunque las siguientes consultas no produzcan errores el objeto mantendrá el error de la primera consulta en dicho atributo.

Las consultas a la base de datos se realizaron con las siguientes funciones, todas reciben por parámetro un String *query* con la consulta. Ez_sql valida el String para evitar inyección de SQL.

`$db->query(query)` : Ejecutar consultas del tipo INSERT, DELETE y UPDATE, estas consultas podrían actualizar el valor de *'insert_id'* y *'rows_affected'*. Retorna un valor boleano según el éxito de la consulta.

`$db->get_var(query)` : Retorna el valor de un campo en un registro consultado usando SELECT.

`$db->get_row(query)` : Genera un objeto cuyos atributos son nombrados igual que los campos de la tabla y el valor de estos corresponde al valor del campo de ese registro. La consulta debe ser SELECT que retorne solo un registro.

`$db->get_results(query)` : Retorna un array de objetos - con el mismo formato que en *'get_row()'* - a partir de los registros devueltos por una consulta SELECT, si esta no arroja resultados retorna null.

El siguiente código ejemplifica el uso del API de ez_sql, se realiza una consulta desde PHP a la base de datos. Selecciona todos los campos de la tabla *'categoria* cuyos registros cumplan con la condición. La tabla tiene 3 campos: *id*, *numero_categoria* y *fk_id_ordenamiento*.

```
// Codigo php - Genera la consulta
$query = 'select * from categoria '
         .' where fk_id_ordenamiento > 119 AND fk_id_ordenamiento < 123';
//Ejecuta la consulta y guarda los resultados en $categoria
$categorias = $db->db->get_results($query);
//Muestro el resultado
$db->db->vardump($categorias);
```

La función *vardump()* imprime los objetos del array, su posición y sus datos:

```
Array
(
    [0] => stdClass Object
        (
            [id] => 201
            [numero_categoria] => 1
            [fk_id_ordenamiento] => 120
        )

    [1] => stdClass Object
        (
            [id] => 203
            [numero_categoria] => 1
            [fk_id_ordenamiento] => 122
        )

    [2] => stdClass Object
        (
            [id] => 204
            [numero_categoria] => 2
            [fk_id_ordenamiento] => 122
        )
)
```

Más información y ejemplos sobre el uso de ez_sql y más detalles sobre la clase, sus métodos y atributos en el archivo ez_sql_help.htm de la librería o en la página oficial.

Anexo F: Notación de un archivo JSON.

La estructura general está compuesta en pares de la forma *clave,valor*. Su sintaxis es bastante simple y permite representar todo tipo de variables.

Las reglas de notación solo son las siguientes:

i. Los objetos están definidos entre llaves '{}'.
ii. Los vectores se definen entre [].
iii. Los atributos se definen como atributo : valor.
iv. Los Strings se definen entre comillas dobles ("), los valores numéricos solo se escriben como tal.

A continuación se muestra un ejemplo de mensaje codificado en JSON, se definen dos claves: Responsable y Empleados. La primera es un objeto con 4 atributos con sus valores, la segunda es un vector con dos objetos y sus valores.

```
{
"responsable":
    {
    "Nombre": "Juan",
    "Edad": 28,
    "Aficiones": ["Música", "Cine", "Tenis"],
    "Residencia": "Madrid"
    },
"empleados":
    [
        {
        "Nombre": "Elena",
        "Edad": 26,
        "Aficiones": ["Música", "Cine"],
        "Residencia": "Madrid"
        },
        {
        "Nombre": "Luis",
        "Edad": 31,
        "Aficiones": ["Teatro", "Cine", "Fútbol"],
        "Residencia": "Madrid"
        }
    ]
}
```

Anexo G: Formato del objeto generado por la función ‘toArray’.

La biblioteca nestedSortable extiende el uso del método *toArray* de la función Sortable de jQuery UI para funcionar con multiniveles de anidación. Para aplicar la función a una colección se usa la siguiente sentencia:

```
$('Selector').nestedSortable('toArray', {startDepthCount: 0})
```

Con *‘startDepthCount’* se fija el valor por defecto del atributo *‘depth’* del objeto que se generado. A continuación se muestra el formato del objeto:

```
{
     'item_id': itemId,
     'parent_id': parentId,
     'depth': depth,
     'left': left,
     'right': right,
}
```

Los atributos utilizados por la lógica del sistema son:

item_id : Corresponde a lo que está después del guion bajo '_' en el atributo id del elemento *'<li>'*.

parent_id : Ídem a item_id, referencia al elemento que contiene al actual.

depth : Nivel de profundidad de la anidación, comienza en '*startDepthCount*', por defecto 0.

El array generado tiene por defecto en la posición inicial 0 un objeto con valores no válidos, desde la posición 1 se generan objetos validos que representan a los elementos de la lista. La profundidad (o nivel) de anidación de un elemento se guarda en *depth*, los objetos validos comienzan con profundidad 1 y con el atributo *parent_id* con valor *null* porque no están contenidos en otro elemento.

Cuando un elemento es parte de un subnivel el atributo *depth* aumenta proporcional al nivel de anidación en el que se encuentra y *parent_id* tiene el mismo valor que el atributo *item_id* del elemento padre.

Anexo H: Guardar una categoría de grupos de tarjeta.

Las siguientes funciones PHP manipulan los datos y gestionan la comunicación con la base de datos para guardar una categoría de una prueba de ordenamiento de tarjetas. El script responde a las solicitud AJAX realizadas por el método*$.ajax()* de jQuery.

Primero se conecta a la base de datos y recupera los parámetros de la consulta pasados por el método POST; la función *crearCategoria()* crea el registro de la categoría y envía el vector de objetos que representan a las tarjetas de esa categoría *$arrayTarjetasColumna* para su procesado y guardado en la base de datos; la tercera función, *guardarTarjetasSubCategoria()* inserta los nuevos registros en la tabla *subcategoria* usando los datos de las tarjetas recibidos desde la vista en el vector *arrayObjTarjetasColumna*.

```
//Código principal del script PHP que atiende la petición POST.
//Conecta con la BD y recupera los parámetros desde la variable POST
if($_POST){
     //Genera la conexión con la BD
     include_once '../lib/conexion/conexion.class.php';
     $db = new conexion();
     if( $db->conectar()){
        //Decodifica los datos desde el JSON a un array de objetos PHP.
        $arrayTarjetasColumna =
               json_decode($_POST['arrayObjTarjetasColumna']);
        $numCategoria = $_POST['numCategoria'];
        $idOrdenamientoActual = $_POST['idOrdenamiento'];
        //Crea las nuevas categorías con sus subcategorias
        echo  crearCategoria($numCategoria,  $idOrdenamientoActual,  $db->db,
                    $arrayTarjetasColumna);
     }
     $db->desconectar();
}else{
     echo 'Existe un problema en el paso de datos, intente nuevamente'
          .' más tarde';
}

//Crea la categoría y maneja las tarjetas presentes en subcategorias.
function crearCategoria($numCategoria, $idOrdenamiento, $db,
                          $arrayTarjetasColumna){
     //Crea el registro de la categoría
    $queryInsertarCategoria = 'insert into categoria'
```

```
                        .' (numero_categoria, fk_id_ordenamiento) values'
                        .' ('.$numCategoria.','.$idOrdenamiento.')';
     if($db->query($queryInsertarCategoria) && !$db->last_error){
    //Usa el id de la categoría recién creada
          $idCategoriaPadre = $db->insert_id;
          return guardarTarjetasSubCategoria($arrayTarjetasColumna,
                        $idCategoriaPadre, $db);
    }else{
        echo 'No se pudo guardar el categoría';
    }
}

//Recorre las tarjetas creando registros en la tabla subcategoría
function guardarTarjetasSubCategoria ($arrayTarjetas, $idCategoriaPadre,
                                            $db){
     //Un registro en subcategoría es una tarjeta dentro de una categoría
     //Tarjetas con igual depth y parent_id serán parte del mismo //subgrupo
     $primero= TRUE;
     $fueronGuardadas = TRUE;
     //Recorre las tarjetas creando registros en la tabla subcategoría
    foreach ($arrayTarjetas as $tarjeta) {
        if($primero){
            $primero=FALSE; //El primero no tiene datos válidos.
        }else{
            if(empty($tarjeta->parent_id)){
              //Si no tiene padre, el mismo hace de padre
              $parent_id = $tarjeta->item_id;
            }else{
                $parent_id = $tarjeta->parent_id;
            }
            //Genera la consulta.
            //'depth' -1 quita el objeto por defecto de  $arrayTarjetas.
            $queryInsertarSubCategoria= 'insert into subcategoria(nivel,'
              .' fk_id_categoria_padre, fk_id_tarjeta, fk_tarjeta_padre)'
              .' values('.($tarjeta->depth -1).','.$idCategoriaPadre.','
              .'.$tarjeta->item_id.','.$parent_id.')';
            //Revisa que la consulta se haya realizado y existan
            //resultados.
```

```
            $fueronGuardadas =
                queryExitosa($db->query($queryInsertarSubCategoria),
                                !$db->last_error) && $fueronGuardadas;
        }
    }//Fin FOREACH
    unset($tarjeta);
    return $fueronGuardadas;
}
```

Anexo I: Código para cargar el estado final de una prueba.

El código que plasma el estado final de una prueba se muestra a continuación, se omiten las funciones simples de validación y una porción del script PHP que genera la vista HTML para exponer solo el trozo con mayor carga lógica y funcional.

Las funciones generan dinámicamente la estructura de niveles de las categorías según los datos de los registro de la tabla *subcategoria* y de la tabla *tarjeta* que la componen.

```
//===== Inicio script para generar el código HTML de la página =======//
<?php
if($_POST){
     //Datos pasados vía POST usando el botón 'Estado Final'
     $idEstudio = $_POST['idEstudio'];
     $idOrdenamiento = $_POST['idOrdenamiento'];
}else{
//La función error redirige a una página donde se informa sobre el error.
     error('No tiene autorización para utilizar este recurso');
}
//Incluye y establece la conexión a la base de datos (Omitido)

//Obtiene una lista de objetos con las categorías que pertenecen al
//ordenamiento con id igual a $idOrdenamiento
$categorias=rescatarCategoriasEstudio($idOrdenamiento, $db);
}?>

<!-- Codigo de la cabecera HTML de la pagina -->
<div class="span9" id="agrupamientos">
     <div class="row-fluid" id="categorias">
<?php
//Recupera el grupo de registros de la tabla subcategoria
foreach ($categorias as $indice => $categoria){
     $querySubCategoria= 'select * from subcategoria where '
                              .' fk_id_categoria_padre='.$categoria->id;
     $registrosSubCategoria = $db->db->get_results($querySubCategoria);

//Crea la categoría y su lista de tarjetas en orden
     if(!empty($registrosSubCategoria) && $db->db->num_rows){
          echo '<ul class="span2 categoria" id="'.++$indice.'">';
          echo crearSubCategoria($registrosSubCategoria, $db);
```

```
          echo '</ul>';
     }
} //Fin foreach
?>
     </div>
</div>
```

La función *crearSubCategoria()* contiene la lógica para recrear la estructura de anidación que tenían las tarjetas al momento de guardar. Como resultado se obtiene el código de las tarjetas dentro de una categoría, su jerarquía de anidación y los colores correspondientes.

```
<?php
//============== Funciones de apoyo =================
//Retorna 'true' si la siguiente posición de la lista $registros tiene el
//atributo 'nivel' mayor al actual.
function sgteNivelMayor($registros, $indice){...}

//Ídem al anterior para el caso que 'nivel' sea menor.
function sgteNivelMenor($registros, $indice){...}

//Devuelve como valor positivo la diferencia de niveles entre dos
//registros consecutivos.
function calcularDiferenciaNiveles($registros, $indice){...}

//Escribe la clase al tag 'li' abierto en la función anterior además del
//<div> con el concepto de la tarjeta
function escribirTarjeta($idTarjeta, $db){
     $clase='tarjeta';
     if(esTarjetaCreadaUsuario($idTarjeta, $db)){
          $clase=$clase.' tarjeta-usuario';
     }
     echo 'class="'.$clase.'">';
          echo '<div>';
               //La función retorna el nombre de la tarjeta desde la BD
               echo rescatarNombreTarjeta($idTarjeta, $db);
          echo '</div>';
}
// =========== Función principal ====================
```

```
//Recibe los registros de subcategoria que pertenecen a categoria y la
//conexión a la BD para crear las tarjetas dentro de la categoría.
function crearSubCategoria($registrosSubCategoria, $db){
     //Indice para el array de registros
     $i=0;
     //Pila de cierre de tags de lista utilizada en los subniveles.
     $uls= array();
     do{
        //Inicia la escritura de la tarjeta, se deja abierto el tag para
        //agregar la clase correspondiente luego.
        echo '<li ';
        $sigMayor=false;
        //Usa el id de la tarjeta para obtener sus datos
        escribirTarjeta($registrosSubCategoria[$i]->fk_id_tarjeta, $db);
        //Se ha escrito hasta </div>, aun no se ha cerrado el tag li.

        //El registro siguiente corresponde a un subnivel del actual
        if(sgteNivelMayor($registrosSubCategoria, $i)){
            //Escribe el inicio de lista dentro de la tarjeta actual
            echo '<ul>';
            //Guarda en la pila los tags para cerrar este subgrupo y la
            //tarjeta que lo contiene.
            array_push($uls, '</ul></li>');
            $sigMayor=true;
        }//Fin if
        //Actual es la última tarjeta de su subgrupo.
        if(sgteNivelMenor($registrosSubCategoria, $i)){
            //Cierro la tarjeta actual
            echo '</li>';
            //¿Hay que cerrar más niveles?
            $diferencia =
                   calcularDiferenciaNiveles($registrosSubCategoria, $i);
            //Saca elementos de la pila para cerrar los niveles
            for($c=0;$c < $diferencia ; $c++){
                   echo array_pop($uls);
            }
            $sigMayor=true;
         }//Fin if
```

```
            //No entró a los if anteriores, cierra la tarjeta actual
            if(!$sigMayor){
                 echo '</li>';
            }
            ++$i;
        }while($i< sizeof($registrosSubCategoria));

    //¿La ultima tarjeta de la categoría estaba en el nivel 0 o superior?
    $nivelFinal = $registrosSubCategoria[--$i]->nivel;
    //Cerramos las que quedaron sin cerrar.
    if($nivelFinal){
        for($c=0; $c<$nivelFinal; $c++)
            echo '</ul></li>';
    }
}
?>
//Fin script de creación.
```

Anexo J: Funciones para generar el archivo Excel.

El siguiente script genera un archivo Excel con los datos de las pruebas de ordenamiento de tarjetas de un estudio en específico identificado por $idEstudio.

El archivo Excel es generado usando la biblioteca PHPExcel.

```
<?php
//===== Inicio script =======//
// Incluye las librerías (código omitido)
//Conecta a la base de datos (código omitido)
//Toma todas las pruebas realizadas para este estudio
$query='select * from ordenamiento where fk_id_estudio='.$idEstudio;
$idOrdenamientosDiferentes = $db->db->get_results($query);

if(!empty($idOrdenamientosDiferentes) && isset($db->db->num_rows)
                    && !empty($db->db->num_rows) && !$db->db->last_error){
       //Crea el objeto del libro excel
       $excel = new PHPExcel();
       //Nombra la página activa, por defecto es la primera
       $excel->getActiveSheet()->setTitle("Exp1");
       //Fija la página en la inicial.
       $excel->setActiveSheetIndex(0);
}else{
     error('No hay ordenamientos (Experimentos) en el estudio <br>');
}

//Escribe los resultados de cada prueba en una hoja diferente
$crearHoja=FALSE;
//Recorre todas las pruebas de un Estudio
foreach ($idOrdenamientosDiferentes as $idOrdenamiento) {
    if($crearHoja){
        //Crea una nueva hoja en el libro y la establece como la actual
        crearNuevaHoja($excel);
    }
    //Recuperamos los registros de categoría del ordenamiento
    $query='select * from categoria where'
                            .' fk_id_ordenamiento='.$idOrdenamiento->id;
    $categorias = $db->db->get_results($query);
```

```
    if(!empty($categorias)&& isset($db->db->num_rows)){
        if($db->db->num_rows > 0){
           //Función que escribe los datos por defecto en la hoja de
           //cálculo: Tarjetas del estudio y encabezados de columnas.
            escTitulosListaTarjetas($tarjetas, $excel->getActiveSheet());
            //Relleno la hoja con los valores de resultado de la prueba
            foreach ($categorias as $categoria) {
                    escResultadoExperimento($categoria,$tarjetas,
                                            $excel->getActiveSheet(), $db);
            }
            //Creo la siguiente hoja
            $crearHoja=TRUE;
        }else{
           error('No hay categorías en el ordenamiento');
        }
    }//Fin if
}//Fin foreach

//Genera el archivo Excel con el nombre del Proyecto y del Estudio.
$queryNombres= 'select p.nombre as nombreProyecto,'
                  . ' e.nombre as nombreEstudio'
                  . ' from estudio as e join proyecto as p'
                  . 'on p.id=e.fk_id_proyecto where e.id='.$idEstudio;
$resp=$db->db->get_results($queryNombres);
descargar($excel, $resp[0]->nombreProyecto, $resp[0]->nombreEstudio);
?>
```

La función **escResultadoExperimento** gestiona la escritura de los datos de las tarjetas presentes en una categoría y sus subgrupos en la hoja Excel velando por cumplir con el formato establecido.

La función escResultadoExperimento recibe por parámetro un objeto *$categoria* correspondiente a la categoría actual, un vector de objetos *tarjeta $arrayObjTarjetas* con las tarjetas del estudio, un objeto PHPExcel_Worksheet *$hoja* y la conexión a la base de datos *$db.*

Se conecta la base de datos, recupera los registros de *subcategoria* que representa al orden que tiene cada tarjeta en la categoría, calcula el nivel y busca el id de las tarjetas referenciadas por las claves foráneas. La id de cada tarjeta en el Excel es el campo *numero_tarjeta_estudio* de la tarjeta en cuestión

```
function escResultadoExperimento($categoria, $arrayObjTarjetas, $hoja,
$db){

     // Recupera los registros de la tabla subcategoria
     $query='select * from subcategoria where
                                fk_id_categoria_padre='.$categoria->id;
    $arrayRegistrosSubCategoria=$db->db->get_results($query);

    $pilaContSubNivel = array();
    $contSubNivel=0;
    $contNivel=0;

    foreach ($arrayRegistrosSubCategoria as
                            $indice => $registroSubCategoria) {
         //Busca el numero identificador de la tarjeta en el estudio
         //numero_tarjeta_estudio en base de datos, Id en el Libro Excel.
         $idTarjetaEnExcel= busNumTarjetaEstudio($arrayObjTarjetas,
                                $registroSubCategoria->fk_id_tarjeta);
         //Toma el número de fila de la tarjeta en la hoja excel.
         $filaTarjetaExcel = busFilaContenedoraId($idTarjetaEnExcel,
                                                                  $hoja);

         //Todas las tarjetas en revisión pertenecen a la misma categoria
         setCategoria($categoria->numero_categoria, $filaTarjetaExcel,
                                                                  $hoja);
         //¿El registro corresponde a una tarjeta de un subgrupo?
         if($registroSubCategoria->nivel > 0){
           //Busca el identificador de la tarjeta que es padre del
           //subgrupo al que pertenece la tarjeta actual.
           $idTarjetaPadreEnExcel= busNumTarjetaEstudio
             ($arrayObjTarjetas, $registroSubCategoria->fk_tarjeta_padre);
           setIdPadre($idTarjetaPadreEnExcel, $filaTarjetaExcel, $hoja);
           //Una tarjeta de un subnivel tiene el mismo nivel que la
           //tarjeta padre del subgrupo
           $filaTarjetaPadreEnExcel = busFilaContenedoraId
                                      ($idTarjetaPadreEnExcel, $hoja);
           $nivelTarjetaPadre = busNivelTarjetaPadre
                                   ($filaTarjetaPadreEnExcel, $hoja);
```

```
    setNivel($nivelTarjetaPadre, $filaTarjetaExcel, $hoja);
  }else{     /* La tarjeta no es parte de un subgrupo */
    //Guardamos sus propios datos
    setIdPadre($idTarjetaEnExcel, $filaTarjetaExcel, $hoja);
    setNivel($contNivel, $filaTarjetaExcel, $hoja);
   //Aumenta la ubicación vertical de la tarjeta en la categoría
    $contNivel++;
}

  //¿El siguiente está a un nivel de anidación mayor al actual?
if(sgteNivelMayor($arrayRegistrosSubCategoria, $indice)){
    //La tarjeta actual es un padre
    setPosicionSubnivel(-1, $filaTarjetaExcel, $hoja);
    //El padre del subgrupo será la tarjeta actual.
    setIdPadreSubgrupo($idTarjetaEnExcel,
                                        $filaTarjetaExcel, $hoja);
    //Puede estar iterando dentro de un subgrupo. Se guardar el
    //indicador de profundidad de anidación de la tarjeta dentro
    //de su subgrupo.
    array_push($pilaContSubNivel, $contSubNivel);
    $esPadre=TRUE;
    //Reinicamos el contador
    $contSubNivel = 0;

}else{     //¿Siguiente nivel menor al actual?
    if(sgteNivelMenor($arrayRegistrosSubCategoria, $indice)){
       //Fin de un subgrupo.
      //Utiliza la última posición de subnivel guardada.
        $contSubNivel = array_pop($pilaContSubNivel);
        $contSubNivel++;
      }
      $esPadre=FALSE;
}

//El registro actual es una tarjeta miembro de un subgrupo.
if($registroSubCategoria->nivel != 0 && !$esPadre){
    setPosicionSubnivel($contSubNivel, $filaTarjetaExcel, $hoja);
    $contSubNivel++;
```

```
            //Al ser parte de un subgrupo el id en el Excel de la tarjeta
            //padre ya está en la variable
            setIdPadreSubgrupo($idTarjetaPadreEnExcel, $filaTarjetaExcel,
                                                                $hoja);
        }
    }//Fin foreach.
}//Fin function
```