Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Centro de Ingeniería de Software y Sistemas

Sistema de evaluación para la nivelación de los cursos de la Coordinación de Extensión de la escuela de Idiomas Modernos de la UCV

Trabajo especial de Grado
presentado por los bachilleres
Saraí Adriana Díaz Rivas C.I: 18.761.047
Roger Jesús Rosales Pérez C.I: 18.738.592
para optar al título de
Licenciado en Computación

Tutores:
Profa. Jossie Zambrano
Prof. Sergio Rivas

Caracas, octubre de 2014

Acta

Quienes suscriben, miembros del jurado designado por el Consejo de la Escuela de Computación de la Facultad de Ciencias de la Universidad Central de Venezuela, para examinar el Trabajo Especial de Grado titulado: Sistema de evaluación para la nivelación de los cursos de la Coordinación de Extensión de la Escuela de Idiomas Modernos de la UCV, presentado por los bachilleres Saraí A. Díaz R., C.I.: 18.761.047 y Roger J. Rosales P., C.I: 18.738.592, a los fines de optar por el título de Licenciado en Computación, dejan constancia de lo siguiente: Leído el trabajo por cada uno de los miembros del jurado, se fijó el día 21 de octubre de 2014, a las 3:00 p.m., para que sus autores lo defendieran en forma pública en la Sala Planta Baja III de la Escuela de Computación, mediante una presentación oral de su contenido, luego de lo cual respondieron a las preguntas formuladas. Finalizada la defensa pública del Trabajo Especial de Grado, el jurado decidió aprobarlo. En fe de lo cual se levanta la presente Acta, en Caracas a los veintiún (21) días del mes de octubre del año dos mil catorce (2014), dejando constancia de que actuó como coordinador del jurado la Profesora Jossie Zambrano.

Profa. Jossie Zambrano (Tutor)

Prof. Eugenio Scalise (Jurado)

Prof. Sergio Rivas (Tutor)

Prof. Concettina Di Vasta (Jurado)

Agradecimientos

A Dios

Por colocar en nuestro camino todas las oportunidades que nos llevaron al logro de nuestra más anhelada meta.

A nuestras familias

Por el apoyo incondicional y las palabras de aliento en los momentos que más necesitábamos.

A nuestros amigos

Por hacer de la carrera más que una meta, una experiencia de vida llena de buenos momentos e innumerables anécdotas.

A nuestros profesores

Por formarnos e inspirarnos clase tras clase, son ejemplo de vocación y de compromiso.

A nuestros tutores

Por la paciencia y el tiempo invertido en nosotros. Eternamente agradecidos por su guía en el desarrollo del trabajo más trascendental de nuestras vidas.

A la UCV

Por ser más que una institución, un lugar de refugio e inspiración, testigo y cómplice de nuestro crecimiento personal y profesional. Orgullosos siempre de ser ucevistas.

Resumen

La Coordinación de Extensión de la Escuela de Idiomas Modernos de la Facultad de Humanidades y Educación de la Universidad Central de Venezuela imparte cursos de idiomas al público en general. Las personas interesadas en ingresar a estos cursos, pueden inscribirse en el primer nivel (básico I) o presentar una *prueba de* nivelación. Esta prueba evalúa el conocimiento previo que poseen del idioma, para así garantizar su ingreso a un nivel acorde a sus destrezas. Este proceso de nivelación es llevado manualmente, generando un alto costo en tiempo y recursos, razón por la que surge este Trabajo Especial de Grado con el objetivo general de desarrollar una *Aplicación Web* para automatizar la *prueba de nivelación* de los cursos de la Coordinación de Extensión de la escuela Idiomas Modernos de la *UCV*, para lograrlo, se instancia el *método de desarrollo de software* XP (eXtreme Programming), el cual se centra en la simplicidad y la comunicación fluida entre la Coordinación de Extensión y el equipo de desarrollo.

Palabras clave: Prueba de nivelación, aplicación web, método XP, módulos de gestión, modelos de prueba de nivelación.

Contactos

Saraí Díaz
sarai.diazr@gmail.com
Roger Rosales
roger.rosales589@gmail.com
Jossie Zambrano
Escuela de Computación
jossie.zambrano@gmail.com
Sergio Rivas
Escuela de Computación
sergiorivas@gmail.com

Índice

Índice de Imágenes

Índice de Tablas

Introducción

El lenguaje es una facultad humana que permite a las personas comunicarse y relacionarse al poder comprender y expresar pensamientos, ideas o emociones. Éste es la herramienta básica de toda sociedad para el intercambio de información y difusión del conocimiento.

Actualmente, el desarrollo económico y social de las sociedades ha llevado a un proceso constante de interacción e integración entre naciones, es por ello que el aprendizaje de otras lenguas se hace indispensable para poder desenvolverse en un mundo tan globalizado.

Dado que el conocimiento de idiomas es fundamental para el desarrollo personal y profesional, existen varias organizaciones dedicadas a la enseñanza de estos. Una de estas organizaciones es la Coordinación de Extensión de la Escuela de Idiomas Modernos de la Facultad de Humanidades y Educación de la Universidad Central de Venezuela, que se dedica a impartir cursos de inglés, italiano, alemán, francés y portugués dirigidos a personas mayores de 15 años de edad con la excepción del idioma inglés que ofrece cursos para niños entre 9 y 11 años, y para adolescentes entre 12 y 14 años. Esta categorización por edades también aplica para los distintos niveles ofertados según el idioma: El curso de inglés comprende 12 niveles para adultos, 6 niveles para adolescentes (teens) y 4 niveles para niños; los demás idiomas sólo ofrecen una categoría para adultos con una duración de 9 niveles.

Las personas que desean ingresar a los cursos anteriormente mencionados pueden inscribirse directamente en el primer nivel (básico I) o presentar una *prueba de nivelación* la cual evalúa el conocimiento previo que posee del idioma para así garantizar su ingreso a un nivel acorde a las destrezas evidenciadas.

En la actualidad, la Coordinación de Extensión cuenta con 2 sistemas online que les permite automatizar parte de sus procesos. Uno de estos sistemas es *ACEIM*, el cual es utilizado para planificar y gestionar los cursos e inscribir a los estudiantes. El otro sistema existente se llama *aTesT*, encargado de la gestión de todas las diferentes evaluaciones impartidas por esta organización.

Aunque existen sistemas que automatizan algunas actividades que la Coordinación de Extensión realiza, el proceso de gestión y aplicación de la *prueba de nivelación* es llevado

a cabo manualmente por el personal administrativo, haciéndolo vulnerable a cometer errores y dependiente de los recursos humanos disponibles al momento de realizar la prueba. Es por ello que el presente trabajo tiene como objetivo *desarrollar una aplicación web para automatizar la aplicación de la prueba de nivelación de los cursos de la Coordinación de Extensión de la escuela Idiomas Modernos de la UCV,* y como objetivos específicos:

- Configurar el método XP para el desarrollo de software.
- Desarrollar módulos de gestión de contenido, gestión de modelos de *prueba de nivelación* y gestión de estudiantes de nivelación.
- Desarrollar un módulo que permita presentar la *prueba de nivelación* y corregirla automáticamente.

La *Aplicación Web* desarrollada permite a la Coordinación de Extensión reducir costos en tiempo y en recursos humanos al elaborar y aplicar de forma automatizada la *prueba de nivelación* del curso de inglés. Bajo esa premisa, este trabajo de investigación se estructura en 4 capítulos descritos a continuación:

Capítulo 1: *Planteamiento del Problema*. En este capítulo se presenta todo lo relacionado a la *prueba de nivelación*: La Coordinación de Extensión de la escuela de Idiomas Modernos de la *UCV* como organización y la problemática existente dentro de la misma con los procesos relacionados a la aplicación de la *prueba de nivelación*, los cuales son llevados a cabo manualmente.

Capítulo 2: Tecnologías Web. En este capítulo se describen las herramientas y tecnologías usadas en el desarrollo de este trabajo, las cuales son agrupadas en tecnologías del lado del cliente, tecnologías del lado del servidor y otras tecnologías que enriquecen o facilitan el desarrollo de la aplicación, tal como un plug-in o un manejador de versiones.

Capítulo 3: Desarrollo de la aplicación. En este capítulo se especifican las configuraciones realizadas al método de desarrollo de software *XP* (eXtreme Programming), describiendo las iteraciones llevadas a cabo en la creación de la Aplicación, detallando las tareas realizadas y las entregables resultantes.

Finalmente, se exponen las conclusiones, las recomendaciones y se listan las referencias bibliográficas que brindan soporte a este Trabajo Especial de Grado.

Capítulo 1. Planteamiento del Problema

Este capítulo describe el contexto de la problemática relacionada al proceso de la aplicación de la *prueba de nivelación* del curso de inglés que imparte la Coordinación de Extensión de la escuela de Idiomas Modernos de la Facultad de Humanidades y Educación de la *UCV.* La coordinación se encarga de instrumentar proyectos para la promoción de las actividades académicas y profesionales que se desarrollan en dicha escuela, teniendo como principales objetivos:

- Prestar servicios de traducción e interpretación, asistencia técnica o investigación.
- Organizar eventos de intereses afines a la escuela, tales como talleres, seminarios, jornadas, encuentros, simposios, congresos y cursos de idiomas.

Para llevar a cabo éste último, desde el año 2005 la Coordinación de Extensión ofrece cursos de francés, portugués, alemán, italiano e inglés en horario semanal y sabatino. Debido a que el inglés se ha convertido en el idioma líder en las comunicaciones interculturales y de negocio a consecuencia del proceso de globalización, su aprendizaje genera interés en personas con objetivos profesionales o personales a nivel internacional, tal como reflejan los datos registrados de la distribución de estudiantes en el año 2013 (ver imagen 1.1).

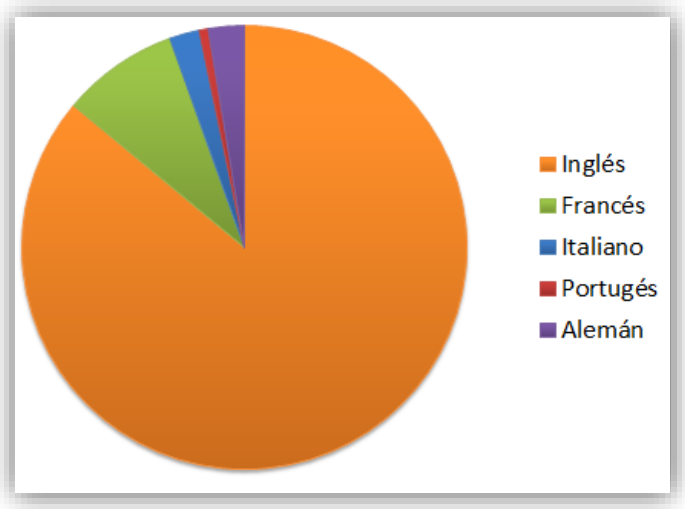


Imagen 1.1: Distribución de estudiantes en el año 2013

Es por ello que el idioma inglés es ofrecido a distintas audiencias con una estructura de niveles de acuerdo al objetivo del programa. A continuación se presenta en la imagen 1.2 los niveles de los cursos de inglés según la audiencia.

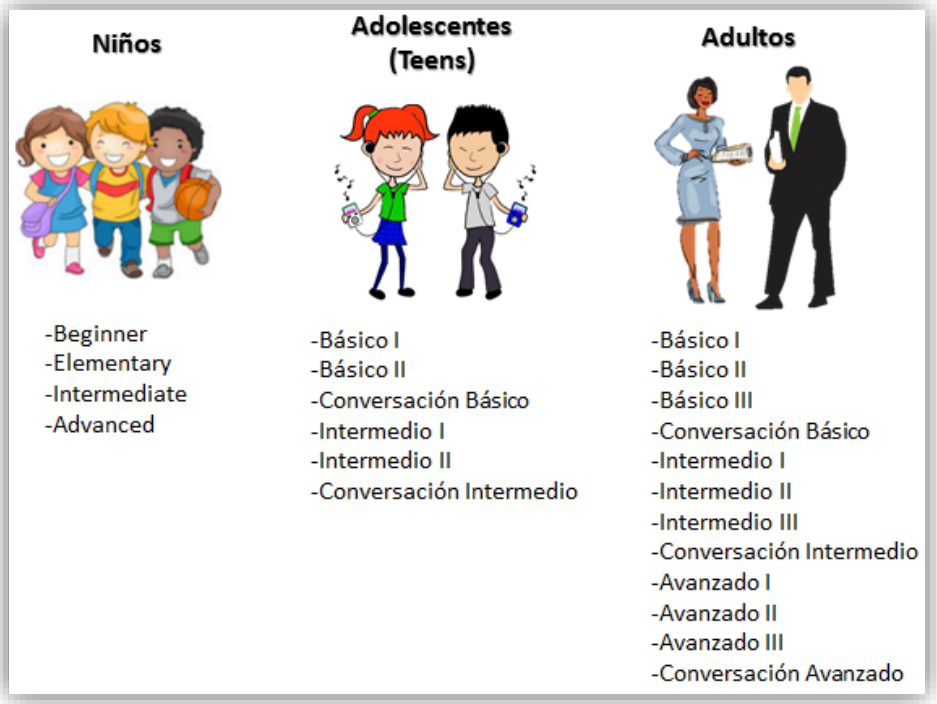


Imagen 1.2: Modalidades del curso de inglés con sus respectivos niveles

Además, la Coordinación de Extensión ofrece los idiomas de francés, portugués, alemán e italiano sólo para una audiencia mayor de 15 años (adulta) con una estructura de niveles similar al curso de inglés para adultos y una duración de 9 semanas (equivalente a 54 horas académicas).

Para ingresar en alguno de los cursos mencionados, se lleva a cabo el siguiente proceso de inscripción para el nivel básico del programa:

1. Generar la planilla de inscripción a través del portal web de la coordinación, registrando los datos personales del estudiante y el horario en el cual desea asistir al curso.
2. Realizar depósito bancario del monto correspondiente al nivel básico I.
3. Entregar la planilla de inscripción y el comprobante de pago en las oficinas de la coordinación.

Pero, si un nuevo ingreso aspira comenzar en un nivel superior a básico I, debe presentar una *prueba de nivelación* para así evaluar la comprensión lectora, auditiva y gramatical que posee del idioma. Esta prueba es parte del proceso de nivelación, el cual se detalla a continuación:

1. Generar la planilla de nivelación a través del portal web de la coordinación, registrando los datos personales del estudiante.
2. Presentar la prueba oral que consiste en una breve entrevista con la coordinadora para así determinar si el estudiante está apto para realizar la parte escrita de la nivelación.
3. Realizar depósito bancario del arancel correspondiente.
4. Entregar la planilla de nivelación y el comprobante de pago en las oficinas de la coordinación.
5. El personal administrativo imprime la prueba de nivelación tantas veces como estudiantes inscritos para la nivelación hayan.
6. Presentar la prueba escrita.
7. Calificación de la prueba.
8. Notificación del nivel asignado según los resultados obtenidos.

Este proceso es efectuado en un lapso de 2 semanas, en donde las tareas de reproducción y calificación de las pruebas son llevadas a cabo por el personal administrativo manualmente.

La parte escrita del proceso de nivelación, es un componente fundamental y potencialmente automatizable que tiene un formato estándar de 130 preguntas para ser respondidas en 1 hora y 30 minutos, todas estas preguntas tienen la misma ponderación y están desglosadas en 5 partes, a saber: *Listening, Grammar, Vocabulary, Reading y Writing* que a continuación se describen (ver imagen 1.3):

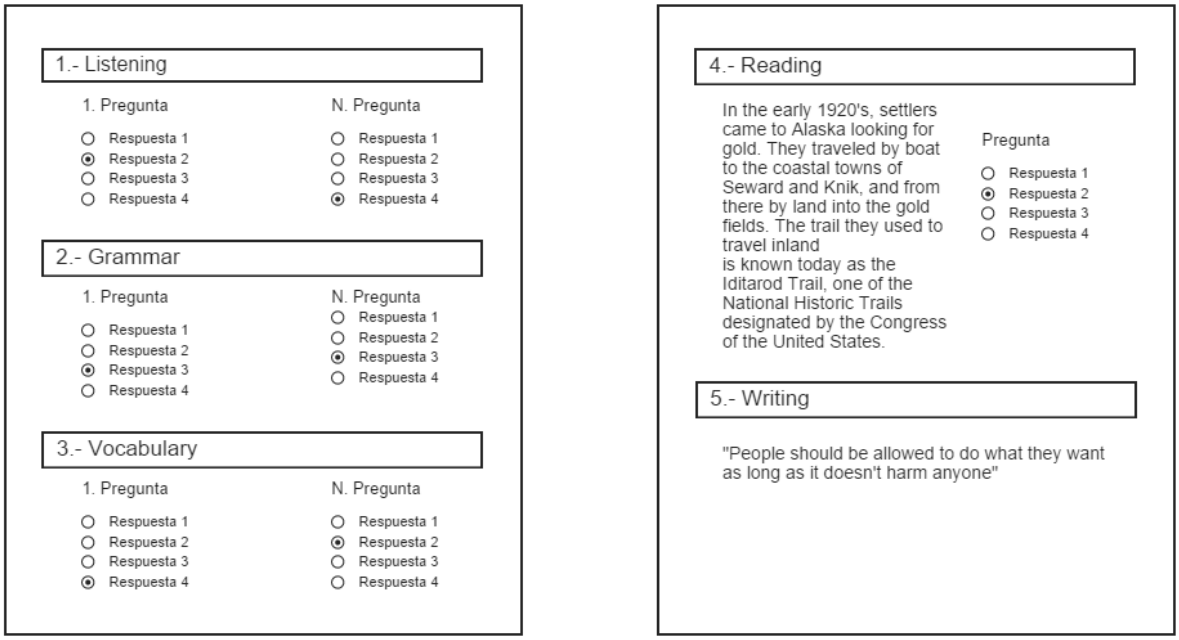


Imagen 1.3: Parte escrita de la prueba de nivelación

La comprensión auditiva o *Listening* consta de 35 preguntas de selección simple que permiten medir la destreza lingüística de interpretar el discurso oral, presentando 18 preguntas correspondientes a pequeñas conversaciones de las cuales se desea que seleccione la respuesta verdadera o más similar a lo escuchado; y 17 preguntas relacionadas a oraciones de las cuales deberá seleccionar la opción que mejor describa a cada una.

Las secciones de *Grammar* y *Vocabulary* evidencian la habilidad para captar la información léxica y derivar interpretaciones de oraciones y discursos al comprender 40 preguntas de selección simple para escoger la opción que mejor complete la oración o conversación presentada.

La comprensión lectora o *Reading* permite comprobar la percepción de las ideas más importantes de un escrito, es por ello que se muestran 3 textos, cada uno con 5 preguntas de selección simple.

El apartado de *Writing* es opcional por lo que está exento de ponderación; se presentan tópicos para escribir un breve ensayo de 250 ó 300 palabras. Esto sólo aporta valor en el proceso de asignación de niveles, es decir, cuando el puntaje obtenido está dentro del umbral aceptado por 2 niveles, entonces la coordinadora revisa el *Writing* que le permite comprobar si el estudiante cuenta con el dominio necesario del idioma para estar en un nivel u otro.

La forma de contestar las 130 preguntas de la *prueba de nivelación* es a través de una hoja de respuesta, tal como se ilustra en la imagen 1.4, lo cual hace el proceso de corrección, un esfuerzo físico repetitivo y vulnerable a cometer errores debido a que es realizado manualmente por la coordinadora de los cursos de idiomas sin contar con la asistencia de la máquina especializada para su lectura.

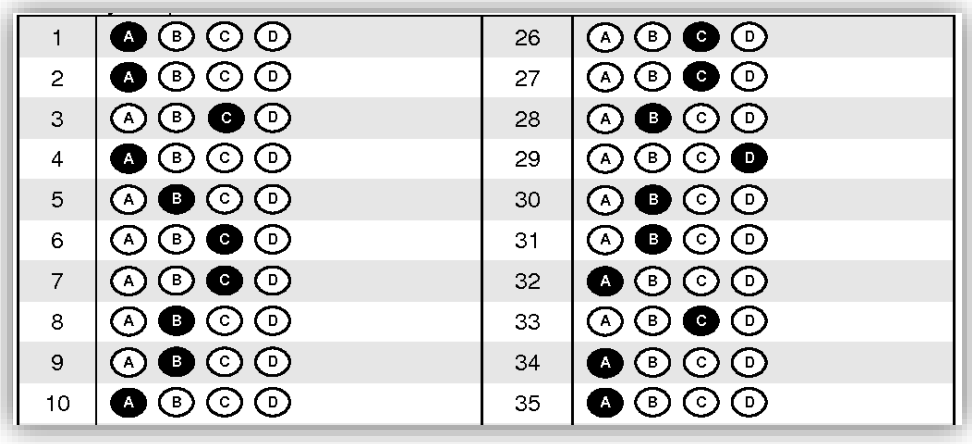


Imagen 1.4: Hoja de respuestas de la prueba de nivelación.

Este proceso de nivelación se realiza en cada período de inscripción que se repite 4 veces al año, generando costos en tiempo y en recursos humanos, entre otras limitaciones mencionadas a continuación:

- Corrección manual de cada prueba. Cada período cuenta con una participación aproximada de 80 personas; si son 4 períodos al año, entonces se corrige manualmente alrededor de unas 320 pruebas al año.
- El tiempo invertido para la corrección de las pruebas es de dos (2) horas aproximadamente.

- No es posible generar modelos personalizados de pruebas debido a que existe sólo un modelo físicamente de 14 páginas que se debe reproducir tantas veces como estudiantes hayan para la nivelación, es decir que, para un período con una participación aproximada de 80 personas se imprimirían 1120 páginas.

Para mejorar las debilidades expuestas, se sugiere la automatización de la parte escrita de la *prueba de nivelación* a fin de:

- Generar distintos modelos de prueba.
- Corregir las pruebas de forma automatizada.
- Disponer de los resultados de manera inmediata.
- Asistir en el proceso de inscripción, al sugerir el nivel según el puntaje obtenido.

Esto es posible a través del uso de las Tecnologías de Información y Comunicación (TIC) para así complementar la plataforma existente llamada *aTesT,* encargada de la gestión de todas las diferentes evaluaciones que imparte la Coordinación de Extensión. Dado que el principal portal web de esta organización es un sistema llamado *ACEIM* que sólo se utiliza para inscripciones, planificación y gestión de cursos.

Es por ello que la aplicación desarrollada se integra al sistema *aTesT* añadiendo funcionalidades que proveen asistencia al proceso de gestión, aplicación y generación de modelos de pruebas de nivelación, permitiendo así aplicar la *prueba de nivelación* online a los estudiantes indicados por la coordinadora o administrador del sistema, es decir, aquellas personas que se inscribieron en la nivelación del curso de inglés y aprobaron la prueba oral.

Apartando el proceso administrativo necesario para llevar a cabo la *prueba de nivelación*, el proceso de aplicar la parte escrita de la prueba online se ilustra en la imagen 1.5.

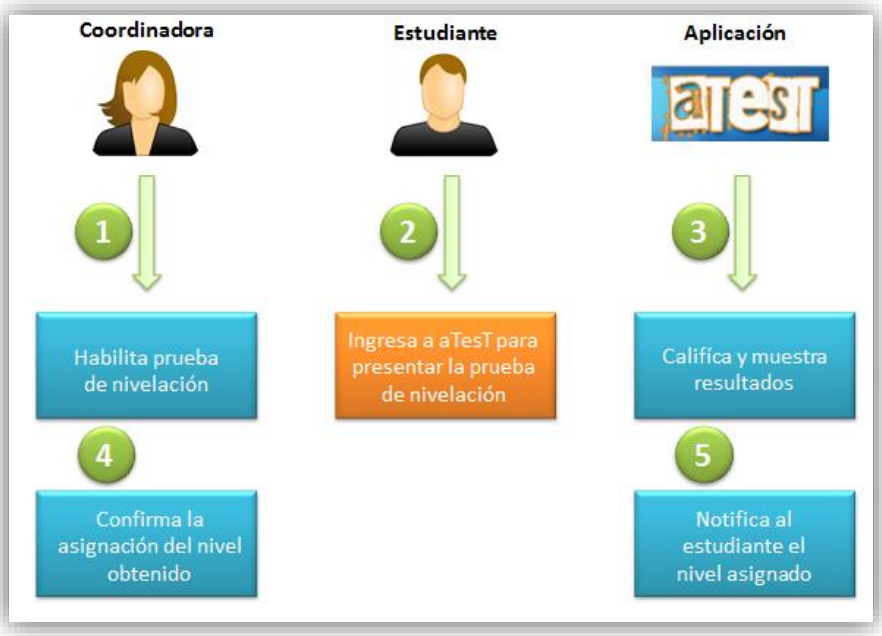


Imagen 1.5: Proceso de aplicar la prueba de nivelación online

Aunque la aplicación puede ser accedida remotamente por cualquier estudiante autorizado desde cualquier computador, la Coordinación de Extensión expresó su interés en que se realizara dentro de sus instalaciones, es por ello que, previendo las limitantes de infraestructura e inconvenientes que puedan surgir al momento de presentar la prueba de forma online, la aplicación cuenta con la opción de imprimir en PDF el modelo seleccionado, asegurando así, la continuidad del proceso de nivelación de ser necesario presentar la prueba de forma manual.

Éstas y otras consideraciones se implementaron a través de un conjunto de técnicas y herramientas denominadas tecnologías web que serán explicadas en el siguiente capítulo.

Capítulo 2. Tecnologías Web

En este capítulo se exponen las bases teóricas usadas para el desarrollo de la *Aplicación Web*, contemplando los conceptos y descripciones de las tecnologías utilizadas en la arquitectura Cliente/Servidor debido a que la masificación de la Internet ha convertido la Web en una herramienta para acceder y compartir información en todo el mundo a tan sólo un *click* al hacer uso de un conjunto de tecnologías que aportan características específicas a la estructura, comunicación y presentación de la información.

2.1 Aplicación Web

Una *Aplicación Web* es un software diseñado para ayudar al usuario a realizar tareas específicas, que puede ser accedida a través de una conexión de red y en la que todas o algunas de sus partes se descargan de una red (Internet o Intranet) cada vez que se ejecuta. Este término puede referirse a las aplicaciones basadas en navegadores que se ejecutan dentro del navegador web del usuario, así como también a las aplicaciones de escritorio o móviles que acceden a la red para obtener información adicional. (PC Magazine, 2014)

Una de las ventajas de las *Aplicaciones Web*, es la facilidad de mantener y actualizar dichas aplicaciones sin la necesidad de distribuir e instalar un software en, potencialmente, miles de clientes. Además de tener la posibilidad de ser ejecutadas en múltiples plataformas al emplear tecnologías que permiten su portabilidad; estas tecnologías se dividen en tecnologías del lado del cliente y en tecnologías del lado del servidor, cada una con funciones específicas que serán explicadas en este capítulo.

Otra ventaja que poseen las *Aplicaciones Web,* es que esta puede ser accedida por uno o más *clientes* mediante el envío de peticiones a un *servidor* central en el cual permanece alojada la aplicación, todo esto se logra a través de una conexión a *Internet*; es aquí donde entra la *Arquitectura Cliente-Servidor*, la cual se detalla a continuación.

2.1.1 Arquitectura Cliente-Servidor

Arquitectura Cliente - Servidor es un modelo de computación en el que el servidor, entrega y administra la mayor parte de los recursos y servicios que se consumen por el cliente, estos servicios pueden incluir el acceso de aplicaciones, almacenamiento, intercambio de archivos, entre otros. Este tipo de arquitectura tiene uno o más ordenadores cliente conectados a un servidor central a través de una conexión de red o Internet. (Janssen, http://www.techopedia.com/, 2012) (Ver imagen 2.1).

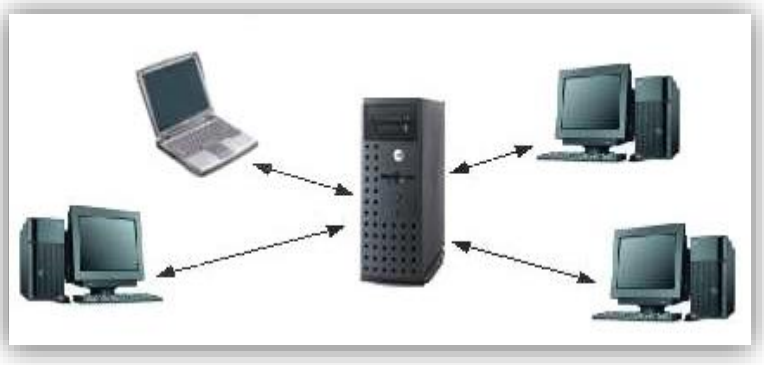


Imagen 2.1: Arquitectura cliente-servidor

La *Arquitectura Cliente - Servidor* funciona cuando el equipo cliente envía un recurso o solicitud de proceso al servidor a través de la conexión de red, que luego es procesado y entregado al cliente, como se observa en la imagen 2.2. Un ordenador servidor puede gestionar varios clientes simultáneamente, mientras que un cliente se puede conectar a varios servidores a la vez, cada uno proporciona un conjunto diferente de servicios, como los antes mencionados. En su forma más simple, la Internet también se basa en una *Arquitectura Cliente - Servidor*, donde el servidor Web sirve a muchos usuarios simultáneos con la página Web y de datos o sitios web. La principal ventaja de esta arquitectura es que facilita la separación de funciones según su servicio, permitiendo situar cada función en la plataforma más adecuada. (Mora, 2002).

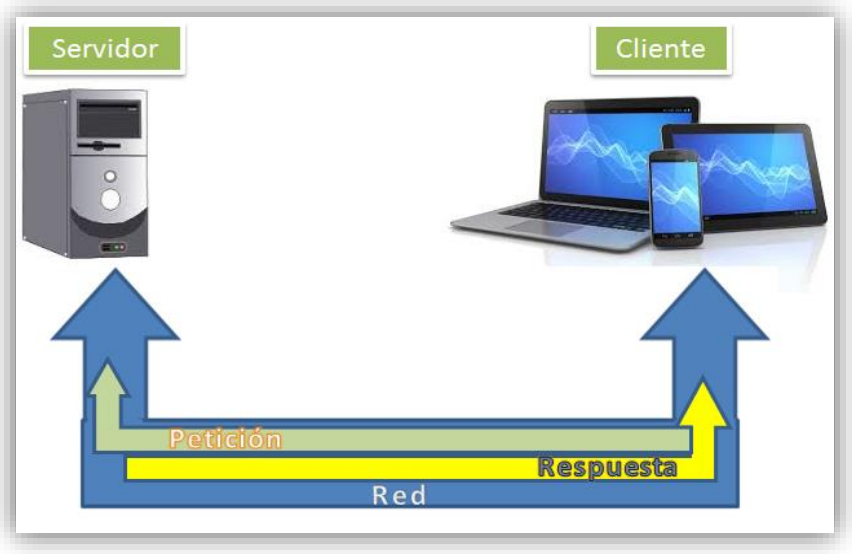


Imagen 2.2: Esquema petición-respuesta de la arquitectura cliente-servidor

La *Arquitectura Cliente- Servidor* se hace útil en el desarrollo de la aplicación para automatizar la *prueba de nivelación*, ya que con ella se hace posible la comunicación de cada uno de los aspirantes a la prueba con la aplicación.

Para la construcción de *Aplicaciones Web* es necesario buscar soluciones tecnológicas que ayuden al desarrollo, implementación y posterior uso de la aplicación. Para ello se necesita definir que tecnologías se pueden usar tanto para la presentación de la información como para procesar las peticiones de recursos que el cliente solicita.

A continuación se detallarán algunas de las tecnologías de código abierto que pueden ser usadas en el lado del *Cliente* o en el lado del *Servidor*.

2.2 Tecnologías del lado del Cliente

Las tecnologías del lado del cliente se refieren a aquellos componentes de software que permiten a un cliente realizar operaciones a través de un navegador web que se ejecuta en una computadora y que se conecta a un determinado servidor. Este comportamiento es posible dado que los navegadores se encargan de la visualización de *Aplicaciones Web* al interpretar sus documentos, estos documentos pueden estar escritos en uno o más lenguajes, lenguajes como *HTML* (*HyperText Markup Language*), JavaScript, *VisualBasicScript*, *Flash*, entre otros.

Como desarrolladores, es necesario acceder a los documentos previamente mencionados para manipularlos y lograr presentar la información de la manera que se desee, es por ello que existe el *DOM* (*Document Object Model*) que proporciona una representación estructural del documento, permitiendo la modificación de su contenido o su presentación visual. Esencialmente, comunicando las aplicaciones web con los scripts o los lenguajes de programación, los cuales forman parte de las tecnologías del lado del cliente utilizadas en el desarrollo de esta aplicación que a continuación se describen:

2.2.1 Ajax

AJAX, acrónimo de *JavaScript asíncrono y XML* (*Asynchronous JavaScript And XML,* por sus siglas en ingles), que como se indica en sus siglas está basado en el lenguaje *JavaScript,* lenguaje de script orientado a objetos que brinda a los desarrolladores la capacidad de crear páginas web dinámicas; es un conjunto de técnicas de programación para el desarrollo *Aplicaciones Web* interactivas. Estas técnicas involucran la capacidad de hacer sin problemas una actualización en la *Aplicación Web* o en una sección de ella con una petición que realiza el servidor, sin necesidad de refrescar la página completa. (Janssen, Techopedia, 2013).

Las aplicaciones web tradicionales tienden a seguir el patrón que se muestra en la Imagen 2.3 en donde en un primer paso la página es cargada, luego el usuario realiza diversas acciones en esta como llenar un formulario o hacer clicks en distintos enlaces, las cuales son presentadas a una aplicación del lado de servidor para su procesamiento mientras que el usuario espera, hasta que finalmente el resultado es enviado como un documento *HTML* al navegador web para su interpretación y visualización, lo que produce que la página se refresque completamente.

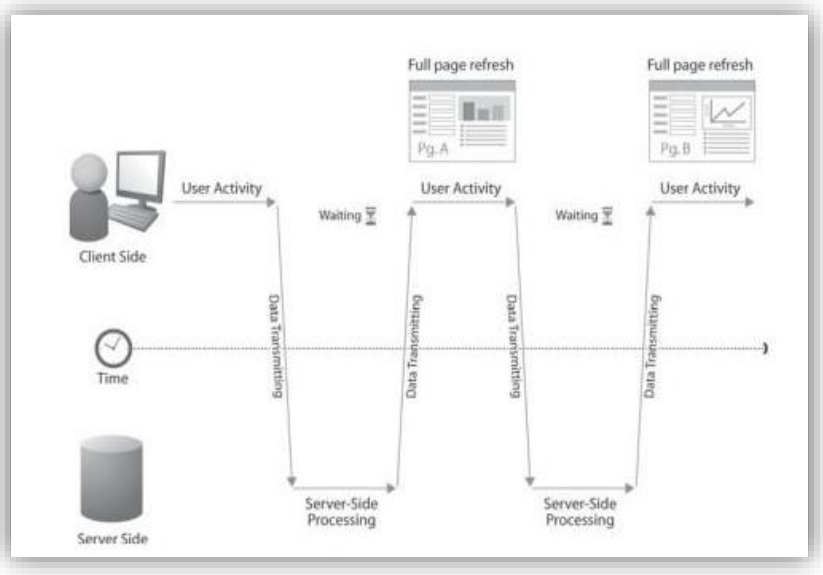


Imagen 2.3. Flujo tradicional de comunicación en una aplicación web

Mientras que esta técnica es simple y fácil de implementar, tiene como desventaja el tiempo que tome el proceso de retransmitir los datos que permitan construir la nueva página una y otra vez con el fin de presentar la aplicación en su nuevo estado, implicando una insatisfacción en el usuario al prolongar el tiempo de espera.

Es por ello que las aplicaciones implementadas con *AJAX* usan un modelo significativamente distinto en donde las acciones del usuario conllevan a una comunicación en segundo plano con el servidor para obtener sólo los datos necesarios para actualizar la página, en respuesta a las acciones realizadas. Este proceso permite así que el usuario pueda realizar otras acciones mientras que el navegador web presenta la data retornada por el servidor, sólo la porción relevante de la aplicación es actualizada. La Imagen 2.4 ilustra este proceso.

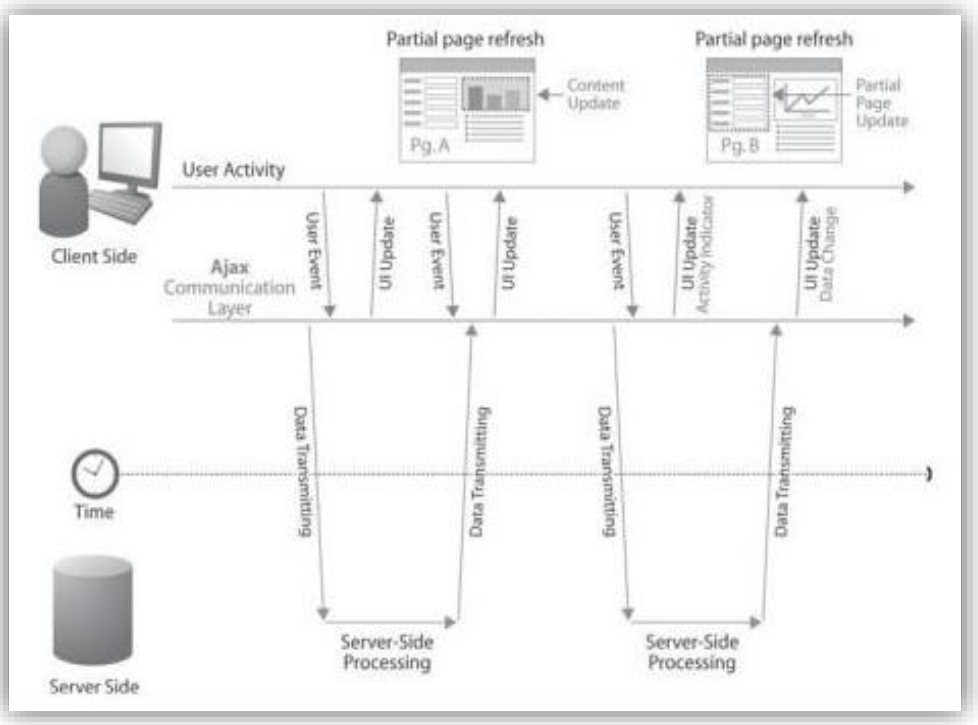


Imagen 2.4. Flujo de comunicación con AJAX

Dentro de los principales beneficios que conlleva el uso de *AJAX* destacan:

- *AJAX* cubre una amplia gama de tecnologías web que se utilizan para iniciar una *Aplicación Web* mientras se está comunicando con el servidor en segundo plano. Esto beneficia al usuario, ya que no interfiere o interrumpe la página Web que él o ella está utilizando.

- *JavaScript* no es el único lenguaje de script del lado del cliente que utiliza la programación *AJAX*; *VBScript* y otros idiomas tienen este tipo de funcionalidad, pero JavaScript es el más popular.

- Existen tecnologías del lado del servidor que son interpretadas por el servidor, para las cuales este lenguaje es independiente ya que es compatible con cualquier tipo de lenguaje de programación web de lado del servidor como *PHP, ASP.NET, JSP*, entre otros.

- A pesar de lo que su nombre indica, *AJAX* no tiene que correr de una manera asincrónica (en segundo plano), ni tampoco tiene que utilizar *XML*. De hecho, se utiliza más a menudo la notación de objetos de *JavaScript*.

Para este proyecto se hace importante el uso de *AJAX* principalmente porque durante el uso de la aplicación es necesario enviar peticiones al servidor sin interrumpir las acciones realizadas por el usuario, permitiéndole así realizar múltiples tareas sin necesidad de mantenerse refrescando la aplicación.

AJAX interactúa con varios lenguajes para llevar a cabo la comunicación con el servidor web e intercambiar datos de manera asíncrona. Uno de estos lenguajes y el más utilizado, como se mencionó anteriormente es *JavaScript*, el cual es descrito a continuación:

2.2.2 JavaScript

JavaScript, como se mencionaba previamente en la definición de *AJAX*, es un lenguaje ligero e interpretado, orientado a objetos, más conocido como el lenguaje de script que brinda a los desarrolladores la capacidad de crear páginas web dinámicas al interactuar con el usuario; siendo *ECMAScript* el *JavaScript* estándar, y a partir de 2012, todos los navegadores modernos soportan completamente *ECMAScript* 5.1. Los navegadores más antiguos soportan por lo menos *ECMAScript* 3 (Mozilla Developer Network, 2014).

Una de las principales ventajas de *JavaScript* es que permite reconocer eventos, es decir, si un usuario realiza una acción, ésta puede ser identificada y proceder a realizar algún proceso en específico, por ejemplo, el cambio del valor de un formulario, o la pulsación de un enlace.

El código *JavaScript* de los manejos de eventos puede ser interno, es decir, que está embebido en el código *HTML* de la página web encerrado entre etiquetas <script> </script>, o externo en donde las instrucciones *JavaScript* están en un archivo con extensión *.js* separado del archivo con código *HTML* y luego se incluye en este último.

Para la aplicación de la automatización de *prueba de nivelación*, es importante el manejo de eventos para las distintas acciones que pueda realizar el usuario dentro de ella, ya que mediante estos eventos puede indicársele si se presenta algún error o bien si fue exitosa la acción que llevó a cabo, generando así mayor interactividad entre la aplicación y el usuario. Entre las acciones más comunes en las que se hizo uso de *JavaScript* para el manejo de eventos en esta *Aplicación Web*, fue en formularios y botones.

La popularidad de *JavaScript* en el desarrollo de aplicaciones web condujo al surgimiento de un conjunto de *Frameworks* y librerías que mejoran las prácticas de programación con *JavaScript*, librerías como *JQuery* que se explica a continuación:

2.2.3 JQuery

JQuery es una librería escrita en lenguaje *JavaScript* que simplifica la interacción con documentos *HTML*, provee manejo de eventos, animaciones, manipula el *DOM* e interactúa con *AJAX*. *JQuery* está diseñado para cambiar la manera de escribir en *JavaScript* (Jquery, 2014).

JQuery se especializa en permitir al desarrollador seleccionar los elementos en una página, determinar que navegador se está usando y cuáles son sus capacidades; además uno de sus principales atractivos son los efectos visuales que produce: sombras, fondos, entre otros.

A partir de librerías como *JQuery* han surgido librerías derivadas de esta que aportan más funcionalidades y efectos. Una de las librerías basadas en *JQuery* más usadas durante el desarrollo de aplicaciones es *JQuery UI*, donde la presente aplicación no está exenta de su uso y la cual se detalla a continuación:

2.2.4 JQuery UI

JQuery UI es una librería de componentes que aporta un conjunto de plugins, widgets y efectos visuales para la creación de aplicaciones web. Cada componente o módulo se desarrolla de acuerdo a la filosofía de *JQuery* (*encuentra algo, manipúlalo*) (Jquery UI, 2014).

JQuery UI ofrece acceso a un conjunto completo de controles de interfaces de usuario (widgets), permite añadir comportamientos complejos a los elementos (interacciones), y cuenta con una API que brinda transiciones animadas para dichos elementos (efectos).

En el desarrollo de la aplicación se utilizaron los siguientes controles configurables:

- Accordion: Muestra secciones plegables de contenido para presentar la información en una cantidad limitada de espacio (ver Imagen 2.5).

Imagen 2.5: Menú con efecto acordeón

- Button: Mejora elementos de formulario estándar, como los botones, haciéndolos personalizables según el estado en el que se encuentre el elemento. Por Ejemplo si el cursor se encuentra *encima* o *activo* en el botón (ver imagen 2.6).

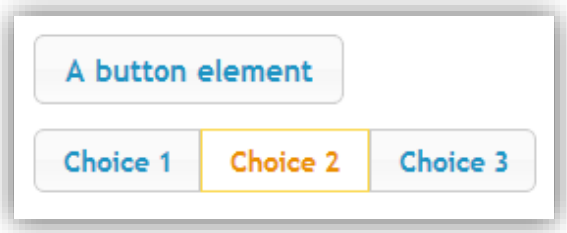


Imagen 2.6: Botones con JQuery UI.

- Dialog: Despliega el contenido de una vista sobre el contenido de otra vista o página de manera interactiva, a modo de captar la atención del usuario o resaltar información importante (ver imagen 2.7).

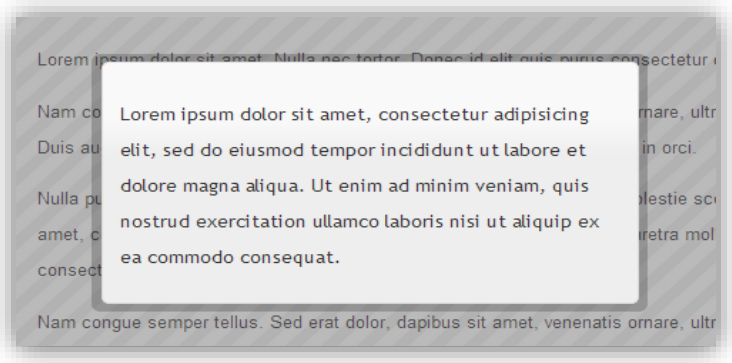


Imagen 2.7: Diálogo en ventana modal

El uso de *widgets* como los mencionados anteriormente en el desarrollo de la aplicación permitió resaltar contenidos, hacer énfasis en las acciones realizadas por el usuario y mayor interactividad con éste.

2.3 Tecnologías del lado del Servidor

El servidor realiza operaciones para generar respuestas a las peticiones del usuario cuando la información o funciones a las que éste quiere acceder están o no disponibles en el lado del cliente.

El propósito principal de las tecnologías que se encuentran del lado del servidor es que permiten modificar el contenido de una *Aplicación Web* de forma online al crear una interfaz dinámica entre el cliente y el servidor, mediante el uso de lenguajes de programación como Ruby on Rails, el cual será descrito a continuación.

2.3.1 Ruby on Rails

Ruby es un lenguaje de programación orientado a objetos de código abierto enfocado en la simplicidad y productividad, fue creado por Yukihiro “matz” Matsumoto mezclando partes de sus lenguajes favoritos (*Perl, Smalltalk, Eiffel, Ada, y Lisp*) para formar un nuevo lenguaje que incorporara tanto la *Programación Funcional* como la *Programación Imperativa* (Ruby on Rails, 2014). Entendiendo como *Programación Funcional* al enfoque que implica crear el problema como un conjunto de funciones que se deben ejecutar; mientras que la *Programación Imperativa* permite al desarrollador escribir el código que

especifica los pasos que se deben realizar para cumplir el objetivo (Microsoft Developer Network, 2014).

Ruby on Rails es un *Framework* que facilita el desarrollo, despliegue, y mantenimiento de *Aplicaciones Web*. Se llama *Framework* a un conjunto de códigos fuentes o librerías que proveen funcionalidades comunes a un grupo de clases de aplicaciones (DocForge, 2014).

El desarrollo en *Ruby on Rail*s está diseñado para reducir el acoplamiento entre la lógica de negocios y la de presentación.

Ruby on Rails utiliza un patrón de diseño arquitectónico llamado *Modelo-Vista-Controlador (MVC)* que separa una aplicación en tres componentes principales: El *Modelo*, la *Vista* y el *Controlador* (Microsoft Developer Network, 2014). Este modelo fue introducido por Trygve Reenskaug, uno de los desarroladores de *Smalltalk* en el Centro de Investigación de Xerox en Palo Alto en 1979, el cual ayuda a separar el acceso a los datos y la lógica del negocio de la manera en la que se muestra al usuario (Leff & Rayfield, 2002).

- *Modelo*: Representa la data y las reglas que permiten el acceso a la misma. Es la conexión entre la vista y el controlador notificándoles la ocurrencia de un cambio en el estado de la aplicación (i.e. una petición del usuario), esto permite que las vistas asociadas al modelo actualicen la presentación de los datos y que los controladores cambien sus comandos disponibles para ello.
- *Vista*: Muestra el contenido de un modelo en un formato adecuado para la interacción (interfaz de usuario). Específica cómo debería presentarse la data de un modelo, si dicha data cambia, la vista debe actualizar su presentación como se requiera.
- *Controlador*: Traduce las interacciones que el usuario tiene con la vista en acciones que realizará el modelo.

La relación existente entre los componentes puede apreciarse en la Imagen 2.8

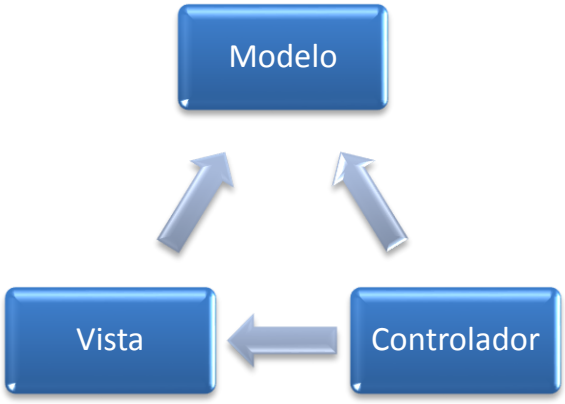


Imagen 2.8: Diagrama MVC (Modelo-Vista-Controlador).

Uno de los beneficios claves del *MVC* es el permitir asociar varias vistas a un modelo para proporcionar presentaciones distintas (separación vista/modelo). También permite cambiar la manera en que una vista responde al usuario sin cambiar su presentación (separación vista/controlador) (Gamma, M. Vlissides, Johnson, & Helm, 2012). En la práctica, las vistas y los controladores tienden a combinarse en un solo objeto por estar relacionados. Por ejemplo, el *Controlador* valida una petición de datos lo que causa que el resultado sea retornado en una *Vista* (PC Magazine, 2014).

La adaptación que implementa *Ruby on Rails* del patrón *MVC* y sus funcionalidades se muestran en la Imagen 2.9, donde la *Vista* está compuesta de plantillas (templates) que pueden estar en distintos formatos, pero las más usadas son las *HTML* con código *Ruby* embebido; el *Modelo* consta de clases derivadas de la librería *ActiveRecord* que permite presentar la data de una base de datos como objetos y embeberla con métodos de la lógica de negocio; el *Controlador* se encarga de manejar y darle respuesta a las peticiones *HTTP* aunque también puede generar vistas *XML*. En *Rails*, el *Controlador* y la *Vista* son manejados de forma conjunta por el *Framework Action Pack*.

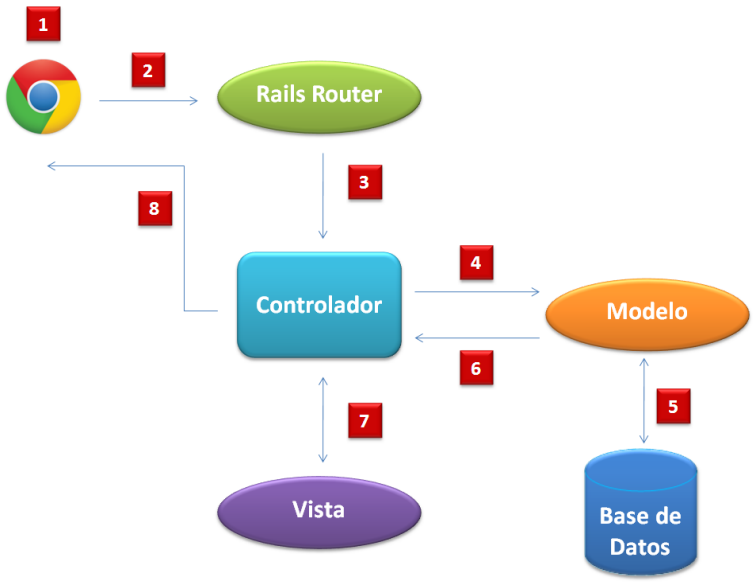


Imagen 2.9: Diagrama del funcionamiento del patrón MVC en Ruby on Rails

En la imagen anterior se ilustra el funcionamiento del patrón *MVC* en *Ruby on Rails* que se explica a continuación:

1. El usuario realiza una petición a través del navegador, haciendo click a un enlace o escribiendo una dirección de un sitio web en la barra de direcciones del navegador.
2. La petición recibida es manejada por un *Controlador* en específico (paso 3), mediante el *Enrutador de Ruby*. El *Controlador* es una clase *Ruby* en la cual varios métodos están definidos para el análisis de las peticiones del usuario.
4. El *Modelo* se encargan de interactuar con la base de datos (Paso 5) y lleva a cabo la validación, asociación, transacciones y demás operaciones con la data.
6. El *Modelo* manipula los datos almacenados y realizar consultas SQL para dar respuesta a lo solicitado.
7. El *Modelo* envía la data obtenida al *Controlador*, el cual la almacena en una memoria temporal.
8. La *Vista* recibe la data enviada por el *Controlador* y genera la salida *HTML*.
9. El *Controlador* retorna el *HTML/XML* y la meta data al navegador.

Los módulos anteriores de *aTesT* fueron desarrollados con *Ruby* bajo *Rails* por lo que se hace imperativo utilizar éste lenguaje para el desarrollo de la aplicación para *prueba de nivelación* de Inglés. Otras de las ventajas que supone el uso de esta tecnología para el desarrollo de la aplicación es que promueve las buenas prácticas de programación, como por ejemplo, ayuda a evitar la repetición de código, con el uso de layouts, helpers, entre otros; ahorra líneas de código y facilita el mantenimiento, la sintaxis del código se muestra de manera sencilla y entendible para casi cualquier programador. Otra de las ventajas por la que se decide hacer uso de esta tecnología es la facilidad de implementación de las operaciones CRUD (Create, Read, Update, Delete) del protocolo *HTTP,* en código eso se traduce en utilizar *POST*, *GET*, *PUT* y *DELETE.*

Cuando existen tecnologías como *Ruby on Rails*, tecnologías que suelen estar en constante actualización, existe la posibilidad de que ciertas versiones no posean compatibilidad con versiones anteriores, es por ello que surgen herramientas que permiten integrar distintas versiones para trabajar en un mismo proyecto, una de las tecnologías que se usa es el *Manejador de Versiones de Ruby*, el cual se detalla a continuación:

2.3.2 RVM

El *Manejador de Versiones de Ruby* (RVM) fue creado por Wayne E. Seguin en el año 2007. Éste además de hacer la instalación de *Ruby* más sencilla, permite instalar y manejar múltiples copias de *Ruby* en el sistema, así como también varias implementaciones de *Ruby* (Ruby on Rails, 2014)

RVM permite el despliegue de proyectos cada uno con contenido propio y en un ambiente dedicado. Además, proporciona características que no son tradicionalmente soportadas en la mayoría de instalaciones *Ruby*. La versión anterior del sistema *aTesT* en el cual esta implementada la aplicación para la *prueba de nivelación*, se utilizó la versión 1.9 de *Ruby*, para éste proyecto se utiliza la versión 2.0 por lo que durante el desarrollo se hizo útil el *Manejador de Versiones de Ruby*, para el despliegue de cada versión.

Al desarrollar una *Aplicación Web* es necesario contar con un repositorio de datos que permita almacenar y gestionar la información contenida en la misma, es por esto que el software desarrollado usa *MySQL* como manejador de base de datos, el cual se detalla a continuación.

2.3.3 MySQL

Es un sistema manejador de bases de datos de código abierto. Se llama código abierto al software que esté disponible para ser usado y/o modificado por cualquier persona; y se llama base de datos a una colección estructurada de datos, que para acceder o procesar dichos datos requiere de un sistema que proporcione estas herramientas.

MySQL fue desarrollado por *MySQL AB*, una compañía de software fundada en Suecia por David Axmark, Allan Larson y Michael Widenius, actualmente pertenece a Oracle Corporation al ser vendida en el año 2010 por Sun Microsystems, quien en el año 2008 había adquirido dicha compañía. El nombre de *MySQL* surgió de la combinación del nombre de la hija de Michael Widenius, My, y las iniciales SQL que indican "*Lenguaje de Consulta Estructurado" (Structured Query Language).*

MySQL maneja bases de datos relacionales donde los datos son almacenados de forma separada en diferentes tablas, de tal forma que, para garantizar la consistencia e integridad de las mismas, existen reglas que el manejador hace cumplir (seleccionar el tipo de relación entre las tablas, tipo de dato, entre otros); haciendo de *MySQL* uno de los sistemas manejadores de bases de datos más populares, del cual destacan las siguientes características:

- Velocidad, confiabilidad y escalabilidad. Puede ser ejecutado en equipos dedicados o personales, compartiendo recursos con otras aplicaciones, servidores web, etc. O configurarlos para que *MySQL* aproveche mejor dichos recursos (memoria, CPU, entrada/salida).
- Cliente/Servidor o sistemas embebidos. Contiene una amplia variedad de herramientas para administrar el servidor multi-hilos que ofrece. También puede ser embebido y asociarlo a una aplicación standalone.
- Código abierto. Incentiva a los desarrolladores a contribuir con el proyecto y mejorarlo, así como también a usarlo como motor de base de datos en las aplicaciones que realicen, por lo que la mayoría de los softwares en el mercado utilizan *MySQL*.

El uso del sistema manejador de bases de datos *MySQL*, así como el uso del lenguaje *Ruby*, se utiliza principalmente porque la versión anterior de la aplicación fue desarrollada utilizando éste manejador, por lo que para el desarrollo de esta nueva aplicación también se realizó con éste manejador, además de ofrecer las ventajas antes mencionadas.

2.4 Otras Tecnologías

Actualmente existen tecnologías que permiten facilitar la tarea como desarrollador de software, herramientas como *HAML, SASS, SCSS y CoffeeScript*, a continuación se explica un poco más acerca estas tecnologías:

2.4.1 HAML

Lenguaje de Marcado Abstracto HTML (HTML Abstraction Markup Languange) creado por Hampton Catlin, su implementación y desarrollo en Ruby lo llevó a cabo Nathan Weizenbaum, actualmente el encargado de su mantenimiento es Norman Clarke (HAML, 2014).

Este lenguaje es usado para describir de una forma elegante el *HTML* de cualquier documento web sin el uso de código embebido, además elimina la necesidad de escribir código *HTML* en las plantillas (template) pues el mismo es una descripción del *HTML*.

En la imagen 2.10 se muestra una porción de código en *HAML* y su equivalente en *Ruby* (*ERB*):

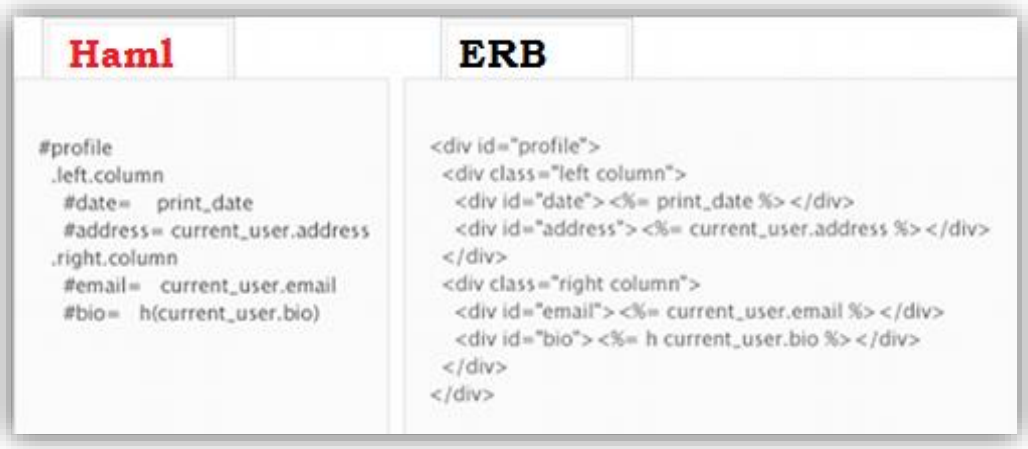


Imagen 2.10: Código en HAML y su equivalente en ERB.

2.4.2 SASS y SCSS

SASS es un meta-lenguaje creado por Hampton Catlin que provee una sintaxis más simple y elegante usado para describir el estilo de un documento web más allá de las limitaciones del *CSS* porque implementa diferentes tipos de características útiles para la creación de hojas de estilos más manejables (Sass: Syntactically Awesome Style Sheets, 2014)

SASS es una extensión de la última versión de *CSS* (*CSS3*) razón por la cual éste lenguaje tiene dos tipos de sintaxis:

- *SCSS* (*Sassy CSS*). Es la principal y la mejor, la cual representa un superconjunto de la sintaxis empleada en *CSS3*, por lo que cualquier hoja de estilo *CSS3* es válida como una hoja de estilo *SCSS*.
- *SASS*. Inspirada en la simplicidad de *HAML*, aunque actualmente no es la principal sintaxis fue creada pensando en hacerla más concisa y no semejante al *CSS*.

2.4.3 CoffeeScript

Es un lenguaje de programación que compila a *JavaScript* cuya sintaxis está inspirada en otros lenguajes de programación: *Ruby y Pytho*n (MacCaw, 2012). *CoffeeScript* es un intento de exponer las partes buenas de *JavaScript* en una forma más sencilla, dado que el código se compila uno-a-uno a su equivalente en *JavaScript* sin la necesidad de un intérprete en tiempo de ejecución, lo cual permite el uso de librerías de *JavaScript* en *CoffeeScript* y viceversa (CoffeeScript, 2014).

Una de las principales diferencias con *JavaScript* es que éste toma en cuenta las sangrías en vez de las tradicionales llaves, reduciendo un tercio o la mitad del código original escrito en *JavaScript* (MacCaw, 2012).

Durante el desarrollo de la *Aplicación Web* es necesario contar con otras herramientas que permitan, en caso de que el desarrollo de la aplicación dependa de varias personas que conforman un grupo de desarrollo, manejar el código fuente producido por cada uno los desarrolladores del equipo. Para éste tipo de tareas se encuentran herramientas cuyo funcionamiento y ventajas se detallan a continuación.

2.4.4 Git

Git es un sistema orientado al desarrollo de aplicaciones que permite un control distribuido de versiones de código haciendo énfasis en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente. *Git* es un software diseñado por Linus Torvalds en el año 2005. Se ha convertido desde entonces en un sistema gratuito de control de versiones con funcionalidad plena para cualquier desarrollador.

La principal ventaja de esta herramienta es que cada directorio en *Git* es un repositorio con grandes capacidades de rastreo de versiones y manejo de historia, que no depende de un acceso a una red especifica ni a un servidor central, características que se hicieron necesarias durante el desarrollo de la aplicación, ya que en múltiples ocasiones el equipo de desarrollo implementaba funcionalidades por separado, la cuales debían ser integradas al código fuente, esta herramienta se encargaba de realizar ese trabajo con el manejo de versiones y la integración de cambios realizados.

Entre las características más relevantes de esta herramienta se encuentran:

- Fuerte apoyo al desarrollo distribuido, por ende rapidez en la gestión de ramas y mezclado de diferentes versiones y fusión de cambios. Cada programador cuenta con una copia local del historial del desarrollo entero, y los cambios se propagan entre los repositorios locales.
- Los almacenes de información pueden publicarse por *HTTP*, *FTP*, *rsync* o mediante un protocolo nativo, ya sea a través de una conexión *TCP/IP* simple o a través de cifrado *SSH*.
- Gestión eficiente de proyectos grandes, dada la rapidez de gestión de diferencias entre archivos.

2.4.5 Simple Player

Simple Player es un plug-in para *JQuery* que permite a los desarrolladores web controlar archivos de audio soportados por la etiqueta de audio de *HTML5* (formato *mp3*, *ogg* o *wav*). También soporta estilo a través de CSS.

Este plug-in fue desarrollado en el año 2010 por un colaborador de *GitHub* (plataforma de desarrollo colaborativo de software) con la iniciativa de reemplazar el tradicional flash player debido a que el uso de flash consume más memoria y esto no debería de ser un problema si sólo se requiere reproducir un archivo de audio.

La implementación de éste plug-in consiste en incluir los respectivos archivos *CSS* y *JavaScript* para hacer posible la inicialización dentro de la aplicación invocando la función .player(), que crea una etiqueta <audio> de *HTML5* en la cual también se puede especificar parte de su configuración como color, ancho y alto de la barra de progreso, control del volumen, etc.

En la imagen 2.11 se puede apreciar el uso dado a éste plug-in en el desarrollo de la aplicación debido a que se requiere poder gestionar los audios cargados que complementan el repositorio de preguntas para la parte de Listening de la *prueba de nivelación*.

Imagen 2.11: Uso del Simple Player plug-in

La principal ventaja que nos ofrece éste plug-in y por el cual se decidió utilizarlo, es la facilidad con la que permite modificar los estilos de la etiqueta de audio de html5, permitiendo realizar modificaciones en características del reproductor como ancho, alto, color, color de fondo de la barra de progreso, estilo y volumen por defecto.

2.4.6 DataTables

DataTables es un plug-in para *jQuery* basado en los fundamentos de la mejora progresiva, lo cual agregará controles avanzados de interacción a cualquier tabla *HTML* debido a que al inicializar el objeto jQuery.dataTable, la información sobre la tabla se lee directamente de la página *HTML* (DataTables, 2014).

Dentro de sus características principales destacan:

- Longitud de paginación variable.
- Filtrado rápido.
- Ordenación Multi-columna.
- Soporta *Ajax*, *JSON*, *PHP*, *C #*, *Perl*, *Ruby*, *AIR*, *Gears*.
- Compatible con *jQuery UI ThemeRoller*, el gestor de temas y estilos de *JQuery*.
- Amplia variedad de plug-ins (*Editor*, *TableTools*, *FixedColumns*, entre otros).

En la imagen 2.12 se observa el uso del plug-in dentro de la aplicación desarrollada, ésta permite al usuario realizar varias acciones (buscar, ordenar, paginar) en una misma interfaz.

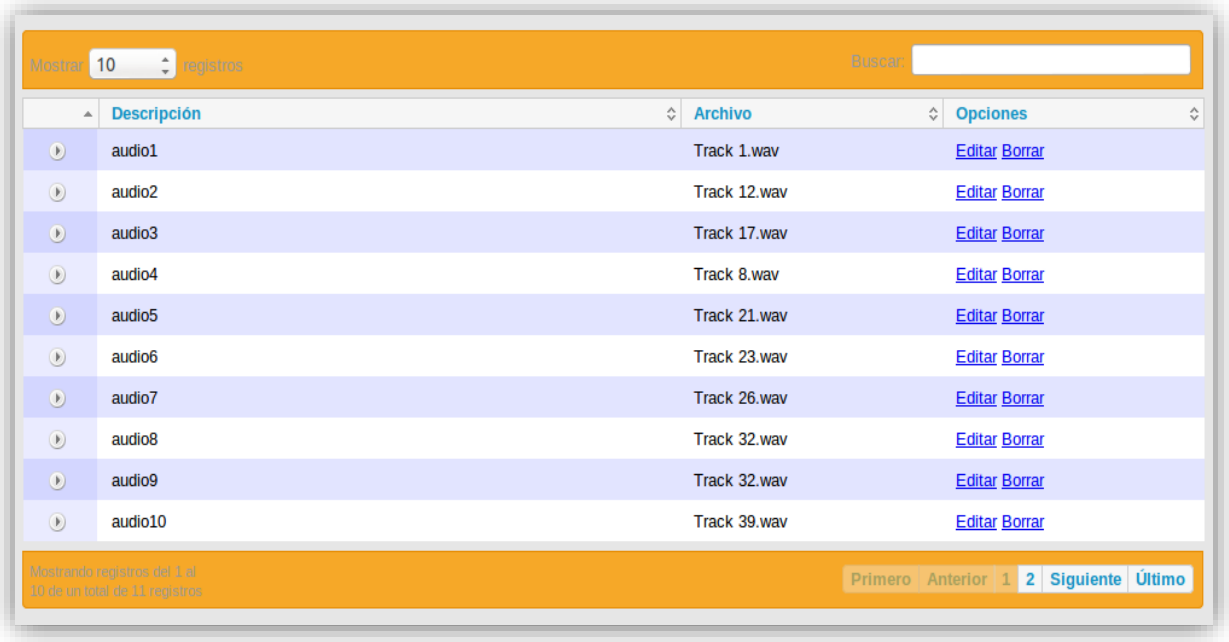


Imagen 2.12: Uso del DataTables plug-in

El uso de éste plug-in nos ofrece múltiples ventajas, entre ellas, la que consideramos más importante, es que con solo una instanciación o inicialización del objeto DataTable, se crea una tabla con paginación, filtrado y búsqueda, y limitación de filas, aspectos importantes y deseables en tablas que pueden tener gran cantidad de registros o filas.

2.4.7 SmartWizard

SmartWizard es un plug-in para *jQuery* si se desea lograr una interfaz tipo W*izard* (asistente) dado que permite agrupar el contenido en secciones, por lo que ahorra espacio en la página, y también ofrece una interfaz limpia y con estilo para los usuarios (Tech Laboratory - Smart JQuery plugins, 2014).

Dentro de sus principales características destacan:

- Soporta *Ajax*.
- Efectos de transición dentro de la navegación de secciones.
- Navegación por teclado.
- Despliegue de las secciones en estilo horizontal o vertical.

El uso de éste plug-in consiste en incluir los respectivos archivos *CSS* y *JavaScript* para luego realizar su inicialización dentro de la aplicación desarrollada invocando a la función .smartwizard() y a la vez se puede configurar atributos como el tipo de efecto de transición, habilitación de botones o de navegación por teclado entre otros.

La imagen 2.13 muestra el uso dado a éste plug-in en la aplicación desarrollada: un asistente para la creación de modelos de la *prueba de nivelación*, dónde cada sección (paso) pertenece al contenido de los módulos de dicha prueba (*Grammar, Vocabulary, Reading, Writing y Listening*).

Imagen 2.13: Uso del SmartWizard plug-in

Durante el desarrollo de la aplicación, desde la fase de diseño se buscó una manera de concentrar el contenido de cada uno de los módulos de *Grammar, Vocabulary, Reading, Writing y Listening,* de manera que el usuario pudiera enfocarse en cada módulo y ofrecerle la ventaja de saber que módulos ha realizado y cuantos les queda por contestar, por lo que éste plug-in contiene lo necesario para cumplir con las características deseadas y permite inclusive agregar funcionalidades adicionales al mismo.

2.4.8 TimeTo

TimeTo es un plug-in para *jQuery* que permite añadir una cuenta atrás de un modo muy sencillo. Permite múltiples opciones de personalización y su aspecto viene definido por estilos *CSS* (Webintenta, 2014).

La implementación de éste plug-in consiste en incluir los respectivos archivos *CSS* y *JavaScript* para proceder a su inicialización invocando la función .timeTo() la cual tiene opciones de configuración como callback para llamar a una función cuando la cuenta llegó a cero, modificar el tipo y tamaño de fuente a utilizar, elegir un tema, entre otros.

La imagen 2.14 muestra la implementación del plug-in con una inicialización de 2 horas para proceder con la cuenta atrás. En la aplicación desarrollada, éste dato es parametrizable de manera que al crear un modelo de *prueba de nivelación*, se indique la duración de la misma, razón por la que se usa éste complemento que además posee un diseño sencillo, pero relevante.

Imagen 2.14: Uso del SmartWizard plug-in

Además de usar las tecnologías previamente mencionadas, también se requiere el uso de un método de desarrollo de software para garantizar la calidad del software y su desarrollo en el tiempo establecido, en el siguiente capítulo se define el método de desarrollo a utilizar y se explican las bases del mismo.

Capítulo 3. Desarrollo de la Aplicación

La construcción de un software exitoso se debe a las directrices establecidas por un método de desarrollo que conduzca a un resultado de alta calidad que satisfaga al cliente, es por ello que tomando en cuenta el contexto del problema y la solución propuesta, se decide aplicar un enfoque ágil, el cual surge en el año 2001 cuando Kent Beck y otros 16 notables desarrolladores, escritores y consultores conocidos como la "Alianza Ágil" firmaron el "Manifiesto para el desarrollo ágil de software", el cual establecía:

- Valorar a los individuos y sus interacciones sobre los procesos y las herramientas.
- Valorar al software en funcionamiento sobre la documentación extensa.
- Valorar la colaboración del cliente sobre la negociación del contrato.
- Valorar la respuesta al cambio sobre el seguimiento de un plan.

Este manifiesto hace que todos los métodos de desarrollo ágiles promuevan el trabajo en equipo y el desarrollo iterativo e incremental del software. Razón por la que en esta investigación se instancia y configura el método de desarrollo de software *XP* (eXtreme Programming), el cual se basa en la simplicidad de las soluciones implementadas y la comunicación fluida entre el cliente y el equipo de trabajo.

El *método XP* se centra en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo de software, por lo que establece varios roles con responsabilidades únicas para así asegurar la premisa de "si desean avanzar en el trabajo, tendrán que hablar entre ustedes" [O'Reilly, M., Extreme Programming Pocket Guide, 2003].

A continuación se describen los lineamientos propuestos por el método *XP* para el desarrollo de un software:

3.1 Método de Programación Extrema (XP)

Existen estándares y normativas que determinan los pasos para la creación de un software (ciclo de vida del software) al definir los procesos, las actividades y las tareas involucradas en el desarrollo, la explotación y el mantenimiento de un producto de software, abarcando la vida del sistema desde la definición de los requisitos hasta la finalización de su uso (ISO, 2013).

El ciclo de vida propuesto por *XP* divide el desarrollo del software en bloques de tiempo, llamado iteraciones, en donde se construyen un conjunto de funcionalidades especificas en un período de tiempo estimado.

La filosofía de éste método, creado por Kent Beck en 1999, describe que sus principios y prácticas son de sentido común, pero llevados al extremo (Beck, 2004). Abarca un conjunto de reglas y prácticas que ocurren en el contexto de 4 fases del marco de trabajo: planificación, diseño, codificación y pruebas. En la imagen 3.1 se ilustra el proceso XP y se observan artefactos y tareas claves asociadas a cada fase.

Imagen 3.1: Fases del método XP

- En la fase de planificación se especifican los requisitos del software y se distribuyen las tareas a realizar para dar respuesta al usuario en un tiempo determinado. Para ello se utilizan Historias de Usuario que describen los requerimientos funcionales y/o no funcionales del sistema, registrando características como fecha, tipo de actividad y prioridad técnica y del cliente, hecho que incide en la planificación pues estas fichas son descompuestas en tareas de programación para ser implementadas durante una iteración que generalmente abarca de 1 a 3 semanas.
- El diseño permite crear una estructura organizada de la lógica del sistema. *XP* establece que los diseños deben ser sencillos, si alguna parte del sistema es de desarrollo complejo, lo apropiado es dividirla en varias. Para esta fase los artefactos

propuestos son los prototipos de interfaz y las tarjetas *CRC* (Clase-Responsabilidad-Colaboración), usadas para mostrar información de eventos y funcionalidades relacionadas.

- En la fase de codificación, *XP* recomienda el trabajo en pareja para agilizar la respuesta a los requerimientos plasmados en las historias de usuario y asegurar la calidad de las soluciones desarrolladas. Asimismo, recomienda la integración continua de código para evitar problemas de compatibilidad e interfaz, generando así un ambiente que ayuda a descubrir los problemas desde el principio.
- *XP* propone realizar pruebas de integración y validación del sistema frecuentemente para que el equipo conozca su avance de acuerdo a lo planificado, además de realizar pruebas de aceptación para garantizar la satisfacción del usuario con el producto desarrollado.

La propuesta original de éste método contempla un equipo de desarrollo integrado por varios roles con tareas específicas. En la tabla 3.1 se detallan estos roles con sus respectivas asignaciones.

Rol	Tarea
Programador	Producir el código que lleve a cabo las funcionalidades del software y describir las pruebas unitarias del mismo.
Cliente	Escribir las historias de usuario y las pruebas funcionales para validar la implementación del software.
Tester	Ayudar al cliente a escribir las pruebas funcionales, ya que éste las ejecuta regularmente para comunicar al equipo sus resultados
Tracker	Realizar el seguimiento del progreso de cada iteración verificando el grado de acierto entre lo planificado y lo llevado a cabo.
Coach	Proveer directrices al equipo para garantizar la aplicación de las prácticas XP.
Consultor	Instruir al equipo con sus conocimientos de surgir un problema en algún tema en específico.
Big Boss	Coordinar todas las tareas para que el equipo trabaje efectivamente, además es el vinculo entre el cliente y los programadores.

Tabla 3.1: Roles del método XP

Una de las cualidades más destacables del método *XP* es su sencillez, tanto en su aprendizaje como en su aplicación, reduciendo así los costos de implantación en un equipo de desarrollo. Siendo una de las razones por la que en esta investigación se decide utilizar éste método con ciertas modificaciones.

3.2 Configuración del método de desarrollo de software *XP*

Para el desarrollo de la aplicación web se decidió usar el método de desarrollo *XP*, el cual centra sus procesos en la simplicidad, la comunicación y la satisfacción del usuario; brinda flexibilidad en la definición de los requerimientos del sistema y reduce los tiempos de desarrollo. Es por esta razón que *XP* es el método que mejor se adecua al contexto en el cual se enmarca el presente trabajo.

A continuación se describen las configuraciones del método de desarrollo de software *XP* aplicadas en éste trabajo de acuerdo a los roles utilizados, la estructuración de las fases y las iteraciones realizadas:

3.2.1 Roles

En la propuesta original del método se especifican siete (7) roles distintos, sin embargo, para el desarrollo de esta aplicación se cuenta con un equipo reducido, por lo que es necesario reestructurar la asignación de responsabilidades según los recursos disponibles, generando como resultado los siguientes roles:

- Usuario: El usuario o cliente es la persona o ente que impulsa el proyecto, definiéndolo según sus necesidades. Cuanto más precisos sean sus requerimientos y más frecuente su participación, mayores serán las probabilidades de que el proyecto tenga éxito. Este rol lo lleva a cabo la Coordinación de Extensión de la Escuela de Idiomas Modernos.
- Programador: El desarrollador o programador trabaja en estrecha colaboración con el usuario para captar sus requerimientos y plasmarlos en un formato específico llamado Historias de Usuario, el cual permite una mayor legibilidad y comprensión al momento de planear y desarrollar las funcionalidades que contendrá la aplicación para dar respuesta a dichos requerimientos. Éste rol, además de llevar a cabo el análisis de requerimientos, se encarga de la planificación, diseño, programación y ejecución de las pruebas.

- Entrenador (Coach): Es la persona encargada de proveer directrices al equipo para garantizar la aplicación de las prácticas XP; en éste proyecto, las responsabilidades de éste rol son llevadas a cabo por la tutora.
- Consultor: Es la persona designada para instruir al equipo con sus conocimientos de surgir algún problema en un tema en específico. Durante el desarrollo de la aplicación, el tutor realizó las acciones asociadas a éste rol.

Además de la repartición de responsabilidades por roles, el *método XP* abarca un conjunto de buenas prácticas que ocurren en el contexto de cuatro (4) fases, cuya aplicación en esta investigación se ilustra en la imagen 3.2 y se explica a continuación.

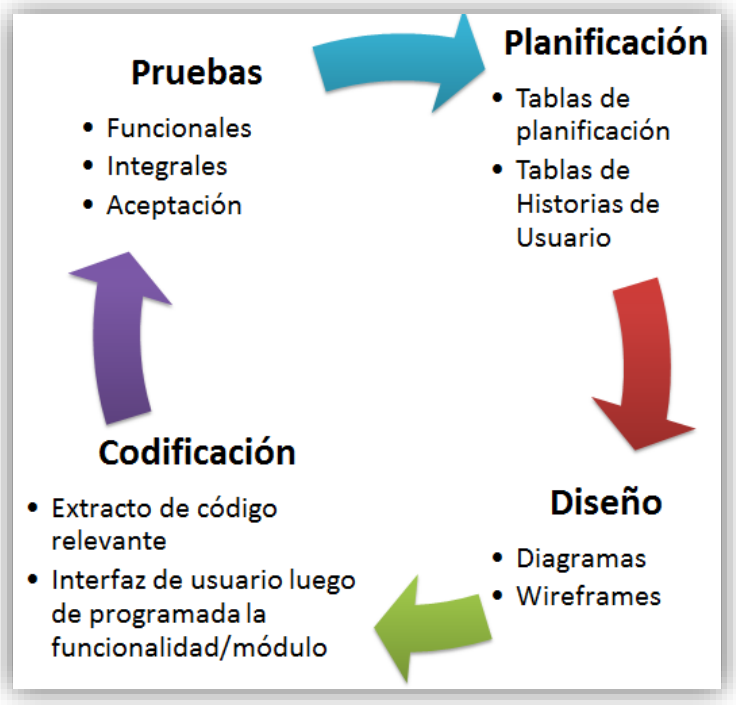


Imagen 3.2: Fases del método XP configurado

3.2.2 Planificación

Es el proceso de recolectar los requerimientos funcionales de la aplicación según las necesidades expresadas por el cliente, para así trazar objetivos por tiempo y prioridad que permitan planificar dichos requerimientos. Para ello se utilizan las Historias de usuario, las cuales se escriben en lenguaje natural y contienen descripciones cortas de lo que la

aplicación debe realizar, éstas deben tener el detalle mínimo de las funcionalidades a desarrollar para que los programadores puedan realizar una estimación poco riesgosa del tiempo que llevará su programación. Cuando llegue el momento de la implementación, los desarrolladores dialogan directamente con el cliente para obtener más detalles de ser necesario.

Como parte de la configuración del *método XP,* en éste trabajo se utilizan tablas descriptivas como artefacto de la fase de planificación, para así detallar aquellas Historias de Usuario a las que se va a dar respuesta en cada iteración, su duración y tipo (ver imagen 3.3).

Nombre de Iteración			
Descripción			
Duración de iteración			
Historias de Usuario		Duración	Tipo

Imagen 3.3: Formato de Planificación

Otro artefacto correspondiente a esta fase son las Historias de Usuario cuyo formato se muestra en la imagen 3.4. Éstas deben ser planificadas en un tiempo entre una y tres semanas. Si la estimación es superior a tres semanas, debe ser dividida en dos o más historias. Si es menos de una semana, se debe combinar con otra historia.

Titulo	Nombre de la Historia de Usuario
Duración	
Descripción	

Imagen 3.4: Formato de las Historias de Usuario

3.2.3 Diseño

En esta fase se definen estrategias que permiten proponer soluciones a los requerimientos descritos en la fase de planificación y así facilitar la programación a realizar en la fase de codificación. Para ello, se establece como entregable, prototipos de interfaces que permitan visualizar las funcionalidades de la aplicación esbozando su contenido y la disposición física de los elementos.

Se decide utilizar diagramas y la técnica de *Wireframing* que ofrece herramientas para esquematizar el diseño y contenido a presentar, generando así, una guía visual de los componentes de la aplicación (ver imagen 3.5).

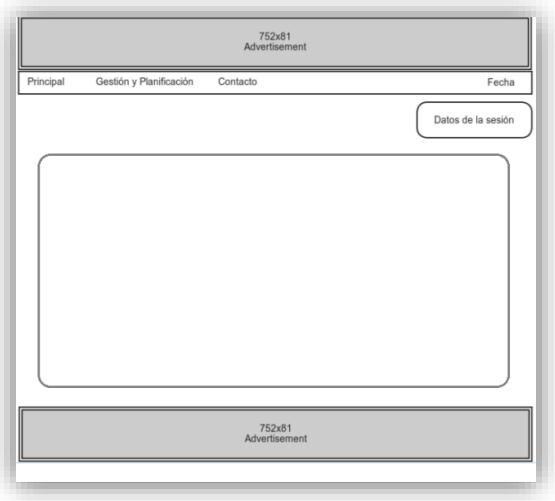


Imagen 3.5: Formato de prototipos (Wireframing)

3.2.4 Codificación

Luego de crear las Historias de Usuario, y realizar el prototipo correspondiente a cada una de ellas, el desarrollador se centra en lo que debe implementarse, para ello el método XP sugiere que 2 programadores trabajen en una misma computadora para complementar los conocimientos y asegurar la calidad de las soluciones.

Los entregables en esta fase son componentes de software totalmente funcionales, de los cuales se muestran extractos del código más relevante o las vistas resultantes, que luego, para verificar su correcto comportamiento se realizan pruebas en la siguiente fase.

3.2.5 Pruebas

En esta fase se comprueba que la aplicación desarrollada, o el componente de ella más recientemente desarrollado, se comporte como debería, es decir, se realizan pruebas funcionales para verificar la robustez de cada una de las funcionalidades implementadas. Las pruebas funcionales comprenden un conjunto de condiciones de entrada que permiten comprobar la salida esperada de todas las funcionalidades programadas, de acuerdo a los requerimientos del cliente.

Adicionalmente, se aplican pruebas unitarias para asegurar que un conjunto de funcionalidades actúen correctamente por separado, y para validar el comportamiento esperado de la aplicación como un todo, se realizan pruebas de integración. Estas últimas consisten básicamente en verificar que todos los módulos desarrollados se comporten correctamente de manera conjunta. En especial, aquellos componentes de software que se comunican con los sistemas existentes (*ACEIM* y *aTesT*).

Además de las pruebas mencionadas para verificar el correcto funcionamiento del software desarrollado, se llevan a cabo pruebas de aceptación para evaluar el grado de satisfacción del cliente con la aplicación.

Todas las fases descritas anteriormente conforman los procesos por los cuales debe pasar un proyecto que se desarrolla según las directrices que proporciona el método XP, y para cumplir con las premisas de adaptabilidad y entregas tempranas de software, promulgadas por la metodología ágil, éste desarrollo debe ser realizado por partes, es decir, por cada módulo o grupo de funcionalidades que comprenda la aplicación, se ejecutan todas las etapas de desarrollo mencionadas en un tiempo determinado, éste proceso se denomina iteración.

Por esta razón el proyecto se divide en iteraciones de aproximadamente 3 semanas de duración. Al comienzo de cada iteración los programadores seleccionan, las Historias de Usuario que serán implementadas.

A continuación se describen las distintas iteraciones y tareas llevadas a cabo durante el desarrollo de la aplicación para la gestión de pruebas de nivelación.

3.3 Iteraciones

El desarrollo de la aplicación se divide en bloques de trabajo para así descomponer las tareas necesarias para llevar a cabo el producto final en partes claramente identificables. Cada una de estas partes se llama iteraciones, las cuales agrupan el desarrollo de un conjunto de funcionalidades que dan respuesta a los siguientes requerimientos:

1. Almacenar preguntas especificando su tipo (*Grammar, Vocabulary, Reading, Writing* y *Listening*) con sus respectivas opciones de respuesta.
2. Almacenar audios y textos.
3. Modificar preguntas, audios y textos almacenados.
4. Buscar preguntas, audios y textos almacenados para generar un modelo de *prueba de nivelación*.
5. Automatizar la presentación de la *prueba de nivelación*.
6. Mostrar aleatoriamente las preguntas al presentar la *prueba de nivelación*.
7. Calcular automáticamente el puntaje obtenido al presentar la *prueba de nivelación* y mostrar el nivel asociado a éste.
8. Visualizar el *Writing* de cada estudiante que presentó la *prueba de nivelación*.
9. Poder cambiar el nivel resultante luego de presentar la *prueba de nivelación* aun cuando es asignado automáticamente.
10. Inscribir al estudiante en el nivel indicado y en el horario de su preferencia.

Estos requerimientos son detallados en Historias de Usuarios a lo largo de las 6 iteraciones que se realizan para la generación de artefactos de manera incremental, tomando en cuenta, para cada iteración, las 4 fases de desarrollo descritas en la sección 3.2. A continuación una descripción breve de las mismas:

- Iteración 0: Se refiere a la planificación inicial y a la preparación del entorno de desarrollo.
- Iteración 1: Se desarrolla un conjunto de funcionalidades que permiten al usuario contar con un módulo que gestione el contenido de la aplicación.
- Iteración 2: Se planifican las Historias de Usuario correspondientes a la creación de modelos de *prueba de nivelación*, generando como entregable el módulo de gestión de modelos.

- Iteración 3: Abarca la implementación de un módulo que permite al usuario la selección del modelo de *prueba de nivelación* de los modelos previamente creados, y la gestión de los estudiantes autorizados a presentar la *prueba de nivelación*.
- Iteración 4: Comprende el desarrollo de las funcionalidades que permiten a los estudiantes autorizados ingresar a la aplicación y presentar la *prueba de nivelacion*.
- Iteración 5: A partir de la iteración 1, en la fase de pruebas, se llevan a cabo pruebas funcionales y de integración que verifican el correcto comportamiento de las funcionalidades desarrolladas, además de sesiones de reunión con el cliente para obtener su opinión sobre los componentes de software entregados. Razón por la que en esta iteración sólo se aplican pruebas de aceptación para ponderar el grado de satisfacción de los usuarios con el sistema en general.

A continuación se detallan las iteraciones mencionadas, describiendo lo realizado en cada una de sus fases de desarrollo:

3.3.1 Iteración 0: Planificación Inicial

En esta iteración se conciben las primeras ideas de diseño de interfaces de usuario contemplando el desarrollo de las funcionalidades e interacciones con el cliente, además del diseño preliminar de la base de datos.

En el desarrollo de esta iteración se aplican las fases de planificación y diseño, las cuales se explican a continuación:

3.3.1.1 Planificación

En esta fase se establece las directrices a seguir para darle respuesta a los requerimientos del usuario, tal como se ilustra en la tabla 3.1.

Iteración 0	
Descripción	Elaboración de ideas conceptuales de la aplicación, instalación y configuración del entorno de desarrollo en Ruby on Rails e Interfaces principales
Duración de iteración	4 días

Historia	Duración	Tipo
Construcción de *WireFrames* para vistas de Usuario	8 horas	Nuevo
Diseño e implementación de base de datos	8 horas	Nuevo
Recolección de material básico para aprendizaje del Lenguaje	6 horas	Nuevo
Desarrollo de Interfaz de inicio y Autenticación	8horas	Nuevo
Desarrollo de página principal de profesor o Administrador	3 horas	Nuevo

Tabla 3.1: Planificación de la iteración 0

Estas tareas planificadas corresponden a las necesidades expresadas por el cliente y registradas en Historias de Usuario, tal como lo muestra la tabla 3.2

Titulo	**Construcción de *WireFrames* para vistas de usuario**
Duración	8 horas
Descripción	Esquematizar el diseño y contenido de la aplicación.
Titulo	**Desarrollo de Interfaz de inicio y autenticación**
Duración	4 horas
Descripción	La aplicación debe poseer una página de inicio que le presente a los distintos usuarios información relevante que le permita realizar diferentes funcionalidades. Además de módulo de autenticación que permita separar contenidos presentados dependiendo del usuario autenticado.
Titulo	**Diseño e implementación de Base de Datos**
Duración	8 horas
Descripción	Se debe diseñar e implementar una base de Datos que permita almacenar la información necesaria para la aplicación. Además, esto permite visualizar de forma general la información que se requiere para trabajar con la aplicación.

Titulo	Desarrollo de Interfaz principal de usuario autenticado
Duración	3 horas
Descripción	Se debe desarrollar la vista principal luego de que el usuario ha realizado la autenticación. Si es estudiante se le habilita la opción de inscribirse en la *prueba de nivelación*. Si es profesor o Instructor puede realizar otras tareas a nivel de administrador.

Tabla 3.2: Historias de usuario de la iteración 0

Además de la planificación inicial del proyecto, en esta iteración se lleva a cabo la configuración del entorno de programación, necesario para el desarrollo de la aplicación. La Configuración del entorno incluye principalmente, la selección de versiones, instalación y configuración de tecnologías como *Ruby, Rails, RVM, MySQL y Git*, todas bajo la plataforma *Linux.* Una vez instaladas, se procede entonces a la creación de tablas pertinentes al sistema e importación de datos en las mismas.

El siguiente paso abarca la carga del código fuente de *aTesT,* instalación de gemas necesarias, levantamiento del servidor y posterior ejecución con el fin de comprobar que la aplicación no arroja errores durante la misma.

3.3.1.2 Diseño

Una vez configurado el ambiente de desarrollo se procede con el diseño de la base de datos, identificando tablas, campos y tipos de datos. Entre las tablas creadas están las de preguntas, respuestas, modelo, opciones, entre otras. (Ver imagen 3.6)

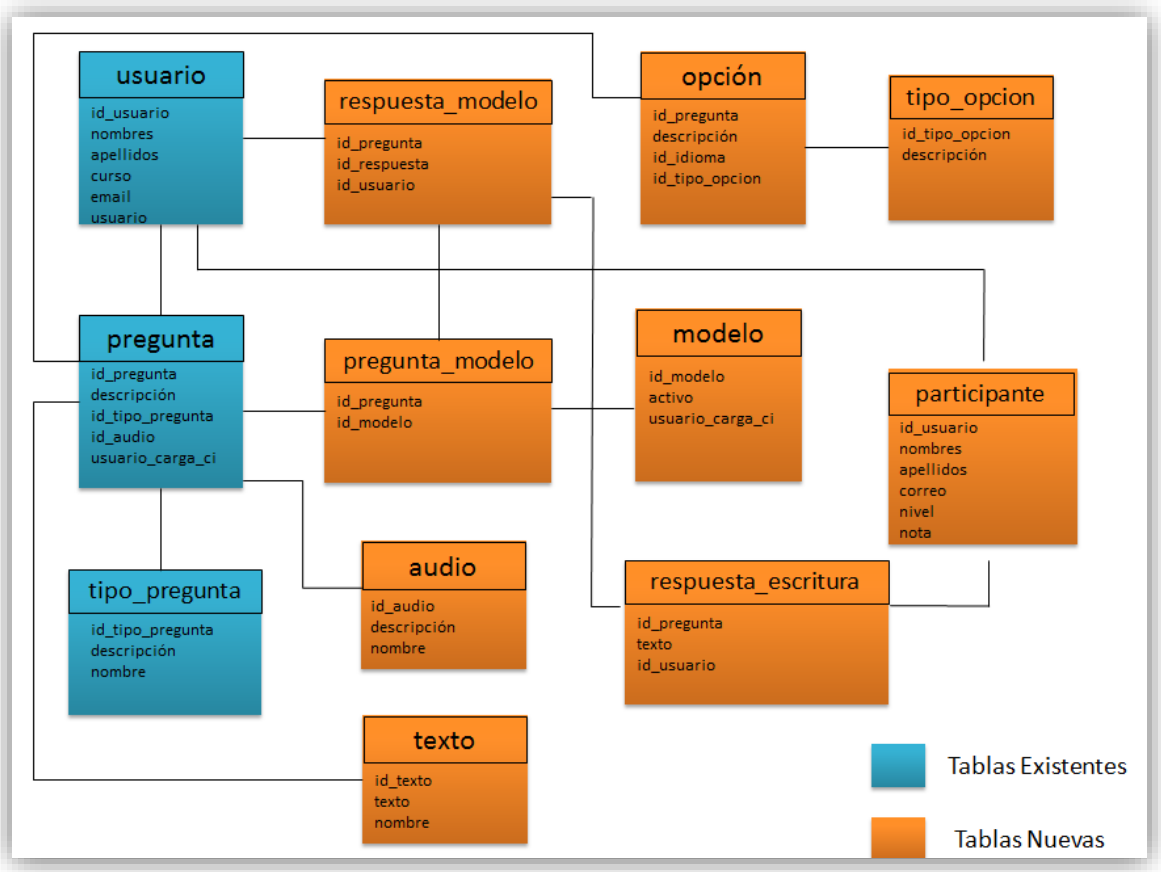


Imagen 3.6: Modelo Relacional de la Base de Datos

Además, se realiza el diseño de prototipos de interfaces de la *Aplicación Web*, construyendo así una guía de las funcionalidades a desarrollar en las iteraciones siguientes. Ejemplo de esto es la imagen 3.7, en la cual se puede observar la interfaz del menú de administrador una vez éste se haya autenticado en el sistema *ACEIM*, en ella se puede apreciar un nuevo enlace llamado *Prueba de Nivelación*, el cual dirige al usuario al sistema *aTesT* en dónde se encuentran todos los módulos correspondientes a la *prueba de nivelación*.

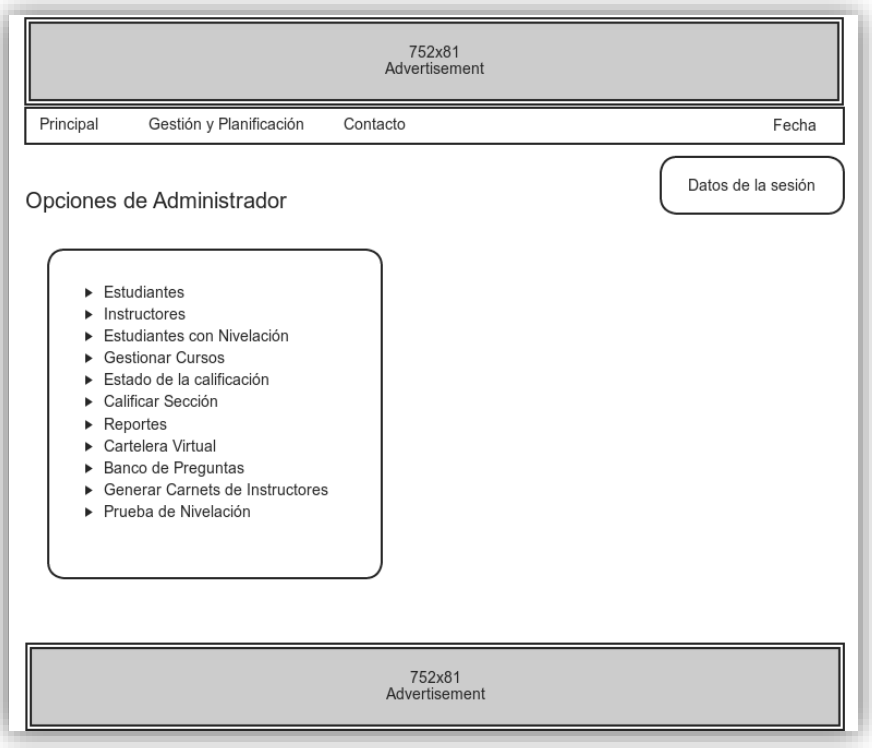


Imagen 3.7: Wireframe de la página principal del administrador ACEIM

3.1.2.3 Codificación

Para la creación de las estructuras de datos necesarias para almacenar la información relacionada a la aplicación, se decide utilizar una de las ventajas que ofrece *Ruby on Rails*, las migraciones. Las migraciones permiten la creación de tablas en el manejador de bases de datos *MySQL* usando código *Ruby*, generando, a través de un solo comando, todas las tablas indicadas con su respectiva estructura (ver imagen 3.8).

```
class CreateModelo < ActiveRecord::Migration
  def change
    create_table :modelo do |t|
      t.string :titulo
      t.string :tiempo
      t.boolean :activo
      t.string :usuario_carga_ci
      t.string :status
      t.string :pregpornivel
      t.timestamps
    end
  end
end
```

Imagen 3.8: Migración para la creación de la tabla Modelo

3.1.2.4 Pruebas

Con el fin de comprobar el cumplimiento de los objetivos planteados para esta iteración, se realizan pruebas funcionales creando cada una de las tablas identificadas en la fase de diseño mediante el uso de migraciones y verificando que efectivamente fueron creadas en la base de datos.

Adicionalmente, se realizan numerosas pruebas sobre la autenticación de usuarios, comprobando además que las validaciones de esta funcionalidad se realizan de manera correcta.

3.3.2 Iteración 1: Gestión de Contenido

Durante esta iteración se desarrolla el grupo de funcionalidades base para la aplicación: gestión de contenido, dónde el usuario puede agregar, editar o eliminar preguntas correspondientes a cada módulo de la *prueba de nivelación* (Listening, Gramática, Vocabulario, Lectura y Escritura).

Esta iteración comprende todas las fases de desarrollo (planificación, diseño, codificación y pruebas), las cuales se detallan a continuación:

3.3.2.1 Planificación

Se establecen las tareas a realizar durante esta iteración para desarrollar las funcionalidades que permitan satisfacer los requisitos del cliente, tal como se muestra en la tabla 3.3

Iteración 1		
Descripción	Desarrollo de la barra de menú, más las funcionalidades pertenecientes al ítem de menú Gestionar Contenido: agregar, editar y eliminar preguntas o audios.	
Duración de iteración	2 semanas	
Historia	**Duración**	**Tipo**
Construcción de *WireFrames* para vistas de Usuario	8 horas	Modificación

Barra de Menú	8 horas	Nuevo
Gestionar preguntas	42 horas	Nuevo
Gestionar audios	42 horas	Nuevo

Tabla 3.3: Planificación de la iteración 1

En éste sentido, se utilizaron las Historias de Usuario descritas en la tabla 3.4.

Titulo	**Construcción de *WireFrames* para vistas de usuario**
Duración	8 horas
Descripción	Prototipo de interfaz para las funcionalidades correspondiente a Gestionar Contenido (audios y preguntas).
Titulo	**Barra de Menú**
Duración	8 horas
Descripción	La aplicación debe facilitar la navegación de manera que el usuario pueda acceder a las principales opciones ofrecidas sin importar en que interfaz se encuentre.
Titulo	**Gestionar Preguntas**
Duración	42 horas
Descripción	Se deben desarrollar las funcionalidades de agregar, editar y eliminar preguntas de manera que desde una misma interfaz el usuario visualice las preguntas disponibles y pueda realizar las acciones mencionadas.
Titulo	**Gestionar Audios**
Duración	42 horas
Descripción	Se deben desarrollar las funcionalidades de agregar, editar y eliminar audios de manera que desde una misma interfaz el usuario visualice los audios disponibles y pueda realizar las acciones mencionadas.

Tabla 3.4: Historias de usuario de la iteración 1

3.3.2.2 Diseño

Los prototipos de interfaz realizados para esta iteración muestran una modificación de la iteración 0: la barra de menú, de tal forma que las opciones de éste módulo siempre estén accesibles al usuario desde cualquier parte de la aplicación.

También se diseña un prototipo de interfaz del sub-menú Audio, funcionalidad encargada de gestionar (agregar, editar, eliminar) los archivos de audios indicados por el usuario para luego poder ser usados en el módulo de *Listening* al crear un modelo de *prueba de nivelación*. (Ver imagen 3.9)

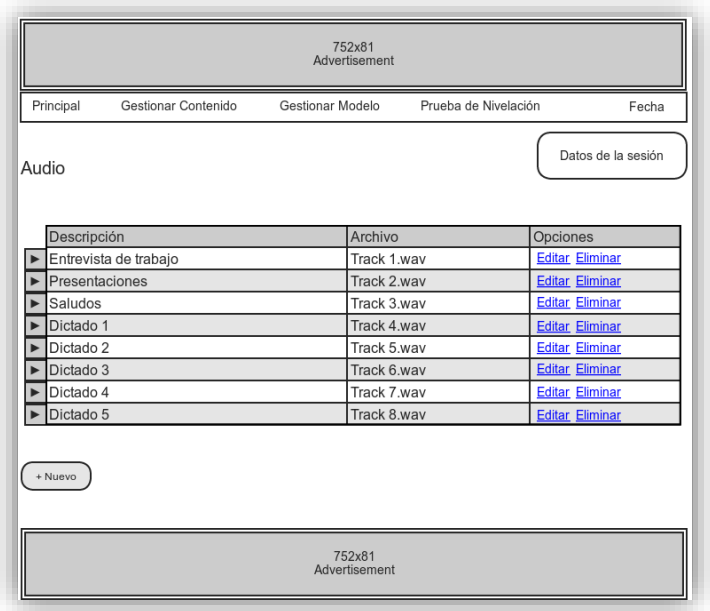


Imagen 3.9: Wireframe de Gestionar Contenido – Audio

De manera similar se creó el prototipo de interfaz del sub-menú Pregunta, funcionalidad encargada de gestionar (agregar, editar, eliminar) las preguntas indicadas por el usuario según su tipo (Listening, Gramática, Vocabulario, Lectura y Escritura) para luego poder ser usadas al crear un modelo de *prueba de nivelación*.

3.3.2.3 Codificación

Bajo la premisa de realizar la mayor cantidad de acciones posibles desde una misma vista, se utiliza el plug-in *DataTables* para el grupo de funcionalidades de Gestionar Contenido (preguntas y audios), siendo la acción de *agregar,* una de las principales funcionalidades en éste módulo, por lo que se ilustra un extracto de su código en la imagen 3.10.

```
def guardar_pregunta
  @pregunta = PreguntaNivelacion.new
  @pregunta.idioma_id = 'IN'
  @pregunta.tipo_categoria_id = params[:opcion][:pregunta][:categoria]
  @pregunta.descripcion = params[:opcion][:pregunta][:descripcion]
  @pregunta.audio_id = params[:opcion][:pregunta][:selectaudio]
  @pregunta.texto_id = params[:opcion][:pregunta][:selecttexto]
  @pregunta.usuario_carga_ci = session[:usuario].ci

  @tp = TipoPregunta.where(:id => params[:opcion][:pregunta][:tipopregunta]).first
  if @pregunta.descripcion.nil?
    flash[:mensaje] = "El campo de pregunta no debe ser vacío"
    redirect_to :action => "agregar_pregunta"
  end
  if @tp.nil?
    flash[:mensaje] = "Debe seleccionar un tipo de pregunta"
    redirect_to :action => "agregar_pregunta"
  else
      @pregunta.tipo_pregunta_id = @tp.id
      if @pregunta.tipo_pregunta_id =='LI' and Audio.all.empty?
          flash[:mensaje] = "Debe agregar un audio"
          redirect_to :action => "agregar_audio"
      return
  end
```

Imagen 3.10: Extracto del código de Gestionar Contenido

Adicionalmente, se desarrolla las opciones de visualizar, editar y eliminar las preguntas o audios listados utilizando el plug-in *DataTables*, el cual permite mostrar todo el contenido con paginación y búsqueda, tal como se muestra en la imagen 3.11.

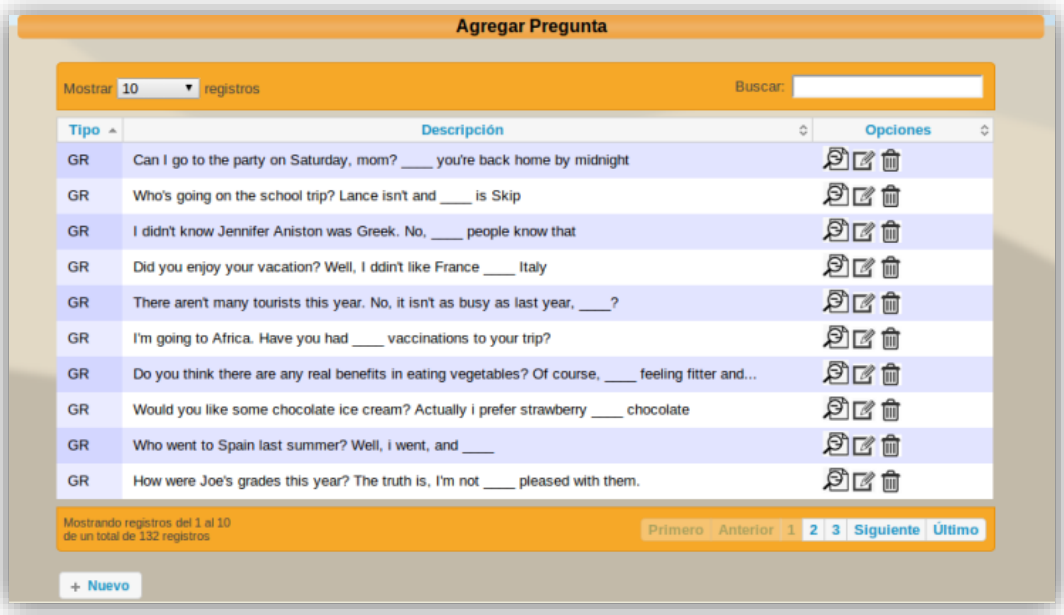


Imagen 3.11: Vista de Gestionar Contenido - Agregar Pregunta

3.3.2.4 Pruebas

La aplicación debe permitir agregar archivos de audio con su respectiva descripción; editar las descripciones de los archivos cargados, y eliminar aquellos que el usuario desee, por lo que se realizaron pruebas funcionales que verifican los procesos de almacenamiento, edición y eliminación de archivos de audio.

Otro resultado que se espera de la aplicación es poder agregar preguntas indicando las posibles opciones con su respuesta correcta, además de editar el contenido y eliminar aquella que el usuario desee; motivo por el cual se realizan pruebas funcionales almacenando varias preguntas de todos los tipos (Listening, Gramática, Vocabulario, Lectura y Writing) para asegurar el comportamiento esperado.

Como resultado de esta fase, en la imagen 3.12 se muestra el proceso selección de un archivo de audio.

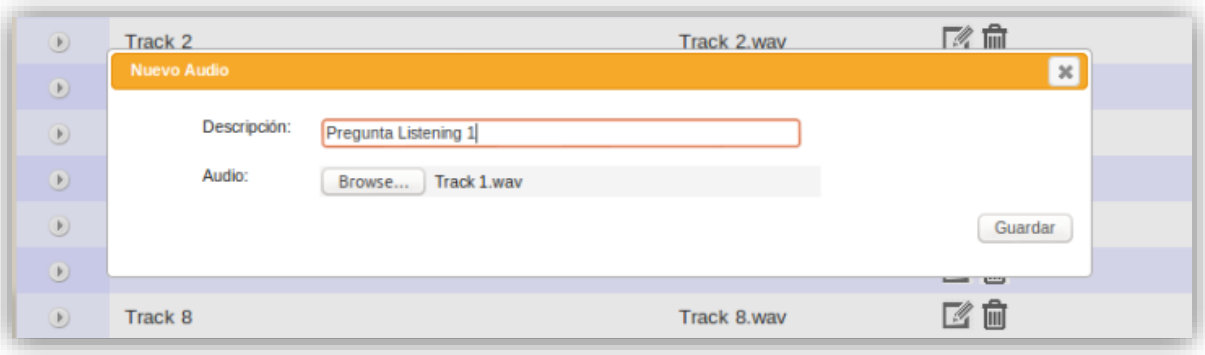


Imagen 3.12: Vista de Gestionar Contenido - Nuevo Audio

Una vez elegido el archivo de audio con la pregunta de *Listening*, se procede a la carga y posterior visualización en el listado de audios cargados. En la imagen 3.9 se observa que el audio fue cargado de manera exitosa.

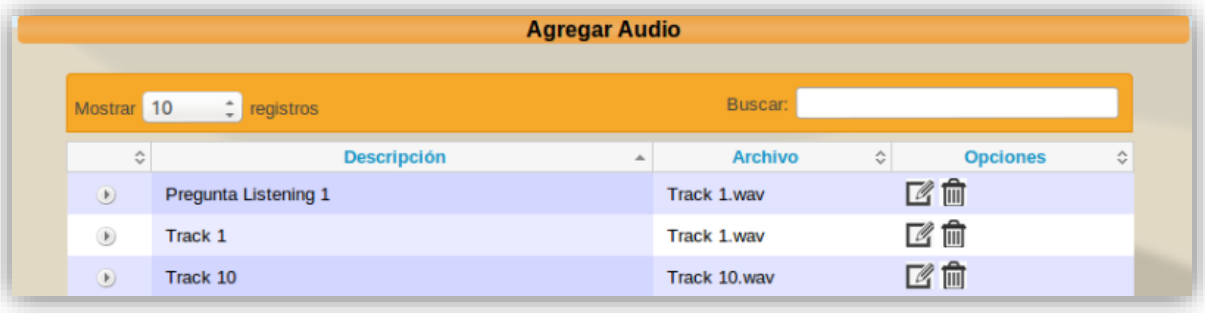


Imagen 3.13: Vista de Gestionar Contenido – Audio Agregado

3.3.3 Iteración 2: Gestión de Modelos de Pruebas de Nivelación

En esta iteración se desarrolla el principal grupo de funcionalidades para la aplicación que se denomina *gestión de modelo*, dónde el usuario puede crear, editar o eliminar modelos de *prueba de nivelación.*

Para asegurar el resultado esperado de éste ciclo de desarrollo se consideran las siguientes fases:

3.3.3.1 Planificación

En esta fase se describen los pasos a seguir que garanticen la programación de un módulo completamente funcional que responda a los requerimientos del cliente, tal como lo muestra la tabla 3.5

Iteración 2		
Descripción	Desarrollo de las funcionalidades pertenecientes al ítem de menú Gestionar Modelo de *prueba de nivelación*: crear, editar y eliminar modelos de *prueba de nivelación*.	
Duración de iteración	2 semanas	
Historia	**Duración**	**Tipo**
Construcción de *WireFrames* para vistas de Usuario	8 horas	Nuevo
Crear modelo de *prueba de nivelación*	32 horas	Nuevo
Vista previa	20 horas	Nuevo
Editar modelo de *prueba de nivelación*	32 horas	Nuevo

Tabla 3.5: Planificación de la iteración 2

Este plan de acción está soportado en las necesidades expresadas por el cliente y registradas en las historias de usuario que muestra la tabla 3.6.

Titulo	**Construcción de *WireFrames* para vistas de usuario**
Duración	8 horas
Descripción	Prototipo de interfaz para las funcionalidades correspondiente a Gestionar Modelo (crear y editar).
Titulo	**Crear modelo de *prueba de nivelación***
Duración	32 horas
Descripción	Se debe desarrollar un conjunto de funcionalidades que permitan listar las preguntas disponibles según su tipo, de manera que puedan ser seleccionadas y asociadas a la creación de un nuevo modelo de *prueba de nivelación*.

Titulo	Vista Previa
Duración	20 horas
Descripción	Se debe mostrar las preguntas hasta el momento seleccionadas de la forma en que el usuario final (estudiante de nivelación) las apreciaría.
Titulo	**Editar modelo de prueba de nivelación**
Duración	32 horas
Descripción	Se deben listar los modelos almacenados de manera que el usuario seleccione aquel al cual desea editar su contenido o eliminar.

Tabla 3.6: Historias de usuario de la iteración 2

3.3.3.2 Diseño

Los prototipos de interfaz realizados para esta iteración contemplan una guía visual de un asistente para la creación de modelos de la *prueba de nivelación*, permitiendo dividir en módulos o pasos el asistente de acuerdo al tipo de pregunta que se presenta (*Listening*, *Grammar*, *Vocabulary*, *Reading* y *Writing*), tal como se refleja en la imagen 3.14.

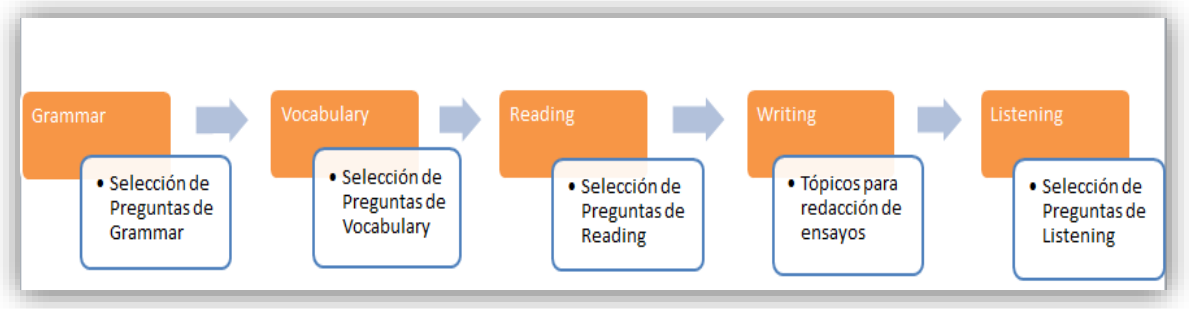


Imagen 3.14: Selección de preguntas del Modelo – Crear Modelo

Una de las funcionalidades que contiene la aplicación es la de Vista Previa, que al momento de crear un modelo, ofrece al usuario la posibilidad de apreciar cómo se mostraría a los estudiantes inscritos para presentar la *Prueba de Nivelación* las preguntas que ha seleccionado hasta ese momento; el prototipo de dicha funcionalidad se observa en la imagen 3.15.

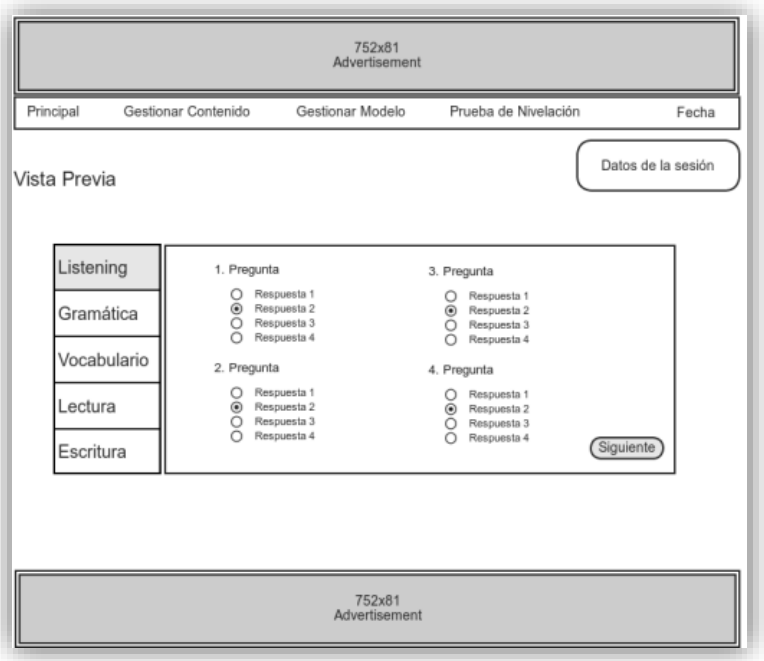


Imagen 3.15: Wireframe de Gestionar Modelo – Vista Previa

3.3.3.3 Codificación

En esta fase se usan los plug-in *SmartWizard* y *DataTables* para que las interfaces de Crear y Editar Modelo muestren, por cada módulo de la *prueba de nivelación*, el listado de las preguntas disponibles de manera que el usuario seleccione aquellas que desee asociar al modelo que está creando o editando (ver imagen 3.16). Además de contar con un paso inicial de configuración, en el cual se indica el nombre del modelo y el umbral de puntuación con su respectiva asignación de niveles.

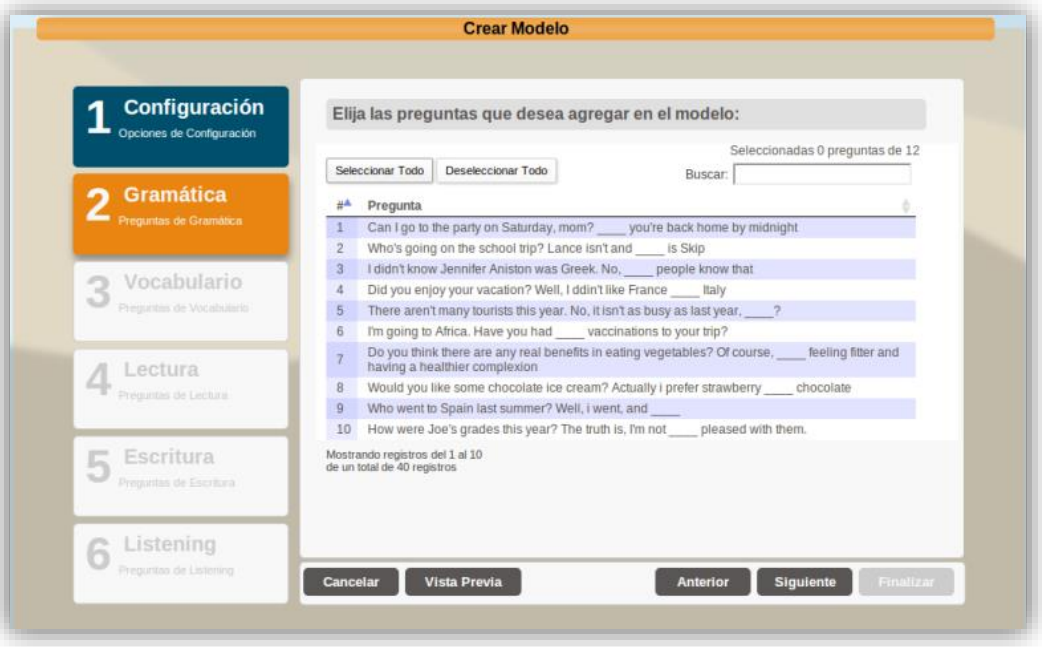


Imagen 3.16: Vista de Gestionar Modelo – Crear Modelo

Una de las acciones más relevantes desarrolladas en esta iteración es la de vista previa que, como se menciona en la fase anterior, permite visualizar la disposición física de las preguntas del modelo que se está creando (ver imagen 3.17).

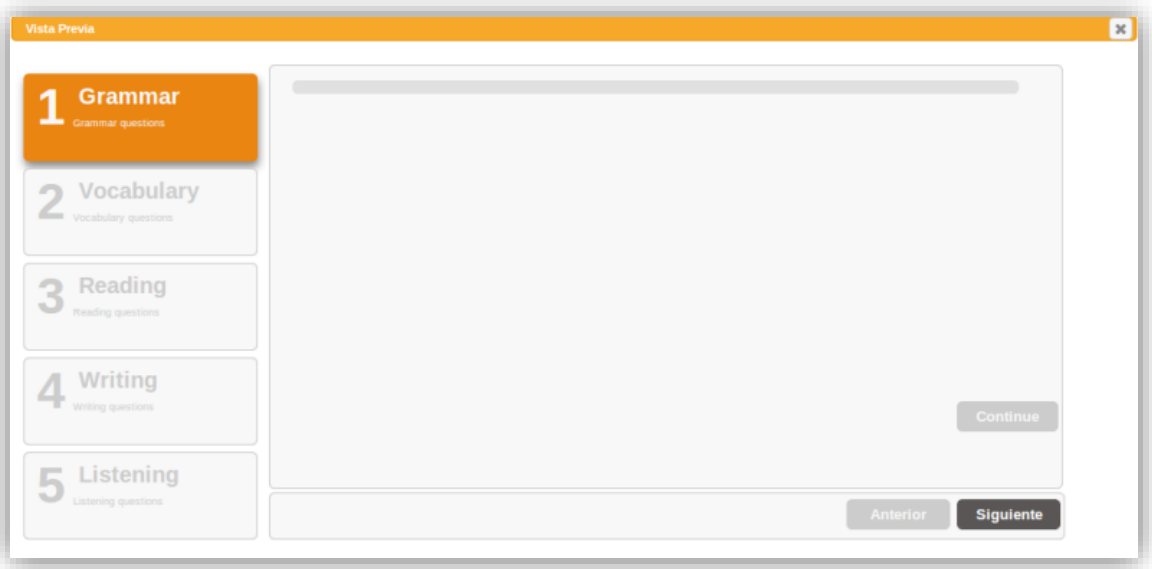


Imagen 3.17: Vista de Gestionar Modelo - Vista Previa

3.3.3.4 Pruebas

Para comprobar el cumplimiento de los objetivos planteados en esta iteración, se realizan pruebas funcionales creando decenas de modelos a fin de verificar el comportamiento de la funcionalidad de Vista Previa.

Adicionalmente, después de crear estos modelos existe la opción de modificar su contenido a través de la interfaz de Editar Modelo, en donde el usuario puede agregar y quitar preguntas o eliminar el modelo si así lo desea. Por consiguiente, éste conjunto de funcionalidades se ejecutaron repetidas veces y variando el orden de su ejecución para cada uno de los modelos creados de manera que se pudiese verificar la exactitud de las salidas. En la imagen 3.18 se observa la vista de Editar Modelo luego de haber creado un Modelo de *Prueba de Nivelación*.

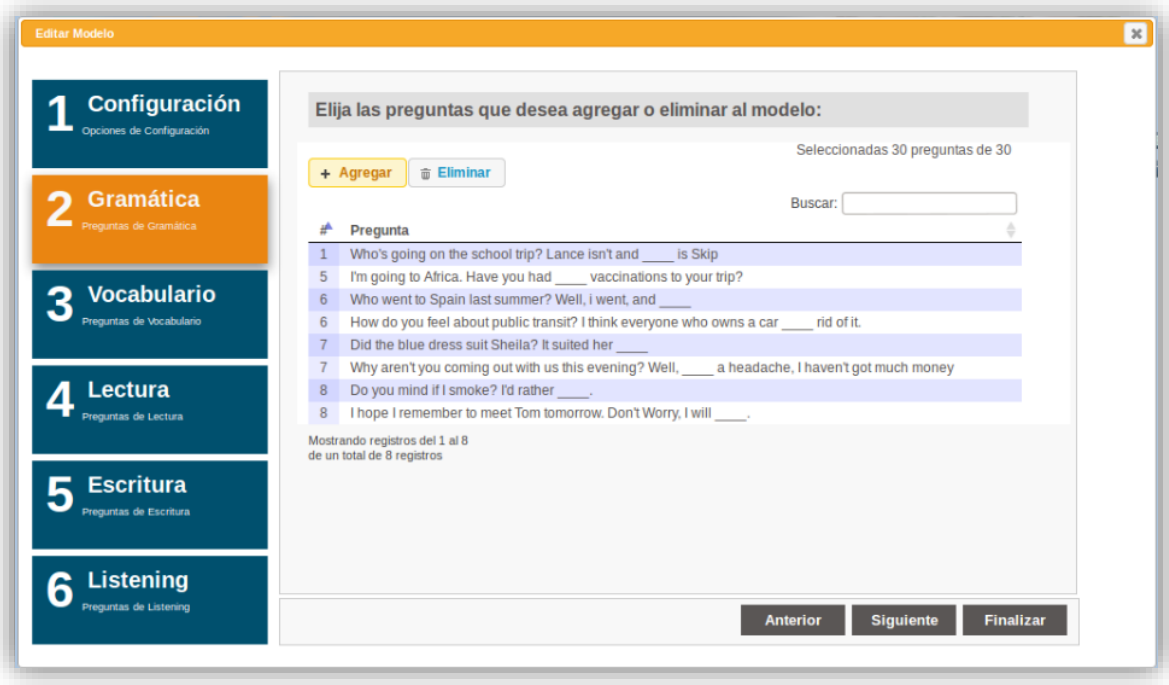


Imagen 3.18: Vista de Editar Modelo

3.3.4 Iteración 3: Gestión de Estudiantes de Nivelación

Durante esta iteración se desarrolla el grupo de funcionalidades inherentes a la *Prueba de Nivelación*: Seleccionar Modelo, dónde el usuario selecciona el modelo a ser utilizado como *Prueba de Nivelación* que será aplicado a los estudiantes inscritos para presentar la misma.

Adicionalmente, se desarrolla una interfaz para visualizar los estudiantes autorizados, la cual muestra información relevante a ellos: Datos personales, nota obtenida y nivel asociado.

Asimismo, se agrega un grupo de funcionalidades relacionadas a lo desarrollado en la iteración 1, pero no contemplada en la planificación de la misma, por lo que al identificar la necesidad de manejar los textos de la sección Reading de la *Prueba de Nivelación* igual que el contenido de la sección Listening, se agrega una interfaz más al grupo de funcionalidades de Gestionar Contenido (Audios, Preguntas y Textos).

Para garantizar el entregable estipulado en esta iteración se aplicaron las siguientes fases de desarrollo:

3.3.4.1 Planificación

La estructuración de las tareas a realizar en esta iteración para dar respuesta a los requerimientos del cliente se muestra en la tabla 3.7

Iteración 3		
Descripción	Agregar funcionalidades al ítem de menú Gestionar Contenido: agregar, editar y eliminar textos. Además del desarrollo del grupo de funcionalidades correspondientes al ítem de menú *Prueba de Nivelación*: Seleccionar Modelo.	
Duración de iteración	1 semana	
Historia	**Duración**	**Tipo**
Construcción de *WireFrames* para vistas de Usuario	8 horas	Modificación
Gestionar Textos	8 horas	Nuevo

Seleccionar Modelo	20 horas	Nuevo
Listar estudiantes autorizados	4 horas	Nuevo

Tabla 3.7: Planificación de la iteración 3

De acuerdo a la planificación mencionada, las funcionalidades a programar y las modificaciones a realizar en éste ciclo de desarrollo se basan en las Historias de Usuario mostradas en la tabla 3.8.

Titulo	**Construcción de *WireFrames* para vistas de usuario**
Duración	8 horas
Descripción	Prototipo de interfaz para las funcionalidades correspondiente a Gestionar Contenido (textos) y *Prueba de Nivelación* (autorización del modelo a utilizar y los estudiantes a participar).
Titulo	**Gestionar Textos**
Duración	8 horas
Descripción	Se deben desarrollar las funcionalidades de agregar, editar y eliminar textos de manera que desde una misma interfaz el usuario visualice los textos disponibles y pueda realizar las acciones mencionadas.
Titulo	**Seleccionar modelo de prueba de nivelación**
Duración	20 horas
Descripción	Se debe desarrollar una interfaz en la cual el usuario pueda elegir el modelo que desea utilizar para la *Prueba de Nivelación* que automáticamente autorice a los estudiantes de nivelación del período actual.
Titulo	**Listar estudiantes autorizados**
Duración	4 horas
Descripción	Se deben desarrollar las funcionalidades de buscar, agregar y eliminar estudiantes para el modelo de *Prueba de Nivelación* escogido.

Tabla 3.8: Historias de usuario de la iteración 3

3.3.4.2 Diseño

En esta fase se crea el prototipo de interfaz del sub-menú Texto, encargada de gestionar (agregar, ver, editar, eliminar) los textos indicados por el usuario para luego poder ser usados en el módulo de Reading al generar un modelo de *prueba de nivelación*.

Además, se realiza un bosquejo de las funcionalidades requeridas para la interfaz de usuario que lista los estudiantes autorizados a presentar la *prueba de nivelación*, tal como se ilustra en la imagen 3.19.

Imagen 3.19: Eventos asociados a la interfaz de Listar Estudiantes

3.3.4.3 Codificación

Para esta iteración fue necesario modificar el menú de Gestionar Contenido, a fin de agregar las funcionalidades pertinentes al manejo de los textos que son usados en el módulo de Reading, razón por la que se implementa el plug-in *DataTables*, el cual también se utiliza para listar los estudiantes autorizados a presentar la *Prueba de Nivelación* que previamente con la implementación del plug-in *SmartWizard*, asegura la selección de un modelo y los estudiantes con acceso a éste.

La funcionalidad más relevante desarrollada en esta iteración es la de listar los estudiantes autorizados a presentar la prueba, debido a que una vez presentada dicha prueba, la interfaz de usuario muestra la puntuación obtenida y permite revisar el *Writing,* modificar el nivel automáticamente asignado e inscribir al estudiante (ver imagen 3.20).

Imagen 3.20: Vista de Gestionar Estudiantes – Lista de Estudiantes

3.3.4.4 Pruebas

La aplicación debe permitir seleccionar un modelo de *prueba de nivelación,* de tal forma que estos usuarios autorizados (estudiantes) puedan presentar dicha prueba al ingresar al sistema. En éste sentido, se realizaron pruebas funcionales para validar el comportamiento de la interfaz Seleccionar Modelo, ejecutando distintas combinaciones de las funcionalidades que ofrece esta vista.

Finalmente, se agregan varios textos usando la vista homónima perteneciente al menú de Gestionar Contenido, la cual permite visualizar dichos textos además de editar y eliminar aquellos que el usuario seleccione.

En la imagen 3.21 se observa la vista en donde se eligen los modelos de la *prueba de nivelación* a ser aplicados, luego de la codificación y al momento de las pruebas.

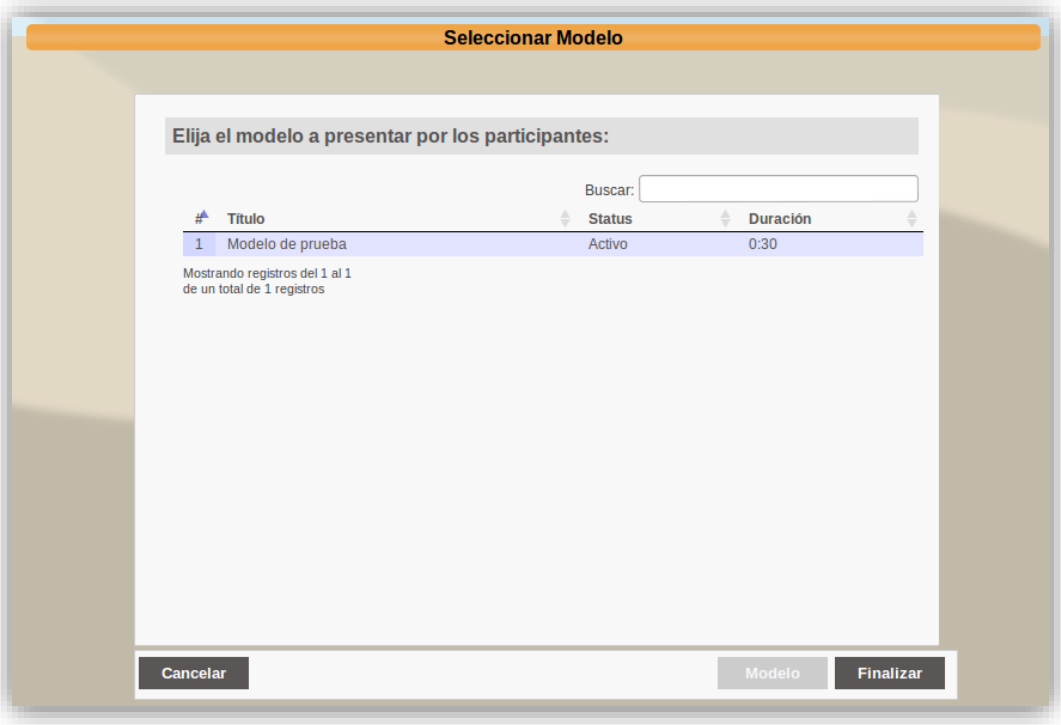


Imagen 3.21: Vista de Selección de Modelo de Prueba de Nivelación

3.3.5 Iteración 4: Presentar Prueba

En esta iteración se desarrolla la interfaz que interactúa con los usuarios autorizados a presentar la *Prueba de Nivelación*, los cuales, a través de un wizard, seleccionan las respuestas que consideren correctas a las preguntas presentadas por módulos (Listening, Gramática, Vocabulario, Reading y Writing) de manera que al finalizar la prueba, el puntaje obtenido y el nivel asociado a éste serán mostrados al Administrador en la vista de Listar Estudiantes.

En el desarrollo de esta iteración se emplean las fases de planificación, diseño, codificación y pruebas, las cuales se detallan a continuación:

3.3.5.1 Planificación

Esta fase establece los lineamientos a seguir para obtener un módulo completamente funcional que permita a usuarios autorizados presentar la *prueba de nivelación*, tal como se muestra en la tabla 3.9.

Iteración 4		
Descripción	Desarrollo de las funcionalidades pertenecientes al ítem de menú Presentar prueba *de nivelación*	
Duración de iteración	1 semana	
Historia	**Duración**	**Tipo**
Construcción de *WireFrames* para vistas de Usuario	8 horas	Nuevo
Presentar Prueba	32 horas	Nuevo

Tabla 3.9: Planificación de la iteración 4

Esta planificación responde a los requerimientos plasmados en las Historias de Usuario reflejadas en la tabla 3.10.

Titulo	**Construcción de *WireFrames* para vistas de usuario**
Duración	8 horas
Descripción	Prototipo de interfaz para las funcionalidades correspondiente al menú Presentar Prueba que sólo será visible para los usuarios autorizados a presentar la *Prueba de Nivelación*.
Titulo	**Presentar Prueba**
Duración	32 horas
Descripción	Se debe desarrollar una interfaz que permita a los estudiantes inscritos en nivelación, presentar la *Prueba de Nivelación*: Mostrar las preguntas por módulo (Listening, Gramática, Vocabulario, Reading y Writing) y almacenar las opciones seleccionadas por el usuario.

Tabla 3.10: Historias de Usuario de la iteración 4

3.3.5.2 Diseño

En esta etapa se realiza el prototipo de la interfaz del menú Presentar, cuya funcionalidad es mostrar las preguntas de la *Prueba de Nivelación* y almacenar las respuestas indicadas por el usuario, tal como se observa en la imagen 3.22.

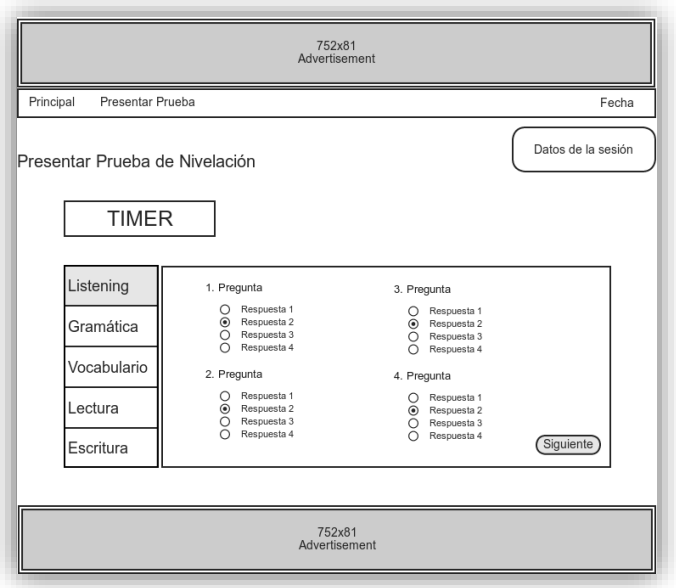


Imagen 3.22: prueba de nivelación de Presentar Prueba de Nivelación

3.3.5.3 Codificación

Para esta iteración se implementa el plug-in *SmartWizard* que permite presentar cada módulo de la *prueba de nivelación* en pasos o secciones para que el usuario pueda seleccionar la opción que considere correcta a cada una de las preguntas mostradas. Además, se utiliza el plug-in *TimeTo* para indicarle al usuario la duración de la prueba a través de un timer.

La acción más importante desarrollada en esta iteración es el guardar las respuestas de los estudiantes que estén presentando la *prueba de nivelación*, es por ello que en la imagen 3.23 se muestra un extracto de su código.

```
def guardar_respuesta
  usr= Participante.where(:usuario_ci => session[:usuario].ci).first
  modelo = usr.id_modelo
  tipo = params[:modulo]
  usr.update_attribute(:timer, params[:timer])
  arrayo = params[:opcion_id].to_s.split(',')
  arrayp = params[:preguntas_id].to_s.split(',')

  i=0
  while i<arrayp.length
    p = RespuestaModeloUsuario.new
    p.id_pregunta = arrayp.at(i)
    p.id_modelo = modelo
    if tipo == 'WR'
      w = RespuestaWriting.new
      w.id_pregunta = arrayp.at(i)
      w.texto = params[:texto]
      w.usuario_ci = session[:usuario].ci
      w.save
      p.id_respuesta_usuario = RespuestaWriting.last.id
    else
      p.id_respuesta_usuario = arrayo.at(i)
    end
    p.usuario_ci = session[:usuario].ci
    p.save
    i=i+1
  end
```

Imagen 3.23: Extracto del código de Presentar Prueba

3.3.5.4 Pruebas

Para comprobar el cumplimiento de los lineamientos establecidos en esta iteración, se realizan pruebas funcionales para verificar que el menú de *Presentar Prueba* esté accesible a los usuarios autorizados: Estudiantes inscritos para presentar la *Prueba Nivelación.*

También se realizan pruebas funcionales para asegurar que los estudiantes respondan de forma secuencial a todas las preguntas presentadas, siempre y cuando quede tiempo en el timer. Se muestran alertas si faltan preguntas por contestar, o se cierra la sesión y no continúa con la presentación de la prueba debido a que su duración expiró. En la imagen 3.24 se observa las alertas arrojadas por la aplicación para el caso donde el estudiante desee avanzar a la siguiente lista de preguntas sin haber contestado alguna.

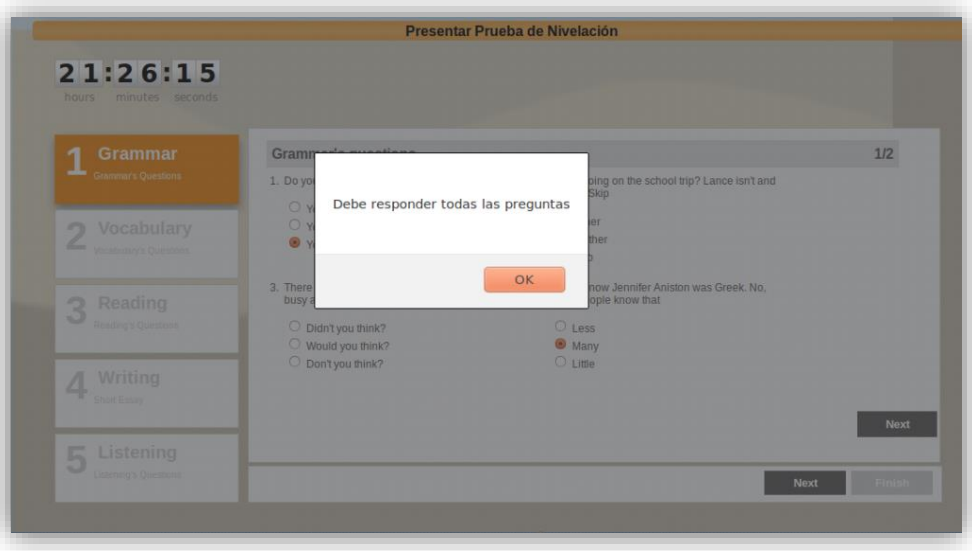


Imagen 3.24: Alerta – El estudiante desea avanzar sin contestar alguna pregunta

Por otra parte, también se alerta al usuario cuando el tiempo de la prueba ha expirado, la imagen 3.25 evidencia esta funcionalidad.

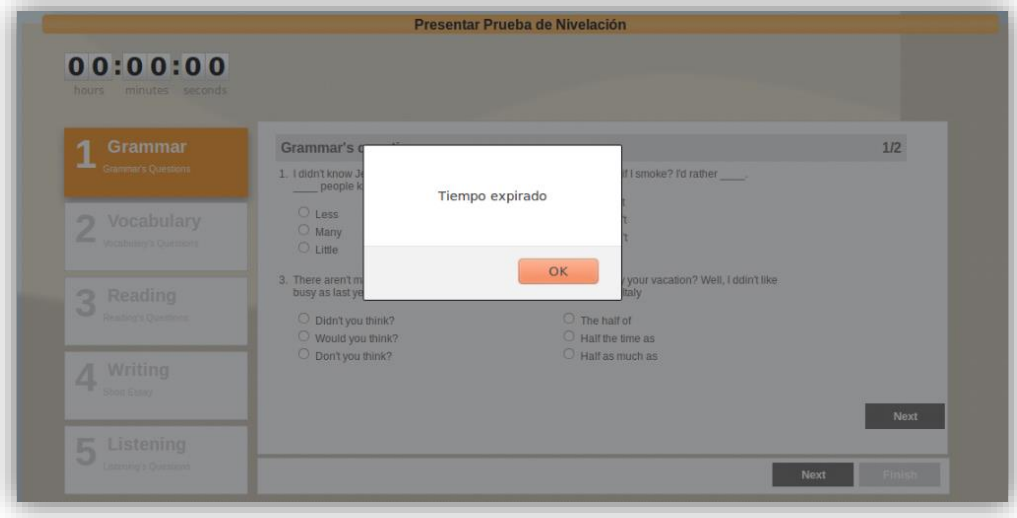
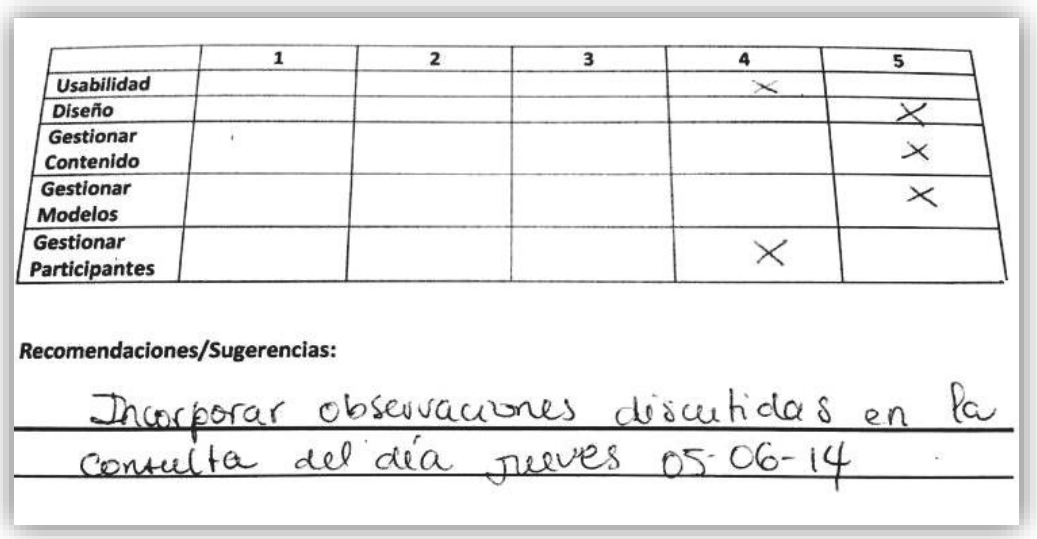


Imagen 3.25: Alerta – El estudiante ha agotado el tiempo para presentar la prueba

3.3.6 Iteración 5: Pruebas

Esta iteración se enfoca en la ejecución de diversos casos de prueba, que permiten determinar si el software desarrollado es parcial o totalmente satisfactorio para el cliente, para ello se realizaron pruebas de aceptación. Estas pruebas contemplan un grupo de criterios basados en las Historias de Usuario permiten evidenciar el grado de satisfacción del cliente con la aplicación desarrollada. Un ejemplo de ello, es la lista de comprobación que se observa en la imagen 3.26



	1	2	3	4	5
Usabilidad				X	
Diseño					X
Gestionar Contenido					X
Gestionar Modelos					X
Gestionar Participantes				X	

Recomendaciones/Sugerencias:

Incorporar observaciones discutidas en la consulta del día jueves 05-06-14

Imagen 3.26: Lista de comprobación aplicada al cliente

Esta lista de comprobación, respondida por la coordinadora de los cursos de idiomas, demuestra la receptividad del cliente al software entregado al medir su grado de satisfacción a través de una escala de apreciación (siendo 5 la máxima ponderación).

Es importante destacar que el principal usuario involucrado en el proceso de nivelación es la coordinadora de idiomas, quien manualmente corrige cada una de las pruebas aplicadas, es por ello que su aceptación total del diseño del software y de los módulos de gestión de contenido y de modelos, es indicador del cumplimiento de los objetivos trazados en éste trabajo.

El módulo de gestión de estudiantes y la usabilidad de la aplicación obtuvieron una ponderación de 4 en la escala anteriormente mencionada. Esto puede deberse a la transición de un proceso manual a uno automatizado, aunado al proceso de familiarización que debe realizar el cliente con las nuevas funcionalidades del sistema debido a que el módulo de gestión de estudiantes se comunica con el sistema *aTesT* para poder llevar a cabo la inscripción de estos estudiantes.

Esto se debe a la regla de negocio que estipula que el sistema *aTesT* es el encargado de la inscripción, planificación y gestión de los cursos, y el sistema *aTesT* sólo comprende la elaboración de los exámenes que se imparten.

Tomando en cuenta las sugerencias expresadas por el usuario, se realizan las modificaciones pertinentes para garantizar una aceptación completa de la aplicación. Además de generar una nueva lista de comprobación con mayor detalle y enfocada al módulo de Presentar Prueba, la cual es aplicada a 10 usuarios al azar y los resultados obtenidos se reflejan en la tabla 3.11.

Pregunta	**1**	**2**	**3**	**4**	**5**
¿Considera usted que la aplicación dispone de una navegación fácil de entender?				10%	90%
¿Considera usted que las funcionalidades de la aplicación son fáciles de reconocer?				10%	90%
¿Considera usted que la aplicación provee feedback a las acciones que realizó?				20%	80%
¿Considera usted que la aplicación posee una interfaz simple, sencilla y organizada?				10%	90%
¿Considera usted que la aplicación está diseñada para minimizar las posibilidades de cometer errores?				10%	90%
¿Considera usted que la aplicación utiliza un lenguaje consistente y apropiado?			20%	10%	70%

Tabla 3.11: Resultados de la lista de comprobación aplicada a 10 usuarios

Los resultados de la lista de comprobación permiten afirmar la aceptación del software desarrollado en la simulación de la presentación de la *prueba de nivelación*, debido a que 90% de los encuestados considera que las funcionalidades de la aplicación son fáciles de reconocer y que dispone de una navegación fácil de entender, además de poseer una interfaz simple, sencilla y organizada diseñada para minimizar las posibilidades de cometer errores; lo que se traduce a una aplicación intuitiva y robusta.

El resto de los resultados obtenidos reflejan las observaciones y/o sugerencias realizadas por los usuarios y descritas a continuación:

- Inconsistencia en el lenguaje utilizado. Se muestran partes en inglés y otras en español. Para éste punto, los usuarios expresan inconsistencias de lenguajes al momento de leer instrucciones y preguntas, ya que si bien las preguntas se expresan en inglés, las instrucciones se presentan es español, por lo que sugieren la homologación de escritos al inglés, ya que al ser una *prueba de nivelación* del inglés, los textos deben ser expuestos en éste idioma.

  Efectivamente como lo expresan los usuarios de la *prueba de nivelación*, existía una discrepancia de idiomas en la *prueba de nivelación*, en la imagen 3.27 se observa éste comportamiento.

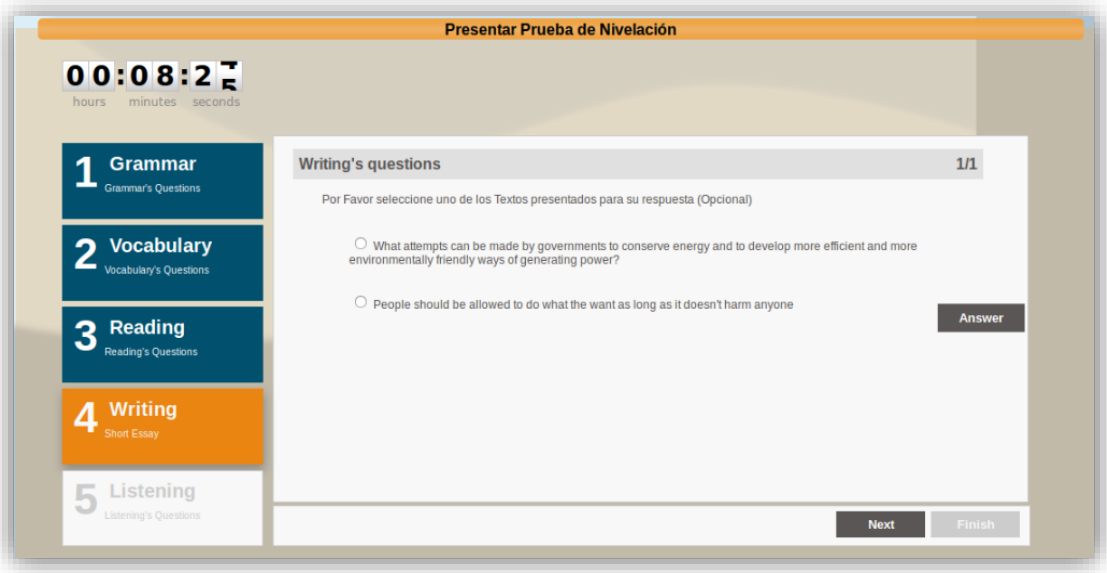


Imagen 3.27: Inconsistencia de inglés y español en la misma vista

- El mensaje de selección de horarios no está lo suficientemente explícito. Para esta observación, los usuarios al momento de realizar la selección de los dos horarios tentativos en los que desearían realizar el curso, manifiestan su confusión sobre qué horarios se referían y cuál era el propósito de los mismos; por lo que se les ofrece una breve explicación de la función de dichos horarios.

La imagen 3.28, muestra la veracidad de las observaciones realizadas por los distintos usuarios al momento de la selección de horarios.

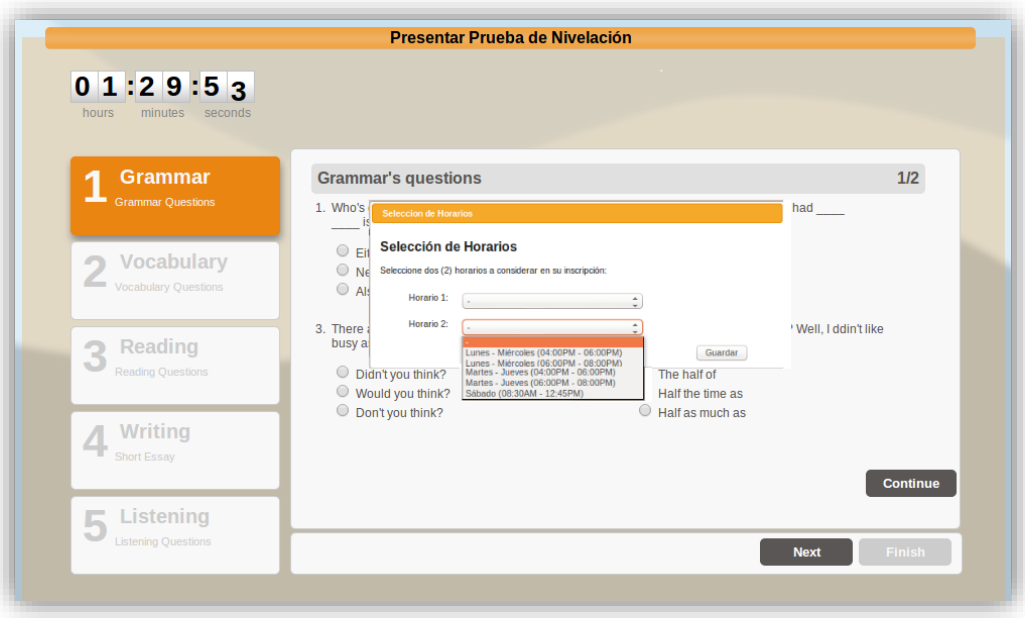


Imagen 3.28: Selección de Horarios Tentativos

Es por ello que se lleva a cabo las mejoras sugeridas, cambiando las indicaciones para presentar la *prueba de nivelación* mostradas en español a inglés; además de ampliar las instrucciones dadas para la selección horarios para así garantizar una total aceptación de la aplicación.

A continuación se muestra cada una de las correcciones realizadas luego de recibir las observaciones y sugerencias de parte de los usuarios de la prueba de nivelación.

En la imagen 3.20 se observa, que la inconsistencia del idioma en la presentación de texto informativo fue corregida.

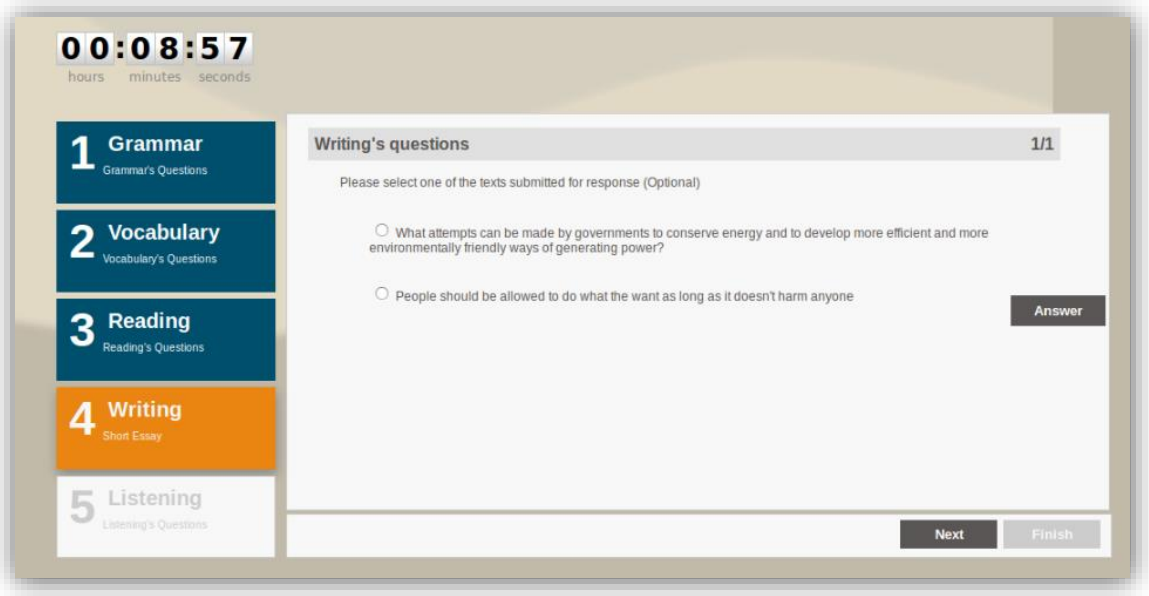


Imagen 3.29: Corrección de Textos Inglés-Español

Además, se hizo más explícito y claro el mensaje de la ventana de selección de horarios opcionales del curso (ver imagen 3.21).

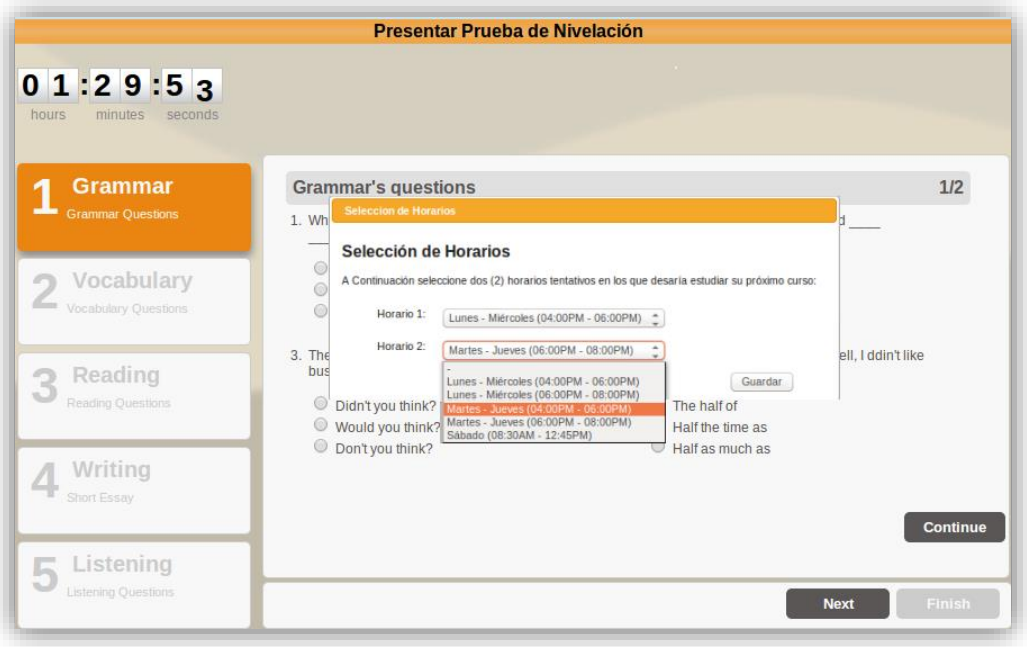


Imagen 3.30: Corrección de Mensaje de Selección de Horarios Tentativos

Conclusiones

Este Trabajo Especial de Grado surgió de la necesidad expresada por la Coordinación de Extensión de la Escuela de Idiomas Modernos de la Facultad de Humanidades y Educación de la Universidad Central de Venezuela de optimizar el proceso de nivelación de los cursos de inglés.

Es por ello que se planteó la automatización de la prueba de nivelación de inglés para dar respuesta a un proceso llevado manualmente que genera un alto costo en tiempo y recursos, razón por la cual se realizó un análisis exhaustivo de la situación actual que permitiera comprender la problemática existente para así determinar las bases teóricas y prácticas a seguir en el desarrollo de una solución a la necesidad evidenciada.

Esta solución se logró al desarrollar una aplicación web para automatizar la prueba de nivelación de inglés, cumpliendo así el objetivo principal de esta investigación. Para ello, se decidió configurar el método de desarrollo XP, principalmente, por su grado de detalle en la fase de planificación, debido a que propone como artefacto las Historias de Usuario, que permiten recopilar de forma concisa los requerimientos del usuario; hecho que facilitó la planificación inicial y por el cual se decidió agregar tablas descriptivas que indican las Historias de Usuario a considerar y el esfuerzo estimado para dar respuesta a cada una de ellas. De esta manera, se pudo dividir estratégicamente el desarrollo de la aplicación en iteraciones de acuerdo a los grupos de funcionalidades identificados en la planificación inicial. Cada iteración generó como salida un entregable o producto de software funcional desarrollado aplicando las fases de planificación, diseño, codificación y pruebas; a excepción de la primera y última iteración debido a que en estas sólo se preparó el entorno de desarrollo y se verificó el correcto comportamiento de la aplicación, respectivamente.

De la configuración del método *XP* se destaca la buena experiencia de la práctica de programación en pareja que permitió a los programadores tener mayor disciplina y compromiso con el desarrollo de la aplicación para así garantizar la calidad del trabajo. Esta práctica también sirvió para compensar la debilidad apreciada en la distribución de roles indicados en la propuesta original del método, debido a que éste plantea 7 responsabilidades en específico cada una asignada a un rol distinto, pero que en esta investigación, la mayoría de dichas responsabilidades se re-asignaron a los programadores por ser los únicos recursos disponibles, es por ello que el aplicar la programación en pareja permitió aligerar la carga de trabajo que esto suponía.

La aplicación desarrollada es parte de una plataforma existente, lo que significó un mayor esfuerzo en la curva de aprendizaje relacionada a las tecnologías ya utilizadas, y en a la integración de los nuevos componentes de software desarrollados. Pero el uso de las Tecnologías de Información y Comunicación (TIC) y de todas las herramientas descritas en el capítulo 2, permitieron construir un software con funcionalidades vanguardistas y garantizar la legibilidad del código al ser estándares mundiales facilitando así, el desarrollo de los módulos planteados para la aplicación.

A continuación se describen los módulos exitosamente desarrollados que permiten presentar la prueba de nivelación de forma automatizada:

- Gestión de contenido: Módulo usado por el administrador de la aplicación para agregar, editar y eliminar preguntas, audios o textos que serán usados al momento de crear modelos de la *prueba de nivelación*. Permitiendo así la personalización del contenido de la prueba; a diferencia del proceso llevado manualmente que comprende un único modelo de prueba sin poder cambiar las preguntas.
- Gestión de modelos: Conjunto de interfaces que ofrecen al usuario administrador las opciones de crear, editar y eliminar modelos de prueba. Ofreciendo así la posibilidad de crear más de un modelo de prueba para ser usado en distintos períodos de inscripción; a diferencia del proceso llevado manualmente que consta de un único modelo de prueba que se repite en cada período.
- Gestión de estudiantes de nivelación: El administrador utiliza éste módulo para seleccionar el modelo que será presentado por los estudiantes inscritos en nivelación. Luego, también puede visualizar todos los estudiantes con su puntación obtenida y el nivel correspondiente. De esta manera, el administrador evita el esfuerzo físico y las acciones repetitivas que conlleva la corrección de las pruebas y la asignación de niveles.
- Presentar prueba: Módulo habilitado sólo para los estudiantes de nivelación, el cual permite contestar cada una de las preguntas del modelo de prueba de nivelación previamente escogido por el administrador. Permitiendo así la aplicación de la *prueba de nivelación* de forma automatizada reduciendo los costos en tiempo y en recursos humanos que conlleva un proceso manual.

La existencia de otros sistemas (*ACEIM* y *aTesT*) hizo necesaria la integración del software desarrollado a dichos sistemas, debido que *ACEIM* es el sistema designado a la inscripción, planificación y gestión de los cursos; mientras que *aTesT* es el sistema

encargado de generar todos los exámenes posibles a realizar por la Coordinación de Extensión. Por esta razón, la aplicación desarrollada se agregó a la plataforma de *aTesT*, y adicionalmente se implementó un enlace *prueba de nivelación* en el menú del administrador en *ACEIM* para que dirija al usuario a la interfaz correspondiente en *aTesT*.

A través de la participación activa y la observación simple aplicada al personal administrativo durante varios procesos de nivelación, se puedo entender más de cerca los detalles de las actividades y procesos realizados. Involucrarse directamente con el problema, hizo más fácil su entendimiento y la búsqueda de soluciones para el mismo.

Una vez completado el desarrollo de la aplicación y su integración con las plataformas existentes, se realizaron las pruebas funcionales y de aceptación que demostraron el cumplimiento de los objetivos planteados y la satisfacción del usuario con el software entregado que, a través del uso de las tecnologías de información, fue puesto en producción, convirtiendo éste trabajo especial de grado en una solución a un problema de la vida real.

Recomendaciones y Trabajos futuros

El software desarrollado resuelve un problema específico aplicable a una necesidad generalizada, es decir, la Coordinación de Extensión de la Escuela de Idiomas Modernos de la Facultad de Humanidades y Educación de la Universidad Central de Venezuela realiza el proceso de nivelación manualmente para todos los cursos de idiomas que imparte, pero el alcance de esta investigación sólo contempló el idioma inglés.

Debido a que éste Trabajo Especial de Grado cumple con su objetivo de automatizar la *prueba de nivelación* de inglés, éste puede servir como base para futuros trabajos que deseen ampliar el alcance y automatizar las pruebas de nivelación para los demás idiomas ofertados.

Adicionalmente, para garantizar la calidad y vanguardia del software desarrollado en el tiempo, se requiere de un mantenimiento preventivo, adaptativo y correctivo, además de las posibles mejoras que a continuación se listan:

- Adjuntar imágenes en el sub-menú Textos en Gestionar Contenido, esto podría ser de utilidad al usuario si, por ejemplo, desea escanear un escrito de un libro.
- Módulo de reportes y/o estadísticas, éste podría indicar cuantos estudiantes han presentado la *prueba de nivelación* con el mínimo, el máximo y la media de las puntuaciones obtenidas para todos los modelos aplicados.
- Parametrizar el orden y título de los módulos de la *prueba de nivelación* para que pueda ser utilizado en otros idiomas.

Referencias Bibliográficas

Beck, K. (2004). *Extreme Programming Explained: Embrace Chage, 2nd. Edition.* Addison-Wesley Professional.

Chromatic. (2003). *Extreme Programming Pocket Guide.* O'Reilly Media.

CoffeeScript. (2014). Recuperado el 16 de 07 de 2014, de http://coffeescript.org

DataTables. (2014). Recuperado el 16 de 07 de 2014, de http://www.datatables.net

DocForge. (2014). Recuperado el 15 de 05 de 2014, de http://docforge.com

Gamma, E., M. Vlissides, J., Johnson, R., & Helm, R. (2012). *Design Patterns: Elements of Reusable Object-Oriented Software.* Prentice Hall.

HAML. (2014). Recuperado el 16 de 07 de 2014, de http://haml.info/

Janssen, C. (18 de Enero de 2012). *http://www.techopedia.com/*. Recuperado el 16 de Julio de 2014, de techopedia: http://www.techopedia.com/

Janssen, C. (2013). *Techopedia*. Recuperado el 15 de Julio de 2014, de http://www.techopedia.com/definition/24402/asynchronous-javascript-and-xml-ajax

Jquery. (2014). Recuperado el 15 de Julio de 2014, de http://jquery.com/

Jquery UI. (2014). Recuperado el 15 de Julio de 2014, de http://jqueryui.com/

Leff, A., & Rayfield, J. T. (2002). *Web-Application Development Using the Model/View/Controller Design Pattern.*

MacCaw, A. (2012). *The Little Book on CoffeeScript.* O'Reilly Media.

Microsoft Developer Network. (2014). Recuperado el 15 de 05 de 2014, de http://msdn.microsoft.com

Mora, S. L. (2002). *Programación de Aplicaciones Web.* Club Universitario.

Mozilla Developer Network. (4 de Julio de 2014). Recuperado el 15 de Julio de 2014, de https://developer.mozilla.org/en-US/docs/Web/JavaScript

O'Reilly Media. (2003). *Extreme Programming Pocket Guide.*

PC Magazine. (2014). Recuperado el 2014 de Julio de 15, de http://www.pcmag.com/encyclopedia/term/39803/client-server-architecture

Ruby on Rails. (05 de 2014). Recuperado el 15 de 05 de 2014, de http://www.rubyonrails.org

Sass: Syntactically Awesome Style Sheets. (2014). Recuperado el 16 de 07 de 2014, de http://sass-lang.com/

Tech Laboratory - Smart JQuery plugins. (2014). Recuperado el 16 de 07 de 2014, de http://www.techlaboratory.net

Webintenta. (2014). Recuperado el 16 de 07 de 2014, de http://webintenta.com/index.php