UNIVERSIDAD SAN FRANCISCO DE QUITO

Diseño y desarrollo de un sistema basado en Web que permite la automatización del procesamiento de becas de estudio: Sistema de Información de la Comisión Fulbright

Byron Santiago Calisto Ingavélez

Tesis de grado presentada como requisito
para la obtención del título de Ingeniería en Sistemas

Quito

Julio del 2007

Universidad San Francisco de Quito

Colegio Politécnico

HOJA DE APROBACIÓN DE TESIS

Diseño y desarrollo de un sistema basado en Web que permite la automatización del procesamiento de becas de estudio: Sistema de Información de la Comisión Fulbright

Byron Santiago Calisto Ingavélez

Lorena Balseca, M.S.
Directora de la Tesis …….

Fausto Pasmay, M.S., M.B.A.
Miembro del Comité de Tesis …….

Enrique Carrera, Ph.D.
Miembro del Comité de Tesis …….

Fernando Romo, M.S., M.B.A.
Decano del Colegio Politécnico …….

Quito, julio del 2007

DEDICATORIA

Dedico el presente trabajo a mis padres, Byron Alfonso Calisto Acosta y Gladys Angélica Ingavélez Ayala, quienes me apoyaron durante toda mi vida en todo aspecto, especialmente en mi decisión de seguir esta carrera que ahora concluye en su etapa más importante. Muchas gracias por haber estado a mi lado siempre y en todo momento.

AGRADECIMIENTOS

Agradezco a todo el personal de la Comisión Fulbright del Ecuador, en especial a su Directora, la Sra. Susana Cabeza de Vaca, Ph.D., quien me permitió desarrollar este proyecto además de brindarme su total apoyo durante el desarrollo del mismo.

RESUMEN

Este proyecto denominado “Diseño y desarrollo de un sistema Web que permite la automatización de procesamiento de becas de estudio” abarca el diseño y el desarrollo de un sistema, basado en tecnologías Web, que permite archivar en una base de datos los datos de aplicantes a becas y becarios de forma organizada en un servidor centralizado, y a la vez maneja los procesos de selección de aplicantes como becarios de programas de estudios ofrecidos por la Comisión Fulbright. La aplicación es accesible desde la red interna (*intranet*) de la Comisión Fulbright mediante un Navegador de la Web estándar. Se describe las tecnologías utilizadas, la mayoría de ellas gratuitas y de código abierto (*open source*), utilización de metodologías y patrones de diseño estandarizados, implementación del sistema, instalación y puesta en funcionamiento del mismo, pruebas de rendimiento general, y finalmente los manuales técnicos y de usuario con descripciones estructurales y funcionales del sistema.

ABSTRACT

This project, called “Design and development of a Web application for the automation of scholarship data processing”, is about the design and development of an application using Web technologies. This application has a database to store, in a central server, all the data related to applicants and scholarship grantees. It also handles the process of selection of grantees for the academic programs offered by the Fulbright Commission. The application can be accessed from any computer connected to the LAN (local area network) of the Fulbright Commission offices, through any standard web browser. This thesis describes the technologies used for the project (most of them free and open source); the use of methodologies and standardized design patterns; the development of the system; and the installation and deployment of the application. Finally, a technical manual and a user manual are included with the structural and functional descriptions of the application.

TABLA DE CONTENIDO

CUERPO DE LA TESIS

Capítulo 1
1.1 Introducción a la Tesis _____ 1
1.2 Sobre la Comisión Fulbright _____ 1
1.3 Motivación _____ 2
1.4 Propuesta _____ 3
Capítulo 2
2.1 Proceso de selección de Becarios _____ 4
2.1.1 Aplicación y apertura de concursos _____ 4
2.1.2 Preselección _____ 6
2.1.3 Entrevista y selección final _____ 9
2.2 Requerimientos del sistema _____ 12
2.2.1 Ingreso de aplicantes _____ 12
2.2.2 Ingreso y apertura de Programas _____ 13
2.2.3 Aplicación a Programas Abiertos _____ 14
2.2.4 Procesos de pre-selección y selección _____ 15
2.2.5 Mantenimiento de la aplicación _____ 17
2.2.6 Usuarios del sistema _____ 17
2.2.7 Importación de datos _____ 18
2.2.8 Aplicación e Interfaz de usuario _____ 18
Capítulo 3
3.1 Metodología de desarrollo _____ 19
3.1.1 Productos de la metodología en el desarrollo del proyecto _____ 20
3.1.1.1 Inicio _____ 20
3.1.1.2 Elaboración _____ 21
3.1.1.3 Construcción _____ 22
3.1.1.3.1 Primera iteración _____ 22
3.1.1.3.2 Segunda iteración _____ 24
3.1.1.4 Transición _____ 25
3.2 Ambiente de desarrollo _____ 25
3.3 Herramientas de desarrollo _____ 27
Capítulo 4
4.1 Instalación del software _____ 29
4.1.1 Instalación de la plataforma _____ 29
4.1.2 Instalación de la aplicación _____ 30
4.2 Interacción de los componentes _____ 32
4.2.1 Interacción entre cliente y servidor _____ 32
4.2.2 Interacción entre componentes internos _____ 33
4.3 Rendimiento del sistema _____ 35
4.3.1 Ambiente de pruebas _____ 36
4.3.2 Pruebas realizadas en el sistema _____ 37
Capítulo 5
5.1 Conclusiones _____ 39
5.2 Recomendaciones _____ 40

ANEXO 1 – MANUAL TÉCNICO

Público objetivo _____ 44
Objetivo del manual _____ 44
1. Instalación del sistema _____ 44
1.1 Estructura de directorios en la aplicación _____ 47
2. Descripción funcional del Producto _____ 49
2.1 Flujo general de pedidos y respuestas _____ 49
2.2 Procesos generales de manejo de datos _____ 52
2.3 Relación entre capa de datos y negocio _____ 58
2.4 Generador automático de tablas de resultados _____ 59
2.5 Procesos de pre-selección y selección final _____ 64
3. Descripción estructural del Producto _____ 69
3.1 Base de datos _____ 70
3.2 Descripción estructural de la aplicación _____ 72
3.2.1 Modelo _____ 73
3.2.1.1 Capa de datos _____ 74
3.2.1.2 Capa de lógica de negocio _____ 77
3.2.2 Vista _____ 83
3.2.3 Controlador _____ 86

ANEXO 2 – MANUAL DE USUARIO

Público objetivo _____ 94
Objetivo del manual _____ 94
1. Acceso a la aplicación _____ 94
2. Página de autenticación (*Login*) _____ 95
3. Página principal _____ 95
4. Opciones del Menú Principal _____ 96
4.1 Submenú *Applications* _____ 97
4.1.1 *New Application* (Nueva aplicación) _____ 97
4.1.1.1 Información del aplicante/becario _____ 99
4.1.1.1.1 Subsección *General Information* _____ 99
4.1.1.1.2 Subsección *Academic Background* _____ 100
4.1.1.1.2.1 *Add/update university/institution* _____ 101
4.1.1.1.3 Subsección *Exams* _____ 101
4.1.1.1.3.1 *Add exam* (Añadir examen) _____ 102
4.1.1.1.3.2 *Add exam score* (Añadir puntaje) _____ 102
4.1.1.1.4 Subsección *Programs and Grants* _____ 103
4.1.1.1.4.1 *Add/update program* _____ 105
4.1.1.1.4.2 *Edit Grants* _____ 106
4.1.2 *Search Applicants* (Buscar Aplicantes) _____ 107
4.1.3 *Pre-selection* (Pre-selección) _____ 109
4.1.3.1 Página de selección de programa _____ 109
4.1.3.2 Lista de aplicantes al programa _____ 109
4.1.3.2.1 Detalles de aplicación a programa _____ 110
4.1.3.2.1.1 Ingreso de documentos _____ 111
4.1.3.3 Eliminación previa _____ 111

4.1.3.4 Cálculo de *rankings* para Pre-selección _____ 112
4.1.4 *Selection* (Selección) _____ 113
4.1.4.1 Página de selección de programa _____ 113
4.1.4.2 Lista de aplicantes al programa _____ 114
4.1.4.3 Cálculo de *rankings* finales _____ 114
4.2 Submenú *Programs* _____ 115
4.2.1 *Open Programs* (Programas abiertos) _____ 116
4.2.1.1 *Open a program* (Abrir un programa) _____ 116
4.2.1.2 *Future programs* (Programas futuros) _____ 117
4.2.1.3 *Closed programs* (Programas cerrados) _____ 117
4.2.2 *All Programs* (Todos los programas) _____ 118
4.2.2.1 *Add/update program* (Añadir/modificar programa) _____ 119
4.2.3 *Agencies* (Agencias) _____ 119
4.2.3.1 *Add/update agency* (Añadir/modificar agencia) _____ 120
4.2.4 *Pre-programs* (Programas Pre-académicos) _____ 120
4.2.4.1 *Add/update pre-program* (Añadir/modificar pre-programa) _____ 121
4.3 Submenú *Reports* _____ 121
4.3.1 *Report Generator* (Generador de Reportes) _____ 122
4.4 Submenú *Administration* _____ 122
4.4.1 *Manage Users* (Administrador de Usuarios) _____ 123
4.4.2 *Exam types* (Tipos de exámenes) _____ 123
4.4.2.1 *Add/update exam type* (Añadir/modificar tipo de examen) _____ 124
4.4.2.1.1 *Add/update score type* (Añadir/modificar tipo de puntaje) _____ 124
4.4.3 *Degrees* (Títulos/grados) _____ 125
4.4.3.1 *Add/update degree* (Añadir/modificar título/grado) _____ 126
4.4.4 *Grant types* (Tipos de Concesiones) _____ 126
4.4.5 *Fields of study* (Campos de estudio) _____ 127
4.4.5.1 *Add/update field category* (Añadir/modificar categoría) _____ 127
4.4.5.1.1 *Add/update field subcategory* (Añadir/modificar subcategoría) _____ 128
4.4.5.1.1.1 *Add/update field of study* (Añadir/modificar campo de estudio) _____ 128
5. Salida del sistema (*Logoff*) _____ 129
6. Glosario _____ 129

LISTA DE FIGURAS

Figura 2.1 – Apertura de concursos y recepción de documentos _____ 6
Figura 2.2 – Preselección _____ 9
Figura 2.3 – Selección final _____ 11
Figura 3.1 – Diagrama general de casos de uso _____ 21
Figura 3.2 – Casos de uso para iteración 1 _____ 22
Figura 3.3 – Diagrama de objetos de la aplicación _____ 23
Figura 3.4 – Casos de uso para iteración 2 _____ 24
Figura 4.1 – Interacción de los componentes de la aplicación _____ 33
Figura 4.2 – Diagrama de interacción de capas _____ 34
Figura 4.3 – Diagrama detallado de comunicación de subcapas de Modelo _____ 35

ANEXO 1 – MANUAL TÉCNICO

Figura 1.1 – Fijación del *Classpath* del manejador JDBC _____ 45
Figura 1.2 – Creación del *Connection Pool* _____ 46
Figura 1.3 – Creación de la referencia JNDI de la conexión a la base de datos ____ 47
Figura 2.1 – Flujo general de pedidos y respuestas _____ 49
Figura 2.2 – Proceso de pre-selección _____ 64
Figura 2.3 – Proceso de selección _____ 66
Figura 3.1 – Diagrama ER simplificado de la base de datos _____ 70
Figura 3.2 – Diagrama de Clases _____ 73
Figura 3.3 – Diagrama de Modelo _____ 74

ANEXO 2 – MANUAL DE USUARIO

Figura 1.1 – Ejemplo de dirección de acceso _____ 94
Figura 2.1 – Página de autenticación _____ 95
Figura 3.1 – Página principal _____ 96
Figura 4.1 – Formulario de ingreso de datos de Nuevo Aplicante _____ 97
Figura 4.2 – Pestañas de selección de subsección _____ 99
Figura 4.3 – Información General de Aplicante _____ 99
Figura 4.4 – Subsección de Perfil Académico _____ 100
Figura 4.5 – Añadir/modificar Perfil académico _____ 101
Figura 4.6 – Subsección de Exámenes _____ 101
Figura 4.7 – Añadir un examen _____ 102
Figura 4.8 – Añadir un puntaje de examen _____ 103
Figura 4.9 – Subsección de Programas y Concesiones Económicas _____ 104
Figura 4.10 – Añadir o modificar aplicación a un programa _____ 105
Figura 4.11 – Cuadro de selección de campo de estudio _____ 106
Figura 4.12 – Editar Concesiones Económicas _____ 106
Figura 4.13 – Página de búsqueda de aplicantes _____ 107
Figura 4.14 – Página de resultados de búsqueda de aplicantes _____ 108
Figura 4.15 – Página de selección de programa para Pre-selección _____ 109
Figura 4.16 – Lista de aplicantes al programa _____ 110
Figura 4.17 – Detalles de aplicación al programa _____ 110

Figura 4.18 – Formulario de ingreso de documentos _____ 111
Figura 4.19 – Pre-eliminación _____ 112
Figura 4.20 – Lista con rankings para pre-selección _____ 112
Figura 4.21 – Lista de aplicantes pre-seleccionados y eliminados _____ 113
Figura 4.22 – Página de selección de programa para Selección Final _____ 113
Figura 4.23 – Lista de aplicantes pre-seleccionados para selección final _____ 114
Figura 4.24 – Lista de rankings para selección final _____ 115
Figura 4.25 – Lista de seleccionados y eliminados _____ 115
Figura 4.26 – Lista de programas abiertos _____ 116
Figura 4.27 – Formulario de apertura de programa _____ 116
Figura 4.28 – Lista de programas futuros _____ 117
Figura 4.29 – Lista de programas cerrados _____ 117
Figura 4.30 – Lista de todos los programas académicos _____ 118
Figura 4.31 – Formulario de ingreso de programa _____ 118
Figura 4.32 – Lista de agencias _____ 119
Figura 4.33 – Formulario de ingreso de agencia _____ 119
Figura 4.34 – Lista de programas pre-académicos _____ 120
Figura 4.35 – Formulario de ingreso de pre-programas _____ 120
Figura 4.36 – Pantalla de selección de reportes _____ 121
Figura 4.37 – Ejemplo de Reporte _____ 121
Figura 4.38 – Administración de Usuarios _____ 122
Figura 4.39 – Lista de exámenes _____ 123
Figura 4.40 – Formulario de tipo de examen con sus respectivos tipos de puntaje 123
Figura 4.41 – Formulario de tipo de puntaje _____ 124
Figura 4.42 – Lista de títulos/grados del sistema _____ 125
Figura 4.43 – Formulario de título/grado _____ 125
Figura 4.44 – Lista de Concesiones Económicas _____ 126
Figura 4.45 – Formulario de tipo de concesión _____ 126
Figura 4.46 – Lista de categorías de campos de estudio _____ 127
Figura 4.47 – Formulario de categoría con sus respectivas subcategorías _____ 127
Figura 4.48 – Formulario de subcategoría con sus respectivos campos de estudio 128
Figura 4.49 – Formulario de campo de estudio _____ 129

TABLAS

Tabla 3.1 – Costos estimados del proyecto _____ 20
Tabla 4.1 – Tiempos obtenidos en pruebas _____ 37

SEGMENTOS DE CÓDIGO

Segmento 2.1: Filtrado de pedidos en Authentication.java _____ 50
Segmento 2.2: Despachador de acciones en Controller.java _____ 51
Segmento 2.3: Clase base Action.java _____ 52
Segmento 2.4: Métodos principales de DBOperations.java _____ 53
Segmento 2.5: Clase DataSetWrapper.java _____ 56
Segmento 2.6: Método doSearch() en TableQuery.java _____ 60
Segmento 2.7: Método convertDataToTable() en DataTable.java _____ 60
Segmento 2.8: Clase CommonAction.java _____ 61

Segmento 2.9: Cálculo de *ranking* en PreSelectionModel.java _____ 67
Segmento 2.10: Cálculo de *ranking* en SelectionModel.java _____ 68

CAPÍTULO 1

1.1 Introducción a la Tesis

La presente tesis tiene como objetivo la creación de una aplicación que será utilizada al interior de la Comisión Fulbright, una institución del Gobierno de los Estados Unidos que provee becas para estudios de post-grado en universidades de ese país. Esta aplicación consistirá en una base de datos con toda la información de becarios y aplicantes, nuevos y antiguos, herramientas para el mantenimiento de esta información, y manejo de los procesos de selección de aplicantes para becas. La aplicación mantendrá toda esta información en un servidor centralizado, que podrá ser accedido de forma simultánea por múltiples usuarios a través de cualquier computador que posea un Navegador de la Web (Web Browser) y que se encuentre conectado a la red interna de la Comisión Fulbright.

1.2 Sobre la Comisión Fulbright

La Comisión Fulbright del Ecuador es una fundación sin fines de lucro auspiciada por el Gobierno de los Estados Unidos de América para el intercambio educativo entre el Ecuador y los Estados Unidos. Desde 1956 más de 1400 ecuatorianos han viajado a universidades de los Estados Unidos para realizar sus estudios de post-grado o ejercer la docencia, y más de 700 ciudadanos estadounidenses han venido al Ecuador para realizar sus trabajos de investigación, gracias al Programa Fulbright. Es un programa de prestigio nacional e internacional, ser un Becario Fulbright o “Fulbrighter” es un reconocimiento a la capacidad intelectual y de liderazgo de la persona. Actualmente, la Comisión Fulbright recibe el apoyo económico para becas a través del Departamento de Estado de los EEUU y de empresas privadas ecuatorianas, el gobierno ecuatoriano no realiza ningún aporte. La

Comisión Fulbright es conocida también por su Programa de Enseñanza de Inglés o ETP (English Teaching Program), este programa ofrece perfeccionamiento del idioma inglés a todas aquellas personas que posean un nivel intermedio del mismo; los cursos son dictados por profesores norteamericanos hablantes nativos del idioma, y también por profesores ecuatorianos especializados en la enseñanza del inglés. Fulbright también ofrece una biblioteca abierta al público, esta biblioteca posee libros de preparación para los diferentes exámenes que son requeridos por las universidades en EEUU y otros países (TOEFL, GRE, GMAT, SAT, ACT, US-MLE, etc.), libros informativos de universidades en EEUU, y dos computadoras con acceso a Internet y con programas que complementan la preparación de los exámenes anteriormente citados.

1.3 Motivación

La Comisión Fulbright maneja una cantidad considerable de información, que se actualiza y aumenta cada vez que se abre un nuevo concurso de selección de becarios. Esta información se encuentra en hojas de Microsoft Excel y bases de datos de Microsoft Access, dentro de un servidor de archivos. Toda esta información, a pesar de estar en un mismo servidor, se encuentra de manera dispersa. Cada usuario tiene sus propios archivos, lo que provoca duplicación innecesaria de la información. No existen esquemas de organización, lo cual hace que encontrar información sea un proceso difícil, requiriendo a veces recurrir a las gavetas de archivos físicos. Los procesos de preselección y selección de becarios, aunque están bien delineados y organizados, no poseen un sistema automatizado que permita organizar la información producida por los mismos. Por estas razones, el presente proyecto propone la realización de un sistema, desarrollado con tecnologías Web que permiten la centralización y organización de la información en servidor (o servidores)

que luego puede ser accedida rápida y eficientemente por cualquier computador que tenga un navegador Web (*browser*).

1.4 Propuesta

Este proyecto propone el desarrollo de un sistema de control de tareas en el proceso de selección de becarios Fulbright. Resolverá principalmente el problema de la desorganización y dispersión de la información. Se diseñará una base de datos que almacenará la información generada por todos los procesos de selección descritos anteriormente, además de información histórica residente en los actuales archivos. El sistema permitirá el manejo fácil y ágil de toda la información. Será desarrollado mediante tecnologías Web, que centralizan la aplicación en un servidor, la cual es accedida por los clientes utilizando solamente un Web Browser (navegador Web) como Mozilla o Microsoft Internet Explorer, sin necesidad de hardware o software adicional. Las tecnologías Web permiten que los clientes puedan ser de cualquier plataforma (Windows, Linux, Mac OS X, etc.), que varios clientes puedan acceder al sistema simultáneamente, y que la interfaz de usuario sea la ya conocida para navegar en el Web, lo cual hace que los usuarios aprendan rápidamente su uso. La reorganización de la información permite que la búsqueda y obtención de la misma sea mucho más ágil, los usuarios ya no tendrán que re-crear hojas electrónicas o bases de datos, ya no existirá duplicación de la información. Los cálculos requeridos, clasificación de datos, obtención de reportes, etc. serán automatizados por el sistema. Para el administrador del sistema su tarea se ve simplificada ya que al estar todo centralizado, es más fácil realizar las tareas de respaldos y exportación de la información. Se trata, pues, de un sistema que pretende mejorar y facilitar el trabajo de las personas al interior de la Comisión Fulbright.

CAPÍTULO 2

2.1 Proceso de selección de Becarios

El Programa Fulbright contempla tres tipos de programas académicos: para estudiantes de post-grado (maestrías), para profesores (docencia), y el programa de Iniciativa en Ecología de la OEA (Organización de Estados Americanos). Estos tres programas siguen un mismo proceso de de selección, que se puede dividir en tres fases: aplicación y apertura de concursos, preselección, y selección final. A continuación se detalla cada una de estas fases.

2.1.1 Aplicación y apertura de concursos

El 1º. de febrero de cada año se inicia el período de recepción de aplicaciones, esto es para las personas que deseen participar del concurso de programas académicos para maestrías en los EEUU. A mediados de febrero se envía invitaciones a todas las universidades del país para que cada una postule profesores como candidatos para el concurso de becas para profesores en universidades de EEUU. Igualmente a mediados de febrero se envía invitaciones a fundaciones y ONG (Organizaciones No Gubernamentales) para que postulen candidatos para el concurso del programa de Iniciativa en Ecología de la OEA. Los tres concursos se abren oficialmente el 1º. de marzo, y se cierran el 31 de mayo del mismo año, siendo ésta también la fecha límite para la entrega de aplicaciones.

El aplicante requiere entregar los siguientes documentos:

- Formulario de aplicación lleno
- Copia del título o acta de grado. Si todavía no han sido entregados (como en el caso de las personas que recién han egresado de la universidad), se puede entregar

provisionalmente un certificado de egreso, pero debe entregarse el título o acta de grado hasta antes de diciembre del año de aplicación.

- Notas de toda la carrera (transcript), al momento de la selección se tomará en cuenta el GPA acumulado (promedio total) excluyendo notas de tesis o disertaciones. Debe incluir traducción al inglés.
- Copia del Currículum Vítae en español y su traducción en inglés.
- Tres cartas de recomendación de profesores de la universidad o universidades donde estudió el aplicante con su respectiva traducción. Entre las tres se puede incluir una carta de recomendación de algún empleador, pero se prefiere que todas sean de profesores con los que el aplicante haya tomado cursos.
- Ensayos en inglés, básicamente de auto-descripción de la persona que aplica, explicando las razones por las cuales se considera una persona ideal para ser seleccionada como becario/a Fulbright, y cuáles son sus objetivos y metas si resulta seleccionada.
- Resultados de los exámenes TOEFL y GRE/GMAT. En caso de no haber tomado el TOEFL todavía, se puede entregar el resultado de un examen de inglés similar que se realiza en la Comisión Fulbright. El resultado del GRE/GMAT es obligatorio, se lo puede entregar una vez que se haya rendido este examen.

El resultado final de este proceso es la lista completa de candidatos (aspirantes) a los concursos. La información producida de esta fase de la selección actualmente se almacena en diferentes bases de datos de Access dispersas, separadas para cada año, las tablas no han sido diseñadas técnicamente, poseen errores de diseño y muchas veces, por la misma razón de estar separadas en diferentes archivos, las búsquedas de información se dificultan. Parte de esta información es enviada al Departamento de Estado de los EEUU mediante un

sistema especial basado en el Web, pero se desea tener esta información y otras adicionales de forma local. La figura 2.1 muestra el flujo del proceso de aplicación.

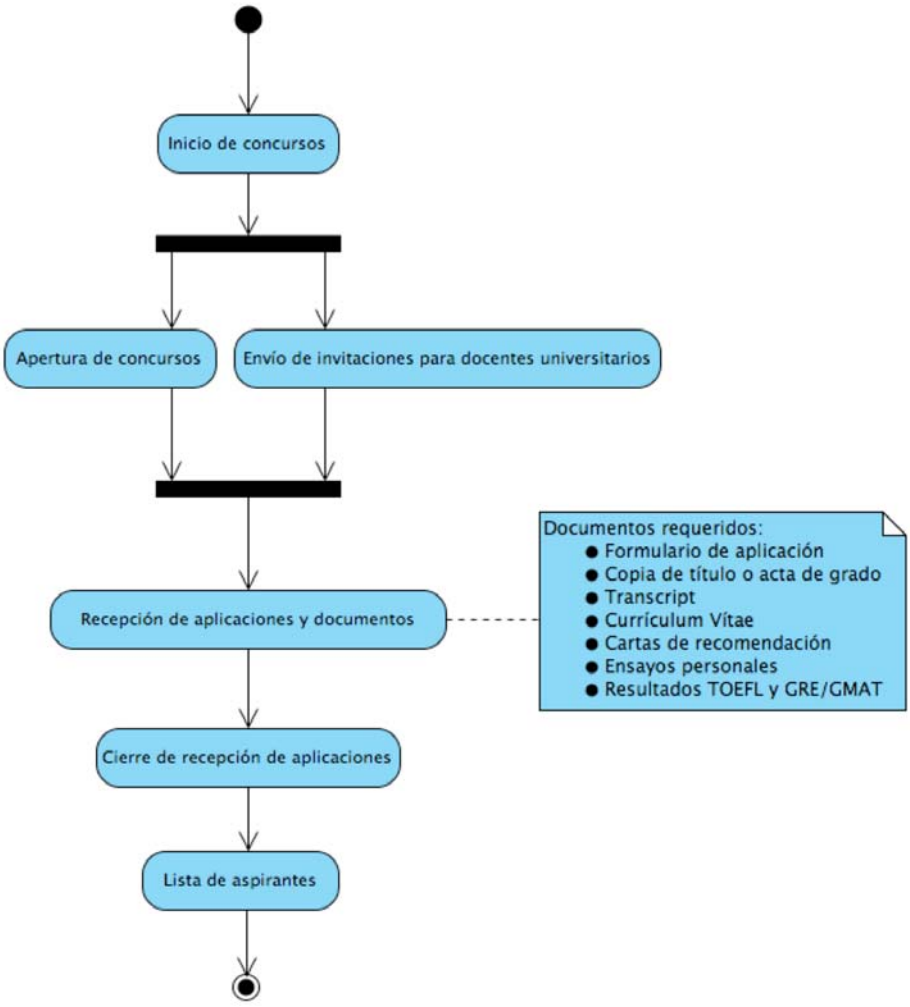


Figura 2.1 – Apertura de concursos y recepción de documentos

2.1.2 Preselección

Todas las carpetas de documentos son revisadas en primera instancia por la persona encargada de la Consejería Académica. Para esto, se toman en cuenta los siguientes criterios de evaluación:

- Revisión de las cartas de recomendación. Se determina si todas son aceptables, o si alguna o algunas de ellas se consideran “débiles”.

- Revisión de ensayos. Se evalúa claridad, objetividad, si las metas y objetivos expuestos son claros.
- Revisión de notas y promedios de la carrera universitaria (transcripts). Si hay repetición de cursos o rendimiento académico bajo, es casi segura la eliminación del aspirante.
- Verificación del campo de estudio. Esto se realiza porque para ciertas especialidades se dificulta mucho conseguir una beca (por ejemplo, las Maestrías en Economía y Finanzas son las de menor apoyo financiero en los Estados Unidos).
- En el caso de que el aspirante desee concursar para una beca de Maestría en Administración de Empresas (MBA), debe comprobar que ha tenido mínimo 3 años de experiencia en el campo laboral después de su graduación.
- Revisión de notas del TOEFL y del GRE/GMAT. En caso de que el aspirante sólo haya rendido la prueba de inglés ofrecida por la Comisión Fulbright, se tomará en cuenta esa nota; sin embargo, si el aspirante llega a ser seleccionado, obligatoriamente debe rendir el TOEFL para ser aceptado en la universidad. El examen GRE/GMAT es obligatorio, y su revisión se realizará una vez que el aspirante lo haya rendido (este examen se ofrece en el país una o dos veces al año).

Los resultados preliminares y las carpetas de los aspirantes pasan a manos de la persona encargada de la Dirección Ejecutiva. El (la) Director(a) Ejecutivo(a), durante el concurso, mantiene contacto frecuente con los aspirantes, realiza entrevistas personales, evalúa características subjetivas como personalidad, facilidad de comunicación, capacidad de liderazgo, etc. El resultado final de todo este proceso es la lista de personas preseleccionadas, en espera y eliminadas. Las personas preseleccionadas automáticamente pasan a la fase final de selección (que será descrita posteriormente). Aquellas personas en

lista de espera podrían pasar a la fase final si alguna de las preseleccionadas desiste de su candidatura; en el caso de personas del interior (de provincia) que queden en lista de espera debido solamente a dificultades de trámites y viajes, a una o dos de ellas es dada la oportunidad de pasar a la siguiente fase. En el caso de la lista de personas eliminadas, se les envía una comunicación escrita de que no han sido calificadas y que podrían concursar en el programa del próximo año.

La información generada en esta fase es de tipo cuantitativo, y actualmente se almacena en hojas de Excel dispersas, no hay integración hacia las bases de datos de Access, hay duplicación de información, y a veces las entidades que se intercomunican (Consejería Académica y Dirección Ejecutiva) obtienen información incompleta o desactualizada. La figura 2.2 es un diagrama de actividad del proceso de preselección aquí descrito.

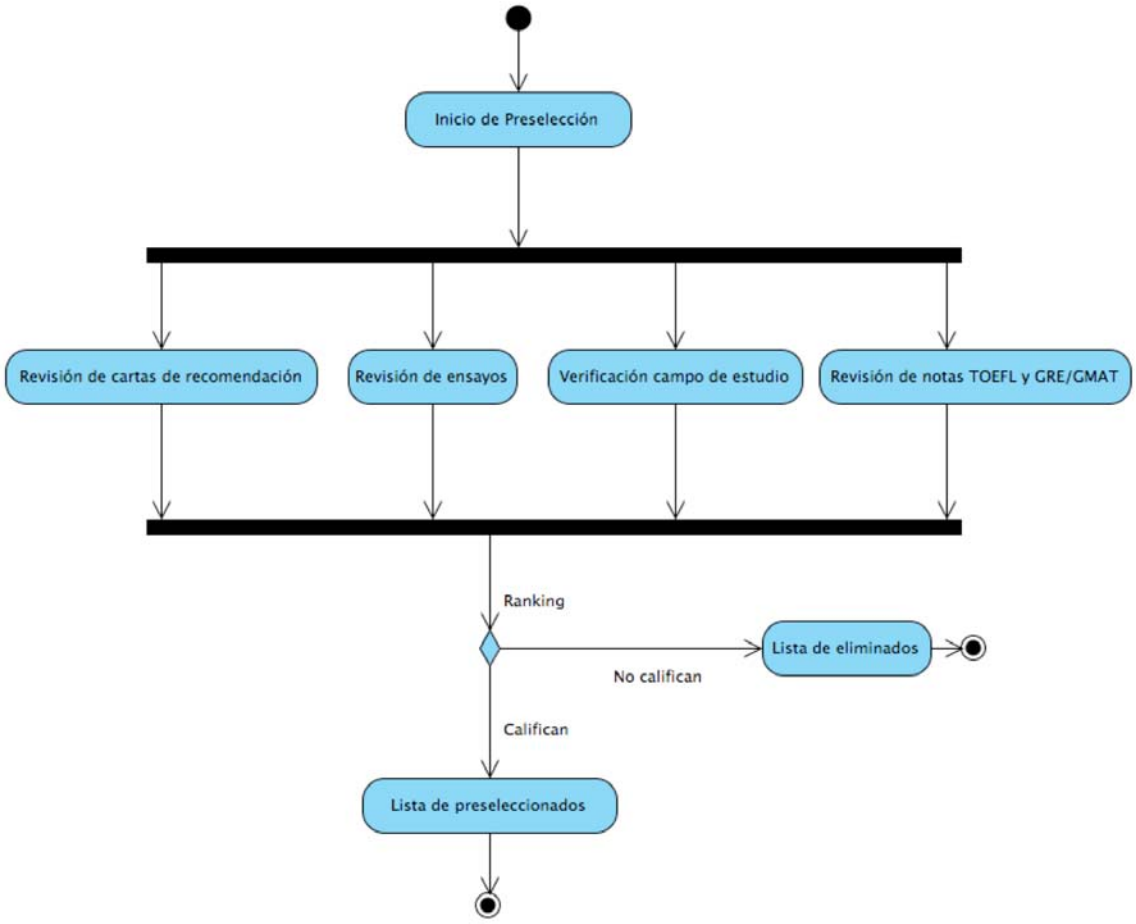


Figura 2.2 – Preselección

2.1.3 Entrevista y selección final

Los candidatos preseleccionados deben pasar por una entrevista ante un grupo de panelistas invitados previo a su selección final. Para esto, los candidatos son divididos en varios grupos según su campo de estudios. Para las entrevistas se dedica un día para cada grupo. El panel de invitados está conformado por los Miembros del Directorio de la Comisión Fulbright, ex-becarios Fulbright que hayan obtenido su título dentro del campo de estudios del grupo que se está evaluando, profesores del campo de estudios, y si están disponibles, becarios Fulbright estadounidenses que se encuentren en el país al momento de las entrevistas. Durante el proceso de entrevistas, ninguno de los empleados de la Comisión Fulbright, incluyendo la persona a cargo de la Dirección Ejecutiva, pueden intervenir. Pero los resultados de las entrevistas son entregados a la persona encargada de la Consejería Académica, que junto a la Dirección Ejecutiva realizarán la selección final. Los panelistas realizarán 9 preguntas a cada candidato, cada panelista posee una hoja de calificación para las respuestas de los candidatos. El panelista califica cada respuesta con un valor del 1 al 5, el puntaje total máximo que puede obtener cada candidato en cada hoja de cada panelista es de 45 puntos. Estos resultados son los que se entregan a Consejería Académica y Dirección Ejecutiva de la Comisión Fulbright para su evaluación final.

Los criterios para la selección final y sus respectivos pesos son:

- Calificación del examen GRE/GMAT: 40%
- Calificación del examen de inglés, ya sea el TOEFL o el ofrecido por la Comisión Fulbright: 10%
- Notas de la carrera, promedio acumulado (GPA acumulado): 35%
- Resultados de las entrevistas con los panelistas: 5%

- Posicionamiento (ranking) provisional: 10%

Con los criterios anteriormente citados se realiza un posicionamiento o “ranking” de los candidatos, siendo el 1er. lugar para aquel que ha obtenido la mayor calificación. Si dos o más candidatos obtienen una misma posición, se realiza un promedio o selección por moda (el último criterio de 10%). La figura 2.3 demuestra el proceso de selección final.

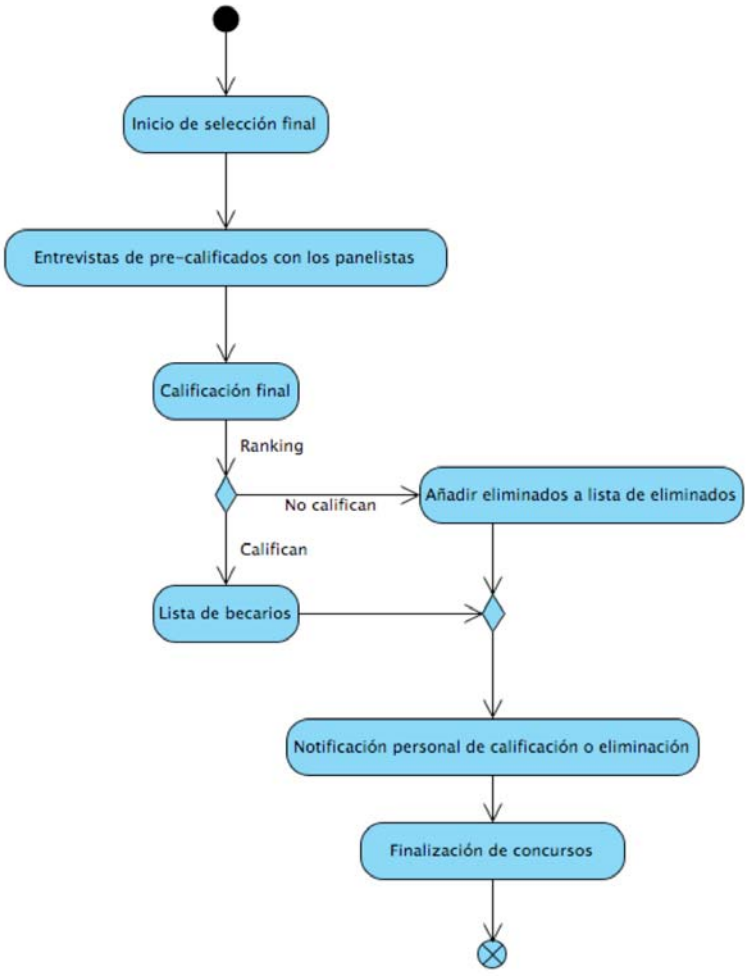


Figura 2.3 – Selección final

El criterio final de selección no depende de los candidatos, sino del dinero que haya asignado el Departamento de Estado de los EEUU a la Comisión Fulbright de Ecuador para el año en curso. Por lo general, todos aquellos que hayan obtenido los primeros lugares de su grupo son seleccionados Becarios Fulbright, y podrán iniciar sus estudios al

año siguiente de su selección. Dependiendo de la cantidad de dinero, se puede seleccionar más candidatos, pero ha habido casos que el dinero no ha sido suficiente y se ha tenido que eliminar a alguno de los ya seleccionados.

El resultado de este proceso son las listas de seleccionados y eliminados. A todos se les comunica por escrito de su selección o eliminación. Aquellos que han sido eliminados pueden aplicar nuevamente, cuantas veces deseen, a concursos en años posteriores.
Los problemas de manejo de información son los mismos de los puntos anteriores. Todo se maneja en hojas de Excel y bases de datos de Access que muchas veces no están relacionadas entre sí, hay duplicación de la información, y en el futuro el buscar información histórica importante sería un proceso muy difícil.

2.2 Requerimientos del sistema

La siguiente es la lista de levantamiento de requerimientos del sistema. Se lo realizó mediante entrevistas a la Directora y a la Consejera Académica de la Comisión. Se han organizado en 8 secciones: Ingreso de aplicantes (requerimientos 1 al 9), Ingreso y apertura de Programas (10 al 14), Aplicación a Programas abiertos (15 al 22), Procesos de pre-selección y selección (23 al 34), Mantenimiento de la aplicación (35 al 39), Usuarios del sistema (40 al 45), Importación de datos (46 al 48), Aplicación e interfaz (49 y 50).

2.2.1 Ingreso de aplicantes

REQ-1 La aplicación debe permitir el ingreso de todos los datos personales de los aplicantes: nombres completos, identificación, estado civil, lugar y fecha de nacimiento

REQ-2 La aplicación debe permitir el ingreso de datos de dirección domiciliaria, dirección del lugar de trabajo, e información de contacto, especialmente direcciones de correo electrónico.

REQ-3 La aplicación debe permitir la posterior edición de los datos personales, direcciones e información de contacto.

REQ-4 La aplicación debe permitir marcar un aplicante como difunto, o desmarcarlo en caso de error.

REQ-5 La aplicación debe permitir al administrador del sistema borrar completamente todos los registros de un aplicante.

REQ-6 La aplicación debe permitir el ingreso de las universidades/institutos por los cuáles ha cursado sus estudios, y mostrarlos como una lista.

REQ-7 La aplicación debe permitir el ingreso de los resultados de exámenes tomados para ingreso a universidades del exterior, como TOEFL, GRE, GMAT, IELTS, etc.

REQ-8 La aplicación debe asignar a cada aplicante un código único, que podrá ser entregado al estudiante para que luego su búsqueda en la base de datos se vea facilitada.

REQ-9 La aplicación debe permitir la búsqueda de información de cualquier aplicante mediante múltiples criterios, como código asignado, cédula/pasaporte, nombres, etc.

2.2.2 Ingreso y apertura de Programas

REQ-10 La aplicación debe permitir el ingreso de cualquier número de programas, y permitir activarlos/desactivarlos para uso en concursos.

REQ-11 La aplicación debe permitir la apertura de un programa para aceptar aplicaciones, que luego aparecerá en las listas de programas abiertos y para aplicaciones al mismo.

REQ-12 Después de la fecha de cierre de un programa, no debe admitir más aplicaciones y debe aparecer en una lista de programas cerrados.

REQ-13 Todos los programas a ser abiertos deben tener una fecha de apertura, pero la fecha de cierre es opcional, pudiendo dejar ciertos programas indefinidamente abiertos.

REQ-14 Debe existir una lista de programas abiertos para información histórica, así estos programas ya hayan sido desactivados. Necesario para la importación de bases de datos antiguas.

2.2.3 Aplicación a programas abiertos

REQ-15 Todos los aplicantes pueden aplicar a uno o más programas abiertos.

REQ-16 El formulario de aplicación sólo debe mostrar aquellos programas que se encuentren abiertos, o aquellos que serán abiertos en el futuro.

REQ-17 El formulario de aplicación debe permitir el ingreso de la universidad en Estados Unidos a la que se aplica, el campo de estudio, la agencia que auspicia el programa, y opcionalmente la escuela/institución/universidad que auspicia localmente al aplicante. Mientras el aplicante no haya sido pre-seleccionado y seleccionado para un programa, no permitirá el ingreso de detalles del programa.

REQ-18 Una vez realizada la aplicación, se debe fijar el programa y no se podrá cambiar este valor, ni la fecha de la aplicación.

REQ-19 Si el aplicante ya ha sido pre-seleccionado y seleccionado para el programa, el formulario permitirá (es decir, desbloqueará) el ingreso de los datos del programa como fecha de inicio y de finalización, departamento, y datos de programa pre-académico.

REQ-20 La aplicación a programas históricos servirá para ingresar directamente los datos de becarios que ya han participado en ellos, y el formulario de ingreso tendrá el mismo formato del formulario estándar de aplicación, pero con los campos de datos de programa y pre-programa totalmente desbloqueados.

REQ-21 Para aplicaciones a programas históricos, y becarios seleccionados para un programa, el programa mostrará una opción para permitir el ingreso de concesiones económicas (Grants).

REQ-22 El programa permitirá, en el caso de becarios seleccionados y becarios históricos, ingresar cualquier número de concesiones económicas (Grants) que contengan: tipo de concesión, cantidad y año/período.

2.2.4 Procesos de pre-selección y selección

REQ-23 El programa permitirá seleccionar desde una lista en cuál programa se correrá el proceso de pre-selección o de selección. La lista debe contener programas abiertos, o cerrados en los últimos 6 meses.

REQ-24 La aplicación debe mostrar la lista de todos los aplicantes al programa, con su respectivo código asignado, nombres completos, indicar si han sido pre-seleccionados, seleccionados, y eliminados, y un botón para ingresar a una página con detalles adicionales.

REQ-25 La pantalla de detalles adicionales de cada aplicante debe contener

información de campo de estudio, GPA, notas de exámenes, cartas de recomendación y ensayos realizados por el aplicante.

REQ-26 En la pantalla de detalles, el usuario podrá ingresar el GPA y grabarlo inmediatamente cuando se presione en un botón destinado para este propósito.

REQ-27 El usuario debe tener la posibilidad de ingresar múltiples cartas de recomendación y ensayos con sus respectivas calificaciones (excelente, muy bueno, regular, débil, muy débil)

REQ-28 El sistema debe analizar si los aplicantes cumplen con los requisitos mínimos para la pre-selección, y aquellos que no cumplan, deben ser notificados al usuario. Si el usuario decide que efectivamente no cumplen con los requisitos, el sistema los eliminará del concurso.

REQ-29 El sistema, basado en las notas de exámenes (TOEFL, GRE/GMAT, etc.), en el GPA, en la calidad de ensayos y cartas de recomendación, deberá calcular un valor que permitirá dar un posicionamiento (ranking) (siendo 1er. lugar el de más alta calificación) a cada aplicante.

REQ-30 Desde la lista calculada de posicionamientos (rankings), el usuario podrá seleccionar aquellos aplicantes que pasarán a la siguiente fase del concurso.

REQ-31 En la lista de aplicantes para selección final, sólo deben aparecer aquellos aplicantes que han sido pre-seleccionados en la fase anterior.

REQ-32 La lista de aplicantes para selección final debe contener, para cada aplicante, un campo que permitirá ingresar la calificación obtenida en la entrevista con los panelistas.

REQ-33 El proceso de selección final realizará un cálculo similar al del REQ-29, con

la adición de la calificación de la entrevista, para desplegar una lista con un ranking final.

REQ-34 El usuario, desde la lista de ranking final, podrá seleccionar aquellos aplicantes que se convertirán en becarios del programa.

2.2.5 Mantenimiento de la aplicación

REQ-35 La aplicación debe permitir el ingreso de cualquier número de agencias auspiciantes de programas.

REQ-36 Aunque actualmente sólo existan 2 programas pre-académicos, la aplicación debe permitir ingresar cualquier cantidad de éstos, debido a que en un futuro hay la posibilidad de que cambien.

REQ-37 La aplicación debe permitir el ingreso de cualquier número de tipos de exámenes, y dentro de cada examen, cualquier número de tipos de puntajes/resultados.

REQ-38 La aplicación debe permitir el ingreso de cualquier número de tipos de títulos académicos (Ing., Ph.D., M.B.A., M.S., etc.)

REQ-39 La aplicación debe permitir, de forma organizada en categorías y subcategorías, el ingreso de cualquier número de campos de estudio.

2.2.6 Usuarios del sistema

REQ-40 La aplicación debe tener un pequeño módulo de mantenimiento de usuarios del sistema, donde se podrá cambiar su contraseña, nivel de acceso y preferencias como el número de resultados por página de búsqueda.

REQ-41 La aplicación debe tener por defecto un Administrador del Sistema,

denotado por el nombre de usuario “admin”, que solamente podrá cambiársele la contraseña y el número de resultados por página. No podrá ser borrado ni cambiado su nivel de acceso.

REQ-42 El Administrador del Sistema tendrá la potestad de añadir, suprimir, y cambiar los usuarios del sistema. Usuarios con menor nivel de acceso solamente podrán cambiar su contraseña y preferencias.

REQ-43 Los niveles de acceso que tendrá el sistema son: Administrador del Sistema, Director, Consejero Académico, Asistente.

REQ-44 Los usuarios Director y Consejero Académico tendrán prácticamente los mismos privilegios del Administrador, pero sin la posibilidad de modificar tipos de títulos, tipos de exámenes, tipos de programas y agencias.

REQ-45 Los usuarios con nivel de acceso Asistente, solamente podrán ejecutar tareas de mantenimiento (modificación de títulos, agencias, tipos de programas y pre-programas, exámenes).

2.2.7 Importación de datos

REQ-46 Todos los datos de las bases antiguas realizadas en Access deben ser trasladados a la aplicación

REQ-47 La base de datos actual debe proveer las estructuras necesarias para permitir los datos de las bases antiguas.

REQ-48 Las bases de datos antiguas no se encuentran normalizadas, el proceso de importación debe repartir automáticamente la información entre las tablas normalizadas de la base de datos nueva.

2.2.8 Aplicación e Interfaz de usuario

REQ-49 La aplicación debe ser accesible desde cualquier computadora dentro de la red de la Comisión Fulbright.

REQ-50 La interfaz de usuario debe ser consistente tanto visual como operativamente en todas sus secciones, procesos, formularios.

CAPÍTULO 3

3.1 Metodología de desarrollo

Para el desarrollo del presente proyecto, se utilizará la metodología de desarrollo iterativa e incremental. Esta metodología consiste en desarrollar el sistema de forma incremental, es decir desarrollarlo en módulos funcionales que se van añadiendo poco a poco al sistema hasta obtener el sistema completo. Esto permite ir aprendiendo de los errores y aciertos en el proceso de desarrollo, y en cada iteración se obtiene un software más estable y con menor número de errores.

El proceso iterativo e incremental consiste básicamente en cuatro grandes fases: Inicio, Elaboración, Construcción y Transición.

- Fase de Inicio: se trabaja con la idea inicial del proyecto, el problema a resolver, el estimado de costos y el estimado de ingresos.
- Fase de Elaboración: se realiza un análisis más profundo del problema, se desarrolla un plan de proyecto y se eliminan los riesgos lo más tempranamente posible. Se debe elaborar una visión del proyecto de cómo el cliente la entiende. Para esto, se utilizan diagramas de casos de uso a niveles alto e intermedio.
- Fase de Construcción: esta es la fase iterativa del proceso, en la que cada iteración consiste de las siguientes sub-fases:
    - Análisis: detalle de los casos de uso asociados. En esta fase se entiende el problema, no se busca la solución.
    - Diseño: se deciden los objetos necesarios, el rol de cada objeto y la interacción entre los mismos.

- Implementación: desarrollo de la solución.
- Prueba: se realizan varios tipos de pruebas como pruebas de funcionalidad, pruebas de unidad (unit testing), etc.

Es importante entregar un código probado, a tiempo y debidamente documentado al final de cada iteración.

- Fase de Transición: en esta fase no se añaden más características ni funcionalidad al sistema, solamente se realizan optimizaciones, corrección de errores, refactoring, etc. En esta fase también se realizan las pruebas por parte del cliente, realización de la documentación para el cliente (manuales de usuario), entrenamientos, etc.

3.1.1 Productos de la metodología en el desarrollo del proyecto

3.1.1.1 Inicio

El presente proyecto pretende desarrollar un sistema, basado en tecnologías Web, para el almacenamiento de datos y automatización del proceso de selección de becarios de la Comisión Fulbright del Ecuador. El problema consiste básicamente en que los datos son almacenados en archivos dispersos, sin organización central, y el proceso de selección de becarios es manual. Para el desarrollo de este sistema, se pretende utilizar tecnologías de código abierto (*open source*) y/o gratuitas para reducir los costos. Se ha realizado la siguiente estimación económica para el proyecto:

Computador para Servidor de Aplicaciones*	US$ 1000.00
Herramientas de desarrollo (licencias)**	US$ 0.00
Sistema operativo para Servidor***	US$ 200.00
Desarrollador de software	US$ 1000.00
Transporte	US$ 100.00
TOTAL	**US$ 2300.00**

*El computador es provisto por la Comisión Fulbright

**Todas las herramientas a utilizar son gratuitas y/o de código abierto

***Se considera una licencia de Microsoft Windows XP Professional. Si se utiliza una distribución gratuita de Linux, este costo baja a US$ 0.00.

Tabla 3.1 – Costos estimados del proyecto

3.1.1.2 Elaboración

El proyecto pretende organizar toda la información referente a aplicantes/becarios en una base de datos centralizada, accesible desde cualquier computador al interior de la Comisión Fulbright, y además proveer automatización de los procesos de selección de nuevos becarios para los programas académicos. El análisis detallado del problema y la propuesta para su resolución ya han sido descritos en las secciones 1.3 y 1.4 de este documento. Posteriormente se realiza el levantamiento de requerimientos de la aplicación (ver sección 2.4.1). El siguiente es un diagrama de casos de uso de alto nivel de cómo se visualiza el proyecto y su funcionalidad con el usuario.

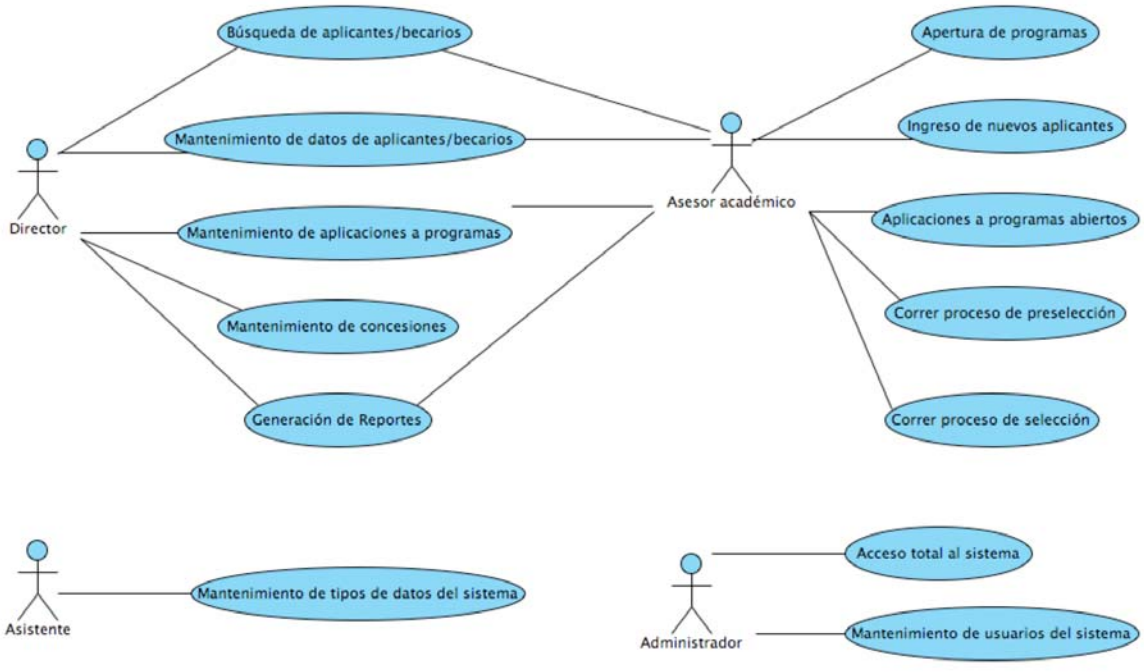


Figura 3.1 – Diagrama general de casos de uso

3.1.1.3 Construcción

El proyecto se desarrolló en dos iteraciones, las cuales se describen a continuación.

3.1.1.3.1 Primera iteración

El problema radica principalmente en la información dispersa en archivos creados por cada empleado de forma independiente. Esto significa que el manejo de información a través de la institución no ha sido estandarizado, por lo menos a nivel de sistemas de información. El sistema a desarrollar propone una base de datos centralizada, que permite organizar toda la información para que tenga un flujo estandarizado a través de los sistemas de información, y que sea fácilmente accesible para cualquier empleado al interior de la institución. El siguiente diagrama muestra los casos de uso del producto.

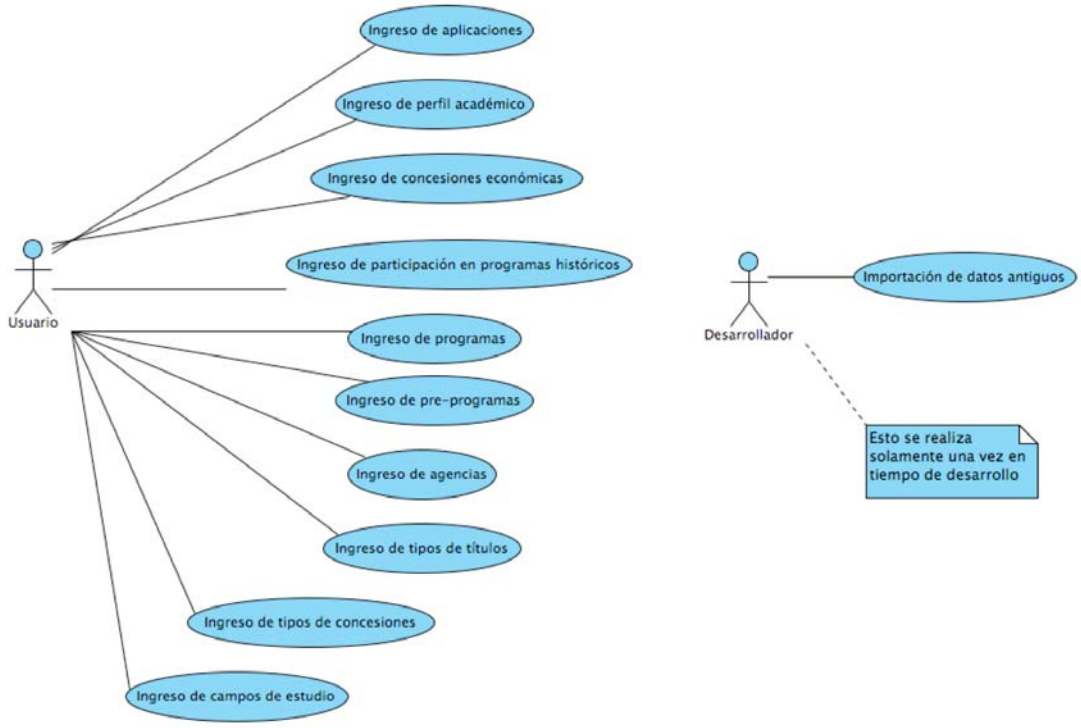


Figura 3.2 – Casos de uso para iteración 1

El sistema se desarrolló en tres capas utilizando el patrón de diseño MVC (Modelo-Vista-Controlador, ver sección 3.2), donde los objetos de la Vista (o Visualización) se encargan

de construir la interfaz de usuario (en este caso, páginas HTML para el *Web Browser* del usuario) y los objetos del Modelo se encargan de las consultas y modificaciones a los datos, la comunicación de ambas capas reguladas por un Controlador (en este caso específico, un servlet de despacho de eventos), y organizados como se muestra en la siguiente figura.

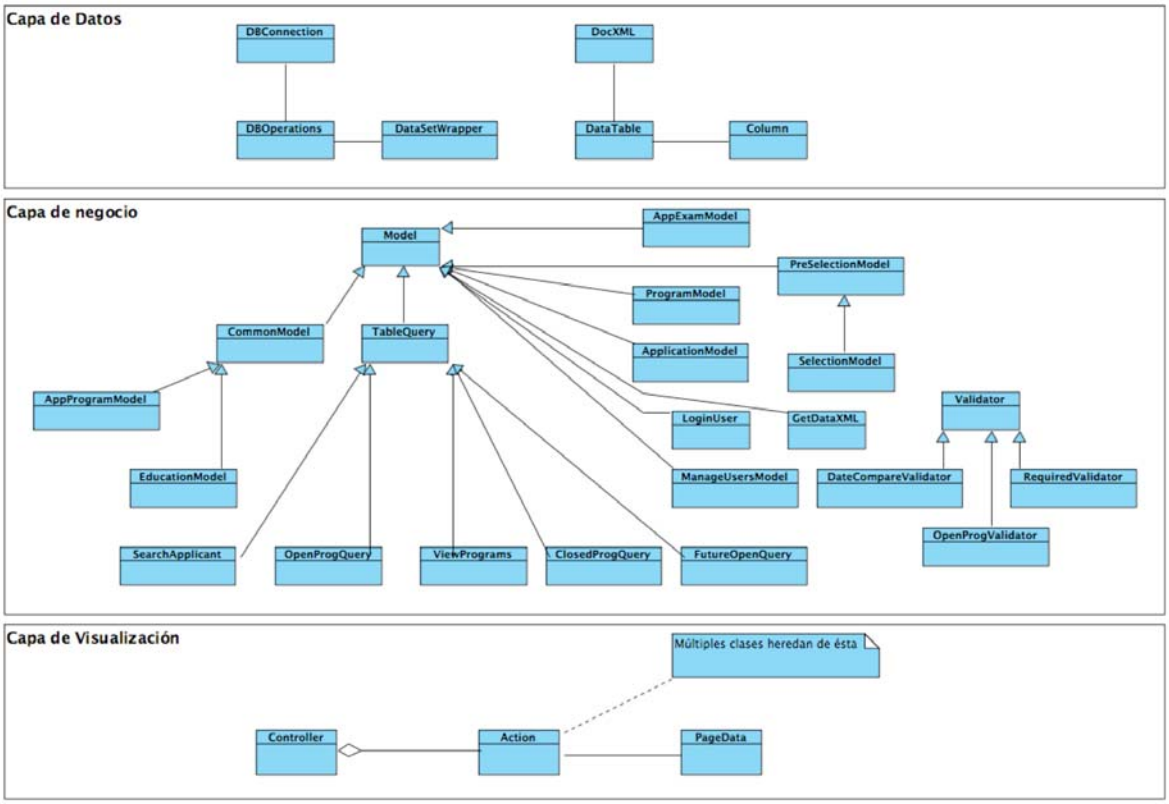


Figura 3.3 – Diagrama de objetos de la aplicación

La figura 3.3 contiene algunos objetos que no se utilizan en la primera iteración, pero se han incluido en el modelo ya que servirán en la segunda iteración. Para mayores detalles de la interacción entre objetos, referirse a la sección 4.2 de este documento, o al Anexo 1, secciones 2 y 3.

El resultado de esta iteración es una base de datos con todos los datos antiguos importados a la misma, y una aplicación Web que posee un solo tipo de usuario (el administrador), y que básicamente permite el mantenimiento de estos datos, y también el ingreso de

información nueva. Las pruebas realizadas en la aplicación demostraron un alto grado de estabilidad y velocidad en las respuestas.

3.1.1.3.2 Segunda iteración

El problema de la organización de la información fue resuelto en la primera iteración, en esta se analizará el problema de la automatización de los procesos de pre-selección y selección de becarios para programas académicos. Actualmente se lo realiza de forma manual, y cada consejero académico realiza el proceso a su manera en archivos de Excel. El sistema utilizará los datos almacenados en la base de datos para realizar los procesos de pre-selección y selección, y para generar los reportes del sistema. La siguiente figura muestra los casos de uso del producto final.

Figura 3.4 – Casos de uso para iteración 2

El diseño es el mismo utilizado en la primera iteración, con la adición de las clases que manejan los nuevos procesos. La flexibilidad del diseño utilizado permite añadir

características al sistema sin cambiar la estructura de clases preexistente (ver figura 3.1.1.3.2). Para mayores detalles de la interacción entre objetos, referirse a la sección 4.2 de este documento, o al Anexo 1, secciones 2 y 3.

El resultado de esta iteración es la aplicación completa, con toda la funcionalidad requerida por la institución (ver requerimientos en la sección 2.4.1). Las pruebas realizadas en la aplicación final se detallan en la sección 4.3.

3.1.1.4 Transición

La aplicación es entregada a la institución, junto con los manuales técnico (ver Anexo 1) y de usuario (ver Anexo 2). El entrenamiento es provisto por el desarrollador a los usuarios del sistema al interior de la Comisión Fulbright.

3.2 Ambiente de desarrollo

Para este proyecto se utilizará un ambiente de desarrollo multicapa, siguiendo las recomendaciones del patrón de desarrollo MVC (Modelo – Vista – Controlador). Este patrón de desarrollo básicamente consiste en tres capas. El Modelo es la capa que contiene la representación de los datos y los procesos de negocio, es decir es un modelo de objetos que representa el negocio en forma lógica. La Vista es aquella que muestra de forma visual el Modelo, es decir presenta la interfaz de usuario y organiza los datos del modelo de tal forma que puedan ser entendidos por los usuarios del sistema. Como se puede ver, el Modelo corresponde a las capas de datos y negocio en el modelo de tres capas, y la Vista corresponde a la capa de presentación. Pero este patrón de desarrollo contiene un elemento adicional, el Controlador. Este consiste en un canal intermedio de comunicación entre la Vista y el Modelo, y su propósito principal es desacoplar en lo posible la dependencia entre

la Vista y el Modelo. Esto permite hacer cambios en el modelo de datos, sin necesidad de hacer cambios a la interfaz de usuario. El Controlador maneja los eventos de usuario generados en la Vista, y canaliza estos eventos al Modelo para que este, a su vez, realice la consulta o las respectivas modificaciones a los datos.

En la actualidad existen varios *frameworks* para Java que implementan el patrón MVC, o proveen partes del mismo. Uno de los más populares es el proyecto de código abierto Apache Struts, que basa su funcionamiento en los valores de archivos de configuración XML para adaptarlos a cualquier proyecto. JavaServer Faces (JSF) provee una librería de controles Web y manejadores de eventos, que efectivamente corresponden a la Vista y el Controlador de una aplicación MVC. Generalmente estos frameworks implementan solamente la Vista y/o el Controlador, para el Modelo existen otras tecnologías que permiten manejar la lógica de negocio y la capa de datos. La tecnología Enterprise JavaBeans (EJB) permite realizar modelos de objetos que manejan la lógica de negocio desde el lado del servidor, y que pueden utilizarse en cualquier tipo de aplicación distribuida. La versión EJB 3.0 incluye una interfaz de desarrollo de aplicaciones (API, Application Programming Interface) de persistencia para mapeo entre objetos y bases de datos relacionales. También existen soluciones de solamente capa de datos como Hibernate, un proyecto de código abierto que implementa una capa de persistencia objetos-BD relacional que puede ajustarse a cualquier tipo de proyecto, incluyendo aquellos que utilicen EJB.

Para el presente proyecto se utilizará una librería de plantillas (templating) que ayuda a reutilizar componentes de la interfaz de usuario y organizan la Vista, un Controlador desarrollado específicamente para esta aplicación y un Modelo de acuerdo a las

necesidades del cliente. El Modelo a su vez está subdividido en dos capas, una capa de datos y una capa de lógica de negocio.

La Vista posee clases específicas que permiten construir las estructuras de datos necesarias para utilizarse junto con el motor de plantillas, y así generar la interfaz de usuario (que es básicamente páginas HTML). El Controlador es un servlet de despacho, que mediante técnicas de herencia y polimorfismo de objetos, despacha la acción a ejecutar en el servidor. La subcapa de negocios del Modelo posee objetos que se encargan de realizar los procedimientos específicos del negocio, como validaciones, procesos de selección, construcción de estructuras de datos, etc. La subcapa de datos del Modelo se encarga de las tareas de consulta y actualización de datos en la base de datos subyacente, y de construir estructuras de datos que se pueden procesar en la subcapa de negocios. Para un mayor detalle de la intercomunicación entre capas, referirse a la sección 4.2 de este documento. Para información más detallada y de procesos específicos, referirse al Anexo 1.
Este ambiente de tres capas permite añadir características y funcionalidad al sistema sin afectar a otros procesos ni a la estructura básica de clases, además de facilitar el mantenimiento del código.

3.3 Herramientas de desarrollo

El presente proyecto se desarrollará en la plataforma Java Enterprise Edition. Esta plataforma se puede ejecutar en cualquier tipo de sistema, ya sea este Microsoft Windows, Linux, Mac OS X, y otras versiones de Unix. La aplicación residirá en un Servidor de Aplicaciones, que provee el contenedor y los servicios necesarios para ejecutarla. El Servidor de Aplicaciones escogido es el Sun Java Application Server, que provee todos los estándares de la especificación Java Enterprise Edition, y es gratuito para desarrollo y

producción. Para el desarrollo en sí de la aplicación se utilizará el IDE (Integrated Development Environment, ambiente integrado de desarrollo) NetBeans 5.5, que incluye herramientas de edición, compilación, trazado de errores y conexión directa con el Sun Application Server para realizar los deployments de la aplicación. Esta herramienta es también gratuita y de código abierto. El sistema de base de datos a utilizar es el MySQL 5.0, un DBMS gratuito (a excepción del soporte técnico opcional), de código abierto. Para administrar esta base de datos, se utilizará el EMS SQL Manager 2005 Lite for MySQL, que también es gratuito (a excepción de la versión completa que es comercial). Se ha procurado utilizar solamente herramientas gratuitas, sin costo para el desarrollador ni para el cliente, para reducir los costos totales.

Con respecto al hardware, la aplicación residirá en una computadora central cuyas características son las siguientes: procesador Pentium 4 HT (HyperThreading) de 3 GHz, 1 GB de RAM, 120 GB de espacio en disco duro, puerto de red 10/100 Mbps. Contiene el sistema operativo Linux Fedora Core 5, que posee una licencia gratuita GPL.

CAPÍTULO 4

4.1 Instalación del Software

4.1.1 Instalación de la plataforma

Para la implementación del sistema, primero se debe instalar el software especial de servidor de aplicaciones y servidor de base de datos. La presente aplicación ha sido diseñada para ser usada con el servidor de base de datos MySQL, que es gratuito y no requiere el pago de licencias para su uso. Para utilizarse con otros servidores de base de datos, como PostgreSQL, Oracle, Microsoft SQL Server, etc., se requeriría la modificación de ciertas sentencias SQL dentro de la aplicación que utiliza funciones específicas de MySQL, y también los stored procedures (SP) deberían ser adaptados a la sintaxis del servidor deseado. La instalación de MySQL depende del sistema operativo del servidor. Para Microsoft Windows y Apple Mac OS X, MySQL incluye un instalador gráfico interactivo que paso a paso permite configurarlo, una vez instalado está listo para usar. En Linux o sistemas operativos tipo Unix, MySQL generalmente se copia a mano, es decir se lo extrae del archivo comprimido de forma manual, se asignan los respectivos permisos y se debe ejecutar ciertos scripts que configuran el servidor de base de datos. En Linux/Unix, se requiere de un mayor conocimiento técnico y de uso de la línea de comandos (shell). MySQL incluye, para ciertas variantes de Linux, paquetes de instalación automática que facilitan este proceso, aunque generalmente requieren correr ciertos scripts después de su instalación. MySQL incluye documentación extensiva (aunque un poco compleja y técnica), que puede ayudar a realizar ciertas instalaciones específicas en plataformas específicas.

La presente aplicación ha sido probada y puesta en producción en el servidor de aplicaciones Sun Java Application Server. Se podría utilizar también otros servidores de aplicaciones y contenedores J2EE como Apache Tomcat, Glassfish, JBoss, Oracle OAS, etc., con simplemente modificar o utilizar los archivos de configuración específicos de cada uno. Sin importar el sistema operativo del servidor, ya sea Windows, Mac OS X, Linux/Unix, Sun Java AppServer incluye un instalador gráfico que guía paso a paso el proceso, y permite tener listo el servidor en poco tiempo.

4.1.2 Instalación de la aplicación

Después de la instalación de la plataforma, se realiza la configuración de la base de datos que la aplicación utilizará. La base de datos, durante la fase de desarrollo, es creada de forma manual mediante un script que contiene estructuras SQL DDL (Data Definition Language) que crean las tablas, columnas, claves primarias, índices y reglas de integridad referencial. Ya que la base de datos requiere de datos prefijados que son creados a medida que la aplicación es desarrollada (como datos de apoyo, datos importados de bases de datos antiguas, etc.), al final del desarrollo, mediante la ayuda de alguna herramienta (como la incluida en MySQL, mysqldump) se crea un script SQL con todo lo necesario para reproducir la base de datos en cualquier servidor. En el servidor de base de datos de producción, se crea la base de datos a partir del script generado, utilizando una herramienta de terceros o mediante las herramientas incluidas en MySQL. Generalmente las herramientas de terceros y comerciales (como MySQL-Front o EMS) facilitan las operaciones de importación y exportación de datos para traslados entre servidores de bases de datos, las herramientas incluidas con MySQL requieren de conocimientos un poco más técnicos y de uso de la línea de comandos.

Una vez realizada la configuración de la base de datos, se debe realizar la conexión entre el servidor de aplicaciones y el servidor de base de datos. Ambos servidores pueden residir en el mismo servidor físico, o en servidores separados. Primeramente se debe conseguir el manejador JDBC (Java Data Base Connectivity) del servidor de base de datos, en este caso específico, de MySQL (Connector-J). Si se utiliza otro servidor, como Oracle, PostgreSQL, etc., se debe conseguir el manejador JDBC específico del mismo. Dentro del administrador del Sun AppServer, se debe configurar el Classpath (ruta de búsqueda de clases Java, es decir de librerías y recursos) para que pueda leer el archivo .JAR que contiene el manejador JDBC. Una vez reiniciado el servidor de aplicaciones, se debe configurar un Connection Pool. Esta técnica de conexión permite abrir múltiples conexiones entre el servidor de aplicaciones y el servidor de base de datos, reutilizando aquellas que no se hayan cerrado para evitar crear demasiadas conexiones y sobrecargar el servidor. El servidor de aplicaciones pedirá el nombre completo de la clase del manejador JDBC (esto se debe consultar en la documentación del manejador), y los parámetros de conexión como dirección del servidor de base de datos, puerto, nombre de base de datos, nombre de usuario y contraseña. Una vez creado el Connection Pool, se debe realizar una referencia en el contenedor de nombres del servidor de aplicaciones. Para la presente aplicación, el Connection Pool será buscado en “comp/env/jdbc/FulbrightDB”. Se debe consultar la documentación del contenedor / servidor de aplicaciones para este procedimiento de registro.

La aplicación, después de haber sido desarrollada, es compilada y empaquetada en un archivo .WAR. Dependiendo de las herramientas de desarrollo (IDEs como Eclipse o NetBeans, o herramientas de línea de comandos como Ant), estas ayudan a armar un archivo .WAR con las configuraciones necesarias para cualquier servidor de aplicaciones.

El deploy de la aplicación (es decir, subir la aplicación al servidor de aplicaciones / contenedor para su funcionamiento) se lo puede realizar de forma manual, copiando el .WAR al directorio de aplicaciones del servidor, o mediante las herramientas de deploy que generalmente se incluyen con el servidor de aplicaciones, o en el ambiente de desarrollo (IDE). Una vez realizado el deploy de la aplicación, está lista para ser utilizada.

4.2 Interacción de los componentes

4.2.1 Interacción entre cliente y servidor

La presente aplicación es una aplicación Web. Esto significa que para acceder a las funciones de la misma, se lo hace de la misma manera que un sitio Web cualquiera: a través de un Navegador de la Web (Web Browser). Al igual que los sitios Web, la aplicación funciona básicamente mediante el esquema de pedido-respuesta. Esto quiere decir que el cliente, para interactuar con el sistema, realiza “pedidos” al mismo, y el servidor, para mostrar sus resultados, devuelve “respuestas” al cliente. La comunicación vía Web Browser tiene la limitación de utilizar el protocolo HTTP (HyperText Transfer Protocol) que no mantiene el estado entre llamadas, es decir, que de una llamada a otra se pierde la memoria de lo que se hizo anteriormente. Este problema de falta de estado (lo que en inglés se conoce como “stateless”), se soluciona mediante variadas técnicas, como lo es mantener la sesión del cliente. Esto se logra mediante técnicas en el servidor y/o el cliente, que mantiene para cada cliente un identificador, ya sea en la memoria del servidor o como un “cookie” (un pequeño segmento de datos, que no tiene ningún significado útil para el cliente pero para el servidor tiene algún uso específico) en el lado del cliente, que lo identifica al servidor como una sesión abierta. Con esta solución, el servidor puede rastrear y distinguir los pedidos de los clientes. El siguiente gráfico (Figura 4.2.1) resume de forma

simplificada el esquema de alto nivel de la interacción y comunicación entre los componentes del sistema.

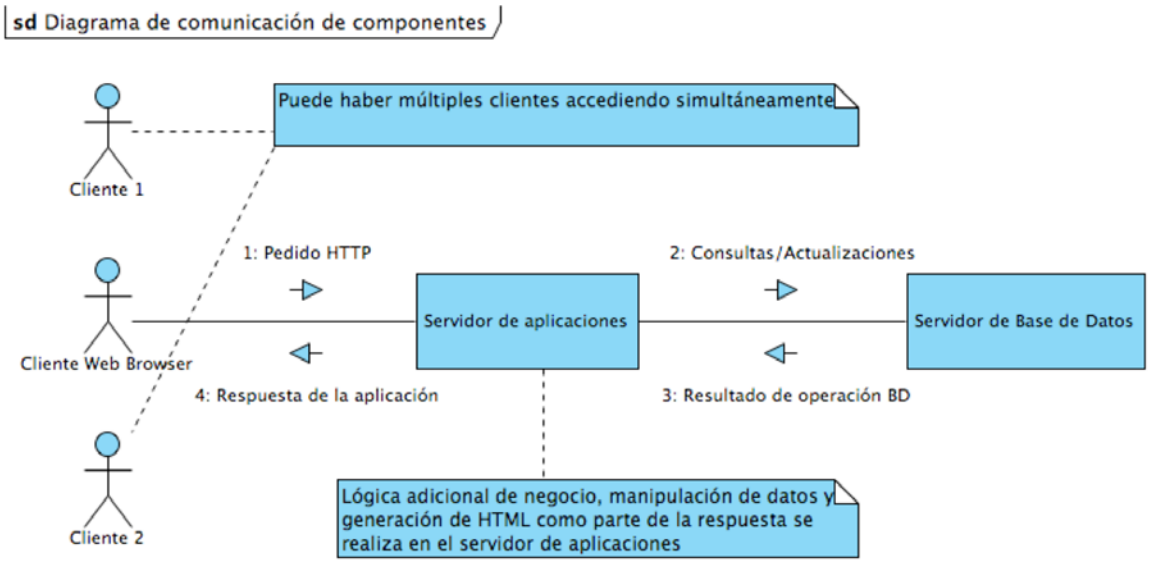


Figura 4.1 – Interacción de los componentes de la aplicación

Generalmente la respuesta del servidor será una página HTML para ser interpretada por el Web Browser, que graficará la interfaz para el usuario, además de los resultados del pedido. La respuesta también podría consistir en un documento XML, especialmente en pedidos de tipo AJAX (Asynchronous Javascript And XML) que es una técnica utilizada por los Web Browsers modernos, en la cual se realiza un pedido al servidor, y la respuesta no refresca toda la página, sino solamente una parte de ella, como por ejemplo actualizar una lista sin afectar a los demás componentes de la página. Esta técnica permite crear aplicaciones con mayor nivel de interactividad y mayor velocidad de respuesta.

4.2.2 Interacción entre componentes internos

La aplicación, como todo software estándar, requiere de datos y procedimientos para manipular estos datos y obtener resultados. La aplicación se encarga de coordinar las tareas de obtención de datos desde la base de datos, procesarlos, formatearlos para enviarlos al

cliente, y construir la respuesta que se enviará al cliente. El servidor de base de datos provee el repositorio para los datos, además de herramientas de consulta y modificación de datos que pueden ser llamadas desde la aplicación.

Para proveer modularidad y mayor facilidad de mantenimiento, la aplicación ha sido creada mediante un patrón de diseño conocido como Modelo-Vista-Controlador (Model-View-Controller, MVC). Este patrón de diseño permite separar la lógica en tres capas, una que controla exclusivamente la lógica de negocio y manipulación de datos (Modelo), y otra que se encarga exclusivamente de la lógica de presentación e interfaz de usuario (Vista), la comunicación y coordinación de las dos capas se realiza mediante una tercera que captura los eventos y canaliza las operaciones (Controlador). La interacción básica entre capas consiste en consultas y modificaciones al Modelo. La Vista realiza un pedido, el cual puede contener consultas o modificaciones al Modelo. El Controlador canaliza el pedido a la operación respectiva del Modelo. Éste al recibir el pedido, realiza la consulta o las modificaciones necesarias a los datos, y a través del Controlador devuelve los resultados a la Vista. La figura 4.2 es un diagrama de las interacciones entre las capas de la aplicación.

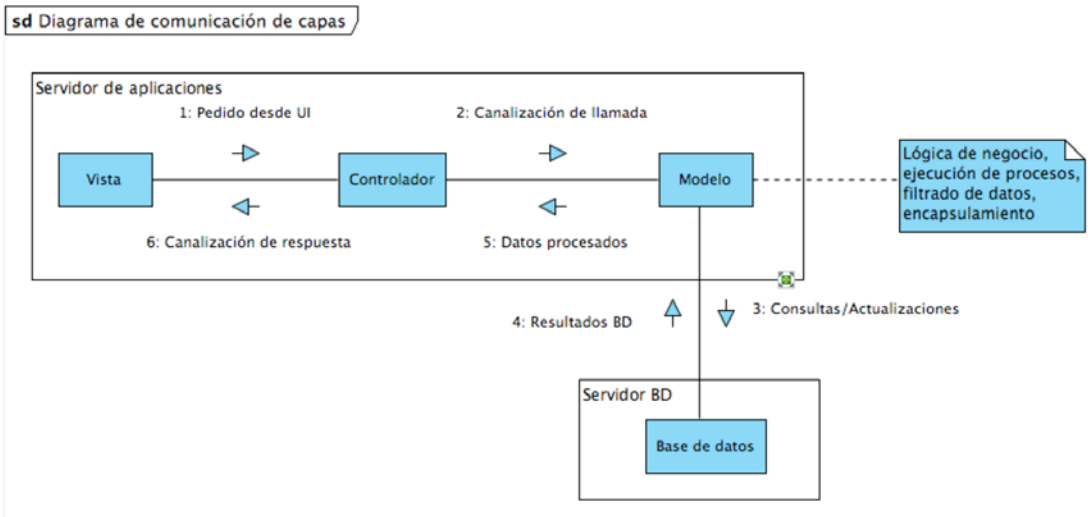


Figura 4.2 - Diagrama de interacción de capas

El Modelo consiste de dos subcapas, una que se encarga de la lógica de negocio, y otra que se encarga de la manipulación de datos. Esta última realiza las consultas y actualizaciones a la base de datos de forma directa, enviando comandos SQL. La lógica de negocio se encarga de procesar los datos obtenidos desde la base de datos, organizarlos, ejecutar algoritmos o procedimientos específicos, y devolver estos datos procesados al cliente.

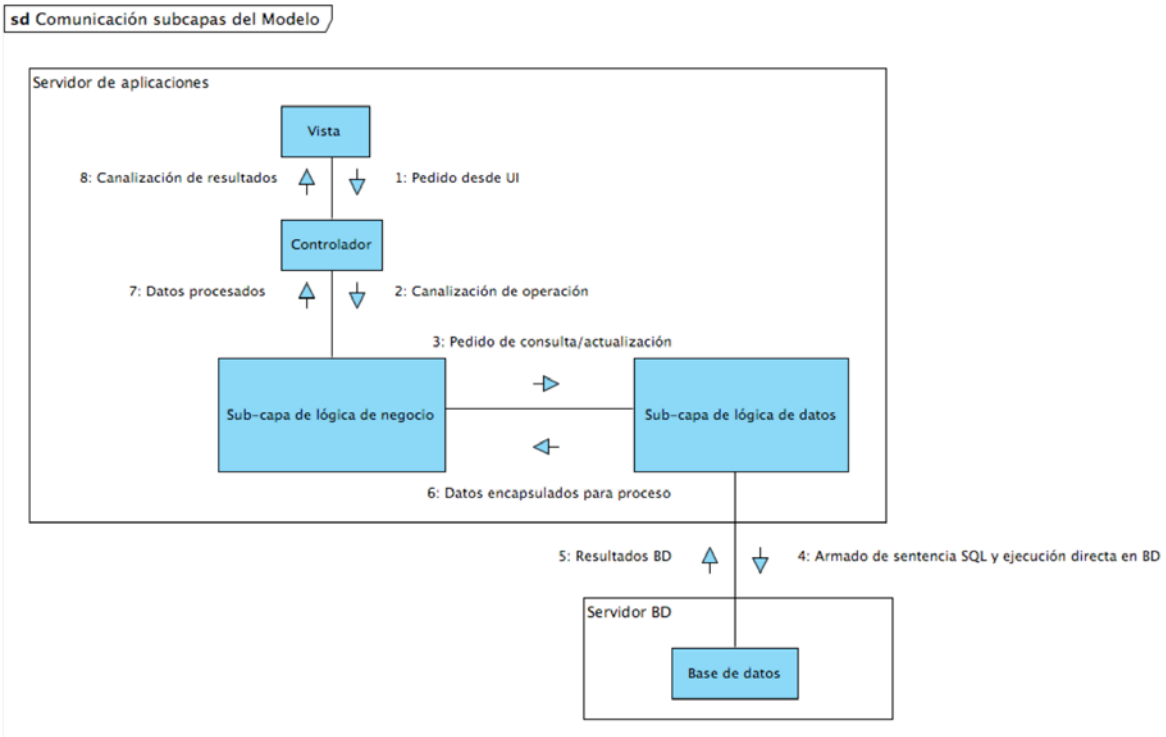


Figura 4.3 – Diagrama detallado de comunicación de sub-capas del Modelo

4.3 Rendimiento del sistema

En general la aplicación, una vez probada en ambiente de desarrollo y en ambiente de producción, posee un nivel de respuesta rápido, casi inmediato, limitado simplemente por las capacidades de dibujo (rendering) del Web Browser y la velocidad de la red. Hay que tomar en cuenta que este sistema no tendrá mucha carga de usuarios, ya que la información y procedimientos dentro del mismo concierne solamente al departamento académico de la Comisión Fulbright, es decir, la carga máxima que se prevé para el sistema de forma

simultánea será entre 5 y 8 usuarios. La red en el ambiente de producción tampoco es un factor limitante, ya que además de poseer pocos usuarios, es de infraestructura Fast Ethernet (100 Mbps, Megabits por segundo) y conmutada (es decir, utiliza un “switch”, con lo que se establecen conexiones de punto a punto sin colisiones). La máquina que contiene la aplicación y la base de datos (es decir, el servidor) ha sido configurada exclusivamente para este propósito y no cumple otras funciones, en otras palabras, es una máquina totalmente dedicada a la aplicación. A continuación se presentan los resultados de las pruebas realizadas para evaluar el rendimiento del sistema.

4.3.1 Ambiente de pruebas

El sistema se probó en el ambiente de producción, es decir en la Comisión Fulbright. Se realizó la instalación de la aplicación en el servidor (basado en Linux Fedora Core), y se realizó un respaldo de la base de datos para poder realizar las pruebas de inserción de datos. Se utilizaron 5 clientes para realizar las pruebas bajo carga normal de uso. Todos los clientes son parte de la LAN de la Comisión Fulbright. En el código se añadió un cálculo de tiempo, que lo mide desde que se realiza el pedido, hasta que se termina su procesamiento en el servidor. No se midieron tiempos de dibujo de página en el Web Browser, ya que estos dependen del tipo de navegador que se esté utilizando, y además el dibujo de las páginas de esta aplicación fue lo bastante rápido, ya que el HTML generado no es pesado y viaja en una conexión de área local.

4.3.2 Pruebas realizadas en el sistema

Se probaron las capacidades de inserción, consulta, y procesamiento de datos. A continuación los resultados obtenidos y promediados de las operaciones más importantes que se realizaron en la prueba:

Operación	Tiempo en milisegundos
Inserción de aplicación de becario con todos los campos llenos, con consulta inmediata después de la inserción	25 mseg
Consulta de datos básicos de becario	8 mseg
Inserción de perfil académico (es decir, universidad en la que el aplicante estudió) con consulta inmediata	101 mseg
Consulta de perfil académico con hasta 3 universidades	52 mseg
Consulta de exámenes de becario/aplicante	95 mseg
Inserción de examen de aplicante con consulta inmediata	98 mseg
Inserción de resultado de examen de aplicante (puntaje) con consulta inmediata	98 mseg
Proceso de apertura de un programa	50 mseg
Consulta de programas abiertos	6 mseg
Apertura de formulario de aplicación a programa	315 mseg
Ingreso de datos del formulario de aplicación a programa con consulta inmediata	80 mseg
Apertura de formulario de aplicación a programa para edición (con datos)	430 mseg
Edición de concesiones económicas	45 mseg
Consulta de programas aplicados (un solo programa)	55 mseg
Consulta de programas aplicados (múltiples programas)	99 mseg
Detalles de aplicante para pre-selección	224 mseg
Proceso de eliminación previa a pre-selección	86 mseg
Proceso de calificación para pre-selección	220 mseg
Marcar como pre-seleccionados a los aplicantes	50 mseg
Grabación de puntajes de entrevista	89 mseg
Proceso de calificación para selección final	203 mseg
Marcar como becarios seleccionados a aplicantes	50 mseg
Consulta de todos los aplicantes, paginada, con 20 entradas por página	13 mseg
Consultas de datos de mantenimiento	Entre 5 y 10 mseg
Promedio de tiempo de operaciones individuales:	103.8 mseg

Tabla 4.1 – Tiempos obtenidos en pruebas

Como se puede observar, los resultados están en el orden de los milisegundos, es decir que la aplicación está respondiendo de manera instantánea para la mayoría de operaciones, tomando en el peor de los casos, medio segundo. Esto significa que los usuarios pueden utilizar esta aplicación Web casi como una aplicación de escritorio, y elimina las quejas

por parte de los usuarios con respecto al rendimiento de la aplicación. Con respecto al uso de recursos de computación, se observó un uso de un promedio de 100 MB de memoria RAM, el servidor de aplicaciones require de mínimo 512 MB, pero el servidor tiene instalado 1 GB completo, con lo que los recursos de memoria no son un problema. La base de datos físicamente ocupa alrededor de 1.11 MB en disco, esto incluyendo alrededor de 6000 registros de datos con toda la información histórica de la Fulbright. La aplicación en sí (excluyendo servidor de aplicaciones / contenedor) compilada, no ocupa más allá de 2.1 MB. Esto significa que puede ser instalada en un servidor de aplicaciones / contenedor más liviano, e incluso una máquina con especificaciones reducidas.

CAPÍTULO 5

5.1 Conclusiones

- El proceso actual de manejo de información en la Comisión Fulbright no posee ningún tipo de organización a nivel de sistemas y todo se encuentra en archivos dispersos, lo que provoca duplicación de información, dificultades en la búsqueda de datos y falta de consistencia en la ejecución de procesos.
- El proceso de selección de becarios, aunque está bien delineado y organizado, y muchos de sus procesos necesariamente deben ser manualmente realizados, no está automatizado en las tareas de cálculos de puntajes para posicionamiento (*ranking*), y al no haber un repositorio central de datos de aplicantes, necesita de procesos manuales de copia de archivos para sincronizar la información entre las personas encargadas de este proceso.
- La metodología de desarrollo iterativa permitió realizar el sistema de forma planificada, primeramente obteniendo un sistema simple pero estable de mantenimiento de datos, y en la siguiente iteración, aprendiendo de los aciertos y errores de la iteración anterior, permitió crear un sistema completo, con todas las características requeridas por la Comisión.
- El diseño en tres capas, unido al patrón MVC (Modelo-Vista-Controlador), permitió desarrollar un código debidamente organizado, de fácil mantenimiento, que permite la adición de nuevas características sin afectar los procesos y el código existente.
- Ahora se tiene un sistema automatizado de procesos de pre-selección y selección, y centralizado de información que puede ser accedido desde cualquier computador al interior de la Comisión Fulbright.

- La aplicación puede correr en un servidor de cualquier plataforma (Windows, Linux, Unix, Mac OS X, etc.) simplemente consiguiendo una máquina virtual de Java, un servidor de aplicaciones / contenedor apropiado y una versión de la base de datos para la plataforma, sin necesidad de recompilar la aplicación.
- El sistema, al ser basado en Web, permite que múltiples usuarios se conecten al mismo tiempo desde cualquier computador al interior de la red de la Comisión Fulbright.
- El sistema posee una velocidad de respuesta en el orden de los milisegundos, lo que se traduce como resultados instantáneos de las operaciones para el usuario.
- La alta velocidad de respuesta hace a la aplicación casi tan ágil como una aplicación de escritorio, si es utilizada dentro de la red local.
- Los recursos de computación requeridos por la aplicación son mínimos, lo que permite ser instalada en un servidor que no necesariamente tenga lo último en tecnología.

5.2 Recomendaciones

- Dado que el sistema es ampliable, se recomienda incorporar tareas como el envío automático de notificaciones vía correo electrónico a aquellos aplicantes que han sido eliminados o seleccionados como becarios, que alivianará la carga a los usuarios de realizar manualmente este tipo de tareas.
- Se recomienda realizar una conexión entre el sistema financiero utilizado en la Comisión Fulbright y el módulo de manejo de concesiones económicas de la aplicación para evitar la duplicación innecesaria de datos, y la integración de los sistemas para evitar duplicación de esfuerzos.
- Ya que la aplicación utiliza funciones específicas de MySQL, si se desea un soporte de tipo comercial se recomienda adquirir el paquete de soporte de MySQL, resultaría

menos costoso que migrar la aplicación a otra solución comercial como Oracle, Microsoft SQL Server, DB2, etc.

- Aunque el sistema se desarrolló utilizando técnicas estandarizadas y aceptadas en la industria, en la actualidad existen nuevos *frameworks* que facilitan aún más el desarrollo de software. Para implementaciones de sistemas similares en el futuro, se recomienda utilizar cualquiera de estos *frameworks* modernos para agilitar el desarrollo y puesta en funcionamiento de los mismos.
- Si los recursos de computación son limitados en el servidor, se recomienda el uso de contenedores más livianos que Sun Application Server, como Apache Tomcat.
- Se recomienda que el administrador del sistema sea alguien capacitado en tareas técnicas de computación, ya que la instalación y puesta en funcionamiento del sistema son procedimientos que requieren de conocimientos técnicos.

ANEXO 1

MANUAL TÉCNICO

TABLA DE CONTENIDO

Público objetivo _____ 44
Objetivo del manual _____ 44
1. Instalación del sistema _____ 44
1.1 Estructura de directorios en la aplicación _____ 47
2. Descripción funcional del Producto _____ 49
2.1 Flujo general de pedidos y respuestas _____ 49
2.2 Procesos generales de manejo de datos _____ 52
2.3 Relación entre capa de datos y negocio _____ 58
2.4 Generador automático de tablas de resultados _____ 59
2.5 Procesos de pre-selección y selección final _____ 64
3. Descripción estructural del Producto _____ 69
3.1 Base de datos _____ 70
3.2 Descripción estructural de la aplicación _____ 72
3.2.1 Modelo _____ 73
3.2.1.1 Capa de datos _____ 74
3.2.1.2 Capa de lógica de negocio _____ 77
3.2.2 Vista _____ 83
3.2.3 Controlador _____ 86

Público objetivo

Este manual está dirigido a desarrolladores de software que deseen conocer el funcionamiento y procedimientos de la aplicación, o realizar mantenimiento del código de la aplicación.

Objetivo del manual

Este manual tiene como objetivo explicar los principales procesos de la aplicación, y su estructura subyacente de base de datos y clases, para facilitar el mantenimiento de la misma.

1. Instalación del sistema

Primeramente se debe configurar el computador que actuará como servidor de la aplicación. Para esto, se debe instalar el servidor de aplicaciones Sun Java Application Server, y el DBMS MySQL 5. Estos paquetes vienen con su respectiva documentación para instalación en cualquier plataforma (Windows, Mac OS X, Linux, etc.).
Una vez instalados estos paquetes base, se debe configurar la base de datos. Se incluye el script SQL de creación de la base de datos, que se puede ejecutar mediante la herramienta "mysqldump" incluida con MySQL, o a través de una interfaz de administración de MySQL como EMS o MySQL-Front. El nombre de la base de datos debe ser "fulbright_db".

Una vez creada la base de datos, se debe realizar la conexión entre MySQL y Sun AppServer. Primero se debe conseguir el manejador JDBC de MySQL (Connector-J) desde el sitio Web de MySQL. Después, se debe ingresar al administrador del servidor de

aplicaciones (generalmente http://[dirección_del_servidor]:4848, consultar los puertos fijados durante la instalación del servidor). Dentro del administrador, se debe ingresar a la página de configuración (Application Server → JVM Settings → Path Settings) y añadir al *Classpath Suffix* la localización del .JAR del manejador JDBC de MySQL.

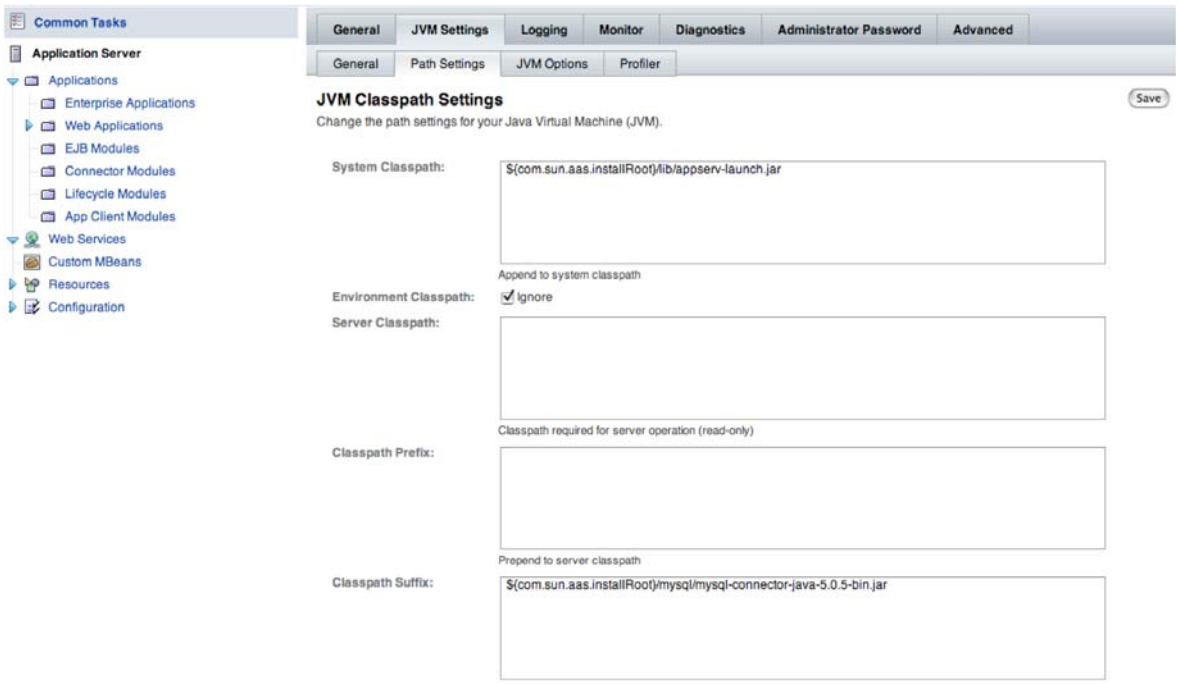


Figura 1.1 – Fijación del *Classpath* del manejador JDBC

Después de reiniciar el servidor, se debe reingresar al administrador, y en la opción Resources → JDBC → Connection Pools, se crea el recurso de conexión hacia la base de datos. El administrador pedirá el nombre completo de la clase del manejador JDBC (para Connector-J es “org.mysql.jdbc.jdbc2.optional.MysqlConnectionPoolDataSource”), y los datos de conexión como nombre del servidor, puerto, nombre de la base de datos, y credenciales de acceso.

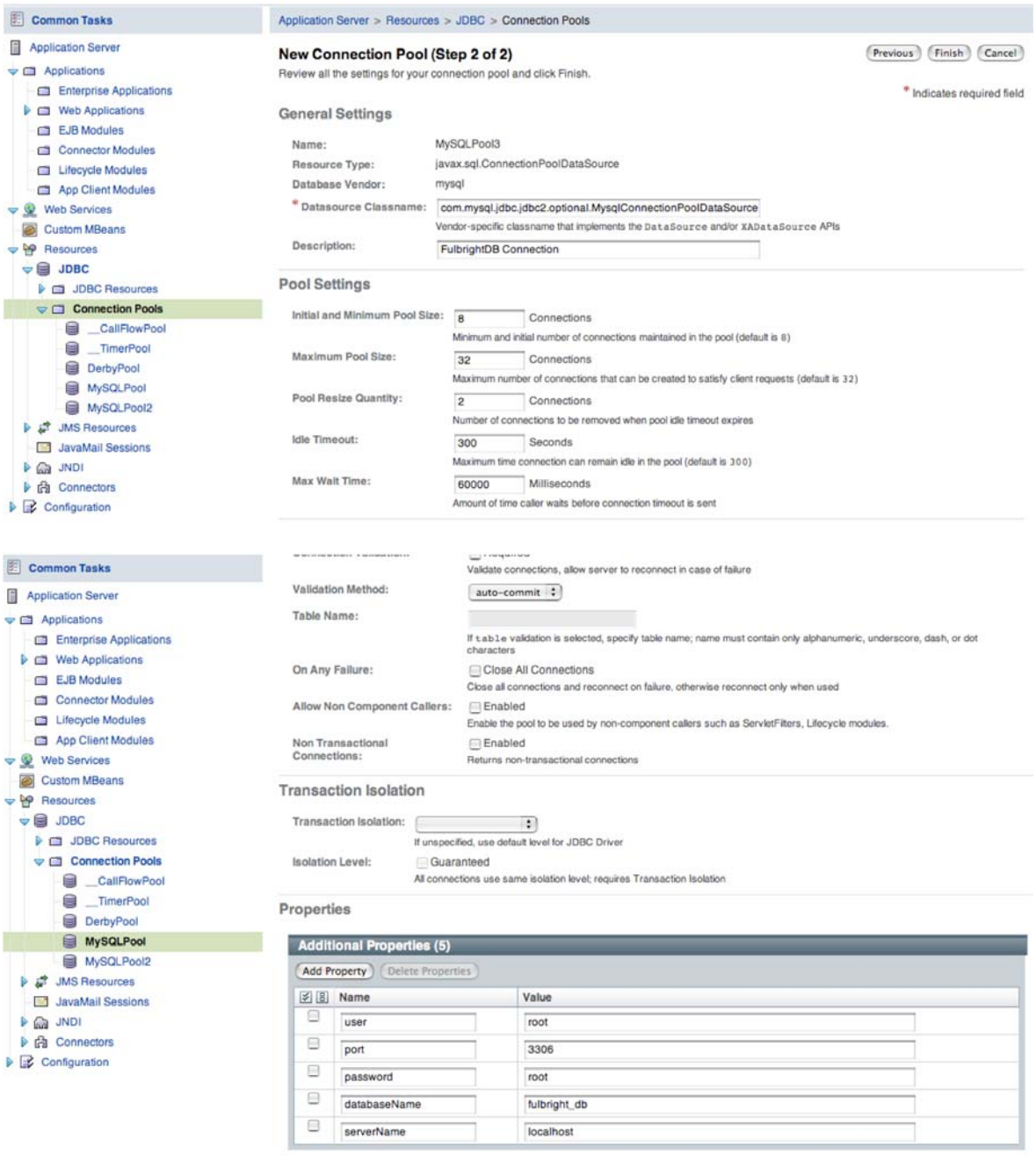


Figura 1.2 – Creación del *Connection Pool*

Una vez creado el *Connection Pool*, se debe crear una referencia a través de Resources → JDBC → JDBC Resources. El servidor pedirá un nombre JNDI para localizar la conexión (en este caso específico, deberá ser “jdbc/FulbrightDB”) y el *Connection Pool* que se creó anteriormente.

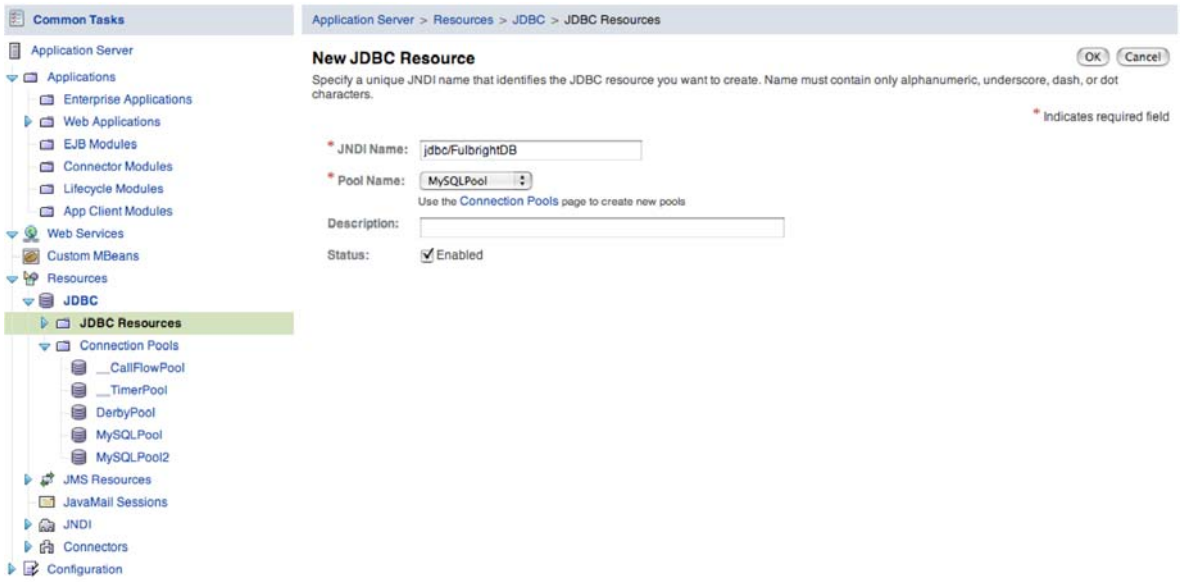


Figura 1.3 – Creación de la referencia JNDI de la conexión a la base de datos

La aplicación se encuentra precompilada y empaquetada en un archivo .WAR configurado para Sun Java Application Server. En el caso de utilizarse algún otro contenedor, se debe reempaquetar la aplicación con un archivo de configuración específico para el contenedor deseado (consultar la documentación del contenedor). Para Sun AppServer, se puede utilizar la herramienta *Deployer* para instalar la aplicación en el servidor, o a través de las herramientas de *Deploy* incluidas en cualquier IDE como Eclipse, NetBeans, JDeveloper, etc. En el caso de otros contenedores, consultar la documentación respectiva.

1.1 Estructura de directorios de la aplicación

El archivo .WAR contiene la siguiente estructura de directorios de la aplicación:

```
Raíz de Fulbright2006.war
+- images
+- js
+- META-INF
+- styles
+- templates
|  +- contents
+- WEB-INF
|  +- classes
|  |  +- org
|  |     +- fulbright
|  |        +- business
|  |        |  +- applications
|  |        |  +- programs
|  |        |  +- reports
|  |        |  +- selection
|  |        |  |  +- data
|  |        |  +- users
|  |        |  +- validators
|  |        |  +- xml
|  |        +- datalayer
|  |        |  +- tables
|  |        +- webapp
|  |           +- actions
|  |           +- filters
|  |           +- listeners
|  |           +- servlets
|  +- lib
+- xml
   +- reports
```

El directorio *images* contiene todos los elementos gráficos de la interfaz de usuario. El directorio *js* contiene todos los JavaScripts utilizados en la aplicación para crear mayor interactividad en la interfaz Web del cliente. El directorio *META-INF* es de uso reservado. El directorio *styles* contiene los archivos con los estilos CSS (*Cascading Style Sheets*) que se aplican a la interfaz de usuario para darle una mejor apariencia. El directorio *templates* contiene todas las plantillas HTML utilizadas por el motor de plantillas FreeMarker para generar la interfaz de usuario. El directorio *WEB-INF* contiene la parte principal de la aplicación. Dentro de este directorio, el subdirectorio *classes* contiene los paquetes con el código compilado de la aplicación. El subdirectorio *lib* contiene las librerías externas (archivos .JAR) utilizadas en la aplicación. Y finalmente el directorio *xml* contiene los archivos XML utilizados en la aplicación.

2. Descripción funcional del Producto

La aplicación ha sido desarrollada en un modelo de 3 capas, siguiendo el patrón Modelo-Vista-Controlador (MVC). La aplicación básicamente funciona mediante llamadas realizadas desde el lado del cliente, en este caso un Web browser, al servidor, y éste a su vez devuelve una respuesta con los resultados de la llamada. Esta es la manera básica en que la mayoría de aplicaciones Web, desde la más sencilla hasta la más compleja, funcionan. Pero para organizar todas estas llamadas, canalizarlas, y realizar las operaciones apropiadas en el servidor, se necesita seguir un patrón estandarizado para garantizar el flujo correcto de los pedidos y respuestas.

2.1 Flujo general de pedidos y respuestas

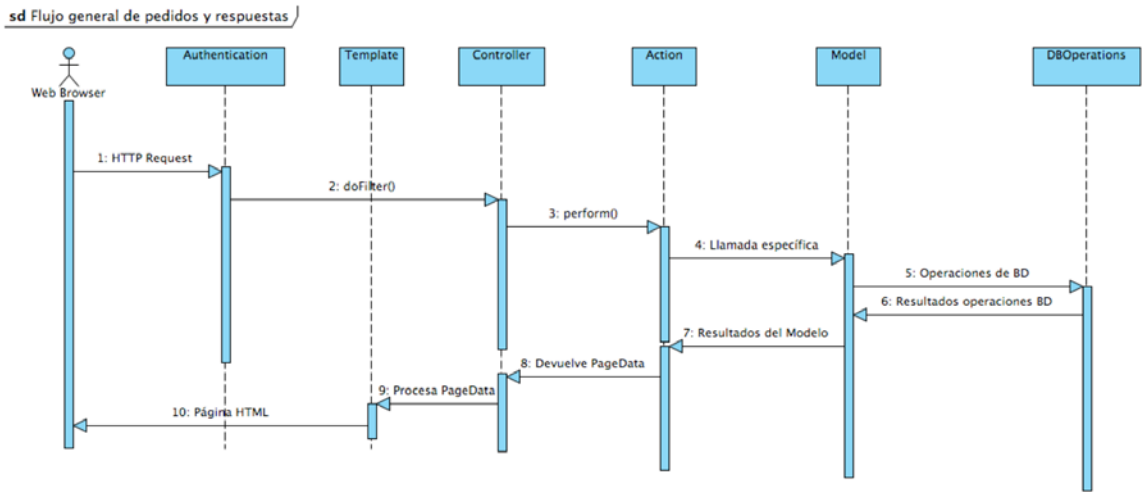


Figura 2.1 – Flujo general de pedidos y respuestas

Los pedidos se originan en el lado del cliente. El Web browser realiza los pedidos mediante el protocolo http, y los métodos utilizados son GET y POST generalmente. El pedido llega al servidor, y antes de pasar por cualquier servlet, primeramente llega al filtro org.fulbright.webapp.filters.Authentication. Este filtro se encarga de validar el pedido,

verificando si pertenece a la sesión de un usuario activo. Si no es así, el pedido es desviado al Controlador con la acción “login”, que lo único que provoca es que el servidor devuelva la página de ingreso al sistema. Si el pedido corresponde a una sesión activa, entonces el pedido se direcciona a org.fulbright.webapp.servlets.Controller. Éste contiene una colección de objetos que heredan de org.fulbright.webapp.actions.Action. Cada uno corresponde a una acción específica, y se seleccionan mediante el parámetro “action” que debe ser obligatoriamente provisto en todos los pedidos al servidor. Mediante polimorfismo, se llama al método perform() del objeto respectivo. En este método se realizan las acciones específicas de cada pedido, que generalmente consisten en evaluar los parámetros, llamar a los objetos de la capa subyacente y ejecutar los procesos de negocio. En el caso de pedidos que impliquen modificaciones al Modelo subyacente, los parámetros del pedido son pasados al objeto (u objetos) del Modelo, y en esta capa se realizan los cambios necesarios. En el caso de pedidos de consulta, se llama a el objeto de Modelo respectivo, que arma la estructura necesaria con los datos de la consulta, y es devuelta al objeto de acción. Como paso final, el objeto de acción crea un objeto de tipo org.fulbright.webapp.actions.PageData, que contiene los datos en una estructura que puede ser manipulada por el motor de plantillas (templating) Freemarker. Objetos del tipo PageData son siempre el resultado de los llamados a perform(). El objeto Controller combina los datos provistos en el objeto PageData (como los provistos en los métodos getPageLoad() y getTplData()), las plantillas son interpretadas mediante los objetos de Freemarker, y las páginas resultantes son enviadas en la respuesta del servidor hacia el cliente en forma de HTML estándar.

Segmento de código 2.1: Filtrado de pedidos en Authentication.java

```
public void doFilter(ServletRequest request, ServletResponse response,
            FilterChain chain)
            throws IOException, ServletException {
```

```
        HttpServletRequest req = (HttpServletRequest) request;
        HttpSession session = req.getSession();
        String resource = req.getServletPath();
        String ext = resource.substring(resource.lastIndexOf(".") + 1);

        if (!ext.equals("js") && !ext.equals("css") && !ext.equals("gif")
&& !ext.equals("jpg")) {
            if (session.getAttribute("userObj") == null &&
request.getParameter("user") == null) {

request.getRequestDispatcher("/Controller?action=login").forward(request,
response);
                return;
            }
        }

        chain.doFilter(request, response);
    }
```

Segmento de código 2.2: Despachador de acciones en Controller.java

```
protected void processRequest(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
        request.setCharacterEncoding("UTF-8");
        String screen = request.getServletPath();
        String act = request.getParameter("action");

        if (screen.equals("/index"))
            act = "index";

        Action action = (Action) actions.get(act);
        PageData result = null;

        try {
            action.setContext(getServletContext());
            result = action.perform(request);
        } catch (NullPointerException ex) {
            ex.printStackTrace();
        }

        if (result == null) {
            result = new PageData(request.getSession());
            result.setPageLoad("error.html");
        }

        loadHTMLForm(request, response, result);
    }

    protected void loadHTMLForm(HttpServletRequest request,
    HttpServletResponse response, PageData pgData)
    throws ServletException, IOException {

        Configuration tplCfg = (Configuration)
            getServletContext().getAttribute("tplCfg");

        Template tpl = tplCfg.getTemplate(pgData.getPageLoad());

        response.setContentType("text/html;charset=UTF-8");
```

```
        PrintWriter out = response.getWriter();

        try {
            tpl.process(pgData.getTplData(), out);
        } catch (Exception ex) {
            ex.printStackTrace();
            out.println("Error loading template.");
        }

        out.flush();
        out.close();
    }
```

Segmento de código 2.3: Clase base Action.java

```
public abstract class Action {
    protected ServletContext context;

    public Action() {
    }

    public void setContext(ServletContext ctx) {
        context = ctx;
    }

    public abstract String getName();

    public abstract PageData perform(HttpServletRequest req);
}
```

2.2 Procesos generales de manejo de datos

El paquete org.fulbright.datalayer contiene clases que permiten manejar el acceso a la base de datos y obtener los datos de consultas en estructuras de datos organizadas. El objeto org.fulbright.webapp.DBOperations contiene los métodos de consulta de datos directQuery() y callStoredProcQuery(). El primero acepta una sentencia SQL “select” directa en una cadena de caracteres estándar. El segundo método acepta una cadena con el nombre del procedimiento de base de datos a llamar, y un java.util.ArrayList como segundo argumento, que contiene de forma ordenada los parámetros a pasar al procedimiento. Ambos procedimientos devuelven una estructura org.fulbright.datalayer.DataSetWrapper. Esta estructura contiene, en forma de filas que pueden navegarse con el método fetchRow(), desde la fila 0 hasta la fila n, todos los datos

de la consulta. Cada columna se accede mediante getColumnValue(), que devuelve un valor de tipo Object. Esto quiere decir que se mantiene el tipo de dato de la columna, y debe hacerse el cast apropiado. Internamente, DBOperations utiliza la información de metadatos de la base de datos para convertir los datos obtenidos de la consulta en sus equivalentes en el lenguaje Java. Para el caso de modificaciones de la base de datos, DBOperations provee los métodos simpleInsert(), simpleUpdate() y simpleDelete(). En los dos primeros métodos, se debe proveer el nombre de la tabla, una estructura ordenada tipo java.util.LinkedHashMap con los valores a insertar (en la forma campo, valor), y en el caso de simpleUpdate(), también se debe proveer un LinkedHashMap adicional con los valores de condición. En el método simpleDelete() simplemente se debe proveer el criterio de condición en forma de cadena. Estos métodos devuelven el número de filas afectadas en la tabla en la cual se ejecutaron.

Segmento de código 2.4: Métodos principales de DBOperations.java

```
public DataSetWrapper callStoredProcQuery(String prcName, ArrayList
paramList) {
        ArrayList result = new ArrayList();
        Connection conn = dbConn.getConnection();
        String sp = buildStoredProcCall(prcName, paramList);

        try {
            CallableStatement callSt = conn.prepareCall(sp);

            if (paramList.size() > 0)
                setParams(callSt, paramList);

            ResultSet rs = callSt.executeQuery();

            setResult(rs, result);

            callSt.close();
            rs.close();

        } catch (Exception ex) {
            ex.printStackTrace();
        }

        if (result.size() > 0)
            return new DataSetWrapper(result);

        return new DataSetWrapper(null);
    }
```

```
public DataSetWrapper directQuery(String select) {
        Connection conn = dbConn.getConnection();
        ArrayList result = new ArrayList();
        int rowCount = 0;

        try {
            PreparedStatement prepStat = conn.prepareStatement(select);
            PreparedStatement rcPrepStat = conn.prepareStatement("select
FOUND_ROWS()");

            ResultSet rs = prepStat.executeQuery();
            ResultSet rcRs = null;

            if (rs != null)
                rcRs = rcPrepStat.executeQuery();

            setResult(rs, result);

            if (rcRs != null && rcRs.next())
                rowCount = rcRs.getInt(1);

            prepStat.close();
            rcPrepStat.close();
            rs.close();

            if (rcRs != null)
                rcRs.close();
        } catch (Exception ex) {
            ex.printStackTrace();
        }

        if (result.size() > 0) {
            DataSetWrapper dataSet = new DataSetWrapper(result);
            dataSet.setRowCount(rowCount);

            return dataSet;
        }

        return new DataSetWrapper(null);
    }

public int simpleUpdate(String table, LinkedHashMap fldVal, LinkedHashMap
condit) {
        Connection conn = dbConn.getConnection();
        Iterator it = fldVal.keySet().iterator();
        String update = "update " + dbConn.getDBName() + "." + table + "
set ";
        ArrayList params = new ArrayList();
        int rowAff = 0;

        while (it.hasNext()) {
            String fld = (String) it.next();
            update += fld + " = ?";

            params.add(fldVal.get(fld));
            if (it.hasNext())
                update += ", ";
        }
```

```
        update += " where ";

        it = condit.keySet().iterator();

        while (it.hasNext()) {
            String fld = (String) it.next();
            update += fld + " = ?";

            params.add(condit.get(fld));

            if (it.hasNext())
                update += " and ";
        }

        try {
            PreparedStatement prepStat = conn.prepareStatement(update);
            setParams(prepStat, params);

            rowAff = prepStat.executeUpdate();
            prepStat.close();

        } catch (Exception ex) {
            rowAff = -1;
            ex.printStackTrace();
        }

        return rowAff;
    }

public int simpleInsert(String table, LinkedHashMap fldVal) {
        Iterator it = fldVal.keySet().iterator();
        String insert = "insert into " + dbConn.getDBName() + "." + table
+ " (";
        String values = ") values (";
        ArrayList paramList = new ArrayList();
        Connection conn = dbConn.getConnection();
        int rowAff = 0;

        while (it.hasNext()) {
            String fld = (String) it.next();
            insert += fld;
            values += "?";

            paramList.add(fldVal.get(fld));

            if (it.hasNext()) {
                insert += ", ";
                values += ", ";
            }
        }

        insert += values + ")";

        try {
            PreparedStatement prepStat = conn.prepareStatement(insert);
            setParams(prepStat, paramList);

            rowAff = prepStat.executeUpdate();
            prepStat.close();
```

```
        } catch (Exception ex) {
            rowAff = -1;
            ex.printStackTrace();
        }

        return rowAff;
    }

public int simpleDelete(String table, String condit) {
        Connection conn = dbConn.getConnection();
        int rowAff = 0;

        try {
            String delete = "delete from " + dbConn.getDBName() + "." +
                    table + " where " + condit;
            PreparedStatement prepStat = conn.prepareStatement(delete);
            rowAff = prepStat.executeUpdate();

            prepStat.close();
        } catch (Exception ex) {
            rowAff = -1;
            ex.printStackTrace();
        }

        return rowAff;
    }
```

Segmento de código 2.5: Clase DataSetWrapper.java

```
public class DataSetWrapper {
    private ArrayList dataSet;
    private HashMap currRec;
    private int rowPointer = -1;
    private int rowCount = 0;

    public DataSetWrapper(ArrayList dataSet) {
        setDataSet(dataSet);
    }

    public void setDataSet(ArrayList dataSet) {
        this.dataSet = dataSet;
        currRec = null;
        rowPointer = -1;
    }

    public void setRowCount(int rowCount) {
        this.rowCount = rowCount;
    }

    public int getTotalRows() {
        if (dataSet != null)
            return dataSet.size();
        else
            return 0;
    }

    public int getRowCount() {
        return rowCount;
```

```
    }

    public int getCurrentRow() {
        return rowPointer;
    }

    public void fetchRow(int rowNum) {
        int totRows = getTotalRows();
        rowPointer = rowNum;

        if (rowPointer >= totRows)
            rowPointer = totRows;
        else if (rowPointer < 0)
            rowPointer = 0;

        currRec = (HashMap) dataSet.get(rowPointer);
    }

    public HashMap getCurrentRecord() {
        return currRec;
    }

    public Iterator currRowColumns() {
        return currRec.keySet().iterator();
    }

    public Object getColumnValue(String column) {
        return currRec.get(column);
    }

    public ArrayList getStringContent() {
        return getStringContent(false);
    }

    public ArrayList getStringContent(boolean withNulls) {
        ArrayList toRet = new ArrayList();
        Iterator it = dataSet.iterator();

        while (it.hasNext()) {
            HashMap conv = new HashMap();
            HashMap orig = (HashMap) it.next();
            Iterator mapIt = orig.keySet().iterator();

            while (mapIt.hasNext()) {
                String fld = (String) mapIt.next();
                Object val = orig.get(fld);

                if (!withNulls || (val != null && withNulls)) {
                    if ((val instanceof Float) || (val instanceof
Double)) {
                        String strVal = String.valueOf(val);
                        String dec = strVal.split("\\.")[1];

                        if (dec.length() < 2)
                            conv.put(fld, strVal + "0");
                        else
                            conv.put(fld, strVal);
                    } else
                        conv.put(fld, String.valueOf(val));
                } else if (val == null && withNulls)
```

```
                    conv.put(fld, null);
                }

                toRet.add(conv);
            }

            return toRet;
        }
}
```

2.3 Relación entre capa de datos y de negocio

Las capas de datos y de negocio componen el Modelo de la aplicación, así que ambas tienen una relación en la cual la capa de negocio depende de la capa de datos. Todos los objetos contenidos dentro del paquete org.fulbright.business necesitan mover datos a través de los objetos de org.fulbright.datalayer. La mayoría de métodos dentro de los objetos de negocios utilizan consultas (queries) propias, específicas para su funcionalidad. Así que el método directQuery() de DBOperations es el más utilizado, con cadenas que contienen las instrucciones SQL Select específicas para las tareas. Pocos métodos, como los contenidos dentro de org.fulbright.business.applications.AppProgramModel, utilizan el método callStoredProcQuery(), ya que hacen llamadas a consultas específicas que han sido pre-programadas en la misma base de datos en forma de procedimientos (stored procedures). Igualmente para modificación de datos, los métodos al interior de las clases de negocio utilizan básicamente los métodos simpleInsert(), simpleUpdate() y simpleDelete(). Manipulando estructuras tipo java.util.LinkedHashMap, se pueden definir los campos a modificar/insertar, y cada método especifica los datos y las tablas a modificar específicos a su función.

2.4 Generador automático de tablas de resultados

El generador automático de tablas de resultados funciona mediante una combinación de XML, funciones en el motor de plantillas y funciones en el servidor. Mediante archivos XML con un formato específico, se definen las propiedades de la tabla a desplegar, como de qué tabla se obtendrán los datos, las columnas a mostrar, formato de las columnas, enlaces a otra página, etc. El corazón del generador se encuentra en la clase org.fulbright.business.TableQuery. Ésta es una clase general, de la cual se puede derivar clases que tengan funcionalidad específica. En el método doSearch(), se lee el archivo XML que es procesado y convertido a una estructura especial mediante la clase org.fulbright.datalayer.tables.DataTable. Se construye la sentencia SQL Select para la obtención de datos de forma automática, y se arman estructuras tanto para las cabeceras de la tabla mediante el método DataTable.getColumnHeaders(), los datos obtenidos de la consulta también se guardan en una estructura especial creada por DataTable.convertDataToTable(). En la capa de Controlador, la acción CommonAction contiene métodos de asignación (setters) para determinar la tabla de la base de datos a manipular, el archivo XML con la plantilla de la tabla, etc. Y finalmente, la plantillas especiales queryTable.html y pageCount.html contienen la lógica de Freemarker necesaria para armar las tablas de forma visual con el HTML apropiado. Los datos de cabecera de la tabla se obtienen del primer elemento del arreglo devuelto por doSearch(), y los datos de la tabla se obtienen del segundo elemento del arreglo devuelto por doSearch(). Los datos de paginación, que es realizada de forma automática en TableQuery, se obtienen con los métodos getCurrentPage() (obtiene la página actual), y getMaxPages() (el número máximo de páginas en la búsqueda). La paginación es posible gracias a las funciones incorporadas de MySQL, que permiten obtener un subconjunto de los datos (con la cláusula LIMIT), y también obtener el conteo total (SELECT FOUND_ROWS()).

Segmento de código 2.6: Método doSearch() en TableQuery.java

```
public ArrayList[] doSearch(String file, boolean withLimit) {
        DBOperations dbOper = getDBOperations();
        file = getContext().getRealPath(file);
        DataTable dt = new DataTable(file);
        String query = dt.buildSelect();

        String relFilter = dt.getRelFilter();
        defaultLink = dt.getLink();

        if (!relFilter.equals("") && !relID.equals("")) {
            relFilter += "'" + relID + "'";
            setFilter(relFilter);
        }

        query += (onFilter != null && !onFilter.equals("") ? " on " +
onFilter : "") +
                (filter != null && !filter.equals("") ? " where " +
filter : "" ) +
                (order != null && !order.equals("") ? " order by " +
order : "") +
                (withLimit ? getLimit() : "");

        System.out.println(query);
        DataSetWrapper dataSet = dbOper.directQuery(query);

        maxRows = dataSet.getRowCount();

        ArrayList headers = dt.getColumnHeaders();
        ArrayList data = dt.convertDataToTable(dataSet);

        ArrayList[] result = {headers, data};

        return result;
    }
```

Segmento de código 2.7: Método convertDataToTable() en DataTable.java

```
public ArrayList convertDataToTable(DataSetWrapper dataSet) {
        if (parsed) {
            ArrayList rows = new ArrayList();
            int totRows = dataSet.getTotalRows();

            for (int i = 0; i < totRows; i++) {
                dataSet.fetchRow(i);

                ArrayList cols = new ArrayList();
                int totCols = columnList.size();

                for (int j = 0; j < totCols; j++) {
                    Column colObj = (Column) columnList.get(j);
                    String value = "";
                    int totFlds = colObj.getFieldTotal();

                    for (int k = 0; k < totFlds; k++) {
                        String fld = colObj.getField(k);
                        Object objVal = dataSet.getColumnValue(fld);

                        if (objVal == null && value.equals(""))
```

```
                            value = "N/A";
                        else if (objVal == null && !value.equals(""))
                            value += "";
                        else {
                            if (objVal instanceof Date) {
                                Date date = (Date) objVal;
                                GregorianCalendar cal = new
GregorianCalendar();
                                cal.setTimeInMillis(date.getTime());

                                int mnt = cal.get(Calendar.MONTH) + 1;
                                int day = cal.get(Calendar.DAY_OF_MONTH);
                                int yea = cal.get(Calendar.YEAR);

                                value += (mnt < 10 ? "0" : "") + mnt +
"/" +
                                        (day < 10 ? "0" : "") + day + "/"
+ yea;
                            } else if (objVal instanceof Boolean) {
                                boolean yesNo = ((Boolean)
objVal).booleanValue();
                                if (yesNo)
                                    value += "yes";
                                else
                                    value += "no";
                            } else {
                                value += String.valueOf(objVal);
                            }
                        }

                        if ((k + 1) < totFlds)
                            value += " ";
                    }

                    cols.add(value);
                }

                rows.add(cols);
            }

            return rows;
        }

        return null;
    }
```

Segmento de código 2.8: Clase CommonAction.java

```
public class CommonAction extends Action {
    protected String table, idField, list, form;
    protected String link, tablePage, xml;
    protected ArrayList vals;
    protected TableQuery tblQry;

    public CommonAction() {
        table = "";
        idField = "";
```

```
    list = "";
    form = "";
}

public void setTable(String table) {
    this.table = table;
}

public void setXml(String xml) {
    this.xml = xml;
}

public void setIDField(String idField) {
    this.idField = idField;
}

public void setList(String list) {
    this.list = list;
}

public void setForm(String form) {
    this.form = form;
}

public void setLink(String link) {
    this.link = link;
}

public void setTablePage(String tablePage) {
    this.tablePage = tablePage;
}

public void addValidator(Validator val) {
    vals.add(val);
}

public String getName() {
    return "common";
}

public PageData perform(HttpServletRequest req) {
    HttpSession sess = req.getSession();
    PageData pgData = new PageData(sess);
    CommonModel cmMod = new CommonModel(req, context);

    tblQry = new TableQuery(req, context);
    vals = new ArrayList();

    setTable(req.getParameter("subAct"));
    setIDField(table + "_id");
    setForm(table + "Form");
    setList(table + "List");
    setXml(table + "List");
    setLink("Controller?action=common&subAct=" + table + "&update=");
    setTablePage("Controller?action=common&subAct=" + table);

    cmMod.setTable(table);
    cmMod.setIDField(idField, CommonModel.INT);

    if (req.getParameter("update") != null)
```

```
            showUpdate(pgData, req, cmMod);

        else if (req.getParameter(idField) != null) {
            addValidator(new RequiredValidator(req));
            doUpdate(pgData, req, cmMod);
        } else
            reloadList(pgData, req);

        return pgData;
    }

    protected void showUpdate(PageData pgData, HttpServletRequest req,
CommonModel cmMod) {
        pgData.setContentsPage(form + ".html");

        if (!req.getParameter("update").equals("new"))
            pgData.setSection("refill",
cmMod.refillUpdate(req.getParameter("update")));
    }

    protected void doUpdate(PageData pgData, HttpServletRequest req,
CommonModel cmMod) {
        doUpdate(pgData, req, cmMod, false);
    }

    protected void doUpdate(PageData pgData, HttpServletRequest req,
CommonModel cmMod, boolean pag) {
        ArrayList reqFld[] = new ArrayList[vals.size()];
        boolean notValid = false;

        for (int i = 0; i < vals.size(); i++) {
            Validator val = (Validator) vals.get(i);
            reqFld[i] = val.validate();

            if (reqFld[i].size() > 0) {
                if (!notValid) {
                    notValid = true;
                    pgData.setSection("refill", val.getFormRefill());
                }
                pgData.setSection(val.valReturn(), reqFld[i]);
            }
        }

        if (notValid)
            pgData.setContentsPage(form + ".html");
        else {
            int rowAff = cmMod.insertRow(req.getParameter(idField));
            reloadList(pgData, req, pag);
        }
    }

    protected void reloadList(PageData pgData, HttpServletRequest req) {
        reloadList(pgData, req, false);
    }

    protected void reloadList(PageData pgData, HttpServletRequest req,
boolean pag) {
        ArrayList[] result = tblQry.doSearch("xml/" + xml + ".xml", pag);

        pgData.setSection("tblHeaders", result[0]);
```

```
        pgData.setSection("tblData", result[1]);
        pgData.setSection("link", link);
        pgData.setSection("tablePage", tablePage);
        pgData.setSection("filter", tblQry.getFilterString());
        pgData.setSection("order", tblQry.getOrder());

        if (pag) {
            pgData.setSection("currPage", new
Integer(tblQry.getCurrentPage()));
            pgData.setSection("maxPage", new
Integer(tblQry.getMaxPages()));
            pgData.setSection("rowsFound", new
Integer(tblQry.getMaxRows()));
        }

        pgData.setContentsPage(list + ".html");
    }
}
```

2.5 Procesos de Pre-selección y Selección final

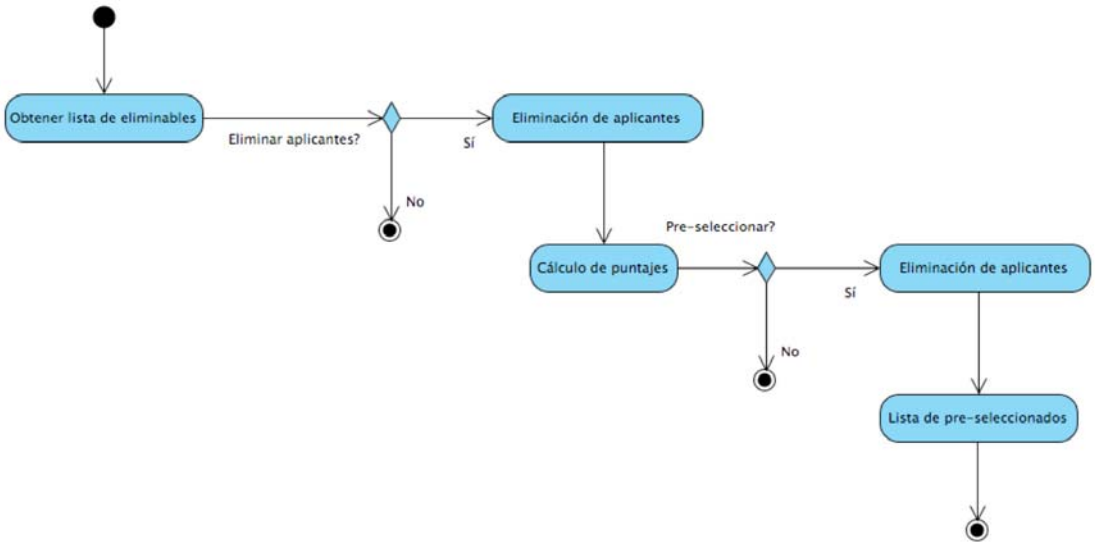


Figura 2.2 – Proceso de Pre-selección

Estos son procesos específicos de negocio de la aplicación, y constituyen el requerimiento central de la misma. El usuario debe ingresar los datos necesarios del aplicante para poder realizar el cálculo de puntaje de cada uno. Estos datos son exámenes obligatorios (TOEFL, GRE ó GMAT), GPA, calificaciones de cartas de recomendación y calificaciones de ensayos. El primer filtro es el proceso de pre-eliminación. Aquellos aplicantes que no

cumplan con los puntajes mínimos en el TOEFL, GRE/GMAT, y GPA, son eliminados en el método getEliminableApplicants() de la clase PreSelectionModel en org.fulbright.business.selection. Este método devuelve una lista con los aplicantes que no cumplen con los requisitos mínimos. Si el usuario ha decidido continuar con el proceso de pre-selección, los aplicantes de esta lista son eliminados en el método markApplicantProgram(), que toma como argumento una lista de aplicaciones en forma de cadena, con valores separados por comas, y un segundo argumento con el estado que se quiere asignar al aplicante, en este caso “eliminado”. Posteriormente la aplicación ejecuta el método preSelectAutoRank(), que calcula el puntaje final de cada aplicante. Todos los valores de puntajes son convertidos a porcentajes y se calcula mediante la siguiente fórmula:

Puntaje pre-selección = (Puntaje % del GRE o el GMAT) * 0.4 + (Puntaje % del TOEFL) * 0.1 + (Puntaje % del GPA) * 0.35 + (Puntaje promedio % de cartas de recomendación) * 0.075 + (Puntaje promedio % de ensayos) * 0.075

La aplicación asigna un posicionamiento provisional, siendo el 1er. lugar para el aplicante con el mayor puntaje, el 2do. lugar para el aplicante con el siguiente mejor puntaje, y así sucesivamente. El usuario manualmente escogerá aquellos que desea que califiquen a la fase de selección final. Una vez realizado esto, la aplicación ejecuta dos veces el método markApplicantProgram(), una para eliminar a aquellos que no fueron marcados, y la segunda vez para pre-seleccionar a aquellos que fueron marcados.

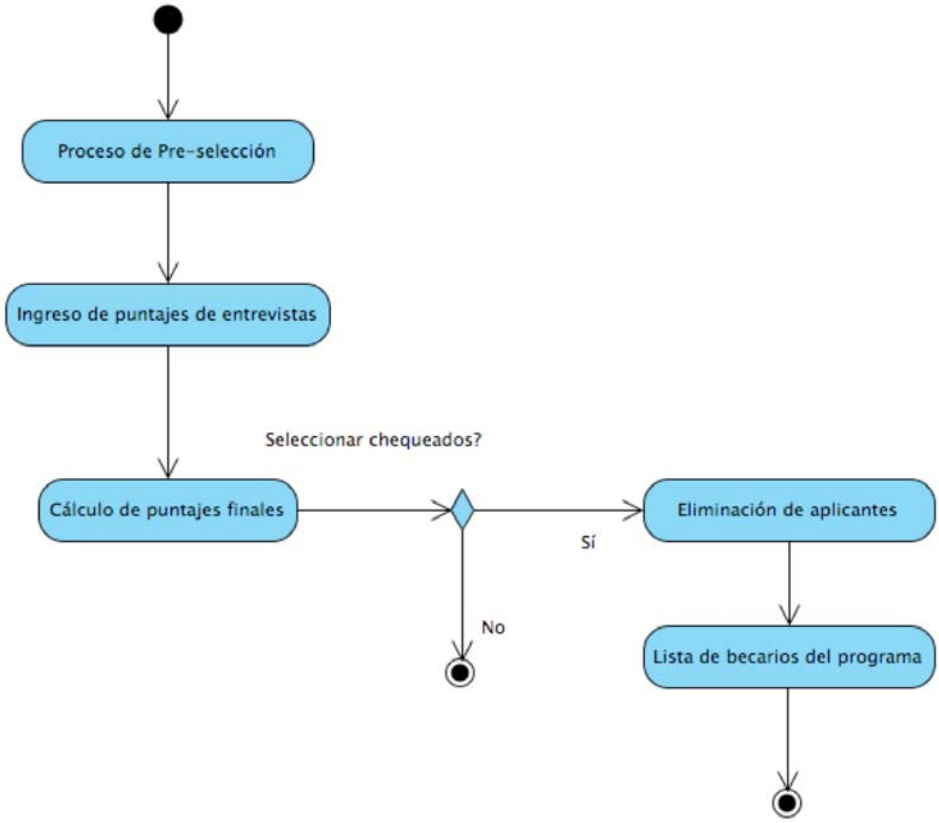


Figura 2.3 – Proceso de selección

En el proceso de selección final, el usuario debe ingresar a través de la interfaz las calificaciones porcentuales de las entrevistas realizadas con los panelistas. Cuando se ejecuta el proceso de selección final, la aplicación ejecuta el método selectAutoRank() de la clase org.fulbright.business.selection.SelectionModel. Esta clase realiza un cálculo de puntaje basado en la siguiente fórmula:

Puntaje final = (Puntaje de pre-selección) * 0.85 + (Ranking provisional convertido a %) * 0.1 + (Puntaje % de entrevista final) * 0.05

Al igual que en el proceso de pre-selección, el usuario a través de la interfaz escoge aquellos aplicantes que serán finalmente seleccionados como becarios del programa. La aplicación corre dos veces el método markApplicantProgram(), una vez para eliminar a

aquellos que no fueron seleccionados y la segunda vez para marcar como seleccionados a aquellos aplicantes que sí fueron escogidos en la interfaz de usuario.

Segmento de código 2.9: Cálculo de *ranking* en PreSelectionModel.java

```
public ArrayList preSelectAutoRank(int openID) {
        LinkedHashMap appList = this.buildApplicantStruct(openID, false,
false, true);

        Iterator keys = appList.keySet().iterator();

        while (keys.hasNext()) {
            String key = (String) keys.next();
            ApplicantData appData = (ApplicantData) appList.get(key);

            double greGmatScore = 0;
            double toeflIeltsScore = 0;
            double gpaScore = 0;
            double letterScore = 0;
            double essayScore = 0;

            if (appData.hasGRE()) {
                ExamData greV = appData.getLatestScore("GRE", "Verbal");
                ExamData greQ = appData.getLatestScore("GRE",
"Quantitative");

                double greTotal = greV.getExamScore() +
greQ.getExamScore();

                greGmatScore = getConvertedScore(greTotal, "maxGRE");
            } else if (appData.hasGMAT()) {
                ExamData gmat = appData.getLatestScore("GMAT",
"Composite");
                greGmatScore = getConvertedScore(gmat.getExamScore(),
"maxGMAT");
            }

            if (appData.hasTOEFL()) {
                double toefl = 0;
                String maxVal = "";

                if (appData.getLatestScore("TOEFL", "Paper") != null) {
                    toefl = appData.getLatestScore("TOEFL",
"Paper").getExamScore();
                    maxVal = "maxTOEFLPB";
                } else if (appData.getLatestScore("TOEFL", "Computer") !=
null) {
                    toefl = appData.getLatestScore("TOEFL",
"Computer").getExamScore();
                    maxVal = "maxTOEFLCB";
                } else if (appData.getLatestScore("TOEFL", "Internet") !=
null) {
                    toefl = appData.getLatestScore("TOEFL",
"Internet").getExamScore();
                    maxVal = "maxTOEFLIB";
                }

                toeflIeltsScore = getConvertedScore(toefl, maxVal);
```

```
            } else if (appData.hasIELTS()) {
                ExamData ieltsData = appData.getLatestScore("IELTS",
"Overall");
                toeflIeltsScore =
getConvertedScore(ieltsData.getExamScore(), "maxIELTS");
            }

            gpaScore = appData.getTranscriptGPA();

            int i = 0;
            Iterator docIt = appData.getLetters().iterator();

            while (docIt.hasNext()) {
                DocumentData docData = (DocumentData) docIt.next();
                letterScore += docData.getDocumentScore();
                i++;
            }

            letterScore /= i;

            i = 0;
            docIt = appData.getEssays().iterator();

            while (docIt.hasNext()) {
                DocumentData docData = (DocumentData) docIt.next();
                essayScore += docData.getDocumentScore();
                i++;
            }

            essayScore /= i;

            double totalScore = greGmatScore * 0.4 + toeflIeltsScore *
0.1 +
                    gpaScore * 0.35 + letterScore * 0.075 + essayScore *
0.075;

            appData.setQualificationScore(totalScore);
            updateAppScore(appData);
        }

        ArrayList sortedApps = this.sortAppList(appList);
        ArrayList toDisp = new ArrayList();

        for (int i = 0; i < sortedApps.size(); i++) {
            ApplicantData ad = (ApplicantData) sortedApps.get(i);
            ad.setRanking(i + 1);

            HashMap adHash = ad.getAsHashMap();
            toDisp.add(adHash);
        }

        return toDisp;
    }
```

Segmento de código 2.10: Cálculo de *ranking* en SelectionModel.java

```
public ArrayList selectAutoRank(int openID) {
```

```
        LinkedHashMap appList = this.buildApplicantStruct(openID, true,
false, true);

        Iterator keys = appList.keySet().iterator();

        while (keys.hasNext()) {
            String key = (String) keys.next();
            ApplicantData appData = (ApplicantData) appList.get(key);

            double interviewScore = appData.getInterviewScore();
            double preQualifyScore = appData.getQualificationScore();
            int ranking = appData.getRanking();

            double rankPerc = 100 / ranking;

            double totalScore = preQualifyScore * 0.85 + rankPerc * 0.1 +
                    interviewScore * 0.05;

            appData.setQualificationScore(totalScore);
        }

        ArrayList sortedApps = sortAppList(appList);
        ArrayList toDisp = new ArrayList();

        for (int i = 0; i < sortedApps.size(); i++) {
            ApplicantData ad = (ApplicantData) sortedApps.get(i);
            ad.setRanking(i + 1);

            HashMap adHash = ad.getAsHashMap();
            toDisp.add(adHash);
        }

        return toDisp;
    }
```

3. Descripción estructural del Producto

El Sistema de Información de la Comisión Fulbright consiste en una base de datos con toda la información, histórica y actual, de los becarios de la Comisión Fulbright, y una interfaz de usuario basada en Web que permite acceso a estos datos y a la lógica de negocio. A continuación se describirá con detalle cada uno de los componentes del sistema.

3.1 Base de datos

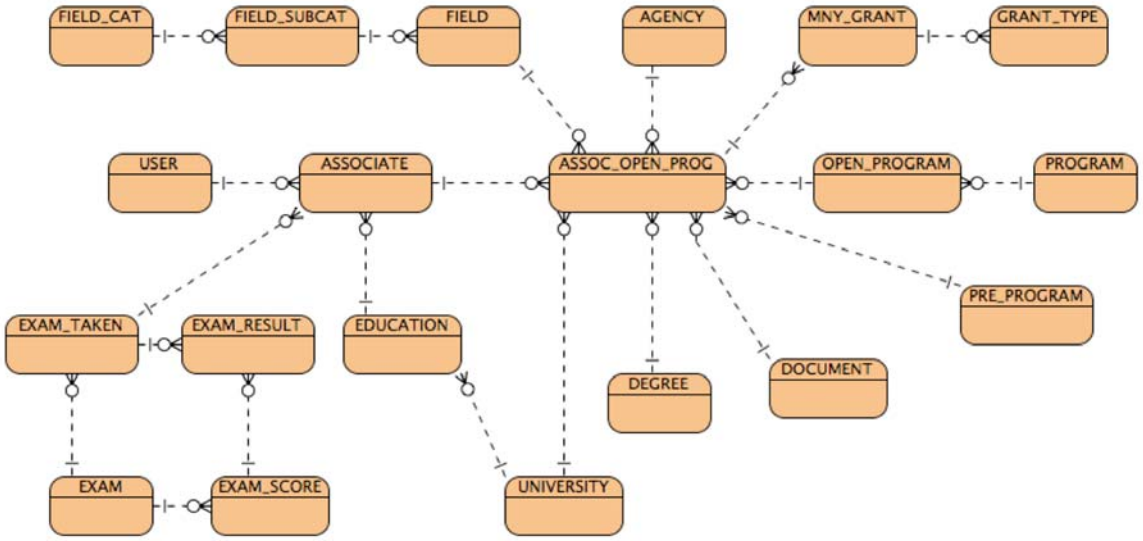


Figura 3.1 – Diagrama ER Simplificado de la base de datos

La base de datos de la aplicación ha sido desarrollada en MySQL 5.0. Consiste de las siguientes tablas:

- ASSOCIATE: Contiene toda la información de cada aplicante/becario.
- PROGRAM: Contiene la información de programas académicos.
- PRE_PROGRAM: Contiene la información de programas pre-académicos (que se realizan antes del programa académico).
- AGENCY: Contiene información de las agencias que manejan los programas académicos.
- GRANT, MNY_GRANT: Contienen información de concesiones económicas a cada programa de cada becario.
- GRANT_TYPE: Contiene información de tipos de concesiones económicas.
- EDUCATION: Contiene información del perfil académico de cada aplicante/becario.

- EXAM: Contiene información acerca de exámenes de admisión de diferentes tipos (GMAT, GRE, TOEFL, etc.)
- EXAM_TAKEN, EXAM_RESULT, EXAM_SCORE: Contienen información acerca de exámenes de admisión tomados por cada aplicante/becario.
- FIELD_CAT, FIELD_SUBCAT, FIELD: Tablas que contienen información organizada por categorías y subcategorías de campos de estudio.
- DEGREE: Contiene información de títulos/grados otorgados (B.A., B.S., Master, Ph.D., etc.)
- OPEN_PROGRAM: Contiene información de apertura de programas académicos a los cuales el público puede aplicar.
- UNIVERSITY: Contiene información de las instituciones de educación en los Estados Unidos y en Ecuador.
- USER: Contiene información de los usuarios del sistema (información de Login, roles, último acceso, etc.)
- ASSOC_OPEN_PROG: Tabla de cruce que contiene toda la información del aplicante/becario con respecto a la aplicación de un programa académico.

Las relaciones entre tablas se pueden observar en el diagrama entidad-relación. Como se puede observar en el diagrama ER, el corazón de la base de datos es la tabla ASSOC_OPEN_PROG, que contiene la información cruzada de los detalles de la aplicación de un becario/aplicante a un programa académico.

Para consultas específicas o de complejidad mayor, se utiliza la nueva capacidad de MySQL 5 para procedimientos residentes en la base de datos (stored procedures, SPs). Los SPs desarrollados para esta base de datos son:

- prQueryApplicant(VARCHAR appID IN): Permite obtener un aplicante/becario por su identificador numérico (appID).
- prQueryAppProg(VARCHAR appID IN, INTEGER assOpID IN): Permite consultar la información cruzada de la aplicación de un aplicante/becario a un programa por identificador de aplicante (appID) e identificador de aplicación a un programa abierto (assOpID).
- prQueryProgram(SMALLINT progID IN): Permite consultar un programa académico por su identificador (progID).
- prUserLogin(VARCHAR fldUsername IN, VARCHAR fldPassword IN): Permite realizar la comprobación de existencia de un usuario en el sistema para ingresar a la aplicación, según el nombre de usuario (fldUsername) y contraseña (fldPassword) proporcionados.

El resto de consultas, inserciones, actualizaciones y borrado se realizan desde la capa de datos de la aplicación. La conexión desde la aplicación hacia la base de datos se realizará a través de los servicios de Connection Pooling del servidor de aplicaciones Sun, además de un manejador JDBC (Java DataBase Connectivity) provisto por MySQL.

3.2 Descripción estructural de la Aplicación

La presente aplicación está basada en el patrón de desarrollo MVC (Modelo-Vista-Controlador). Se trata de un proyecto de aplicación Web, que correrá en el servidor de aplicaciones Sun AppServer y se lo puede acceder a través de cualquier Web browser (Mozilla Firefox 1.5 o superior, Microsoft Internet Explorer 6 o superior, Opera 8 o superior, Apple Safari) desde cualquier computador conectado a la red de la Comisión

Fulbright. Internamente el proyecto está organizado en el paquete org.fulbright, dentro del cual encontramos sub-paquetes con las capas de la aplicación.

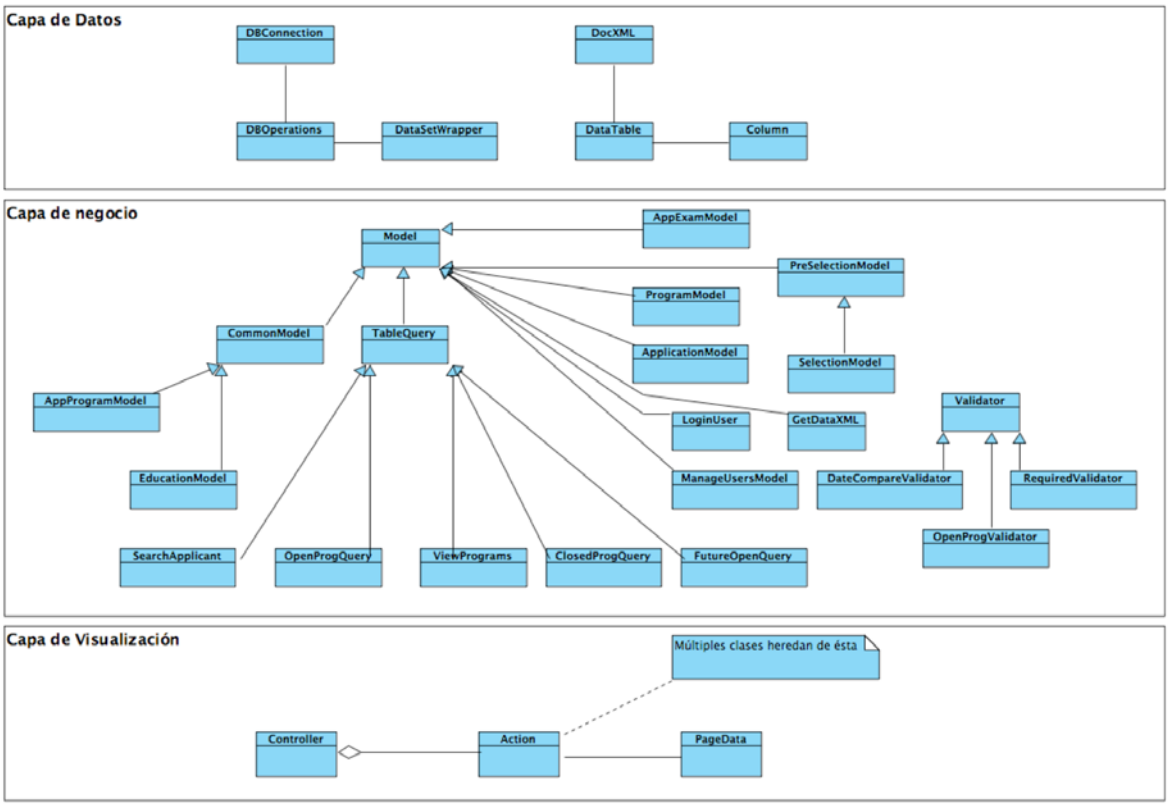


Figura 3.2 – Diagrama de Clases

3.2.1 Modelo

El modelo en esta aplicación está subdividido a su vez en dos sub-capas, Capa de Datos y Capa de Lógica de Negocio. Estas capas se encuentran en los paquetes org.fulbright.datalayer y org.fulbright.business respectivamente. A continuación se realiza una descripción detallada de las clases y su funcionalidad.

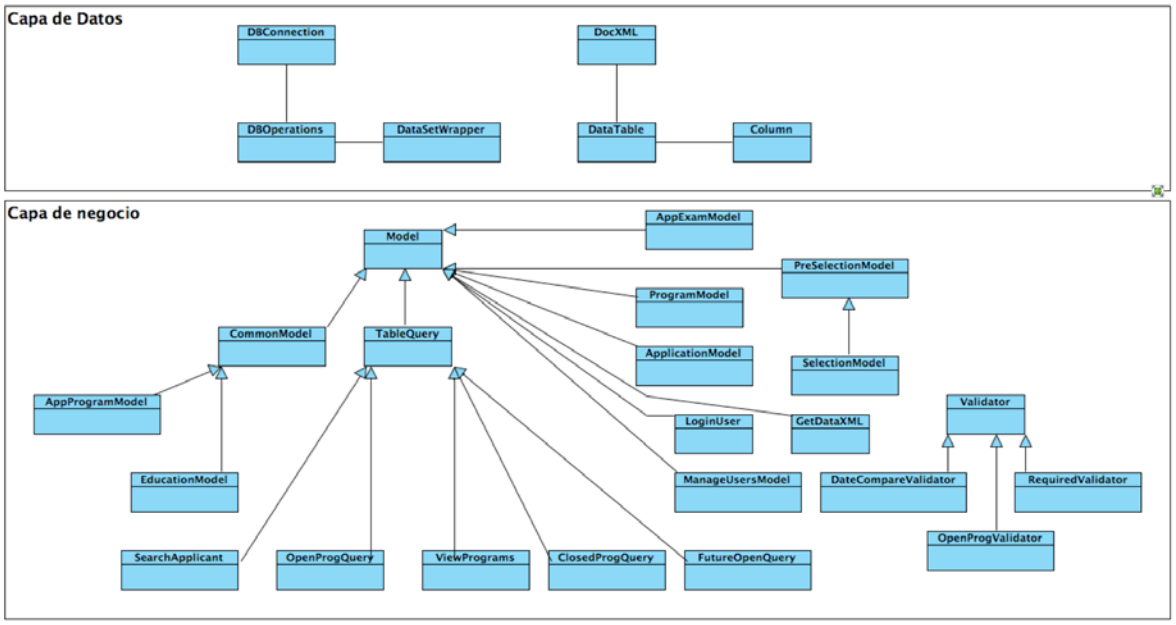


Figura 3.3 – Diagrama del Modelo

3.2.1.1 Capa de datos

Esta capa se encarga de la conexión entre la aplicación y la base de datos; construcción de sentencias SQL para consultas, inserciones, actualizaciones y borrado de datos; y clases de envoltura (wrapping) para los datos obtenidos de las consultas. Las siguientes son las clases contenidas en esta capa:

org.fulbright.datalayer.DBConnection: Esta clase se encarga de conectar la aplicación a la base de datos. Contiene dos métodos de conexión, appServConnect() y standAloneConnect(). El método appServConnect() busca el Connection Pool en el contexto de la aplicación dentro del servidor de aplicaciones. Específicamente, busca el Connection Pool en el recurso “java:comp/env/jdbc/FulbrightDB” que es como se ha configurado en el servidor de aplicaciones para esta aplicación. Este Connection Pool es asignado al objeto interno de tipo java.sql.Connection que puede ser obtenido por la aplicación a través del método getConnection(). El método standAloneConnect() realiza una conexión directa a la base de datos sin obtenerla a través del servidor de aplicaciones, y asigna la conexión al objeto interno de tipo java.sql.Connection. Este método sirve

principalmente para procesos que no se realizan a través de la aplicación Web, como la importación de datos.

org.fulbright.datalayer.DBOperations: Esta clase se encarga de realizar las operaciones en la base de datos. Posee también métodos privados que permiten construir automáticamente sentencias SQL. Los métodos callStoredProcQuery() y callStoredProcUpdate() permiten llamar a los procedimientos residentes de la base de datos (stored procedures). Los métodos simpleInsert(), simpleUpdate() y simpleDelete() permiten realizar operaciones simples de actualización en la base de datos, y devuelven el número de filas afectadas. El método directQuery() permite realizar cualquier tipo de consulta SQL SELECT, y devuelve un objeto org.fulbright.datalayer.DataSetWrapper. Para las inserciones en tablas con clave primaria auto-incremental, existe el método getLastInsertID(), que permite obtener el dato de la clave primaria generado por la base de datos después de la inserción.

org.fulbright.datalayer.DataSetWrapper: Esta clase envuelve un objeto java.sql.ResultSet que contiene generalmente el resultado de una consulta. A través de este objeto, la capa de negocios puede acceder a los datos en filas a través del método fetchRow(), y obtener datos de columnas específicas a través de getColumnValue(). Esta clase también incluye un método útil para la capa de presentación (que generalmente sólo maneja cadenas de caracteres), el método getStringContent(), que convierte todos los datos subyacentes en cadenas de caracteres fácilmente desplegables en un Web Browser.

org.fulbright.datalayer.DocXML: Esta clase permite parsear y leer un documento XML. Se utiliza principalmente para la lectura de las plantillas de tablas en pantalla, aunque técnicamente permite leer cualquier tipo de documento XML. Internamente utiliza las

librerías de DOM (Document Object Model) del paquete org.w3c.dom. Se construye un objeto DocXML con la referencia a un archivo XML a través de un objeto java.io.File. El método getRootElement() permite obtener el elemento raíz del documento XML. El método getSingleElement() obtiene un solo elemento a partir de su elemento padre y nombre de tag. El método getElements() permite obtener una colección de elementos a partir del elemento padre que los contiene y el nombre de tag de los elementos a obtener. El método getElementData() obtiene (en forma de cadena de caracteres, String) el contenido literal de un elemento específico.

Esta capa contiene un paquete interno adicional, org.fulbright.datalayer.tables, que contiene clases específicas para la lectura de plantillas XML para despliegue visual de los datos como tablas HTML en la capa de presentación. Las clases contenidas en este paquete son las siguientes:

org.fulbright.datalayer.tables.DataTable: Esta clase lee un archivo XML que contiene el formato de plantilla para la generación de tablas visuales con datos, utilizadas en la capa de presentación. Utiliza el objeto org.fulbright.datalayer.DocXML para leer la plantilla en su formato XML, y a través de este objeto parsea nuevamente para obtener los datos de columnas, títulos, hipervínculos, etc. También ejecuta automáticamente la consulta en la base de datos para obtener los datos requeridos. Internamente utiliza colecciones de la clase org.fulbright.datalayer.tables.Column para generar cada una de las columnas de la tabla visual.

org.fulbright.datalayer.tables.Column: Clase utilizada por DataTable que contiene información de la columna generada por DataTable para despliegue visual. La información

contenida tiene que ver con título de la columna, alineación, orden, hipervínculos, y los datos contenidos en la misma.

3.2.1.2 Capa de lógica de negocio

Esta capa contiene la funcionalidad en sí de la aplicación. Todos los procedimientos de negocio están contenidos en esta capa, que a su vez hace llamadas a la capa de datos para obtener los datos necesarios. También contiene clases de validación para los datos ingresados por el usuario desde la capa de presentación. Las clases que encontramos en esta capa son:

org.fulbright.business.Model: Esta es la clase base para todas las clases de lógica de negocio. Contiene métodos básicos que permiten obtener la conexión a la base de datos, construir consultas que son enviadas a la capa de datos, obtener el nombre completo del aplicante/becario, obtener el contexto de la aplicación y el pedido desde la capa de presentación.

org.fulbright.business.CommonModel: Esta clase hereda de org.fulbright.business.Model, se utiliza principalmente para operaciones comunes y simples con datos de apoyo, como grados, títulos, nombres de programas, nombres de campos de estudio, etc. También contiene el método refillUpdate(), que en el caso de actualizaciones de datos permite rellenar automáticamente el formulario de datos en la capa de presentación.

org.fulbright.business.TableQuery: Esta clase hereda de org.fulbright.business.Model. Permite realizar una consulta a la base de datos, lee una plantilla XML para generar una tabla visual a través de la clase org.fulbright.datalayer.tables.DataTable, y devuelve los

datos necesarios para que la capa de presentación genere el HTML necesario para dibujar la tabla con los datos. Contiene métodos de uso general como búsqueda, filtración de datos, y paginación, las clases que heredan de ésta hacen la implementación específica de consultas y definición de filtros.

Las clases descritas son clases base, contienen métodos de uso general. Para implementaciones específicas, el paquete org.fulbright.business contiene sub-paquetes con clases que heredan la funcionalidad de las clases base y realizan los procesos específicos requeridos de la lógica de negocio. A continuación se describen las clases contenidas en estos sub-paquetes:

org.fulbright.business.applications.AppProgramModel: Hereda de org.fulbright.business.Model, contiene métodos para consulta y actualización de datos de aplicaciones a programas por parte de los aplicantes/becarios. El método queryPrograms() obtiene los programas a los cuales ha aplicado el becario/aplicante, utiliza el stored procedure específico para esta tarea en la base de datos. El método getProgName() obtiene el nombre del programa por su identificador. El método refillUpdate() permite rellenar un formulario en la capa de presentación cuando se realiza una actualización de datos. El método isHistoric() indica si el programa al cual se ha aplicado es del tipo histórico, es decir que es de información de ex-becarios que aplicaron antes de la implementación de este sistema. El método getGrants() obtiene todas las concesiones económicas de un grupo de aplicaciones a programas. El método getGrantList() obtiene las concesiones económicas de una aplicación en específico. El método insertProgApplication() inserta en la base de datos una nueva aplicación. El método insertGrants() inserta en la base de datos todas las concesiones económicas de un programa específico.

org.fulbright.business.applications.ApplicationModel: Hereda de org.fulbright.business.Model. Contiene métodos de mantenimiento de información de aplicantes/becarios. El método insertApplication() crea un nuevo aplicante/becario o actualiza los datos de uno existente. El método getUpdateRefill() obtiene los datos de un aplicante existente para rellenar el formulario de actualización de datos en la capa de presentación. El método queryApplicant() obtiene los datos de un aplicante en específico. Internamente esta clase posee un método createAppID(), que crea un código único para un aplicante nuevo, basado en la fecha de creación más un código generado a partir de los milisegundos transcurridos en el día. Para garantizar seguridad de hilos (thread-safety) y evitar códigos duplicados, se realiza una comprobación de existencia en la base de datos antes de devolver el código generado.

org.fulbright.business.applications.EducationModel: Clase simple que hereda de org.fulbright.business.Model, permite consultar el perfil académico, es decir la lista de universidades y los títulos obtenidos en cada una por parte del aplicante/becario. El mantenimiento y actualización de esta lista se realiza a través de la clase org.fulbright.business.CommonModel, ya que se trata de un mecanismo de inserción y actualización común y simple.

org.fulbright.business.applications.SearchApplicant: Esta clase se encarga de la función de búsqueda de aplicantes/becarios en el sistema. Hereda de org.fulbright.business.TableQuery. El método performSearch(), con el criterio de búsqueda suministrado, genera el filtro y llama a la función doSearch() de la clase base. El método

performSortPage() permite ordenar los datos de la búsqueda dependiendo de la columna escogida para realizar el reordenamiento.

org.fulbright.business.programs.ProgramModel: Esta clase se encarga del mantenimiento de Programas de Estudio. Hereda de org.fulbright.business.Model. El método getAgencyList() permite obtener la lista de agencias que auspician un programa. El método refillUpdate() permite rellenar el formulario para actualización de datos de un programa.

org.fulbright.business.programs.ViewPrograms: Clase sencilla de búsqueda de programas de estudio. Hereda de org.fulbright.business.TableQuery. Permite desplegar en una tabla todos los programas de estudio ingresados en el sistema.

org.fulbright.business.programs.OpenProgQuery: Clase que permite consultar los programas de estudio abiertos al público para aplicaciones. Esta clase hereda de org.fulbright.business.TableQuery. Despliega en una lista ordenable todos los programas abiertos en el período de tiempo específico de cada uno.

org.fulbright.business.programs.ClosedProgQuery: Clase que permite consultar los programas que han sido abiertos, pero que ya sobrepasaron su fecha límite de cierre. Esta clase hereda de org.fulbright.business.TableQuery. Despliega en una lista ordenable todos los programas que ya han sobrepasado la fecha de cierre. Para determinar si un programa se ha cerrado o no, la aplicación se basa en la fecha del sistema.

org.fulbright.business.programs.FutureProgQuery: Clase que permite consultar y desplegar aquellos programas que tengan una fecha de apertura al público posterior a la fecha actual del sistema. Hereda de la clase org.fulbright.business.TableQuery.

Las siguientes clases implementan la lógica de ingreso al sistema (Login) y mantenimiento de usuarios del mismo. Se encuentran en el paquete org.fulbright.business.users.

org.fulbright.business.users.LoginUser: Esta clase encapsula el procedimiento de autenticación y permisos de ingreso a la aplicación. Extiende de la clase org.fulbright.business.Model, ya que debe leer los datos de usuarios desde la tabla destinada a este propósito en la base de datos. El método performLogin() realiza el proceso de autenticación y permisos de acceso a la aplicación, y devuelve un objeto de tipo org.fulbright.business.users.UserLogged.

org.fulbright.business.users.UserLogged: Esta es una clase que contiene los datos de los usuarios que se encuentren autenticados y utilizando el sistema. Los objetos de esta clase residen en la sesión del usuario mientras se encuentre utilizando el sistema, es decir, hasta que realice un log-out o cierre la ventana principal del Web Browser.

Para la validación de los datos ingresados a través de los formularios de la aplicación, existe un grupo de clases especializadas para este propósito. Todas se encuentran en el paquete org.fulbright.business.validators.

org.fulbright.business.validators.Validator: Esta es una clase abstracta, que sirve como base para todas las clases de validación. Contiene el método getFormRefill(), que rellena

automáticamente los datos del formulario si la validación ha fallado. Tiene tres métodos abstractos que deben ser implementados en las clases concretas que heredan de ésta, validate(), valType() y valReturn().

org.fulbright.business.validators.RequiredValidator: Esta clase implementa un validador de campo requerido. Hereda de org.fulbright.business.validators.Validator, implementa los métodos abstractos de esta clase para realizar la validación del formulario. En su proceso chequea que los campos especificados contengan algún valor para ser válidos, de lo contrario el validador falla y devuelve la lista de los campos que requieren de algún valor.

org.fulbright.business.validators.DateCompareValidator: Esta clase permite comparar fechas en pares, y por cada par compara si la primera es anterior a la segunda, caso contrario el validador falla. Hereda de org.fulbright.business.validators.Validator, e implementa todos sus métodos abstractos.

org.fulbright.business.validators.OpenProgValidator: Esta clase permite determinar si un programa está tratando de ser abierto en un lapso de tiempo en el cual ya ha sido abierto. Si el intervalo de tiempo ingresado se sobrepone a cualquier intervalo de tiempo ingresado para el mismo programa, el validador falla. Hereda de org.fulbright.business.validators.Validator, e implementa todos sus métodos abstractos.

Finalmente, el último paquete de la capa de lógica de negocios es el org.fulbright.business.xml, que contiene clases para exportar datos a formato XML. Es útil para llamadas de cliente tipo AJAX (Asynchronous JavaScript and XML) que generalmente se manejan con respuestas XML.

org.fulbright.business.xml.GetDataXML: Permite obtener los datos de una tabla según un criterio de filtrado, y los convierte a un archivo XML con un formato específico. El método getXML() permite convertir los datos de cualquier tabla en la base de datos. Esta clase hereda de org.fulbright.business.Model.

3.2.2 Vista

La Vista (o capa de presentación) contiene todos los elementos de la interfaz con el usuario. Como se trata de una aplicación Web, la Vista maneja el despliegue de la información y hace los pedidos e ingreso de datos desde un Web Browser. Los pedidos los hace el Web Browser a través del protocolo HTTP (HyperText Transfer Protocol, protocolo de transferencia de hipertexto), y la aplicación responde con contenido HTML que a su vez es interpretado por el Web Browser y desplegado de manera visual al usuario. Todo el contenido visual de la aplicación ha sido organizado en diferentes plantillas HTML, que permiten la reutilización de secciones y combinarlas para crear el contenido completo de las páginas Web que son la interfaz con el usuario de la aplicación.
Todo el contenido HTML y de apoyo visual está contenido dentro del directorio “web”, que a su vez contiene los directorios “images”, “js”, “styles”, “templates” y “xml”. Dentro del directorio “images” están todas las imágenes, gráficos y diseños que mejoran el aspecto visual de la interfaz, además de íconos y símbolos informativos. Todas las imágenes están codificadas según los estándares JPEG y GIF, que son los recomendados para Web en general.

En el directorio “js” se encuentran archivos que contienen código JavaScript. A continuación se describen cada uno de ellos:

common.js: Posee funciones de uso general para cualquier página que los requiera. La función openCalendar() permite abrir el mini-calendario selector de fechas utilizado en algunos formularios para el ingreso sin errores de fechas. La función selectTab() permite alterar el aspecto visual de las pestañas que aparecen en algunas páginas que poseen varias secciones, en otras palabras resalta aquella pestaña que ha sido seleccionada. La función isNumericKeyStroke() puede ser llamada desde cualquier caja de texto HTML (tipo <input type=”text”>) para permitir el ingreso de solamente caracteres numéricos, enteros o con decimales. Y finalmente la función initHttpRequest(), una función que puede ser utilizada en cualquiera de los Web Browsers más populares del medio (Microsoft Internet Explorer, Mozilla Firefox, Apple Safari, Opera, etc.) para inicializar un objeto XMLHttpRequest, que permite hacer llamados al servidor sin refrescar completamente la página (AJAX).

topMenu.js: Posee funciones que permiten el funcionamiento de la barra de menú principal de la aplicación.

validations.js: Posee funciones para marcar aquellos campos que han fallado las validaciones (markInvalid()) y permite re-seleccionar los valores de listas que habían sido seleccionados antes de que falle la validación, con lo que el usuario no se ve forzado a re-seleccionar aquellos valores manualmente (refillSelect).

calendar.js: Contiene todas las funciones utilizadas dentro del control de mini-calendario para llenar campos de fecha sin errores.

programs.js: Contiene funciones de uso exclusivo en la página de aplicación a programas, que permiten deshabilitar o borrar ciertos campos cuando ciertas opciones específicas son escogidas por parte del usuario.

grants.js: La página de ingreso de concesiones económicas (Grants) contiene una lógica compleja para facilitar el ingreso de las mismas mediante una tabla flexible donde el

usuario puede añadir filas e ingresar los valores. Las funciones dentro de este archivo se encargan del manejo de la lógica de aquella página en específico.

En el directorio “styles” se encuentra el archivo maestro de estilos de página CSS (Cascading Style Sheets), styles.css. Permite que todo el contenido HTML tenga un estilo visual consistente en todas las páginas, los elementos simplemente deben utilizar las clases contenidas en este archivo maestro.

El directorio “xml” contiene todas las plantillas para generación automática de tablas de consulta de datos. Estos archivos tienen un formato que es utilizado por el servidor para generar las tablas.

Y finalmente el directorio “templates” contiene todas las plantillas para generación de contenido HTML. La aplicación utiliza el motor de plantillas FreeMarker (software libre, no requiere de compra), que incluye un juego de tags especiales para la manipulación de las mismas dentro de cada archivo HTML. El archivo default.html es la plantilla maestra de contenido. Organiza la cabecera, contenido y pie de página de todas las páginas generadas por la aplicación. El archivo calendar.html permite generar el control de mini-calendario que puede ser incluido en cualquier página-formulario que requiera ingreso de fechas. El archivo fieldSelect.html es una página especial que permite seleccionar campos de estudio mediante listas que se actualizan vía AJAX, y se utiliza dentro de las páginas de aplicación a programas. Y finalmente el archivo login.html, que contiene la página de ingreso a la aplicación donde se pide nombre de usuario y contraseña. Existe un subdirectorio “contents”, que contiene todas las plantillas de todas las páginas de la aplicación, y que junto con la plantilla maestra default.html generan la interfaz de usuario de la aplicación para el usuario.

Todos los pedidos a la aplicación, el Web Browser los maneja a través del protocolo HTTP. Para el pedido de páginas y de consultas de datos generalmente utiliza el método GET, que arma una cadena de consulta (query string) que es incluida en la dirección de la página para pedir información específica, y disparar los eventos de la aplicación. Para el ingreso de datos a través de los formularios, el Web Browser los hará mediante el método POST, que envía los datos como parte del pedido HTTP, fuera de la vista del usuario. Esta actitud permite una operación segura, ya que GET no debe usarse para modificar el estado del servidor, para esto existe el método POST. Las operaciones tipo AJAX utilizan la extensión de JavaScript para pedidos que no refrescan toda la página, sino partes específicas de la misma. En este tipo de operaciones, el servidor generalmente responde con documentos XML, que se procesan a través de las funciones DOM (Document Object Model) de JavaScript. Dada la naturaleza dinámica de JavaScript, se puede modificar contenido y controles sin necesidad de regenerar la página. Para que esta funcionalidad esté disponible, el Web Browser del cliente debe tener activado JavaScript.

3.2.3 Controlador

El Controlador en esta aplicación se encarga del despacho de acciones (eventos). A su vez canaliza los pedidos a los procedimientos de negocio apropiados según la acción requerida. Existe una clase de despacho, a la cual todas las páginas en la Vista se refieren, ya sea a través de pedidos GET directos o en el ingreso de formularios mediante método POST. Para poder despachar las acciones, todos los pedidos deben incluir el parámetro “action”, en el cual se especifica el código de acción a llevar a cabo. Todas las clases del controlador se encuentran dentro de los subpaquetes del paquete org.fulbright.webapp. A continuación se describe la funcionalidad de cada una.

org.fulbright.webapp.servlets.Controller: Esta es la clase de despacho de acciones principal, es un servlet. Internamente posee una colección de objetos tipo org.fulbright.webapp.actions.Action, cada objeto de esta colección se encarga de canalizar el pedido para realizar la consulta o modificación al Modelo subyacente. Los métodos doPost() y doGet() ambos llaman al método processRequest(), que contiene una rutina genérica de llamado de acciones. En esta rutina, se obtiene el código de acción a través del parámetro requerido “action” que viene a través del pedido HTTP. Mediante ese código se obtiene la acción de la colección. Ya que todas las acciones son clases concretas de la clase abstracta org.fulbright.webapp.actions.Action, se llama al método perform(), que es un método abstracto implementado en cada clase concreta. Por lo tanto la llamada a perform() se realiza de manera polimórfica, y el pedido es correctamente canalizado a la acción deseada. Todas las acciones devuelven un objeto de tipo org.fulbright.webapp.actions.PageData, que permite armar una página de contenido HTML a través de la combinación de múltiples plantillas HTML. A continuación se llama al método loadHTMLForm(), que ejecuta los métodos en las clases de la librería del motor de plantillas FreeMarker. La respuesta es enviada de vuelta al cliente como un documento HTML completo.

org.fulbright.webapp.servlets.XMLData: Este es un servlet especial que permite obtener los datos de la base de datos formateados como XML. Se utiliza principalmente para llamados de tipo AJAX. Al igual que el servlet Controller, posee los métodos doPost() y doGet(), que a su vez llaman al método processRequest(). En este caso este método llama directamente a la clase org.fulbright.business.xml.GetDataXML de la capa de negocios y devuelve la respuesta como un documento XML.

Las clases de apoyo del controlador se encuentran en el paquete org.fulbright.webapp.actions, todas representan las diferentes acciones que el cliente Web puede pedir al servidor y realizar las respectivas consultas y modificaciones al Modelo.

org.fulbright.webapp.actions.Action: Clase base abstracta para todas las acciones. Posee el método implementado setContext(), que permite fijar el contexto de la aplicación necesario para obtener los recursos del servidor de aplicaciones por parte del Modelo. El método abstracto getName() permite obtener el nombre o código de acción, las clases concretas lo implementan con su respectivo nombre. Y finalmente el método abstracto perform(), que en cada acción concreta contiene su implementación específica.

org.fulbright.webapp.actions.PageData: Esta clase permite organizar la información necesaria para armar el contenido a desplegar al usuario. Los métodos y estructuras de datos de los objetos de esta clase permiten al motor de plantillas FreeMarker reconocer los datos que se especifican en las plantillas mediante los tags especiales y generarlos como código HTML. FreeMarker lee los datos de estructuras tipo HashMap o ArrayList, el método getTplData() permite obtener el HashMap raíz donde todos los datos a desplegar deben ser incluidos. Ya que la capa de presentación maneja básicamente cadenas de caracteres, generalmente los datos se formatean como colecciones (HashMaps, ArrayLists) de objetos tipo String. Los métodos setPageLoad() y getPageLoad() permiten fijar y obtener, respectivamente, la página de plantilla base (en el caso de esta aplicación, la predeterminada es default.html). El método setSection() permite asignar directamente un valor a la colección raíz de datos a desplegar. Y finalmente el método setContentsPage() permite asignar la plantilla de contenido principal para la plantilla predeterminada.

El resto de clases del paquete org.fulbright.webapp.actions son implementaciones específicas de org.fulbright.webapp.actions.Action, cada una representa una acción de la aplicación que se puede llamar desde la capa de presentación.

Finalmente la aplicación Web cuenta con clases de apoyo que se encargan de funciones específicas del ambiente del servidor de aplicaciones, estas se encuentran en los paquetes org.fulbright.webapp.listeners y org.fulbright.webapp.filters

org.fulbright.webapp.listeners.ContextListener: Esta clase implementa la interfaz javax.servlet.ServletContextListener. Maneja el evento de inicialización del contexto de la aplicación, es decir cuando la aplicación es ejecutada por primera vez en el servidor de aplicaciones (en el proceso de deploy), y también maneja el evento de cierre de la aplicación (en el caso de undeploy o cierre del servidor de aplicaciones). En el evento contextInitialized(), se crea un objeto org.fulbright.datalayer.DBConnection, en el cual se ejecuta el método appServConnect(), se inicializa el motor de plantillas FreeMarker, y se fijan los objetos de conexión y de FreeMarker como atributos del contexto de la aplicación Web en el servidor de aplicaciones. En el evento contextDestroyed(), se cierra la conexión a la base de datos para liberar sus recursos y se remueve su definición como atributo del contexto de la aplicación.

org.fulbright.webapp.filters.Authentication: Esta clase especial es un filtro de pedidos y respuestas del servidor, que se ejecuta siempre que se pide una acción al servlet org.fulbright.webapp.servlets.Controller. Se encarga de determinar si el usuario se ha autenticado para usar la aplicación. Para esto, en el método doFilter() determina si en la sesión del usuario existe el atributo con el objeto org.fulbright.business.users.UserLogged.

Si no existe, entonces elimina el pedido y redirecciona el Web Browser del cliente a la página de autenticación (login.html), donde se puede ingresar el nombre de usuario y contraseña. Si el proceso de autenticación es correcto, el objeto de usuario se carga en la sesión y cada vez que el pedido pasa por el filtro, se lo permite alcanzar hacia el servlet controlador de la aplicación.

ANEXO 2

MANUAL DE USUARIO

TABLA DE CONTENIDO

Público objetivo _____ 94
Objetivo del manual _____ 94
1. Acceso a la aplicación _____ 94
2. Página de autenticación (*Login*) _____ 95
3. Página principal _____ 95
4. Opciones del Menú Principal _____ 96
4.1 Submenú *Applications* _____ 97
4.1.1 *New Application* (Nueva aplicación) _____ 97
4.1.1.1 Información del aplicante/becario _____ 99
4.1.1.1.1 Subsección *General Information* _____ 99
4.1.1.1.2 Subsección *Academic Background* _____ 100
4.1.1.1.2.1 *Add/update university/institution* _____ 101
4.1.1.1.3 Subsección *Exams* _____ 101
4.1.1.1.3.1 *Add exam* (Añadir examen) _____ 102
4.1.1.1.3.2 *Add exam score* (Añadir puntaje) _____ 102
4.1.1.1.4 Subsección *Programs and Grants* _____ 103
4.1.1.1.4.1 *Add/update program* _____ 105
4.1.1.1.4.2 *Edit Grants* _____ 106
4.1.2 *Search Applicants* (Buscar Aplicantes) _____ 107
4.1.3 *Pre-selection* (Pre-selección) _____ 109
4.1.3.1 Página de selección de programa _____ 109
4.1.3.2 Lista de aplicantes al programa _____ 109
4.1.3.2.1 Detalles de aplicación a programa _____ 110
4.1.3.2.1.1 Ingreso de documentos _____ 111
4.1.3.3 Eliminación previa _____ 111
4.1.3.4 Cálculo de *rankings* para Pre-selección _____ 112
4.1.4 *Selection* (Selección) _____ 113
4.1.4.1 Página de selección de programa _____ 113
4.1.4.2 Lista de aplicantes al programa _____ 114
4.1.4.3 Cálculo de *rankings* finales _____ 114
4.2 Submenú *Programs* _____ 115
4.2.1 *Open Programs* (Programas abiertos) _____ 116
4.2.1.1 *Open a program* (Abrir un programa) _____ 116
4.2.1.2 *Future programs* (Programas futuros) _____ 117
4.2.1.3 *Closed programs* (Programas cerrados) _____ 117
4.2.2 *All Programs* (Todos los programas) _____ 118
4.2.2.1 *Add/update program* (Añadir/modificar programa) _____ 119
4.2.3 *Agencies* (Agencias) _____ 119
4.2.3.1 *Add/update agency* (Añadir/modificar agencia) _____ 120
4.2.4 *Pre-programs* (Programas Pre-académicos) _____ 120
4.2.4.1 *Add/update pre-program* (Añadir/modificar pre-programa) _____ 121
4.3 Submenú *Reports* _____ 121
4.3.1 *Report Generator* (Generador de Reportes) _____ 122
4.4 Submenú *Administration* _____ 122
4.4.1 *Manage Users* (Administrador de Usuarios) _____ 123

4.4.2 *Exam types* (Tipos de exámenes) _____ 123
4.4.2.1 *Add/update exam type* (Añadir/modificar tipo de examen) _____ 124
4.4.2.1.1 *Add/update score type* (Añadir/modificar tipo de puntaje) _____ 124
4.4.3 *Degrees* (Títulos/grados) _____ 125
4.4.3.1 *Add/update degree* (Añadir/modificar título/grado) _____ 126
4.4.4 *Grant types* (Tipos de Concesiones) _____ 126
4.4.5 *Fields of study* (Campos de estudio) _____ 127
4.4.5.1 *Add/update field category* (Añadir/modificar categoría) _____ 127
4.4.5.1.1 *Add/update field subcategory* (Añadir/modificar subcategoría) _____ 128
4.4.5.1.1.1 *Add/update field of study* (Añadir/modificar campo de estudio) _____ 128
5. Salida del sistema (*Logoff*) _____ 129
6. Glosario _____ 129

Público objetivo

Este es un manual de referencia para los usuarios y los administradores del sistema.

Objetivo del manual

El objetivo de este manual es proveer una referencia a las funciones de la aplicación y los procedimientos a seguir para ejecutarlas.

1. Acceso a la aplicación

La aplicación solamente se puede acceder desde la red local, utilizando cualquier computador equipado con un Navegador de la Web (*Web Browser*). La aplicación ha sido probada con los siguientes *Web Browsers*, si algún usuario posee uno que no está en esta lista, se recomienda instalar cualquiera de los mencionados aquí (Mozilla y Opera poseen *Web Browsers* que pueden instalarse en prácticamente cualquier plataforma, ya sea Windows, Mac OS X, o Linux):

- Mozilla Firefox 1.5 o superior, y similares (Camino, Mozilla Suite).
- Apple Safari 4.3 o superior, y similares (KDE Konqueror).
- Microsoft Internet Explorer 6 o superior.
- Opera Browser 8 o superior.

El administrador del sistema debe proveer los datos de dirección (URL) y puerto de acceso. Estos datos se escriben en la barra de direcciones, de esta manera:

http://[dirección_del_servidor]:[puerto]/fulbright

El siguiente ejemplo muestra una dirección de acceso a la aplicación:

Figura 1.1 – Ejemplo de dirección de acceso

Una vez que se haya accedido a la página de autenticación (Login, que se verá en detalle en la siguiente sección), se recomienda guardar esta dirección en un marcalibros o favorito del *Web Browser* del usuario.

2. Página de autenticación (*Login*)

Una vez ingresada la dirección de acceso en su *Web Browser*, la aplicación mostrará la página de acceso (autenticación) del sistema, donde el usuario debe ingresar sus respectivas credenciales.

Figura 2.1 – Página de autenticación

El administrador proveerá las credenciales iniciales a cada usuario del sistema, después cada usuario tiene la capacidad de cambiar su contraseña al interior del sistema (Ver Administración de Usuarios). Si las credenciales provistas al sistema son incorrectas, la página mostrará un mensaje de que el usuario no ha podido ser verificado.

3. Página principal

La página principal muestra un mensaje de bienvenida al usuario, la versión del sistema, y una lista de los *Web Browsers* recomendados para el uso de la misma.

Figura 3.1 – Página principal

Como se puede observar en la figura A2.3, en la parte superior de la página se encuentra el Menú Principal de la aplicación. Todas las funciones de la aplicación se acceden desde este Menú. Cada opción en el Menú Principal tiene un submenú, donde se accede a las funciones específicas. Es posible que cada usuario tenga diferentes elementos en sus menús, ya que dependen de su nivel de acceso. Los usuarios con privilegios de administrador tendrán acceso a todas las opciones del menú.

4. Opciones del Menú Principal

El Menú Principal tiene 5 submenús: *Applications* (Aplicaciones), *Programs* (Programas), *Reports* (Reportes), *Administration* (Administración) y *Help* (Ayuda). A continuación se describe las funciones contenidas en cada submenú.

4.1 Submenú *Applications* (Aplicaciones)

Este submenú contiene las funciones para manejo de aplicantes/becarios, búsqueda, y procesos de pre-selección y selección de becarios.

4.1.1 *New Application* (Nueva aplicación)

Esta opción despliega el formulario de ingreso de datos personales para un nuevo aplicante/becario del sistema.

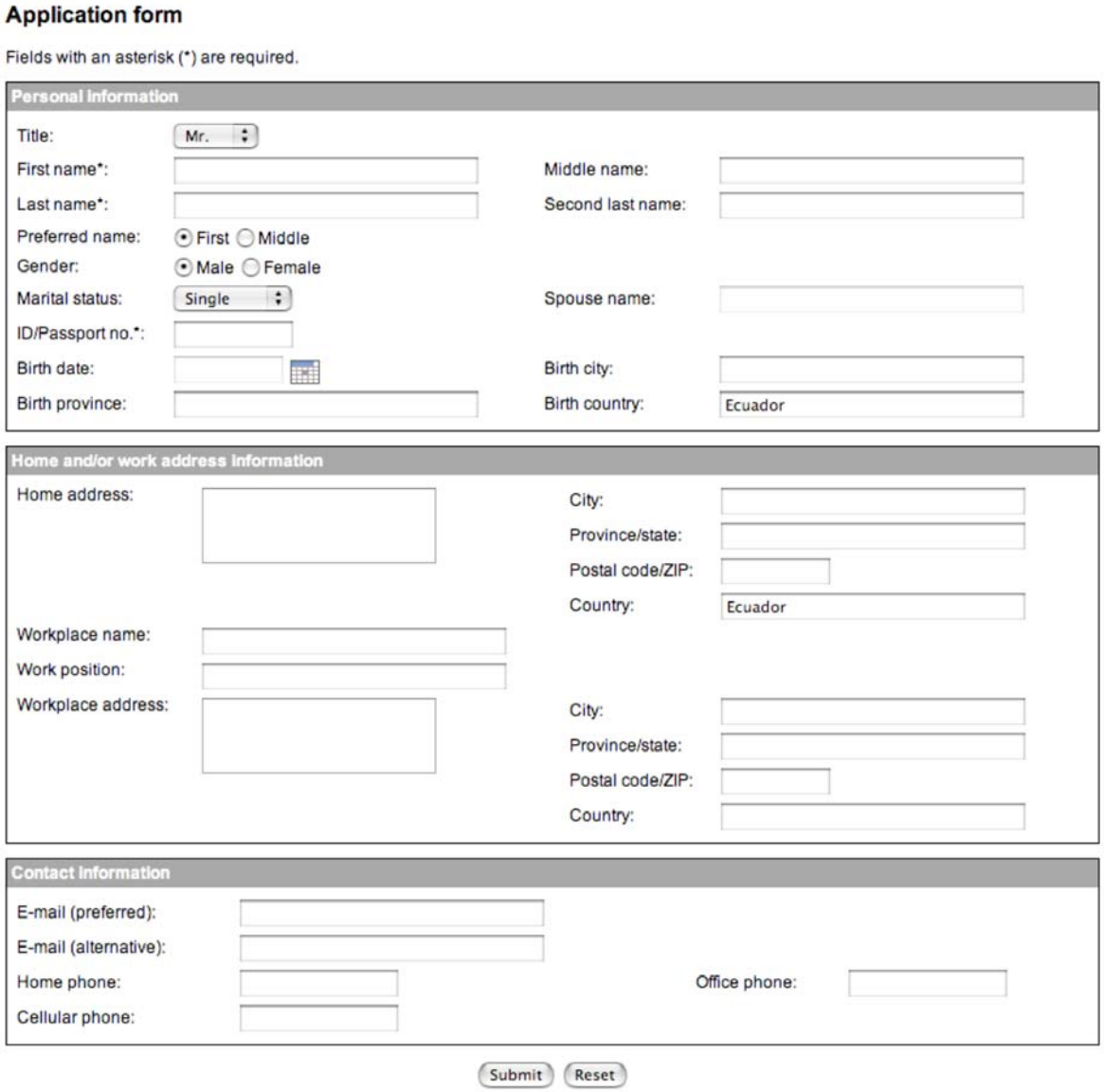


Figura 4.1 – Formulario de ingreso de datos de Nuevo Aplicante

Esta página está dividida en 3 secciones: Datos Personales (*Personal Information*), Datos de Dirección Domiciliaria y de Trabajo (*Home/work address information*) y Datos de Contacto (*Contact Information*).

En la sección de Datos Personales, se puede ingresar todos los datos del aplicante como título, primer y segundo nombres, primer y segundo apellidos, nombre preferido (primero o segundo), género, número de documento de identificación o pasaporte, estado civil, nombre del cónyuge (en el caso de seleccionar estado civil Casado (*Married*)), fecha de nacimiento y lugar de nacimiento (ciudad, estado / provincia, país). De estos campos, primer apellido, primer nombre e identificación son obligatorios. Si no se llenan estos datos obligatorios, el formulario será rechazado por el sistema.

En la sección de Datos de Dirección Domiciliaria y/o Trabajo, permite el ingreso de la dirección, ciudad, provincia/estado, código postal y país. Y en el caso de del Trabajo, también pide el nombre del lugar de trabajo y cargo/posición.

En la sección Datos de Contacto, se pide dos direcciones de correo electrónico, número de teléfono del domicilio, número de teléfono del lugar de trabajo, y número de teléfono celular. De estos datos, la dirección de correo electrónico principal es requerida, si no es ingresada, el formulario será rechazado por el sistema.

En el caso de que el formulario haya sido rechazado por el sistema, volverá a pedir los datos del aplicante/becario y resaltará en amarillo los campos obligatorios que deben ser llenados correctamente.

Si el formulario es aceptado, asignará al aplicante un código que consiste en el año, mes y día del registro, más un número basado en el número de milisegundos transcurridos desde la medianoche hasta la hora que se realizó el registro, en el formato “aaaa-mmdd-milisegn”. Por ejemplo, un aplicante que se ingrese el día 14 de Mayo de 2007 exactamente al mediodía, obtendrá un código “2007-0514-43200000”.

4.1.1.1 Información del Aplicante/Becario

Una vez ingresada una aplicación, o realizado una búsqueda de un aplicante, se mostrará la página de Información del Aplicante/Becario, que contiene en su parte superior 4 subsecciones: *General* (Información General), *Education* (Perfil Académico/Educación), *Exams* (Exámenes) y *Programs* (Programas). Cada subsección se accede a través de las pestañas en la parte superior de la pantalla, debajo del menú principal.

Figura 4.2 – Pestañas de selección de subsección

4.1.1.1.1 Subsección *General Information* (Información General)

Esta página muestra la información personal y de contacto del aplicante.

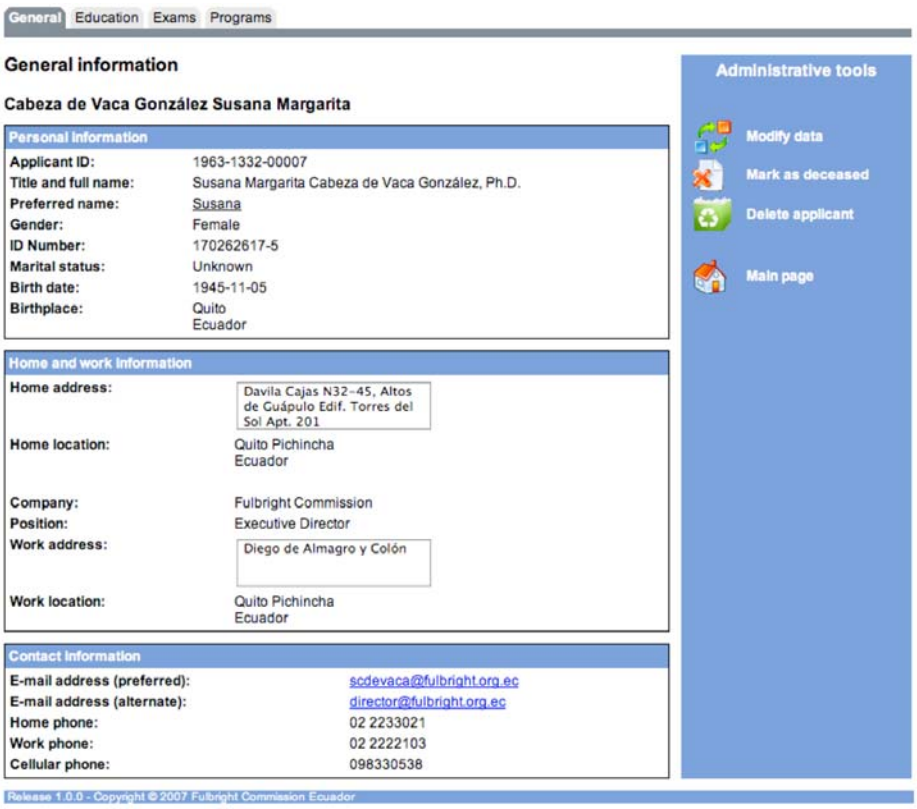


Figura 4.3 – Información General de Aplicante

De la misma manera que en el formulario de aplicación (ver 4.1.1), la información mostrada en esta página está organizada en Datos Personales (*Personal Information*), Datos de Dirección Domiciliaria y/o Trabajo (*Home/work address information*), y Datos de Contacto (*Contact Information*). En la parte derecha de la pantalla se encuentran las Herramientas de Administración (*Administrative Tools*). Estas herramientas permiten editar los datos del aplicante (*Modify Data*), lo que lleva de nuevo al formulario de aplicación (ver 4.1.1), marcar al aplicante como difunto o no (*Mark as Deceased / Mark as not Deceased*), y borrar el aplicante (*Delete applicant*). En el caso de intentar borrar el aplicante (reservado para los administradores del sistema), una caja de texto advertirá que la acción borrará todos los registros asociados con el aplicante, y permitirá continuar o detener la operación de borrado.

4.1.1.1.2 Subsección *Academic Background* (Perfil académico)

Esta pantalla muestra los institutos/universidades y títulos que el aplicante ha obtenido durante su vida académica.

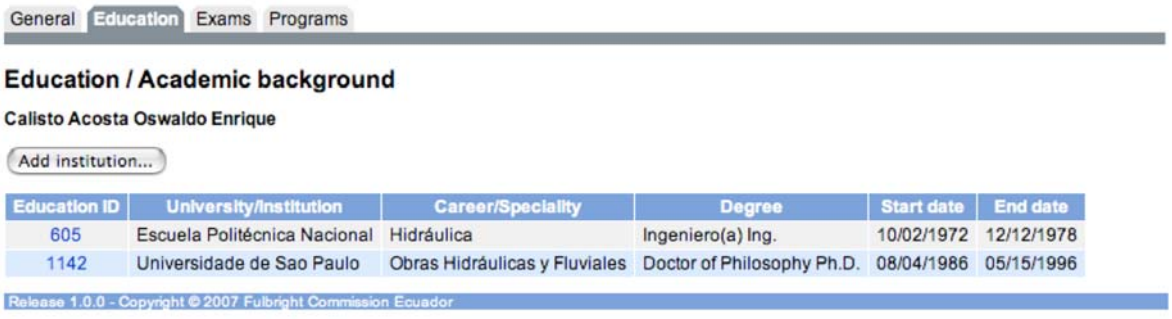


Figura 4.4 – Subsección de Perfil Académico

El botón *Add university/institution* (Añadir universidad/institución) permite añadir una entrada más a la lista. Haciendo clic sobre el identificador de la fila permite modificar la información de una entrada de la lista.

4.1.1.1.2.1 *Add/update university/institution* (Añadir/actualizar universidad/institución)

Esta página permite añadir (o modificar) una entrada al perfil académico del aplicante.

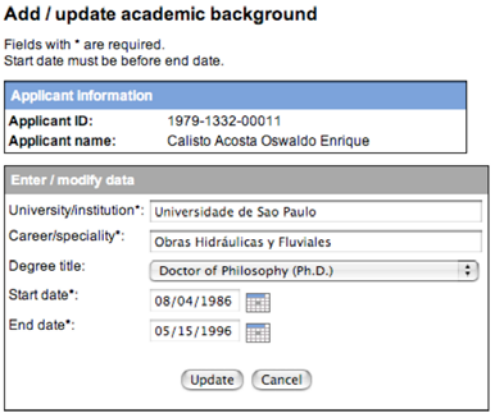


Figura 4.5 – Añadir/modificar Perfil académico

Se debe ingresar el nombre de la universidad/institución, nombre de la carrera, título obtenido, fecha de ingreso y fecha de egreso de la universidad/institución. El botón *Cancel* retorna a la página de perfil académico (ver 4.1.1.1.2) sin realizar ninguna modificación.

4.1.1.1.3 Subsección *Exams* (Exámenes)

Esta subsección permite ingresar los exámenes tomados por el aplicante, como el TOEFL, GRE, GMAT, IELTS, etc.

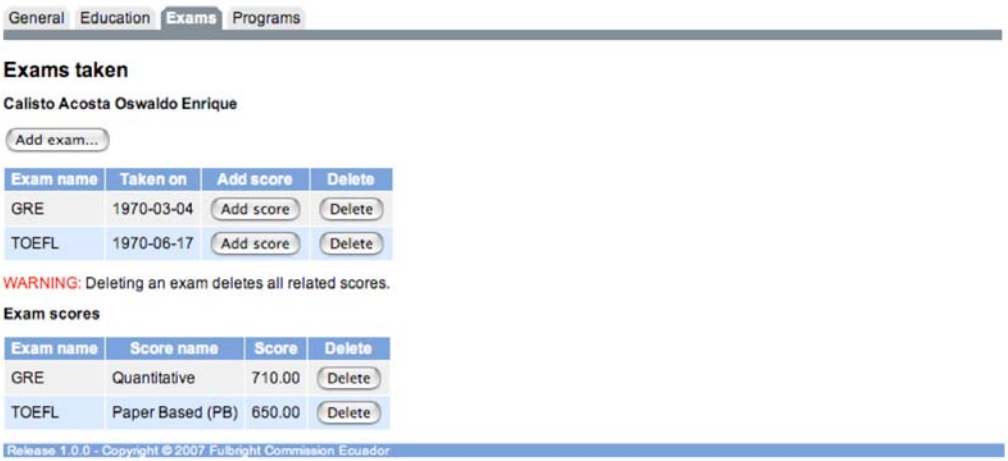


Figura 4.6 – Subsección de Exámenes

La parte superior de la página contiene los exámenes tomados por el aplicante, con su respectiva fecha de rendición. El botón *Add exam* permite ingresar un nuevo examen rendido (ver 4.1.1.1.3.1). Cada entrada posee un botón *Delete* (Borrar), si se presiona este botón, todos los puntajes relacionados con el examen también son borrados. La parte inferior de la página contiene los puntajes obtenidos en cada uno de los exámenes rendidos. El botón *Add score* en cada entrada de cada examen rendido (en la sección superior de la página) permite ingresar un puntaje individual de ese examen (ver 4.1.1.1.3.2). Cada entrada de puntajes posee un botón *Delete* (Borrar), si se presiona este botón, se elimina solamente el puntaje, no el examen.

4.1.1.1.3.1 *Add exam* (Añadir un examen)

Esta página permite añadir un examen rendido por el aplicante.

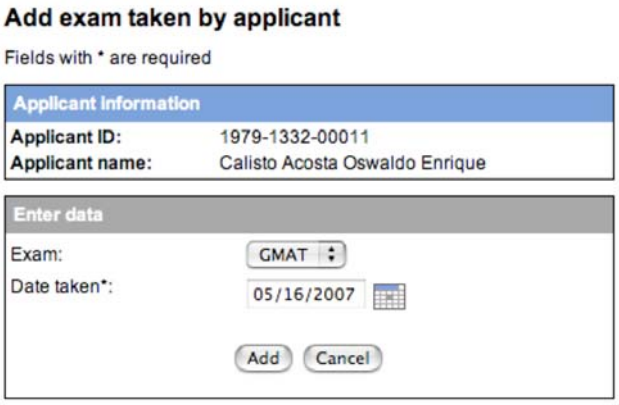


Figura 4.7 – Añadir un examen

Este formulario pide el tipo de examen rendido (que se escoge en la lista) y la fecha de rendición. El botón *Cancel* retorna a la página de exámenes (ver 4.1.1.1.3) sin realizar la adición del examen.

4.1.1.1.3.2 *Add exam score* (Añadir un puntaje de examen)

Esta página permite añadir un puntaje de un examen rendido por el aplicante.

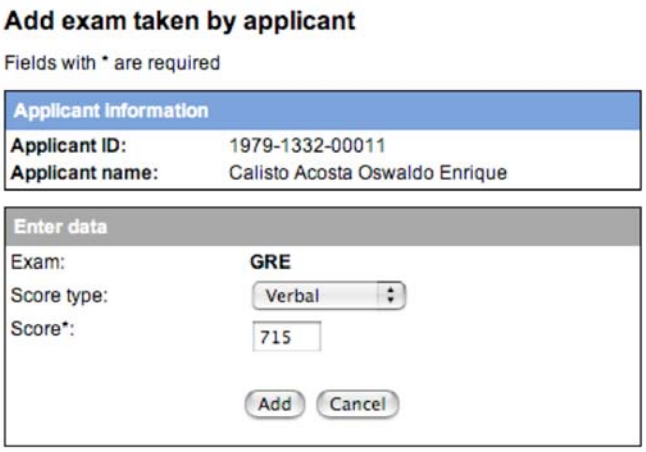


Figura 4.8 – Añadir un puntaje de examen

Este formulario muestra el examen al que pertenece el puntaje, permite escoger el tipo de puntaje de examen a ingresar, y el valor numérico del puntaje según indique el registro físico de resultados del examen. El botón *Cancel* retorna a la página de exámenes (ver 4.1.1.1.3) sin realizar la adición del puntaje.

Nota de advertencia: Este formulario, al ser de carácter general, no realiza validación si el puntaje ingresado está dentro del rango de un tipo de puntaje de examen específico. Se debe cuidar esto ya que ciertos puntajes de ciertos exámenes (TOEFL, GRE, GMAT) se utilizan en los procesos de pre-selección y selección de becarios, un valor incorrecto provocaría un cálculo de puntajes finales incorrectos y por lo tanto, irregularidades en los procesos.

4.1.1.1.4 Subsección *Programs and Grants* (Programas y Concesiones)

Esta subsección muestra todos los programas que el aplicante/becario ha aplicado y/o participado.

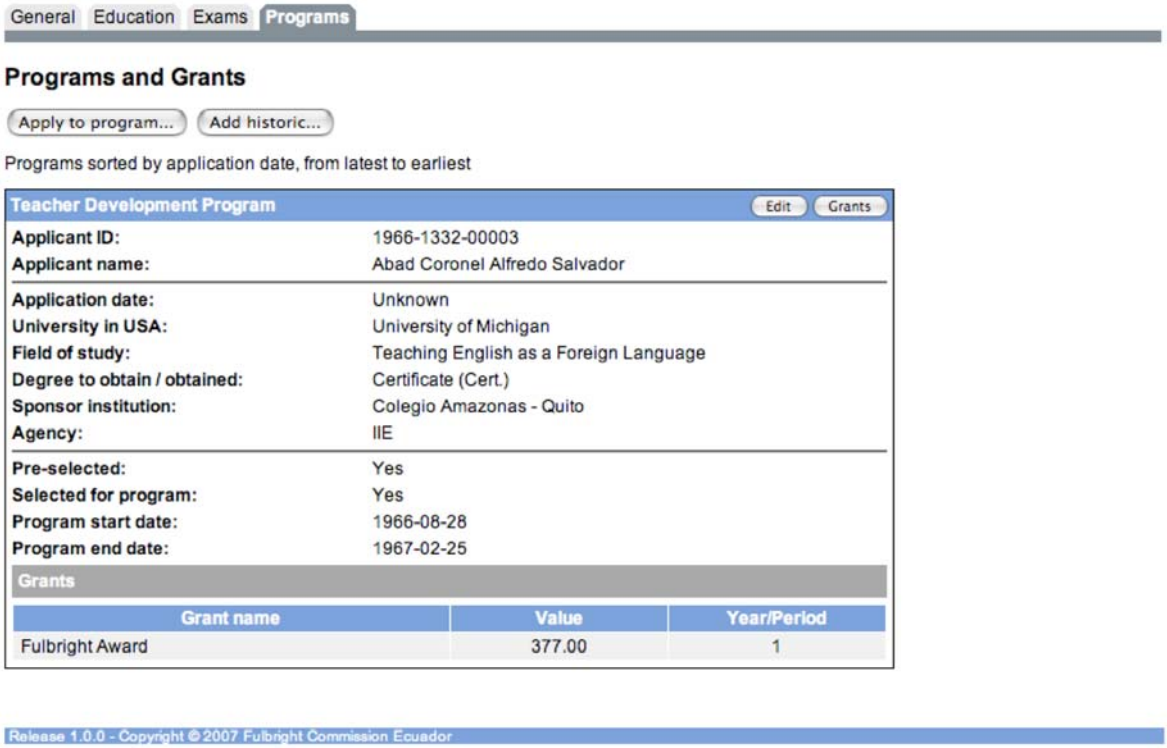


Figura 4.9 – Subsección de Programas y Concesiones Económicas

Cada cuadro contiene los datos un programa al cual se ha aplicado, el título azul de cada uno de estos cuadros es el nombre del programa. Contiene información como fecha de aplicación al programa, universidad, título a obtener/obtenido, agencia que auspicia el programa, campo de estudio, institución local que auspicia al aplicante/becario. En el caso de que el aplicante ha sido seleccionado como becario, o es un programa al cual ha calificado antes de la implementación de este sistema (es decir, programa histórico), muestra los datos de inicio y fin del programa, datos de programa pre-académico, y concesiones económicas realizadas por parte de la Comisión Fulbright al becario. Todos estos datos pueden ser editados mediante el botón *Edit* en la barra de título del cuadro (ver 4.1.1.1.4.1). Las concesiones económicas pueden ser editadas solamente en programas históricos o si el aplicante ha sido seleccionado como becario del programa, mediante el botón *Grants* en la barra de título del cuadro (ver 4.1.1.1.4.2). El botón *Apply to program* (Aplicar a programa) permite registrar al aplicante en un programa abierto (ver 4.2.1.1)

para participar del concurso. El botón *Add historic* sirve para registrar al aplicante como becario de un programa sin pasar por el proceso de selección.

4.1.1.1.4.1 *Add/update program* (Añadir/actualizar programa).

Permite aplicar a un programa abierto o histórico, según el botón que se haya presionado en la página de Programas y Concesiones (ver 4.1.1.1.4).

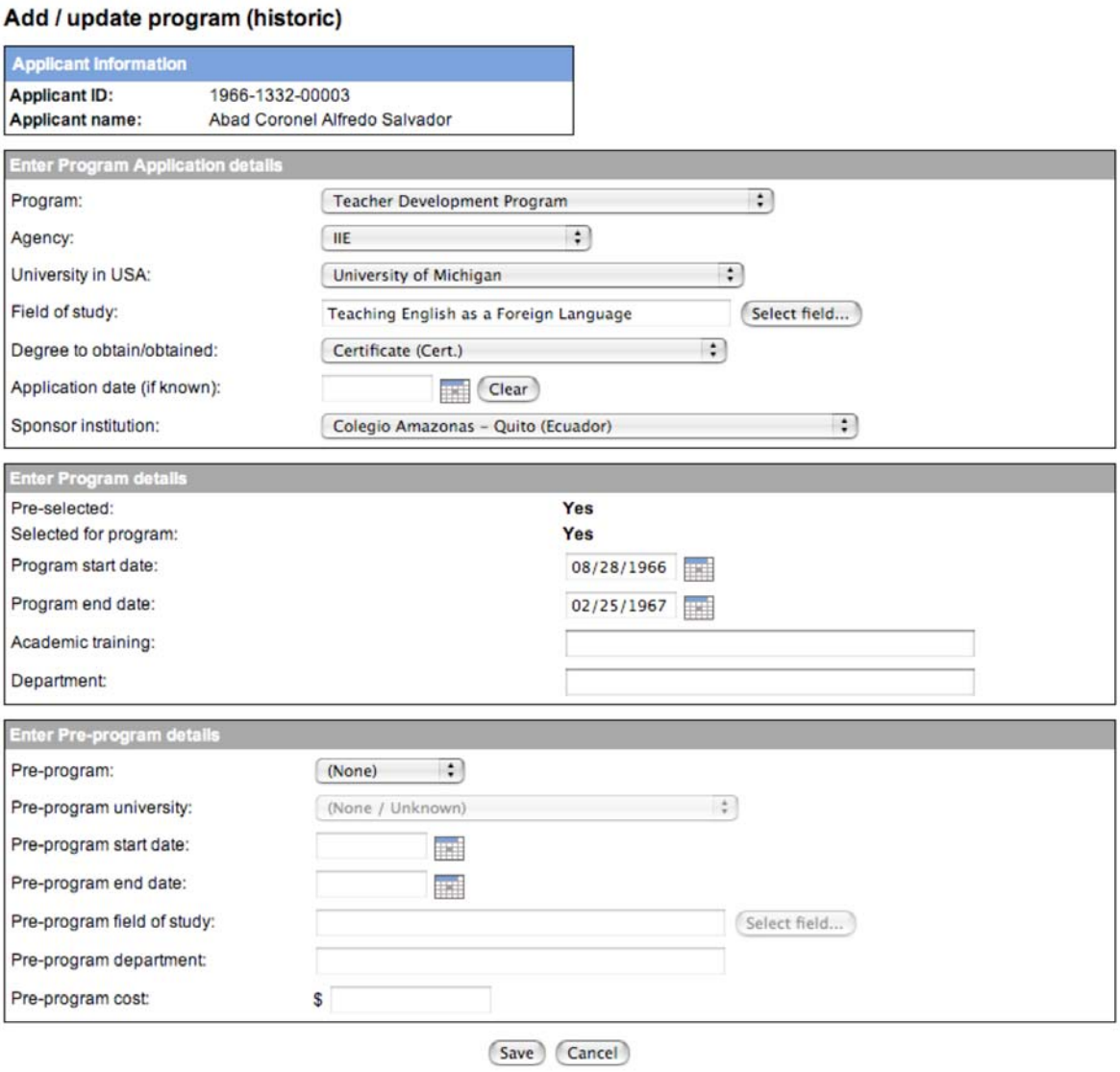


Figura 4.10 – Añadir o modificar aplicación a un programa

En el caso de programas abiertos, se muestra la lista de aquellos programas que en la fecha actual estén abiertos, o aquellos que serán abiertos en fechas futuras. En el caso de programas históricos, se mostrará una lista con todos los programas marcados como para

uso histórico (ver 4.2.1.2). Se debe ingresar la agencia que auspicia el programa, la universidad a la que se aplica, el título a obtener, la institución que auspicia localmente al aplicante, y el campo de estudio. Este último se escoge mediante un cuadro que permite escoger, por categoría y subcategoría, el campo deseado.

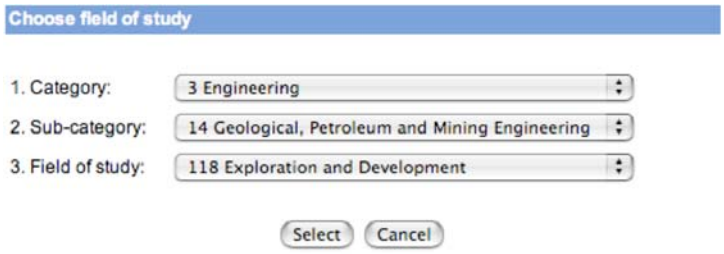


Figura 4.11 – Cuadro de selección de campo de estudio

Si el aplicante ha sido escogido como becario del programa (es decir, ha sido pre-seleccionado y seleccionado) o es un programa histórico, se permite el ingreso de las fechas de inicio y finalización del programa, departamento académico, y datos de programa pre-académico, donde se puede seleccionar el tipo de programa pre-académico, campo, universidad, fechas de inicio y finalización.

En el caso de que se quiera aplicar a un programa abierto, y no existan programas abiertos a la fecha, el mensaje "*No open programs found*" (No se encontraron programas abiertos) aparecerá en lugar del formulario.

4.1.1.1.4.2 *Edit Grants* (Editar Concesiones)

Esta página es accesible solamente a programas donde el aplicante ha sido escogido como becario, o en programas históricos.

Figura 4.12 – Editar Concesiones Económicas

El botón *Add row* permite añadir una fila de información. En cada fila se debe escoger un tipo de concesión económica, ingresar el valor de la concesión, y el año/período en que se realizó la concesión. El botón “X” en cada fila permite remover aquella fila de información. El botón *Save changes* graba todas aquellas filas de información que contengan datos y vuelve a la página de Programas y Concesiones (ver 4.1.1.1.4) El botón *Cancel* no realiza ninguna modificación y retorna a Programas y Concesiones (ver 4.1.1.1.4).

4.1.2 *Search Applicants* (Buscar aplicantes)

Esta función permite buscar aplicantes por múltiples criterios.

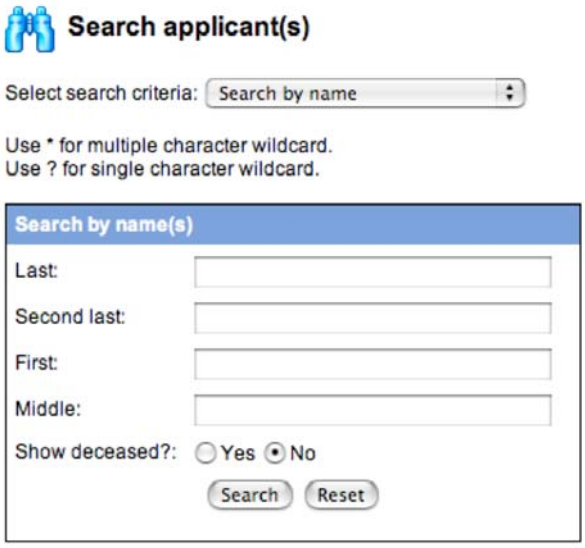
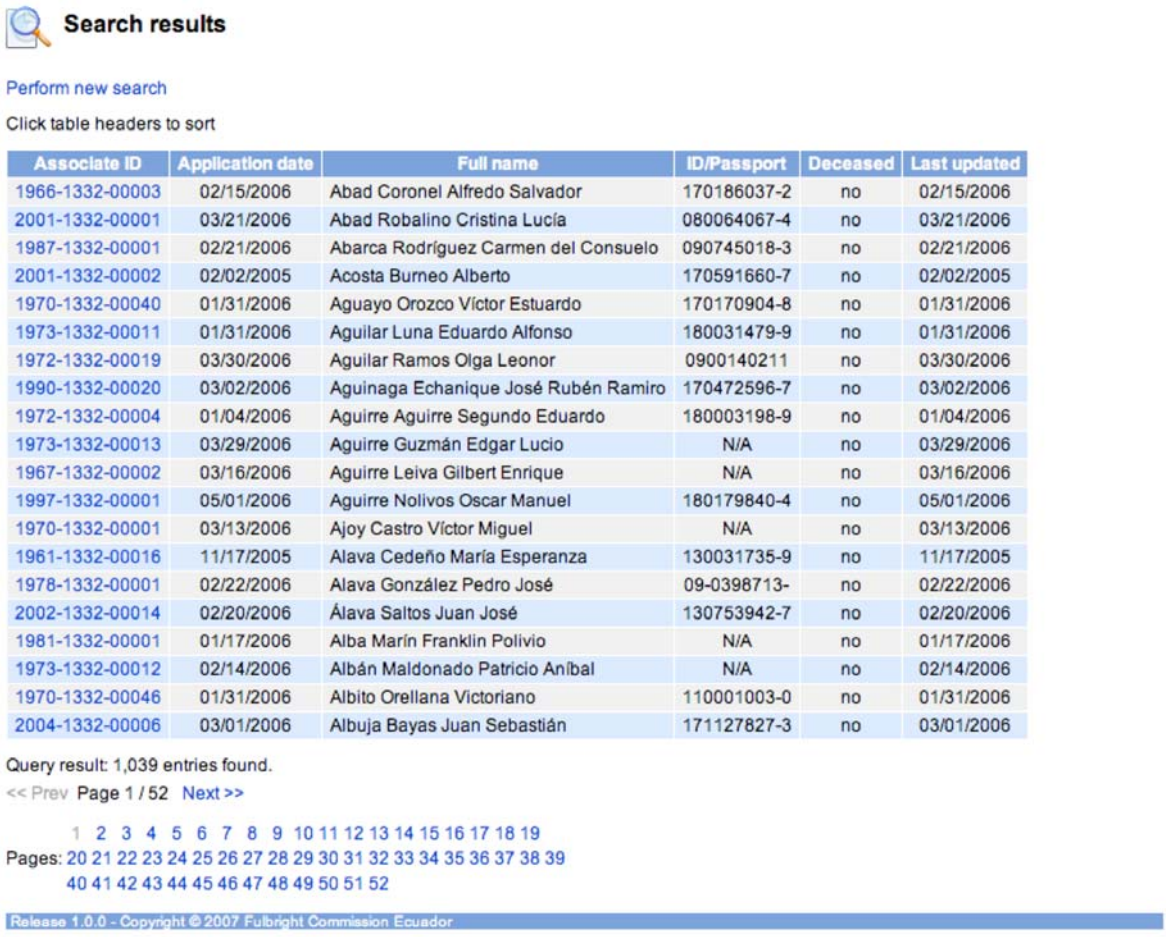


Figura 4.13 – Página de búsqueda de aplicantes

En la lista de la parte superior de la página se puede escoger el criterio de búsqueda deseado. En la parte inferior aparecerá el cuadro con los respectivos campos para filtrar la búsqueda. Estos campos permiten la entrada de comodines (*wildcards*) para generalizar las búsquedas, con el asterisco (*) para representar múltiples caracteres, y el signo de interrogación (?) para representar un solo carácter. Por ejemplo, “C*” encontrará todos los nombres que empiecen con C (“Calle”, “Calderón”, “Campos”, etc.); “Camp?” encontrará

todos aquellos nombres que empiecen con “Camp” y terminen con una letra cualquiera (“Campo”, “Camps”, “Campi”, etc.).

Los resultados de la búsqueda aparecen el la página *Search Results* (Resultados de Búsqueda).



Search results

Perform new search

Click table headers to sort

Associate ID	Application date	Full name	ID/Passport	Deceased	Last updated
1966-1332-00003	02/15/2006	Abad Coronel Alfredo Salvador	170186037-2	no	02/15/2006
2001-1332-00001	03/21/2006	Abad Robalino Cristina Lucía	080064067-4	no	03/21/2006
1987-1332-00001	02/21/2006	Abarca Rodríguez Carmen del Consuelo	090745018-3	no	02/21/2006
2001-1332-00002	02/02/2005	Acosta Burneo Alberto	170591660-7	no	02/02/2005
1970-1332-00040	01/31/2006	Aguayo Orozco Víctor Estuardo	170170904-8	no	01/31/2006
1973-1332-00011	01/31/2006	Aguilar Luna Eduardo Alfonso	180031479-9	no	01/31/2006
1972-1332-00019	03/30/2006	Aguilar Ramos Olga Leonor	0900140211	no	03/30/2006
1990-1332-00020	03/02/2006	Aguinaga Echanique José Rubén Ramiro	170472596-7	no	03/02/2006
1972-1332-00004	01/04/2006	Aguirre Aguirre Segundo Eduardo	180003198-9	no	01/04/2006
1973-1332-00013	03/29/2006	Aguirre Guzmán Edgar Lucio	N/A	no	03/29/2006
1967-1332-00002	03/16/2006	Aguirre Leiva Gilbert Enrique	N/A	no	03/16/2006
1997-1332-00001	05/01/2006	Aguirre Nolivos Oscar Manuel	180179840-4	no	05/01/2006
1970-1332-00001	03/13/2006	Ajoy Castro Víctor Miguel	N/A	no	03/13/2006
1961-1332-00016	11/17/2005	Alava Cedeño María Esperanza	130031735-9	no	11/17/2005
1978-1332-00001	02/22/2006	Alava González Pedro José	09-0398713-	no	02/22/2006
2002-1332-00014	02/20/2006	Álava Saltos Juan José	130753942-7	no	02/20/2006
1981-1332-00001	01/17/2006	Alba Marín Franklin Polivio	N/A	no	01/17/2006
1973-1332-00012	02/14/2006	Albán Maldonado Patricio Aníbal	N/A	no	02/14/2006
1970-1332-00046	01/31/2006	Albito Orellana Victoriano	110001003-0	no	01/31/2006
2004-1332-00006	03/01/2006	Albuja Bayas Juan Sebastián	171127827-3	no	03/01/2006

Query result: 1,039 entries found.

<< Prev Page 1 / 52 Next >>

Pages: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52

Release 1.0.0 - Copyright © 2007 Fulbright Commission Ecuador

Figura 4.14 – Página de resultados de búsqueda de aplicantes

La tabla de resultados contiene el código del aplicante (asignado automáticamente al momento de su registro), nombres completos de los aplicantes, número de documento de identificación, y si está marcado como difunto o no. Si los resultados exceden el número de filas configuradas para el usuario (ver Administración de Usuarios), aparecerán en múltiples páginas seleccionables con los controles en la parte inferior de la página de resultados. Al hacer clic en el código del aplicante deseado, aparecerá la página de

información general del aplicante (ver 4.1.1.1). Al hacer clic en *Perform new search* (Realizar nueva búsqueda), se retorna a la página de ingreso de filtros de búsqueda.

4.1.3 *Pre-selection* (Pre-selección)

Esta función permite ingresar al proceso de pre-selección de aplicantes para la fase final del concurso de un programa.

4.1.3.1 Página de selección de programa

Esta página permite seleccionar el programa en el cual se ejecutará el proceso de pre-selección de finalistas.

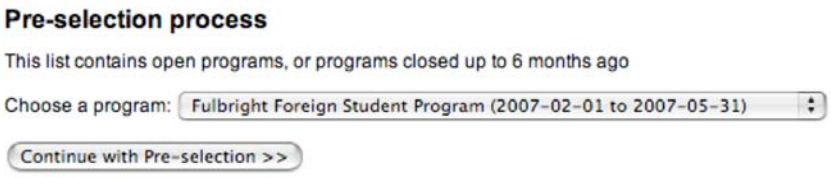


Figura 4.15 – Página de selección de programa para Pre-selección

En la lista aparecerán aquellos programas que se encuentren abiertos, o aquellos programas que hayan sido cerrados hasta 6 meses atrás. Una vez seleccionado el programa, se debe presionar en el botón *Continue with Pre-selection* (Continuar con Pre-selección).

En el caso de que no se encuentren programas abiertos o programas cerrados hasta 6 meses atrás, la aplicación indicará que no es posible realizar ningún proceso de pre-selección.

4.1.3.2 Lista de aplicantes al programa

A continuación se muestra la lista de todos aquellos aplicantes al programa, en una tabla que muestra el código de aplicante, nombres completos, indicación de que ha sido pre-seleccionado o no, indicación de que ha sido eliminado o no, y un botón para detalles adicionales de aplicación (*Details*).

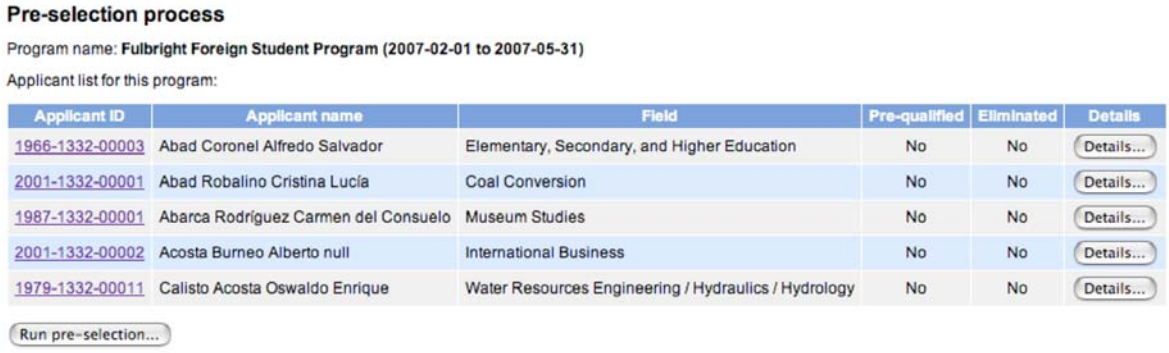


Fig 4.16 – Lista de aplicantes al programa

Cuando se presiona en *Details*, permite editar el promedio acumulado (GPA) del aplicante, ingresar los puntajes de cartas de recomendación y ensayos (ver 4.1.3.2.1).

El botón *Run Pre-selection* (Correr pre-selección) corre el proceso en sí, pasando primero por la eliminación de aplicantes que no cumplen con los requisitos (ver 4.1.3.3).

4.1.3.2.1 Detalles de aplicación a programa

Esta página muestra el promedio acumulado del aplicante (GPA), cartas de recomendación y ensayos con sus respectivos puntajes.

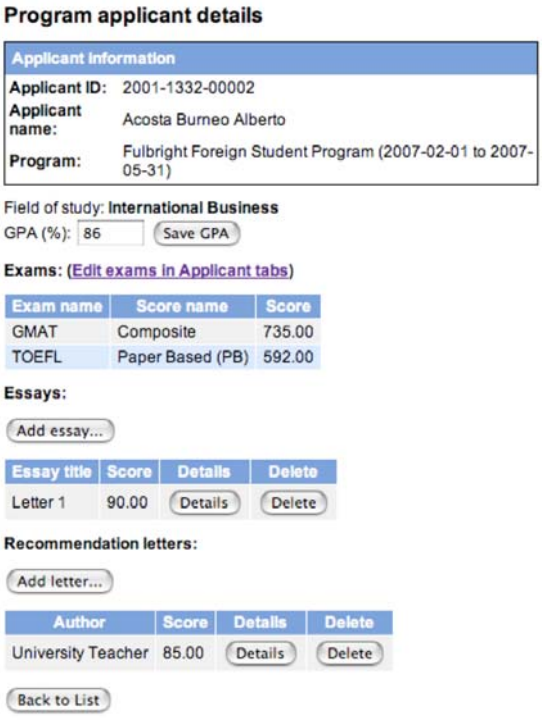


Figura 4.17 – Detalles de aplicación a programa

Para grabar el GPA, se lo ingresa en el cuadro de texto y se presiona en *Save GPA.* Aparecerá *Saved* en letras verdes indicando que el GPA fue ingresado exitosamente. Para ingresar las cartas de recomendación y los ensayos, se debe presionar en *Add letter* o en *Add essay* (ver 4.1.3.2.1.1). Se puede editar cada entrada haciendo clic en el respectivo botón *Update*. Se puede regresar a la lista de aplicantes presionando en el botón *Back to list*.

4.1.3.2.1.1 Ingreso de documentos (ensayos y cartas)

Este formulario permite ingresar los datos de un documento, carta de recomendación o ensayo.

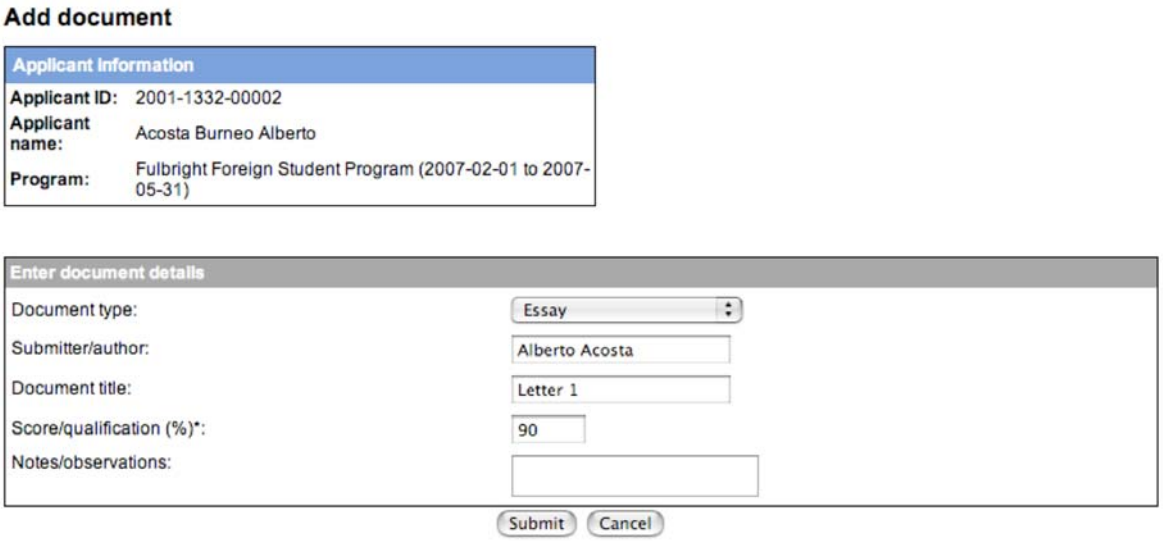


Figura 4.18 – Formulario de ingreso de documentos.

Permite el ingreso del tipo de documento, título del documento, autor del documento, puntaje asignable al documento (necesario para ensayos y cartas, requerido para los procesos de pre-selección y selección) y notas/observaciones. El botón *Cancel* vuelve a la página de Detalles de aplicación a programa (ver 4.1.3.2.1) sin realizar modificaciones.

4.1.3.3 Eliminación previa

Cuando en la lista de aplicantes (ver 4.1.3.2) se presiona en *Run Pre-selection*, primeramente se verifica cuales aplicantes no cumplen con los requisitos mínimos de pre-

calificación, así que se avisa al usuario que tales usuarios serán eliminados para el cálculo de puntajes de la pre-selección.

Figura 4.19 – Pre-eliminación

Si se presiona en *Continue*, los aplicantes mostrados en la lista serán eliminados automáticamente. Si se presiona en *Go Back*, no se realizará la eliminación y retornará a la página de lista de aplicantes (4.1.3.2). Si todos los aplicantes cumplen con los requisitos, entonces un mensaje aparecerá indicando que no se eliminará ningún aplicante cuando se presione en *Continue with Pre-selection*.

4.1.3.4 Cálculo de *rankings* para Pre-selección

Para cada aplicante se calculará un puntaje, basado en sus exámenes (GRE, TOEFL, etc.), calidad de cartas de recomendación, ensayos, y promedio acumulado GPA.

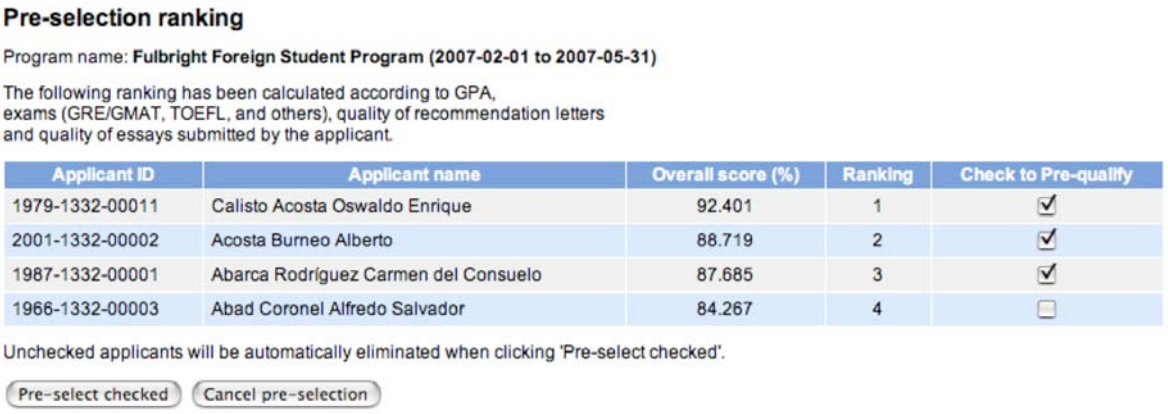


Figura 4.20 – Lista con rankings para pre-selección

Aquel aplicante con el mayor puntaje tendrá *ranking* 1 (primer lugar). El usuario debe marcar aquellos aplicantes que pasarán a la fase final, y presionar sobre el botón *Pre-select checked* (Pre-seleccionar marcados), los aplicantes no marcados serán eliminados. Si se

presiona sobre *Cancel Pre-selection*, ningún aplicante será pre-seleccionado ni eliminado, y retornará a la página de lista de aplicantes (4.1.3.2).

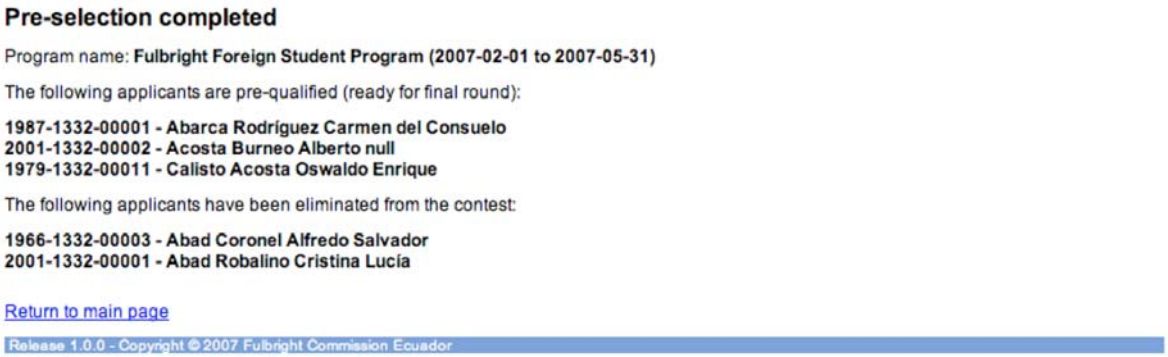


Figura 4.21 – Lista de aplicantes pre-seleccionados y eliminados

Si se han marcado los aplicantes a pre-seleccionar, y se hace clic sobre *Pre-select checked*, una lista con los aplicantes pre-seleccionados y eliminados aparecerá en pantalla.

4.1.4 *Selection* (Selección)

Esta función permite realizar la selección final de becarios para los programas académicos. Para poder correr esta función, se debe haber ejecutado primero la pre-selección para el programa donde se desea realizar la selección.

4.1.4.1 Página de selección de programa

Esta página permite seleccionar el programa en el cual se ejecutará el proceso de selección.

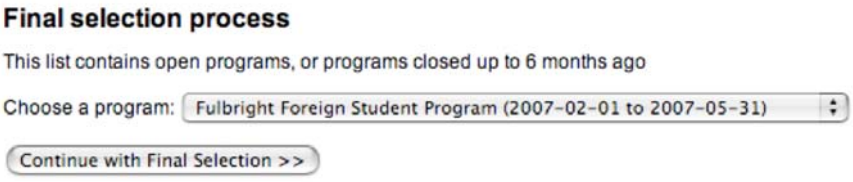


Figura 4.22 – Página de selección de programa para Selección Final

En la lista aparecerán aquellos programas que se encuentren abiertos, o aquellos programas que hayan sido cerrados hasta 6 meses atrás. Una vez seleccionado el programa, se debe presionar en el botón *Continue with Final Selection* (Continuar con Selección).

En el caso de que no se encuentren programas abiertos o programas cerrados hasta 6 meses atrás, la aplicación indicará que no es posible realizar ningún proceso de selección.

4.1.4.2 Lista de aplicantes al programa

De la misma forma como en el proceso de pre-selección (ver 4.1.3.2), aparece una lista de aplicantes del programa, con la diferencia que esta lista solamente contiene aquellos aplicantes que han sido pre-seleccionados para el programa.

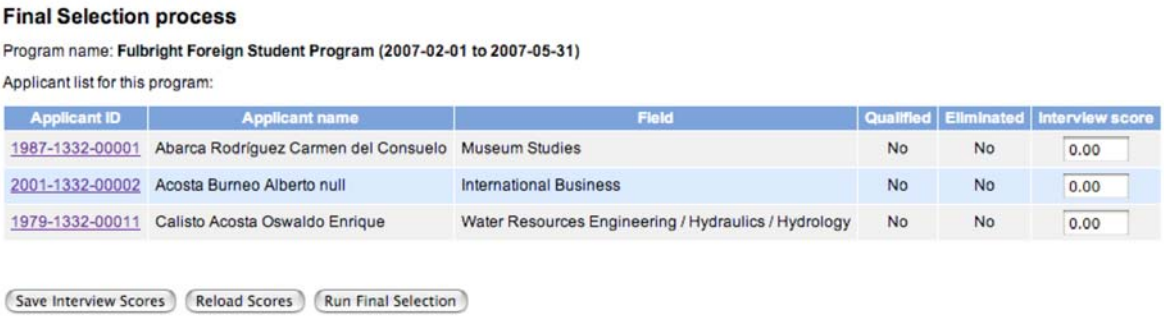
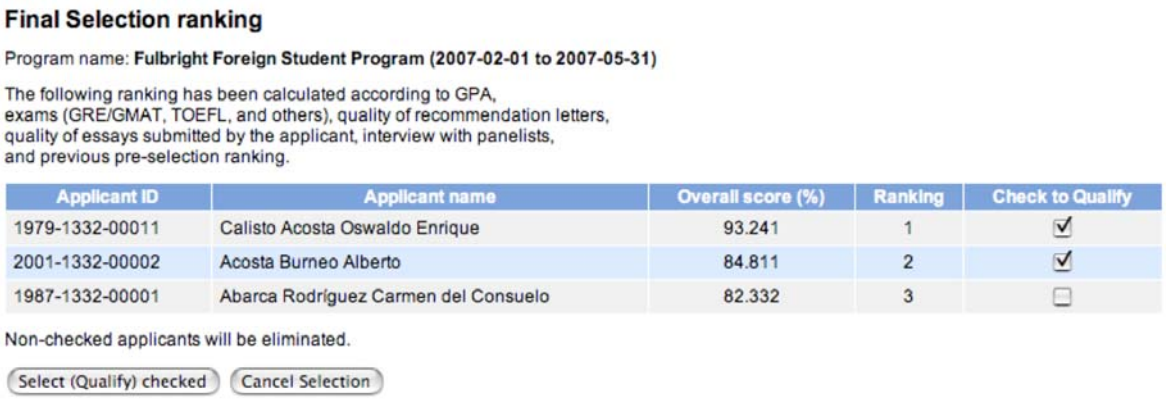


Figura 4.23 – Lista de aplicantes pre-seleccionados para selección final.

Cada aplicante tiene un campo donde se puede escribir el puntaje obtenido en las entrevistas con los panelistas (parte del proceso oficial de selección de becarios), una vez ingresados estos valores se los puede grabar con *Save interview scores* (Grabar puntajes de entrevistas). El botón *Reload Scores* (Re-cargar puntajes) permite obtener los puntajes anteriormente ingresados, en el caso de cometer errores en el ingreso. El botón *Run Final Selection* corre el proceso de cálculo de puntajes finales y permite seleccionar los becarios del programa.

4.1.4.3 Cálculo de *rankings* finales

Una vez presionado el botón *Run Final Selection* en la lista de aplicantes (4.1.4.2), el sistema, basado en el puntaje obtenido en la pre-selección y los resultados de las entrevistas.



Final Selection ranking

Program name: **Fulbright Foreign Student Program (2007-02-01 to 2007-05-31)**

The following ranking has been calculated according to GPA,
exams (GRE/GMAT, TOEFL, and others), quality of recommendation letters,
quality of essays submitted by the applicant, interview with panelists,
and previous pre-selection ranking.

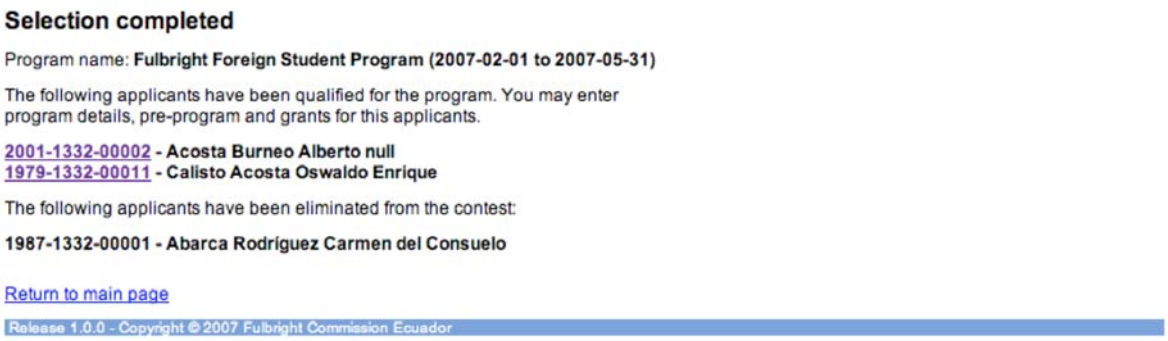
Applicant ID	Applicant name	Overall score (%)	Ranking	Check to Qualify
1979-1332-00011	Calisto Acosta Oswaldo Enrique	93.241	1	☑
2001-1332-00002	Acosta Burneo Alberto	84.811	2	☑
1987-1332-00001	Abarca Rodríguez Carmen del Consuelo	82.332	3	☐

Non-checked applicants will be eliminated.

Select (Qualify) checked | Cancel Selection

Figura 4.24 – Lista de *rankings* para selección final

De la misma manera como se realiza en la pre-selección (ver 4.1.3.4), el usuario debe marcar aquellos que van a ser seleccionados. Después de presionar sobre *Select checked* (Seleccionar marcados), el sistema marcará como seleccionados a aquellos que estén marcados, y como eliminados a aquellos que no estén marcados. Si no se desea continuar con la selección, se debe presionar en *Cancel Selection* y volverá a la lista de aplicantes.



Selection completed

Program name: **Fulbright Foreign Student Program (2007-02-01 to 2007-05-31)**

The following applicants have been qualified for the program. You may enter
program details, pre-program and grants for this applicants.

2001-1332-00002 - **Acosta Burneo Alberto null**
1979-1332-00011 - **Calisto Acosta Oswaldo Enrique**

The following applicants have been eliminated from the contest:

1987-1332-00001 - Abarca Rodríguez Carmen del Consuelo

Return to main page

Release 1.0.0 - Copyright © 2007 Fulbright Commission Ecuador

Figura 4.25 – Lista de seleccionados y eliminados

Después de la selección, aparecerá la lista que resume cuales aplicantes fueron seleccionados como becarios, y cuales han sido eliminados del proceso.

4.2 Submenú *Programs*

Este submenú posee funciones para creación de programas académicos, apertura de programas, y mantenimiento de agencias y programas pre-académicos.

4.2.1 *Open Programs* (Programas abiertos)

Esta función permite abrir programas para aplicaciones, y ver aquellos programas que serán abiertos en el futuro y aquellos que ya han sido cerrados.

Figura 4.26 – Lista de programas abiertos

La lista muestra una tabla con el identificador de cada programa abierto, nombre del programa, fecha de apertura y fecha de cierre. Al hacer clic en *Open program*, se podrá seleccionar cualquier programa activo y darle un período de apertura entre fechas (ver 4.2.1.1). Al hacer clic en cualquiera de los identificadores de programa abierto, se puede modificar las fechas de apertura y cierre.

4.2.1.1 *Open a program* (Abrir programa)

Este formulario permite abrir (o modificar los parámetros de apertura) un programa.

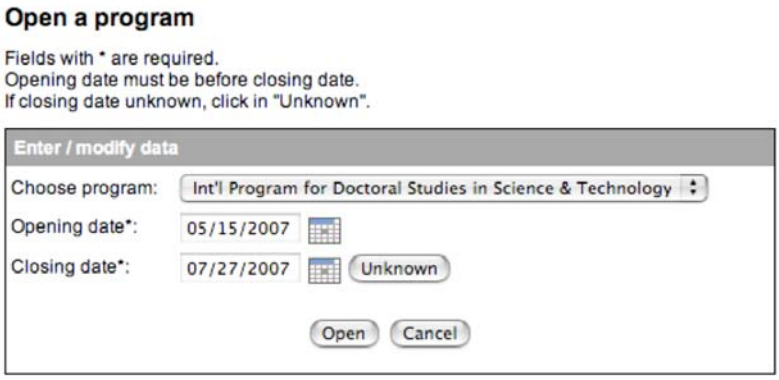


Figura 4.27 – Formulario de apertura de programa

El usuario debe escoger de la lista el programa a abrir, y las fechas de apertura y cierre. Existe la posibilidad de abrir un programa indefinidamente, para lograr esto se debe presionar en el botón *Unknown*, con esto el programa permanecerá abierto siempre (o hasta que en alguna modificación se fije una fecha).

4.2.1.2 *Future programs* (Programas futuros)

Esta lista muestra simplemente aquellos programas que tengan una fecha de apertura posterior a la fecha actual del sistema.

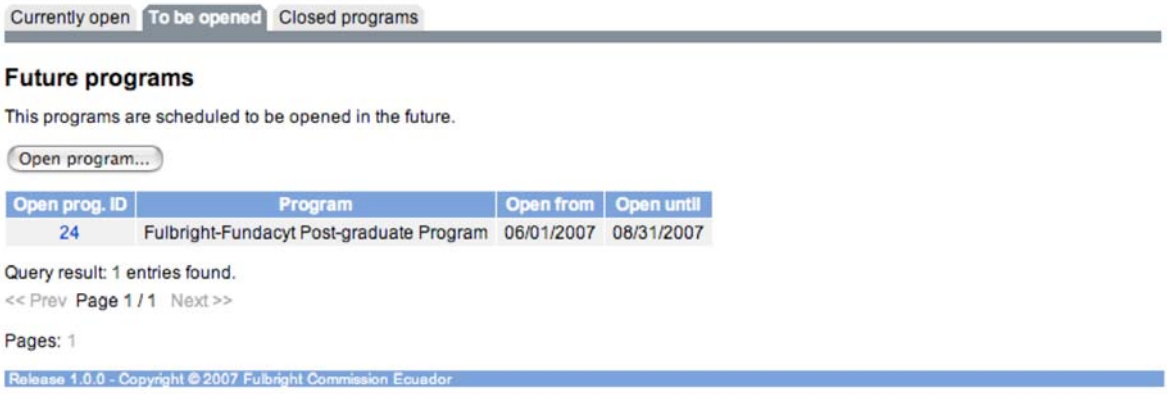


Figura 4.28 – Lista de programas futuros

4.2.1.3 *Closed programs* (Programas cerrados)

Esta lista muestra aquellos programas cuya fecha de cierre ya ha sido sobrepasada (relativa a la fecha actual del sistema).

Figura 4.29 – Lista de programas cerrados

4.2.2 *All Programs* (Todos los programas)

Esta función permite visualizar todos los programas académicos ingresados al sistema, incluso los desactivados.

All Programs list

Add program...

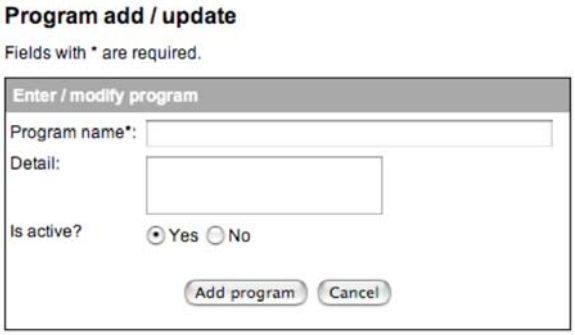
Program ID	Program name	Is active?
1	Fulbright Foreign Student Program	yes
2	Fulbright Faculty Development Program	yes
3	Fulbright-Fundacyt Post-graduate Program	yes
4	Int'l Program for Doctoral Studies in Science & Technology	yes
5	The Study of the United States Program	yes
6	Hubert H. Humphrey Program	yes
7	Fulbright Visiting Scholar Program	yes
8	Fulbright Scholar-in-Residence Program	yes
9	New Century Scholar Program	yes
10	IIE - Non-Fulbright	no
11	IIE - No FC or Gov. Funding	no
12	Multinational Specialist Group Project	no
13	Teacher Development Program	no
14	Teacher Development Workshop	no
15	Travel Grant	no
16	Aid - Fulbright	no
17	Amazon Basin Program	no
18	American Studies Winter Institute	no
19	Fulbright-OAS Ecology Initiative	no
20	The Study of the United States - Secondary Educators	no
21	Unknown Program	no
22	State Alumni	no
23	Fulbright Science and Technology Program	yes

Figura 4.30 – Lista de todos los programas académicos

Al hacer clic en cualquiera de los identificadores de programa, se puede modificar sus datos (ver 4.2.2.1). De la misma manera, se puede añadir un nuevo programa al sistema mediante el botón *Add program* (ver 4.2.2.1).

4.2.2.1 *Add/update program* (Añadir/modificar programa)

Este formulario permite ingresar o modificar un programa académico.



Program add / update

Fields with * are required.

Enter / modify program

Program name*:

Detail:

Is active? Yes No

Add program Cancel

Figura 4.31 – Formulario de ingreso de programa

Se debe ingresar el nombre del programa, y opcionalmente una descripción. También se debe determinar si el programa estará activo para poder ser utilizado en los procesos de apertura de concursos.

4.2.3 *Agencies* (Agencias)

Esta función permite ver y modificar las agencias del gobierno que auspician los programas académicos.

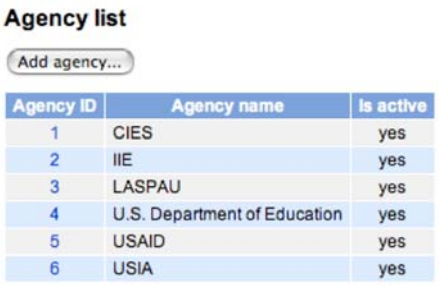


Figura 4.32 – Lista de agencias

Haciendo clic sobre cualquier identificador de agencia, permite realizar modificaciones sobre la misma (ver 4.2.3.1). El botón *Add agency* permite añadir una nueva agencia.

4.2.3.1 *Add/update agency* (Añadir/modificar agencia)

Este formulario permite el ingreso de una agencia.

Figura 4.33 – Formulario de ingreso de agencia

Se debe ingresar el nombre de la agencia, y opcionalmente, una descripción corta de la misma para propósitos informativos. De la misma manera que para los programas

académicos, se debe indicar si está activa y podrá ser utilizada en las diferentes opciones al interior del sistema.

4.2.4 *Pre-programs* (Programas pre-académicos)

Esta función permite visualizar y modificar los programas pre-académicos que se pueden asignar a becarios de programas.

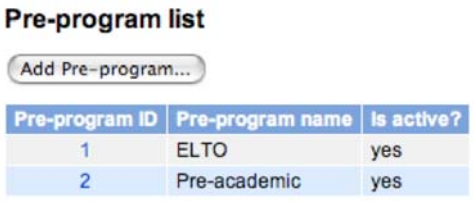


Figura 4.34 – Lista de programas pre-académicos

Al hacer clic sobre cualquiera de los identificadores de pre-programas, permite modificarlos (ver 4.2.4.1). El botón *Add pre-program* permite añadir más programas pre-académicos al sistema (ver 4.2.4.1).

4.2.4.1 *Add/update Pre-program* (Añadir/modificar Pre-programa)

Este formulario permite ingresar o modificar los programas pre-académicos.

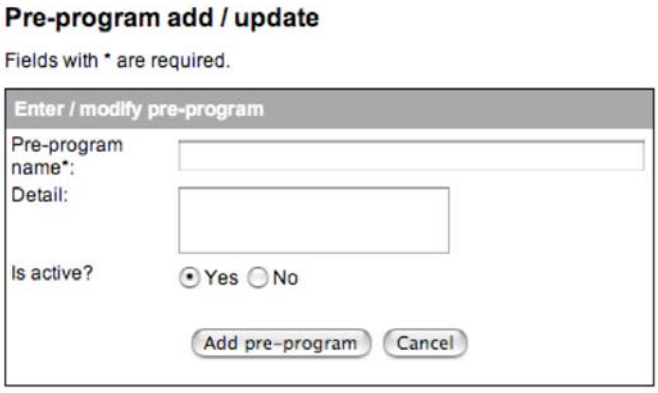


Figura 4.35 – Formulario de ingreso de pre-programas

Se debe ingresar el nombre del pre-programa, opcionalmente una descripción corta del mismo con propósito informativo, y determinar si estará activo para ser utilizado dentro de las diferentes funciones del sistema.

4.3 Submenú *Reports*

Este submenú contiene la función para generación de reportes del sistema, que pueden ser impresos.

4.3.1 *Report Generator* (Generador de Reportes)

Esta función permite configurar los reportes que se deseen obtener.

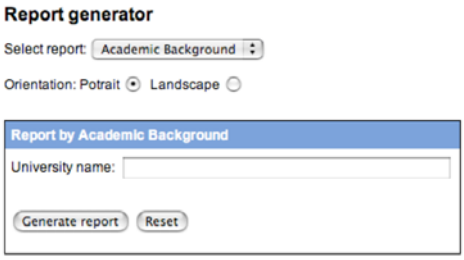


Figura 4.36 – Pantalla de selección de reportes

En la parte superior de la página se escoge el tipo de reporte a generar, y en la parte inferior de la página aparecerán los filtros de información específicos del reporte, que soportan el uso de comodines (de manera similar a la función de búsqueda de aplicantes, ver 4.1.2). El reporte generado se despliega en una ventana aparte, como en la siguiente figura.

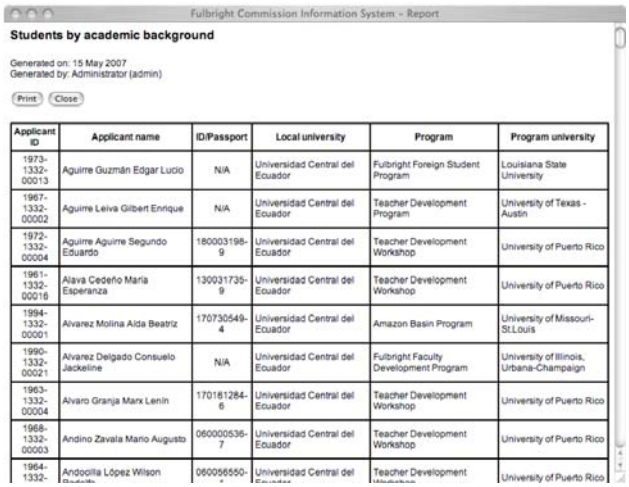


Figura 4.37 – Ejemplo de Reporte

Todos los reportes comparten la siguiente información general. En la parte superior contienen el título del reporte en negrita, y debajo del título la fecha de generación del reporte, y el usuario quien generó el reporte. También contiene dos botones, *Print* que permite imprimir el reporte, y *Close* que cierra la ventana del reporte.

4.4 Submenú *Administration*

Este submenú contiene funciones de administración y mantenimiento del sistema.

4.4.1 *Manage Users* (Administración de Usuarios)

Esta función permite administrar los usuarios del sistema.

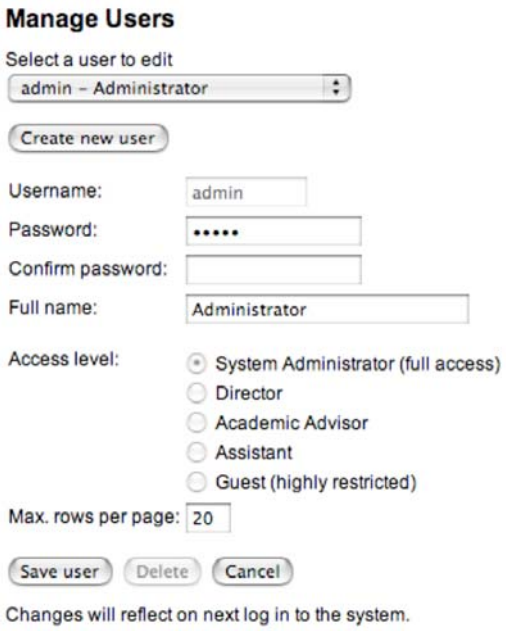


Figura 4.38 – Administración de Usuarios

Solamente los administradores del sistema son capaces de añadir o eliminar usuarios y cambiar niveles de acceso. Los demas tipos de usuarios solamente podrán cambiar su propia contraseña, su nombre completo y el número de resultados por página en consultas de tipo paginado. En la parte superior de la página, el administrador puede escoger el usuario al cual se realizarán las modificaciones (los usuarios con menor nivel de acceso

tienen esta opción bloqueada). El botón *Save user* graba los cambios, que se reflejarán la próxima vez que el usuario ingrese al sistema. El botón *Delete* permite eliminar un usuario (es imposible eliminar el usuario “admin”, ni siquiera los administradores del sistema serán capaces de eliminar este usuario). El botón *Cancel* no realiza ningún cambio y retorna a la página principal del sistema.

4.4.2 *Exam types* (Tipos de exámenes)

Esta función permite añadir o modificar los exámenes del sistema y sus respectivos tipos de puntajes.

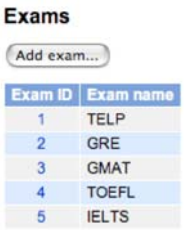


Figura 4.39 – Lista de exámenes

Al hacer clic en cualquiera de los identificadores de examen permite modificarlos (ver 4.4.2.1). El botón *Add exam* permite añadir un nuevo examen (ver 4.4.2.1).

4.4.2.1 *Add/update exam type* (Añadir/modificar tipo de examen)

Permite añadir o modificar un tipo de examen.

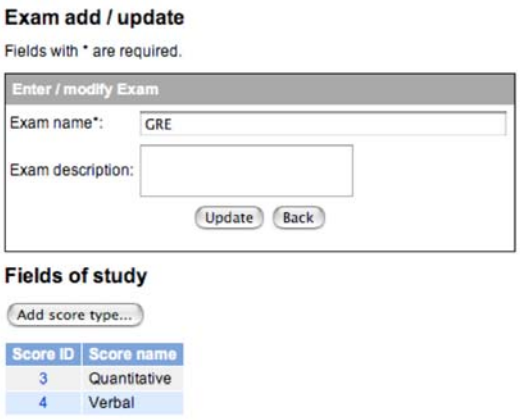


Figura 4.40 – Formulario de tipo de examen con sus respectivos tipos de puntajes

En el caso de ingreso de nuevo programa, se pide nombre de examen. En el caso de modificación de examen, mediante el botón *Add score type* se puede añadir tipos de puntajes (ver 4.4.2.1.1). El botón *Back* permite retornar a la lista de exámenes.

4.4.2.1.1 *Add/update score type* (Añadir/modificar tipo de puntaje)

Este formulario permite ingresar un nuevo tipo de puntaje.

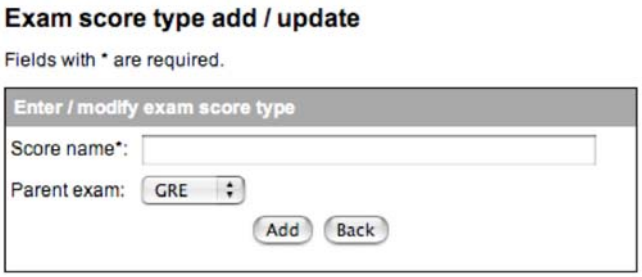


Figura 4.41 – Formulario de tipo de puntaje

La lista escogerá automáticamente el examen padre, pero es posible escoger otro examen como padre de este puntaje. Se debe ingresar el nombre del tipo de puntaje. Una vez ingresado, se debe presionar en *Back* para volver a la descripción del examen padre.

4.4.3 *Degrees* (Títulos/grados)

Esta función permite mantener la lista de títulos/grados del sistema.

Degree list

Add degree...

Degree ID	Degree name	Degree title
1	Associate	Assoc.
2	Bachelor of Arts	B.A.
3	Bachelor of Science	B.S.
4	Master of Arts	M.A.
5	Master of Science	M.S.
6	Doctor of Philosophy	Ph.D.
7	Doctor of Medicine	M.D.
8	Engineer	Eng.
9	Licenciado(a)	Lic.
10	Ingeniero(a)	Ing.
11	Arquitecto(a)	Arq.
12	Doctor(a)	Dr.
13	Master of Laws	L.L.M.
14	Master of Business Administration	M.B.A.
15	Certificate	Cert.
16	Diploma	Diploma
17	Master of Public Health	M.P.H.
18	Non-Degree - Undergraduate	N.D. - UG
19	Non-Degree - Seminar	N.D. - Sem
20	Non-Degree	N.D.
21	Master of Engineering	M.E.
22	Post-doctoral	P.D.
23	Doctor of Business Administration	D.B.A.
24	Doctor of Education	Ed.D.
25	Doctor of Philosophy Candidate	Ph.D. Cnd.
26	Master of Science in Nursing Education	M.S.N.
27	Master of Architecture	M.Arch.
28	Master of Fine Arts	M.F.A.
29	Master of Government Administration	M.G.A.
30	Master of Public Administration	M.P.A.
31	Master of Science in Business	M.S.B.
32	Master of Science in Finance	M.S.F.

Figura 4.42 – Lista de títulos/grados del sistema

Al hacer clic en cualquiera de los identificadores de título/grado, permite modificarlo (ver 4.4.3.1), y al hacer clic en *Add degree* permite añadir un nuevo título/grado.

4.4.3.1 *Add/update degree* (Añadir/modificar título/grado)

Este formulario permite ingresar o modificar un título/grado.

Degree add / update

Fields with * are required.

Enter / modify degree

Degree name*:

Degree title*:

Add degree Cancel

Figura 4.43 – Formulario de título/grado.

Ambos campos son obligatorios, el primero debe contener el nombre completo del título/grado, y el segundo la abreviación del título/grado.

4.4.4 *Grant types* (Tipos de Concesiones)

Esta función permite añadir o modificar los tipos de concesiones económicas que pueden escogerse en la ventana de edición de Concesiones Económicas (ver 4.1.1.1.4.2).

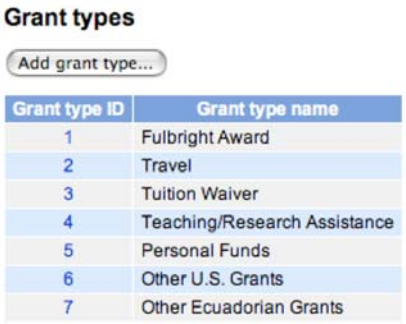


Figura 4.44 – Lista de Concesiones Económicas

Al hacer clic en cualquiera de los identificadores de concesión, permite realizar modificaciones (ver 4.4.4.1). Se puede añadir nuevos tipos de concesiones económicas presionando sobre el botón *Add grant type* (ver 4.4.4.1).

4.4.4.1 *Add/update grant type* (Añadir/modificar tipo de concesión)

Permite ingresar o modificar tipos de concesiones económicas.

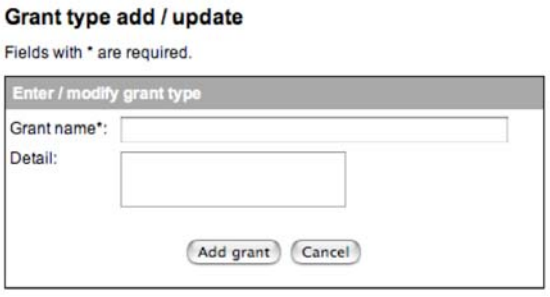


Figura 4.45 – Formulario de tipo de concesión

El campo obligatorio es el nombre del tipo de concesión. Se puede ingresar opcionalmente una descripción del mismo para propósitos informativos.

4.4.5 *Fields of study* (Campos de estudio)

Esta función permite mantener las categorías, subcategorías y campos de estudio del sistema.

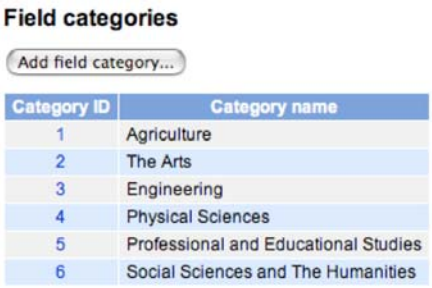


Figura 4.46 – Lista de categorías de campos de estudio

Al hacer clic en cualquiera de los identificadores de categoría de campo de estudio, permite realizar modificaciones (ver 4.4.5.1). El botón *Add category* permite añadir una categoría de campos de estudio (ver 4.4.5.1).

4.4.5.1 *Add/update field category* (Añadir/modificar categoría)

Permite ingresar o modificar una categoría de campos de estudio.

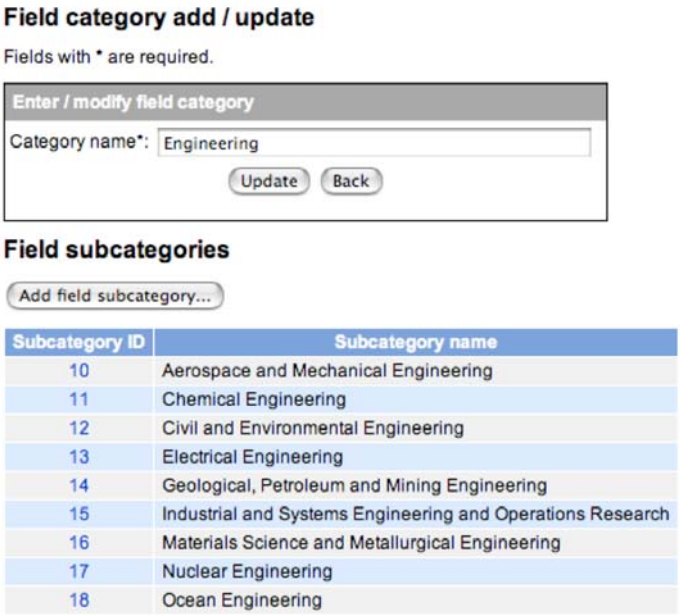


Figura 4.47 – Formulario de categoría con sus respectivas subcategorías

En el caso de ingreso de categoría, solamente se pedirá el nombre de la categoría. En el caso de modificación, se podrá añadir subcategorías al mismo con el botón *Add subcategory* (ver 4.4.5.1.1). El botón *Back* retorna a la lista de categorías.

4.4.5.1.1 *Add/update field subcategory* (Añadir/modificar subcategoría)

Permite ingresar o modificar una subcategoría de campos de estudio.

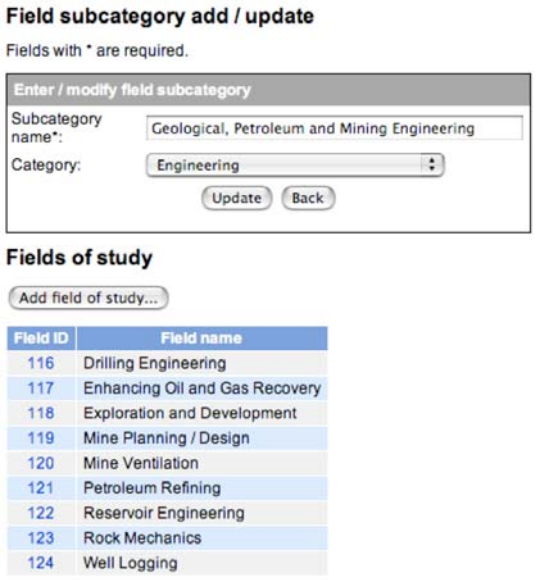


Figura 4.48 – Formulario de subcategoría con sus respectivos campos de estudio

En el caso de ingreso de subcategoría, solamente se pedirá el nombre de la subcategoría, la lista con la categoría padre será automáticamente seleccionada, aunque puede ser cambiada si se desea. En el caso de modificación, se podrá añadir campos de estudio a la misma con el botón *Add field of study* (ver 4.4.5.1.1.1). El botón *Back* retorna al formulario de la categoría padre.

4.4.5.1.1.1 *Add/update field of study* (Añadir/modificar campo de estudio)

Permite ingresar o modificar un campo de estudio.

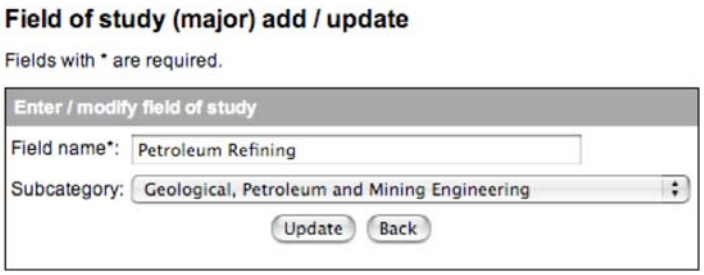


Figura 4.49 – Formulario de campo de estudio

Se debe ingresar el nombre del campo de estudio. La lista de la subcategoría padre será automáticamente seleccionada, aunque se puede cambiar si se desea. El botón *Back* retorna al formulario de la subcategoría padre.

5. Salida del sistema (*Logoff*)

La forma más segura de salir del sistema es mediante la opción de *Logoff*, que se encuentra en la parte derecha de la barra del menú principal, o como un vínculo (*link*) en la página principal que dice "*Click here to logoff*". Si se cierra el *Web Browser* también se realizará un *logoff* del sistema.

6. Glosario

- **Administrador del sistema:** usuario que posee acceso total a la aplicación, no posee ningún tipo de restricción.
- **Agencia:** institución, generalmente del gobierno de los Estados Unidos, que auspicia los programas académicos ofrecidos por la Comisión Fulbright.
- **Aplicante:** cualquier persona inscrita en la Comisión Fulbright, que todavía no ha sido escogida para participar en un programa académico.

- **Autenticación:** proceso que todos los usuarios del sistema deben realizar para ingresar a la aplicación. Para autenticar, se debe proveer el nombre de usuario (ver *Username*) y la contraseña (ver *Password*).
- **Becario:** persona inscrita en la Comisión Fulbright, y que va a participar o ha participado en un programa académico.
- **Campo de estudio:** área del conocimiento (carrera), que se debe especificar cuando se aplica a un programa académico.
- **Concesión económica:** cantidad de dinero categorizada que la Comisión Fulbright desembolsa a un becario para pagar los diferentes gastos del programa.
- **Login:** proceso de Autenticación (ver *Autenticación*).
- **Logoff:** proceso por el cual el usuario abandona el sistema.
- **Navegador de la Web:** programa que permite navegar en el World Wide Web de la red Internet, y muestra la información en forma de *páginas Web*.
- **Nivel de acceso:** categoría que se le asigna a un usuario, y que le impone restricciones en cuanto al uso de funciones dentro del sistema.
- **Password:** clave que se utiliza junto con el nombre de usuario (*Username*) para acceder al sistema al momento de la autenticación (ver *Autenticación*).
- **Perfil Académico:** lista de universidades y/o institutos por los cuales el aplicante/becario ha cursado antes o después de aplicar/participar de un programa académico de la Fulbright.
- **Plataforma:** sistema operativo de la computadora, como por ejemplo Microsoft Windows, Apple Mac OS X, Linux, Unix, etc.
- **Pre-selección:** proceso por el cual se escoge a aquellos aplicantes que cumplan con ciertos requisitos para ser entrevistados por un grupo de panelistas.

- **Programa Académico:** cualquiera de los programas de estudios superiores ofrecidos por la Comisión Fulbright, y a los cuales el público puede aplicar.
- **Programa Pre-académico:** cursos que el becario debe seguir antes de iniciar formalmente el programa académico. Son asignables solamente a becarios ya calificados a un programa.
- **Puerto:** número dentro de una dirección (URL) que indica a qué servicio el usuario desea conectarse en el servidor.
- **Reporte:** informe impreso de los datos al interior de la aplicación.
- **Selección final:** proceso por el cual los aplicantes preseleccionados (ver *Pre-selección*), después de haber sido entrevistados por los panelistas, se determina si califican como becarios o no.
- **URL:** abreviación de *Uniform Resource Locator* (Localizador Uniforme de Recursos), generalmente una dirección para acceder a una página o aplicación Web.
- **Username:** nombre de usuario que se utiliza junto con la clave (*Password*) para acceder al sistema al momento de la autenticación (ver *Autenticación*)
- **Web Browser:** navegador de la Web (ver *Navegador de la Web*).