

UNIVERSIDAD DE BUENOS AIRES  
FACULTAD DE INGENIERÍA

75.99 Trabajo Profesional  
Ingeniería en Informática

wIDE  
Web Integrated Development Environment

**Tutora: Rosa Graciela Wachenchauzer**  
**Fecha de presentación: 25 de febrero de 2011**

**Gastón Kleiman**  
**Padrón: 84.446**



# Índice general

<b>1</b>	<b>Visión</b>	<b>1</b>
1.1.	Objetivo . . . . .	1
1.2.	Alcance . . . . .	2
1.3.	Equipo de trabajo . . . . .	3
<b>2</b>	<b>Planificación</b>	<b>5</b>
2.1.	Metodología de desarrollo . . . . .	5
2.2.	Plan de comunicación . . . . .	6
2.3.	Administración de cambios . . . . .	7
2.4.	Administración de la configuración . . . . .	8
2.5.	Administración de la calidad . . . . .	8
2.6.	Herramientas de desarrollo . . . . .	8
2.7.	Entregables . . . . .	9
<b>3</b>	<b>Requerimientos</b>	<b>11</b>
3.1.	Requerimientos funcionales . . . . .	12
3.2.	Requerimientos no funcionales . . . . .	14
3.3.	Grupos funcionales . . . . .	16
<b>4</b>	<b>Cronograma</b>	<b>19</b>
4.1.	Contenido de las iteraciones . . . . .	19
4.2.	Duración planificada de las iteraciones . . . . .	25
<b>5</b>	<b>Especificación de la API</b>	<b>27</b>
5.1.	Tipos de recursos . . . . .	28
5.2.	Proyectos . . . . .	28
5.3.	Repositorios . . . . .	30
5.4.	Entradas . . . . .	31
5.5.	Protección contra CSRF (Cross-Site Request Forgery) . . . . .	32
<b>6</b>	<b>Detalles de implementación del cliente</b>	<b>33</b>

6.1. Arquitectura . . . . .	33
6.2. Convenciones . . . . .	35
6.3. Detalles de implementación . . . . .	36
6.4. Utilización de componentes de terceros . . . . .	42
<b>7 Detalles de implementación del servidor</b>	<b>43</b>
7.1. Arquitectura global . . . . .	44
7.2. Servidor Web . . . . .	45
7.3. Tareas asincrónicas . . . . .	52
7.4. Panel de administración . . . . .	53
7.5. Proxy para acceso externo a los repositorios . . . . .	54
<b>8 Instalación</b>	<b>55</b>
8.1. Dependencias . . . . .	55
8.2. Supuestos . . . . .	56
8.3. Procedimiento de instalación . . . . .	56
<b>9 Administración y mantenimiento del sistema</b>	<b>61</b>
9.1. Acceso al panel web de administración . . . . .	62
9.2. Creación de nuevos administradores . . . . .	62
9.3. Activación de nuevos usuarios . . . . .	64
9.4. Creación de un nuevo tipo de proyectos . . . . .	65
<b>10 Manual del usuario</b>	<b>69</b>
10.1. Usuarios . . . . .	70
10.2. Listado de proyectos . . . . .	71
10.3. Entorno de desarrollo . . . . .	74
<b>11 Conclusiones</b>	<b>83</b>
<b>Bibliografía</b>	<b>85</b>
<b>Listado de código</b>	<b>89</b>
<b>Carta de aprobación del cliente</b>	<b>91</b>

# Índice de tablas

1.2. Roles, tareas y personas responsables. . . . .	3
3.1. Grupos de funcionalidades . . . . .	17
4.1. Contenido planificado para cada iteración . . . . .	24
4.3. Duración estimada de cada iteración. . . . .	25
7.1. Mapeo de las rutas de la API a los controladores. . . . .	51
10.1. Iconos y sus correspondientes estados . . . . .	77

# Índice de figuras

6.1. Módulos y componentes en la pantalla de edición . . . . .	34
6.2. Diagrama de secuencia de la operación “pull” . . . . .	42
7.1. Diagrama de relaciones entre componentes. . . . .	44
7.2. Diagrama con atributos y relaciones entre modelos. . . . .	47
7.3. Diagrama de clases (modelos). . . . .	48
7.4. Diagrama de clases (controladores). . . . .	50
9.1. Primer paso para la creación del primer administrador. . . . .	62
9.2. Segundo paso para la creación del primer administrador. . . . .	63
9.3. Pantalla principal del panel de administración. . . . .	63
9.4. Creación y activación de un nuevo administrador. . . . .	64
9.5. Activación de un usuario. . . . .	65
9.6. Creación de un nuevo tipo de proyecto. . . . .	66
10.1. Formulario de registro de usuarios . . . . .	71
10.2. Formulario de configuración de la cuenta de un usuario . . . . .	72
10.3. Listado de proyectos . . . . .	72
10.4. Formulario para la creación de un nuevo proyecto . . . . .	73

10.5. Entorno de desarrollo . . . . .	75
10.6. Formulario de configuración de un proyecto . . . . .	76
10.7. Menú contextual . . . . .	77
10.8. Diálogo de “commit” . . . . .	78
10.9. Historial de cambios (changesets) . . . . .	79
10.10 Vista detalle de un cambio . . . . .	80

# CAPÍTULO 1

## Visión

### Índice

1.1. Objetivo . . . . .	1
1.2. Alcance . . . . .	2
1.3. Equipo de trabajo . . . . .	3

### 1.1. Objetivo

Actualmente los integrantes del Club de Robótica de FIUBA [1] utilizan para algunos de sus proyectos un entorno de desarrollo (IDE) web llamado “mbed” [2].

Este entorno permite utilizando tan sólo un navegador web, sin necesidad de instalar software adicional:

- Crear nuevos proyectos.
- Editar archivos dentro de ellos.
- Compilarlos, pudiendo seleccionar la arquitectura para la cual se quiere compilar el proyecto.

Este innovador sistema presenta varias limitaciones, entre ellas:

- Es un producto comercial sólo accesible para quien cuente con un microcontrolador “mbed”.

- Al ser un producto privativo de código cerrado, no es posible realizarle modificaciones.
- No cuenta con ninguna clase de versionado de archivos.
- Sólo es posible acceder y/o manipular los archivos a través de la interfaz web.
- No presenta ninguna forma práctica de importar y exportar repositorios de código.
- No permite que varios usuarios trabajen sobre un mismo proyecto.
- Las opciones de arquitecturas para las que se pueden compilar los programas se limita a los dos productos comercializados por la empresa.

El objetivo de este trabajo profesional es el desarrollo de un entorno de desarrollo web abierto y fácilmente extensible, que supere estas limitaciones.

## 1.2. Alcance

Se requiere un entorno de desarrollo web que cuente con un repositorio versionado por cada proyecto creado.

Desde el sistema se podrán editar archivos a través de un editor web, y guardar los cambios en el repositorio. También debe permitir sincronizar el repositorio almacenado en el sistema, aplicando los cambios realizados en otro repositorio (realizar “pull” de otro repositorio).

Además de poder crear, modificar, mover y borrar archivos y directorios, debe ser posible ver el historial de cambios de un archivo, y volver a una versión anterior.

Para el versionado de los proyectos se definirá una interfaz; de forma tal que se puedan agregar fácilmente nuevos SCM. Dado que el Club de Robótica trabaja con Mercurial [3], se lo utilizará como implementación de referencia.

Es deseable poder acceder a los repositorios utilizando el cliente del sistema SCM, de forma independiente a la interfaz web, pero utilizando su base de datos para la autenticación y autorización de usuarios.

Debe ser posible obtener un archivo binario con el resultado de compilar el proyecto para la arquitectura elegida por el usuario. El módulo de compilación debe ser fácilmente extensible y permitir agregar fácilmente nuevas arquitecturas.



Actualmente el Club de Robótica utiliza Redmine [4] para el seguimiento de sus proyectos, por lo que se contemplará la posibilidad de integrar parte de la funcionalidad del entorno con este sistema.

### 1.3. Equipo de trabajo

En la tabla 1.2 se listan los integrantes del equipo de trabajo, y se especifican las tareas de las que es responsable cada uno.

Rol	Responsable	Tareas
Tutora	Rosa Wachenchauzer	<ul style="list-style-type: none"><li>■ Tutoría del trabajo</li></ul>
Cliente	Lucas Chiesa	<ul style="list-style-type: none"><li>■ Definición de requerimientos.</li><li>■ Pruebas de aceptación.</li></ul>
Desarrollador	Gastón Kleiman	<ul style="list-style-type: none"><li>■ Análisis.</li><li>■ Diseño.</li><li>■ Desarrollo de la aplicación y pruebas.</li><li>■ Elaboración de la documentación.</li><li>■ Instalación en el servidor del cliente.</li></ul>

**Tabla 1.2:** Roles, tareas y personas responsables.



# CAPÍTULO 2

## Planificación

### Índice

2.1. Metodología de desarrollo . . . . .	5
2.2. Plan de comunicación . . . . .	6
2.3. Administración de cambios . . . . .	7
2.4. Administración de la configuración . . . . .	8
2.5. Administración de la calidad . . . . .	8
2.6. Herramientas de desarrollo . . . . .	8
2.7. Entregables . . . . .	9

### 2.1. Metodología de desarrollo

Dado que el equipo de desarrollo está conformado por una sola persona y que al iniciar el proyecto los requerimientos no están 100 % definidos, se adoptará una metodología de desarrollo ágil caracterizada por iteraciones cortas.

Cada iteración incluirá el análisis, diseño, desarrollo, prueba y despliegue de la funcionalidad planificada.

Para maximizar el retorno de inversión del cliente y reducir los riesgos del proyecto, se implementarán en primer lugar las funcionalidades de mayor valor.

Si bien para la planificación de las iteraciones se definirá una duración óptima de dos semanas, este valor no será rígido.

Ya que la obtención temprana de feedback y del compromiso del cliente disminuyen el riesgo del proyecto, será posible disminuir la duración de una iteración cuando esto resulte beneficioso para lograr estos objetivos, y no implique sacrificar la coherencia del conjunto de funcionalidades a implementar.

En cambio se la podrá aumentar en caso de que sea necesario más tiempo para obtener un conjunto coherente de funcionalidades.

Se contará con tres ambientes distintos:

1. Desarrollo.
2. Prueba.
3. Producción.

Los ambientes de desarrollo y de pruebas estarán bajo el control del desarrollador. El de producción se encontrará en un servidor perteneciente a el “Club de Robótica”, y será administrado por sus integrantes.

Una iteración se considerará lista para presentar al cliente cuando se hayan completado en su totalidad todos los entregables planificados para la misma, pase todas las pruebas unitarias desarrolladas al momento y se ejecute exitosamente en el ambiente de prueba.

Al terminar una iteración se la mostrará al cliente, quién deberá efectuar las pruebas de aceptación. No se continuará con la implementación de funcionalidad planificada para la siguiente iteración sin haber obtenido antes la aprobación del cliente.

Durante una iteración se implementará la funcionalidad planificada para la misma y se corregirán los defectos encontrados por el cliente en la entrega de la iteración anterior.

En la primera iteración se hará una toma detallada de requerimientos, los cuales serán volcados en forma de *User Stories* en el capítulo 3.

A partir de la priorización de requerimientos por parte del cliente, se planificarán el contenido del resto de las iteraciones, elaborándose un cronograma con el contenido y duración tentativos de cada una.

Se utilizará un sistema de administración de proyectos para realizar el seguimiento de defectos y pedidos de cambios.

## 2.2. Plan de comunicación

Una comunicación fluida y frecuente con el cliente es esencial para lograr un mayor involucramiento en el desarrollo del proyecto, y para obtener feedback que permita reducir riesgos y evitar el retrabajo.

El “Club de Robótica” se reúne todos los viernes, por lo que para facilitar la asistencia de todos los interesados, los encuentros se realizarán durante las reuniones del club.

Se planificarán dos tipos de reuniones:

**Reunión de avance:** Tendrán una frecuencia semanal. Durante las mismas se mostrará al cliente el progreso de la implementación, se relevarán nuevos requerimientos, se resolverán dudas, y se evaluarán los posibles pedidos de cambio que puedan surgir.

**Presentación de entregables:** Se realizarán al finalizar una iteración. Previo a estas reuniones se tendrá instalado en el ambiente de prueba el resultado de la última iteración.

El objetivo será la obtención de la aceptación del cliente. Por lo tanto se entregará al cliente la última versión de los entregables, y se lo guiará para que realice las pruebas de aceptación.

## 2.3. Administración de cambios

Se utilizará un sistema de administración de proyectos para el registro y seguimiento de los pedidos de cambios.

A continuación se detallarán el proceso de incorporación de pedidos de cambio.

1. El usuario agrega en el sistema de administración un pedido de cambio priorizado.
2. El desarrollador evalúa el impacto del cambio, y realiza una estimación del tiempo de implementación.
3. El usuario y el desarrollador evalúan en conjunto la posibilidad de la implementación del cambio, teniendo en cuenta la prioridad asignada y el impacto y tiempo estimados.

Dado que se cuenta con un tiempo limitado para el desarrollo del proyecto, se contemplará la posibilidad de que la incorporación de un cambio implique no implementar alguna funcionalidad planificada anteriormente.

## 2.4. Administración de la configuración

Como se describió anteriormente, se contará con los siguientes ambientes:

**Desarrollo:** será administrado por el desarrollador, y es el que se utilizará durante la duración de las iteraciones.

Se planean utilizar los siguientes sistemas operativos para el desarrollo:

- GNU/Linux.
- OS X.

**Prueba:** será administrado por el desarrollador, y será sobre el que se desplegará el resultado de cada iteración para que el usuario pueda realizar las pruebas de aceptación.

Correrá sobre el mismo sistema operativo que utiliza el servidor del cliente en el que se instalará el sistema finalizado.

**Producción:** será administrado por un pasante del LABI, y se instalará en este las versiones entregables del sistema.

El código de la aplicación será versionado utilizando `git` [5].

## 2.5. Administración de la calidad

Se escribirán pruebas unitarias para garantizar el correcto funcionamiento de la funcionalidad crítica del servidor.

La realización de pruebas automáticas de la interfaz del usuario web se verá dificultada por la inclusión de gran cantidad de componentes desarrollados utilizando `JavaScript`, producto del gran dinamismo requerido.

Por lo que tanto las pruebas de la interfaz del usuario, como las de integración, se ejecutarán de manera manual al terminar cada iteración.

Las pruebas de aceptación serán llevadas a cabo por el cliente.

## 2.6. Herramientas de desarrollo

El sistema será escrito en los lenguajes `Ruby` y `JavaScript`.

En el ambiente de desarrollo se utilizará `git` [5] para el control de versiones del código fuente, y `Vim` [6] para la edición del mismo.

Los informes se redactarán utilizando `LATEX` [7].

El cliente web será probado utilizando `Firefox 3.6` y `Google Chrome`.

En el entorno de desarrollo el servidor será ejecutado bajo `GNU/Linux` y bajo `Mac OS X`.

El cliente utiliza `Ubuntu` en el ambiente de producción, por lo que se instalará la misma distribución en el ambiente de prueba.

Se utilizarán las siguientes tecnologías:

- `Ruby on Rails 3` [8] para el desarrollo del servidor.
- `RSpec` [9] para escribir pruebas unitarias y de integración.
- `jQuery` [10] como biblioteca base del código JavaScript del cliente web.
- `jQuery UI` [11] como biblioteca base para elementos gráficos (tales como tabs, árboles, etc.).
- `Typus` [12] como framework para el desarrollo del panel de administración.
- `Devise` [13] como framework para la autenticación de usuarios.
- Algún editor de código hecho en JavaScript, dado que hay una gran variedad [14], se evaluarán y probarán distintos, para encontrar el que mejor se integre con el sistema.
- Algún framework para el encolado de trabajos a realizar de forma asincrónica (algunas opciones que se evaluarán son `Delayed_job` [40], `Resque` [15] y `Beanstalkd` [16]).
- Se analizará la opción de utilizar `mercurial-server` [17] para el control de acceso externo a los repositorios. En caso de utilizarlo, será necesario modificarlo para obtener los permisos y credenciales de acceso de la base de datos del sistema a desarrollar.

Todos los frameworks, componentes y librerías que se utilizarán son libres.

## 2.7. Entregables

En la tabla 4.1 se detallan los entregables que se obtendrán como resultado de cada iteración.

Al finalizar el proyecto se contará con el sistema implementado y desplegado en el ambiente de producción, y con una carpeta con el siguiente contenido:

1. Visión.
2. Planificación.
3. Requerimientos.
4. Contenido planificado de las iteraciones.
5. Cronograma.
6. Definición de la API.
7. Detalles de implementación del cliente web.
8. Detalles de implementación del servidor.
9. Manual de instalación.
10. Manual administración y mantenimiento del servidor.
11. Manual del usuario.
12. Conclusiones.
13. Presentación.



# CAPÍTULO 3

## Requerimientos

### Índice

3.1. Requerimientos funcionales . . . . .	12
Usuarios . . . . .	12
Proyectos . . . . .	12
Control de versiones . . . . .	13
Administración . . . . .	14
3.2. Requerimientos no funcionales . . . . .	14
Hardware . . . . .	14
Software . . . . .	15
Performance . . . . .	15
Usabilidad . . . . .	15
3.3. Grupos funcionales . . . . .	16

### Índice de tablas

3.1. Grupos de funcionalidades . . . . .	17
--	----

En este capítulo se presentan en forma de *User Stories* los requerimientos relevados. Estas User Stories se dividieron en grupos funcionales, los cuales fueron priorizados por el cliente.

### 3.1. Requerimientos funcionales

Se requiere un entorno de desarrollo web que cuente con un repositorio versionado por cada proyecto creado.

Desde el sistema se podrán editar archivos a través de un editor web, y guardar los cambios en el repositorio. También debe permitir sincronizar el repositorio almacenado en el sistema, aplicando los cambios realizados en otro repositorio (realizar “pull” de otro repositorio).

Además de poder crear, modificar y borrar archivos y directorios, debe ser posible ver el historial de cambios de un archivo, y volver a una versión anterior.

Para el versionado de los proyectos se definirá una interfaz; de forma tal que se puedan agregar fácilmente nuevos SCM.

Dado que el Club de Robótica trabaja con Mercurial [3], se incluirá una implementación de referencia con este SCM.

Es deseable que se pueda acceder a los repositorios de forma externa, independientemente del sistema web.

Debe ser posible obtener un archivo binario con el resultado de compilar el proyecto para la arquitectura elegida por el usuario. La incorporación de nuevas arquitecturas de compilación debe poder hacerse fácilmente desde la interfaz de administración web.

Actualmente el Club de Robótica utiliza Redmine [4] para el seguimiento de sus proyectos, por lo que se contemplará la posibilidad de integrar parte de la funcionalidad del entorno con este sistema.

Se requiere un panel web de administración que permita realizar la alta, baja y modificación de administradores, usuarios, proyectos y tipos de proyectos.

En las siguientes secciones se listan las User Stories correspondientes a los requerimientos relevados.

#### Usuarios

1. No estando aún registrado, debo poder crear un nuevo usuario.
2. No habiendo ingresado al sistema debo poder recuperar mi usuario en caso de haber olvidado mi contraseña.

#### Proyectos

Como un usuario debo poder:

3. Crear un proyecto:

- a) Clonando otro repositorio.
  - b) A partir de plantillas.
- 4. Borrar un proyecto propio.
- 5. Explorar mediante un árbol los archivos y carpetas pertenecientes a un proyecto propio.
- 6. Agregar archivos a un proyecto propio.
- 7. Borrar archivos de un proyecto propio.
- 8. Mover archivos en un proyecto propio, haciendo Drag&Drop en el árbol del proyecto.
- 9. Editar archivos de un proyecto propio, utilizando un componente que brinde facilidades para la edición de código fuente.
- 10. Seleccionar una plataforma para la cual se compilará el proyecto.
- 11. Obtener un archivo binario compilado para la plataforma seleccionada. En caso de haber errores de compilación, debo poder verlos de forma amigable.
- 12. Agregar/borrar/modificar constantes que serán definidas en tiempo de compilación del proyecto.
- 13. Agregar/eliminar colaboradores a un proyecto, los cuales tendrán acceso de escritura externa al repositorio del proyecto (es decir a través de clientes SCM, y no desde el navegador web).
- 14. Indicar que un repositorio es público, lo cual significa que se permite a cualquier persona, cuente con un usuario registrado en el sistema o no, el acceso de lectura al repositorio mediante el cliente de SCM.

## Control de versiones

Como un usuario debo poder:

- 15. Hacer “pull”, importando al repositorio web cambios hechos en otro externo. En caso de haber conflictos, se los debe poder resolver a través de la interfaz web.
- 16. Hacer “commit”, guardando los cambios realizados.

17. Hacer “revert”, borrando todo cambio realizado a partir del último “commit”.
18. Indicar que un archivo debe versionarse.
19. Indicar que un archivo debe dejar de estar versionado.
20. Deshacer todo cambio hecho a un archivo desde el último “commit”.
21. Ver el historial de modificaciones de un archivo.
22. Cambiar el contenido de un archivo al de una revisión anterior.
23. Elegir qué archivos se incluirán en un “commit”.
24. Antes de hacer un ‘commit’, visualizar los cambios realizados desde la última revisión.
25. Acceder a los repositorios de forma externa (sin utilizar el cliente web).

Cuando se menciona “un archivo”, se hace referencia a un archivo perteneciente a alguno de los proyectos del usuario.

## Administración

Como administrador, debo poder:

26. Crear/modificar/borrar usuarios.
27. Crear/modificar/borrar proyectos.
28. Crear/modificar/borrar la plantilla de los “Makefile” utilizados al realizar la compilación para cada una de las plataformas.
29. Crear/modificar/borrar plantillas utilizadas para la creación de proyectos.

## 3.2. Requerimientos no funcionales

### Hardware

El hardware utilizado para el servidor deberá ser capaz de correr alguno de los siguiente servidores web: Apache 2 [18], nginx [23] o lighthttpd [19],

y de ejecutar el compilador correspondiente a cada una de las arquitecturas que se desee soportar.

La cantidad de almacenamiento secundario necesario será directamente proporcional al tamaño de los repositorios de los usuarios.

## Software

El sistema podrá ser utilizado desde cualquier computadora capaz de ejecutar un navegador web, sin necesidad de instalar ningún software adicional por parte de los usuarios.

El navegador deberá tener soporte para **JavaScript** y **Cookies**. Por lo tanto sólo se garantizará el completo funcionamiento bajo navegadores modernos, tales como **Firefox** [20] versión igual o superior a la 3.6.6, **Google Chrome** [21] y **Safari 5.0** [22].

El servidor funcionará bajo **GNU/Linux**. Se utilizará un servidor web compatible con **Ruby on Rails 3** [8], tales como **Apache 2** [18], **nginx** [23] o **lighttpd** [19].

La persistencia se deberá programar utilizando **ANSI SQL**, de manera tal que no se requiera el uso de un motor de base de datos específico. El cliente expresó ya estar utilizando **SQLite** [24], y se intentará que el sistema sea compatible con el mismo, evitando así que el cliente tenga que mantener otro motor.

## Performance

Se prevé que la aplicación será utilizada por un máximo de 15 usuarios simultáneamente.

Dentro del servidor, las tareas de larga duración deberán ejecutarse de forma asincrónica, de manera tal que estas no afecten el tiempo de respuesta del servidor web.

## Usabilidad

Todas las páginas del sistema deben tener un estilo consistente y presentar una interfaz amigable.

La página que permita explorar, modificar, y compilar un proyecto deberá contar con los siguientes paneles redimensionables:

**Panel de exploración:** contará con un árbol de directorios.

**Panel de edición:** contará con un tab para cada archivo abierto.

**Panel de resultados de compilación:** será una tabla con los mensajes de error y/o advertencia producidos por el compilador.

Tanto los errores como el progreso de todas las actividades deberán ser comunicados al usuario mediante un sistema de notificaciones consistente.

### 3.3. Grupos funcionales

En conjunto con el cliente se dividieron los User Stories en grupos de funcionalidades. Luego se le pidió que asigne una prioridad a cada uno de los grupos.

En la tabla 3.1 se listan estos grupos en orden de prioridad decreciente, detallando los User Stories que incluye cada uno.

Id.	Descripción del grupo de funcionalidades	User Stories
GF1	Exploración y manejo de archivos de los proyectos a través de un árbol de directorios utilizando AJAX.	5–8
GF2	Edición de archivos a través de un componente que soporte syntax highlighting y otras facilidades para la edición de código fuente.	9
GF3	Autenticación de usuarios y manejo de proyectos por usuario.	1, 2, 3, 4
GF4	Versionado de los proyectos almacenándolos en repositorios Mercurial.	15–20
GF5	Compilación para distintas arquitecturas.	10–12
GF6	Ejecución asincrónica de tareas de larga duración, para mejorar la velocidad de respuesta y usabilidad del sistema.	Sección 3.2
GF7	Panel de administración.	26–28
GF8	Interfaz con paneles redimensionables, tabs y notificaciones.	Sección 3.2

Id.	Descripción del grupo de funcionalidades	User Stories
GF9	Funcionalidad avanzada de control de versiones. Por ejemplo: <ul style="list-style-type: none"><li>■ Visualización e interacción con el historial de versiones, tanto de un archivo como del repositorio completo.</li><li>■ Visualización de cambios realizados antes de hacer un “commit”.</li><li>■ Selección de archivos a incluir en un “commit”.</li></ul>	21–24
GF10	Acceso externo (sin necesidad de utilizar la aplicación web) a los repositorios.	25
GF11	Control de acceso a los repositorios, pudiendo definir qué usuarios tendrán acceso a cada uno.	13, 14
GF12	Creación de proyectos a partir de plantillas	3b, 29

**Tabla 3.1:** División de User Stories en grupos de funcionalidades, ordenados por prioridad decreciente.





# CAPÍTULO 4

## Cronograma

### Índice

4.1. Contenido de las iteraciones . . . . .	19
4.2. Duración planificada de las iteraciones . . . . .	25

### Índice de tablas

4.1. Contenido planificado para cada iteración . . . . .	24
4.3. Duración estimada de cada iteración. . . . .	25

## 4.1. Contenido de las iteraciones

La tabla 4.1 detalla el contenido de cada iteración, y los entregables resultantes.

La definición del contenido de cada una se realizó teniendo en cuenta la prioridad, la estimación del tiempo requerido para la implementación y las dependencias de cada uno de los grupos de funcionalidades definidos en el capítulo 3.

Los entregables se acumularán al ir avanzando las iteraciones, y serán actualizados y refinados cuando sea necesario.

Se planifican dos entregas que serán desplegadas en el ambiente de producción; el resto serán instaladas en el ambiente de pruebas.

El primer despliegue en producción se realizará apenas finalizada la implementación de la funcionalidad actualmente brindada por `mbed`, más un

soporte básico de control de versiones. El segundo al finalizar la implementación de toda la funcionalidad.

Iteración	Contenido	Entregables
1	<ul style="list-style-type: none"> <li>■ Relevamiento y análisis de requerimientos.</li> </ul>	Documentos: <ul style="list-style-type: none"> <li>■ Visión.</li> <li>■ Requerimientos.</li> </ul>
2	<ul style="list-style-type: none"> <li>■ Planificación.</li> <li>■ Diseño de la arquitectura del servidor.</li> <li>■ Diseño de la arquitectura del cliente web.</li> <li>■ Instalación y configuración de los ambientes de desarrollo y prueba.</li> </ul>	Documentos: <ul style="list-style-type: none"> <li>■ Ídem</li> <li>■ Cronograma.</li> <li>■ Arquitectura del servidor.</li> <li>■ Arquitectura del cliente web.</li> </ul> Ambientes: <ul style="list-style-type: none"> <li>■ Desarrollo.</li> <li>■ Prueba.</li> </ul>
Tareas, funcionalidades a implementar, y entregables planificados por iteración.		

Iteración	Contenido	Entregables
3	<ul style="list-style-type: none"> <li>■ Exploración y manejo de archivos. (GF1)</li> </ul>	Documentos: <ul style="list-style-type: none"> <li>■ Ídem</li> <li>■ Manual técnico del servidor.</li> <li>■ Manual técnico del cliente web.</li> </ul> Código: <ul style="list-style-type: none"> <li>■ Servidor.</li> <li>■ Cliente.</li> </ul> Ambientes: ídem.
4	<ul style="list-style-type: none"> <li>■ Investigación sobre componentes HTML/JavaScript de edición de código fuente disponibles. (GF2)</li> <li>■ Selección e integración del componente de edición. (GF2)</li> </ul>	Documentos, ambientes, código: ídem
5	<ul style="list-style-type: none"> <li>■ Autentificación de usuarios y manejo de proyectos por usuario. (GF3)</li> <li>■ Versionado de los proyectos. (GF4)</li> </ul>	Documentos, ambientes, código: ídem
6	<ul style="list-style-type: none"> <li>■ Compilación de los proyectos. (GF5)</li> <li>■ Ejecución asincrónica de tareas de larga duración. (GF6)</li> </ul>	Documentos, ambientes, código: ídem

Tareas, funcionalidades a implementar, y entregables planificados por iteración.

Iteración	Contenido	Entregables
7	<ul style="list-style-type: none"> <li>■ Panel de administración. (GF7)</li> <li>■ Interfaz de usuario con paneles redimensionables, tabs y notificaciones. (GF8)</li> </ul>	<p>Documentos:</p> <ul style="list-style-type: none"> <li>■ Ídem</li> <li>■ Manual del panel de administración.</li> <li>■ Descripción de la interfaz de usuario.</li> <li>■ Manual de instalación, administración y mantenimiento del servidor.</li> </ul> <p>Ambientes:</p> <ul style="list-style-type: none"> <li>■ Ídem.</li> <li>■ Producción</li> </ul> <p>Código: ídem. Sistema desplegado en producción.</p>
8	<ul style="list-style-type: none"> <li>■ Funcionalidad avanzada de control de versiones. (GF9)</li> </ul>	<p>Documentos, ambientes, código: ídem</p>

Tareas, funcionalidades a implementar, y entregables planificados por iteración.

Iteración	Contenido	Entregables
9	<ul style="list-style-type: none"> <li>▪ Acceso externo a los repositorios. (GF10)</li> <li>▪ Control de acceso a los repositorios. (GF11)</li> <li>▪ Creación de proyectos a partir de plantillas. (GF12)</li> </ul>	<p>Documentos:</p> <ul style="list-style-type: none"> <li>▪ Ídem</li> <li>▪ Documentación técnica de configuración de repositorios y control de acceso externo.</li> </ul> <p>Código: ídem. Sistema desplegado en producción.</p>
10	<ul style="list-style-type: none"> <li>▪ Optimización del sistema.</li> <li>▪ Corrección de defectos e implementación de cambios pendientes.</li> <li>▪ Documentación final.</li> </ul>	<p>Documentos:</p> <ul style="list-style-type: none"> <li>▪ Ídem</li> <li>▪ Presentación.</li> <li>▪ Informe final.</li> </ul> <p>Ambientes, código: ídem. Sistema final desplegado en producción.</p>

**Tabla 4.1:** Tareas, funcionalidades a implementar, y entregables planificados por iteración.

## 4.2. Duración planificada de las iteraciones

En la tabla 4.3 se encuentra una estimación de la duración de cada iteración, suponiendo que se trabajarán 20 horas hombre por semana.

Iteración	Duración
1	1 semanas
2	2 semanas
3	2 semanas
4	2 semanas
5	3 semanas
6	2 semanas
7	2 semanas
8	2 semanas
9	3 semanas
10	3 semanas
Semanas:	22
Horas:	440

**Tabla 4.3:** Duración estimada de cada iteración.





## Especificación de la API

### Índice

5.1. Tipos de recursos . . . . .	28
5.2. Proyectos . . . . .	28
Operaciones . . . . .	28
5.3. Repositorios . . . . .	30
Operaciones . . . . .	30
5.4. Entradas . . . . .	31
Operaciones . . . . .	31
5.5. Protección contra CSRF (Cross-Site Request Forgery) .	32

En este capítulo se especifica la API REST [37] utilizada por el cliente web para manipular los proyectos y sus repositorios utilizando tecnología AJAX.

A pesar de que todas las rutas utilizadas por el servidor siguen las convenciones utilizadas en APIs REST, en este capítulo sólo se describirán las que son utilizadas directamente por el código JavaScript del cliente, y no a través de formularios.

A menos que se indique lo contrario, el cuerpo de las respuestas del servidor consistirá de un diccionario en formato JSON [43].

Este diccionario contendrá el resultado de la operación correspondiente a la petición realizada por el cliente, y una clave llamada **success**. Esta tendrá como valor 1 en caso de que la operación haya sido exitosa, y 0 en caso de error.

## 5.1. Tipos de recursos

La API definida en este capítulo sirve para manipular tres tipos de recursos distintos:

- Proyectos.
- Repositorios.
- Entradas dentro de los repositorios.

A continuación se definirán cada recurso y las operaciones que se pueden realizar sobre cada uno.

## 5.2. Proyectos

Un proyecto está asociado a un usuario, un tipo de proyecto, y un repositorio de código versionado.

El cliente puede solicitar al servidor que compile un proyecto, y luego obtener los mensajes y el archivo binario producto del proceso de compilación.

También es posible obtener el `Makefile` utilizado por el servidor para compilar el proyecto.

En la siguiente tabla se listan las operaciones aplicables a un proyecto, y su correspondiente ruta. Ninguna de estas toma parámetros.

### Operaciones

Método	Ruta	Operación
POST	/projects/:id/compile	<code>compile</code>
GET	/projects/:id/compiler_output	<code>compiler_output</code>
GET	/projects/:id/binary	<code>binary</code>
GET	/projects/:id/makefile	<code>makefile</code>

### Compilación

La solicitud de compilación se realiza mediante la acción `compile`. Luego el cliente debe realizar *polling* para obtener los mensajes producidos por la compilación, utilizando la acción `compiler_output`. Este proceso es análogo al observado en la figura 6.2.

El resultado de estas dos operaciones es un diccionario.

Al solicitar la compilación, el resultado puede ser por ejemplo:

```
answer = {  
  "success": 1,  
  "compile_status": {  
    "updated_at": 1298100554,  
    "status": "running",  
    "operation": "compile"  
  }  
}
```

Y luego, al solicitar la salida del proceso de compilación, se obtiene:

```
answer = {  
  "success": 1,  
  "compile_status": {  
    "updated_at": 1298100555,  
    "status": "success",  
    "operation": "compile"  
  },  
  "output": [  
    {  
      "resource": "prueba.c",  
      "type": "info",  
      "description": "foo"  
    },  
    {  
      "resource": "prueba.c",  
      "line": 2,  
      "type": "warning",  
      "description": "bar"  
    }  
  ]  
}
```

Para saber cuando se completó el proceso (ya sea con éxito o no), el cliente debe almacenar el *timestamp* devuelto por el servidor, y realizar *polling* hasta que se obtenga uno mayor al guardado.

El estado del proceso puede tomar alguno de los siguientes valores:

**running:** compilando.

**success:** el proyecto fue compilado con éxito, se puede solicitar bajar el binario generado.

**error:** sucedió un error durante la compilación.

Las operaciones `makefile` y `binary` permiten descargar el `Makefile` y el archivo binario generado, respectivamente.

En ambos casos la respuesta del servidor contendrá en el cuerpo el contenido del archivo correspondiente.

### 5.3. Repositorios

Cada proyecto está asociado a un repositorio de código versionado por un sistema de control de versiones (SCM).

A través de esta API es posible consultar el estado del repositorio, y solicitar la ejecución de operaciones al SCM.

#### Operaciones

Método	Ruta	Operación
POST	/projects/:id/repository/revert	<b>revert</b>
POST	/projects/:id/repository/update	<b>update</b>
POST	/projects/:id/repository/commit	<b>commit</b>
POST	/projects/:id/repository/pull	<b>pull</b>
GET	/projects/:id/repository/async_op_status	<b>async_op_status</b>
GET	/projects/:id/repository/summary	<b>summary</b>

#### **commit**

Permite grabar los cambios hechos en el repositorio.

##### Parámetros

Nombre	Tipo	Descripción
<b>files</b>	<b>string[]</b>	Rutas de archivos a incluir en el commit, en caso de estar vacío, el commit incluirá a todos los archivos cambiados.
<b>message</b>	<b>string</b>	Descripción del commit, no puede estar vacío.

#### **revert**

Permite deshacer los cambios hechos desde el último “commit”.

##### Parámetros

Nombre	Tipo	Descripción
<b>files</b>	<b>string[]</b>	Rutas de archivos a restaurar, en caso de estar vacío, se restaurarán todos los archivos cambiados.

#### **update**

Actualiza el contenido del repositorio al correspondiente a la revisión especificada.

Parámetros

Nombre	Tipo	Descripción
<code>revision</code>	<code>integer</code>	Número de revisión

### summary

Devuelve un resumen del estado del repositorio.

La respuesta tiene el siguiente formato:

```
{
  "clean?": true,
  "unresolved?": false,
  "commitable?": false
}
```

### pull

Encola un “pull” desde la URL indicada. Para saber si fue exitoso, el cliente debe realizar *polling* mediante la operación `async_op_status`, como se indica en la figura 6.2.

Parámetros

Nombre	Tipo	Descripción
<code>url</code>	<code>string</code>	URL desde la que se desean importar los cambios

## 5.4. Entradas

Cada repositorio contiene entradas. Una entrada puede ser un archivo o un directorio.

### Operaciones

Las siguientes operaciones sirven para la alta, baja y modificación de las entradas:

Método	Ruta	Operación
GET	/projects/:id/repository/entries	<code>index</code>
GET	/projects/:id/repository/entries/*path	<code>show</code>
POST	/projects/:id/repository/entries/*path	<code>update</code>
PUT	/projects/:id/repository/entries/*path	<code>create</code>
DELETE	/projects/:id/repository/entries/*path	<code>destroy</code>

A continuación se listan las operaciones realizables por el sistema de SCM sobre una entrada:

Método	Ruta	Operación
POST	/projects/:id/repository/entries/*path/add	add
POST	/projects/:id/repository/entries/*path/mark_resolved	mark_resolved
POST	/projects/:id/repository/entries/*path/mark_unresolved	mark_unresolved
POST	/projects/:id/repository/entries/*path/mv	mv
POST	/projects/:id/repository/entries/*path/forget	forget
POST	/projects/:id/repository/entries/*path/revert	revert
GET	/projects/:id/repository/entries/*path/diff	diff

## 5.5. Protección contra CSRF (Cross-Site Request Forgery)

Como medida de protección, esta API requiere que todas las peticiones que se hagan con un método distinto de `GET` incluyan tanto un encabezado llamado `X-CSRF-Token`, como un parámetro llamado `authenticity_token`.

El valor de estos dos campos es el mismo, se extrae de la página de edición del proyecto, y se mantiene constante durante una sesión.

Esto implica que el cliente debe mantener un estado, lo cual contradice los principios de una arquitectura REST.

Sin embargo se tomó la decisión de aplicar este tipo de protección, ya el único consumidor de la API será el cliente web, y la alternativa sería utilizar “API Keys”, lo cual agregaría complejidad innecesaria al mismo.

# CAPÍTULO 6

## Detalles de implementación del cliente

### Índice

6.1. Arquitectura . . . . .	33
6.2. Convenciones . . . . .	35
Nomenclatura . . . . .	35
Estilo de código . . . . .	36
6.3. Detalles de implementación . . . . .	36
Componentes base . . . . .	36
Componentes de la página de edición . . . . .	38
6.4. Utilización de componentes de terceros . . . . .	42

### Índice de figuras

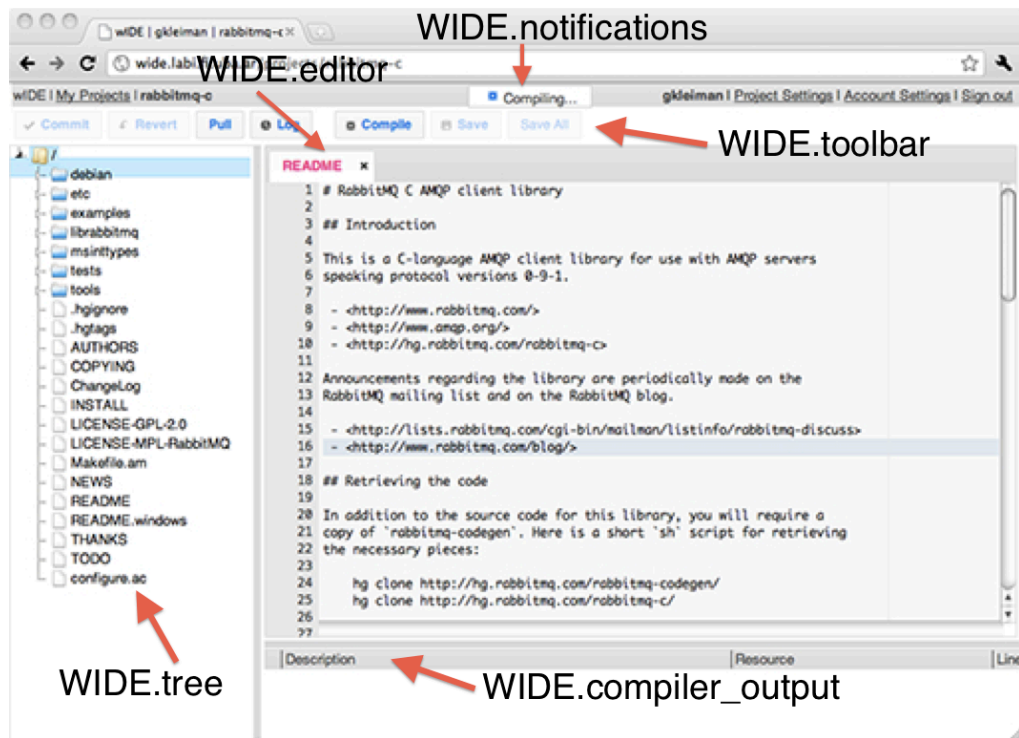
6.1. Módulos y componentes en la pantalla de edición . . . . .	34
6.2. Diagrama de secuencia de la operación “pull” . . . . .	42

## 6.1. Arquitectura

El código JavaScript utilizado por el cliente web está dividido en módulos responsables de agregar comportamiento y dinamismo a cada uno de los

componentes, y de código específico a cada página o diálogo, responsable de la inicialización y del manejo de eventos.

En la figura 6.1 se detalla qué módulo es responsable de cada componente en la página de edición.



**Figura 6.1:** Módulos responsables de cada componente de la pantalla de edición.

Cada componente agrega dentro del objeto global WIDE un objeto con el nombre del módulo, y que contiene sus funciones y atributos públicos.

Debido a que no hay dependencias complejas entre los módulos, no fue necesario incorporar un sistema de carga de módulos y archivos, como por ejemplo `RequireJS` [30].

Para la creación de objetos se utiliza el patrón “Power Constructor”, descrito en “JavaScript: The Good Parts” [31].

Se utiliza el framework CSS de jQuery UI [32] para dar un estilo unificado a todos los componentes, y permitir el uso de temas.



## 6.2. Convenciones

### Nomenclatura

#### Nomenclatura de archivos

Los archivos JavaScript y CSS se nombran de acuerdo a las siguientes reglas:

1. Se utiliza el idioma inglés para nombrar los archivos.
2. Todo archivo perteneciente a un componente externo dependiente de jQuery tiene el prefijo “`jquery.<nombre del componente>`”.

Por ejemplo:

- `jquery.jstree.js`
- `jquery.jstree.overlay.js`

3. Todo archivo perteneciente a un componente externo dependiente de jQuery UI tiene el prefijo “`jquery-ui.<nombre del componente>`”.
4. Todo archivo perteneciente a un componente propio de la aplicación tiene el prefijo “`wide.<nombre del componente>`”.

Por ejemplo:

- Para el componente base: `wide.js`
- Para el componente responsable de la operación de las pestañas y los editores: `wide.editor.js`
- Para los estilos del árbol: `wide.tree.css`

5. Todo archivo que contenga código que debe ser cargado sólo para una página específica tiene el prefijo:

“`page.<controlador>[.<acción>]`”

Por ejemplo:

`page.project.js`: es cargado en todas las páginas relacionadas con proyectos.

`page.project.edit.js`: es cargado en la página de edición de un proyecto.

`page.project.css`: contiene estilos utilizados en todas las páginas relacionadas con proyectos.

6. Todo archivo perteneciente a un diálogo presente en una página, tiene el prefijo:

“page.<controlador>.<acción>.dialog.<nombre del diálogo>”

### Nomenclatura de variables y funciones

Los nombres de funciones y variables siguen las siguientes reglas:

1. Tanto variables como funciones son nombradas en inglés.
2. Los nombres de clases comienzan en mayúscula.
3. Los nombres de funciones y variables se escriben en minúsculas utilizando el guión bajo como separador.

### Estilo de código

Se utiliza el subconjunto de JavaScript definido por JSLint [33] y detallado en el libro “JavaScript: The Good Parts” [31] sin cambios, excepto por la aceptación del uso de los operadores de pre/post suma y resta ( $++$  y  $--$ ).

Se aplica JSLint a todos los archivos con código JavaScript, para verificar que cumplan con el estilo de código definido.

Los únicos objetos almacenados en el ámbito global son:

**jQuery y \$:** pertenecientes a jQuery.

**WIDE:** perteneciente a los componentes propios.

Todo módulo que se quiere hacer público debe agregarse a alguno de esos objetos, evitando así contaminar el ámbito global.

## 6.3. Detalles de implementación

En esta sección se explica el objetivo de cada módulo, y se describen sus métodos y atributos públicos.

### Componentes base

#### Base

Se encuentra en el archivo `wide.js` y contiene funciones utilitarias usadas por el resto de los módulos.

`project_id()`: devuelve el nombre del proyecto actual.

`base_path()`: devuelve el URI correspondiente al actual proyecto.

`repository_path()`: devuelve el URI correspondiente al repositorio del proyecto actual.

`repository_entries_path()`: devuelve el prefijo de los URIs correspondientes a archivos/directorios del repositorio del actual proyecto.

`encode_path(ruta)`: devuelve una cadena de caracteres con la ruta pasada por parámetro correctamente codificada para ser incluida en un URI.

`csrf_token()`: devuelve el nombre del campo utilizado en los formularios para evitar ataques CSRF.

`csrf_param()`: devuelve el valor del campo utilizado en los formularios para evitar ataques CSRF.

### Notificaciones

Se encuentra en el archivo `wide.notifications.js` y expone funciones que permite notificar al usuario tanto del comienzo y fin de actividades, como de errores.

`activity_started(message)`: indica el comienzo de una actividad.

`success(message)`: agrega una notificación indicando que una actividad fue completada.

`error(message)`: notifica un error.

`hide()`: esconde todas las notificaciones.

### Formularios

Se encuentra en el archivo `wide.simple_forms.js` y es responsable de brindar un comportamiento y estilo unificado a todos los formularios presentes en el sistema, mediante la aplicación de clases CSS brindadas por jQuery UI.

No cuenta ni con métodos ni con atributos públicos.

## Paginado

Se encuentra en el archivo `wide.pagination.js` y es responsable de brindar un comportamiento y estilo unificado a todos los links de paginado presentes en el sistema, mediante la aplicación de clases CSS brindadas por jQuery UI.

No cuenta ni con métodos ni con atributos públicos.

## Componentes de la página de edición

### Árbol de directorios

Se encuentra en el archivo `wide.tree.js` y es encargado tanto de la configuración del árbol de directorios, como del manejo de los eventos relacionados con el mismo.

El árbol de directorios se carga con información obtenida del servidor en formato `JSON`, comunicándose con este a través de la API descrita en el capítulo 5.

`refresh(nodo)`: recarga las ramas del árbol descendientes del nodo pasado por parámetro.

`select_node(nodo)`: selecciona el nodo pasado por parámetro.

`focus()`: da foco al árbol.

`unset_focus()`: saca foco al árbol.

### Editor

Se encuentra en el archivo `wide.editor.js` y es responsable con la creación y destrucción de pestañas de edición, y del grabado los cambios realizados en las mismas.

`edit_file(ruta, [numero_de_linea])`: en caso de no existir ninguna pestaña con un editor para el archivo correspondiente a la ruta pasada por parámetro, crea una nueva y le da foco. En caso de ya existir una, le da foco. Si se le pasa un número de línea, mueve el cursor al número de línea correspondiente.

`get_current_editor()`: devuelve el objeto correspondiente al editor de la pestaña actualmente seleccionada.

`remove_editor(índice)`: cierra la pestaña identificada por el índice pasado por parámetro. Si esta contiene cambios sin guardar, se abre un diálogo preguntándole al usuario si desea grabar los cambios antes de cerrar la pestaña.

`save_all()`: guarda todos los cambios realizados.

`save_current()`: guarda los cambios realizados al archivo que se está editando en la pestaña seleccionada actualmente.

`modified_editors()`: devuelve un arreglo con todos los editores que tienen cambios sin grabar.

`focus()`: le da foco al editor de la pestaña actualmente seleccionada.

`dimensions_changed()`: responsable de notificar al editor actual que han cambiado las dimensiones de su contenedor.

## Archivo

Se encuentra en el archivo `wide.file.js`. Expone la clase `WIDE.File`, cuyas instancias representan archivos dentro del repositorio del proyecto actual.

Sobre estas instancias se pueden realizar operaciones relacionadas tanto con la administración de archivos como con el control de versiones.

Las operaciones son realizadas por el servidor de forma asincrónica. El módulo es responsable de notificar al usuario el resultado de las mismas.

Varios métodos de la clase `WIDE.File` toman como parámetro funciones llamadas `success` y `fail`. En todos los casos, `success` será llamada en caso de que la operación asincrónica se ejecute con éxito, `fail` será invocada en caso de que la operación falle.

`File(ruta, es_un_directorio)`: devuelve una instancia representando a un archivo o directorio bajo la ruta pasada por parámetro.

Métodos de cada instancia:

`path()`: devuelve una cadena con la ruta del archivo dentro del repositorio.

`file_name()`: extrae de la ruta el nombre del archivo, y lo devuelve.

`cat(success, fail)`: devuelve una cadena con el contenido del archivo.

`create(success, fail)`: crea un archivo o directorio con la ruta correspondiente a esta instancia.

`log()`: abre una página con la lista de cambios que han modificado este archivo.

Los siguientes métodos están relacionados con operaciones de control de versiones:

- `add(success, fail)`
- `forget(success, fail)`
- `revert(success, fail)`
- `mark_resolved(success, fail)`
- `mark_unresolved(success, fail)`
- `mv(dest_path, success, fail)`
- `rm(success, fail)`
- `diff(success, fail)`

### Barra de herramientas

Se encuentra en el archivo `wide.toolbar.js`. Es encargado de inicializar los botones de la barra de herramientas, y de habilitarlos y deshabilitarlos cuando sea necesario.

`update_scm_buttons()`: habilita o deshabilita los botones “Commit”, “Revert” y “Pull”, de acuerdo al estado del repositorio.

`update_save_buttons()`: habilita los botones “Save” y “Save All” si hay abierto algún editor con cambios sin guardar, los deshabilita en caso contrario.

### Disposición (Layout)

Se encuentra en el archivo `wide.layout.js`. Es encargado de inicializar y mantener la disposición y el tamaño de los componentes.

`layout()`: recalcula la disposición y tamaño de los componentes. Se la debe llamar al cambiar el tamaño de la ventana o de cualquiera de los paneles.

### Compilación

Se encuentra en el archivo `wide.compilation.js`. Es responsable de realizar al servidor pedidos de compilación.

`compile(save_files_if_modified)`: inicia la compilación del proyecto. En caso de que `save_files_if_modified` sea `true`, guarda primero todos los archivos abiertos.

`is_compilation_in_progress()`: devuelve `true` si se está esperando que finalice la compilación del proyecto, `false` de lo contrario.

### Resultados de compilación

Se encuentra en el archivo `wide.compiler_output.js`. Es el responsable del componente que muestra los resultados de la compilación.

`add_output(fila)`: agrega una fila a la tabla. `fila` debe contener los siguientes atributos:

`description`: mensaje.

`type`: tipo de mensaje. Puede ser `warning`, `error` o `info`.

`file_name`: (opcional) nombre del archivo al que corresponde el mensaje.

`line_number`: (opcional) número de línea.

`clear()`: vacía la tabla.

### Operaciones asincrónicas

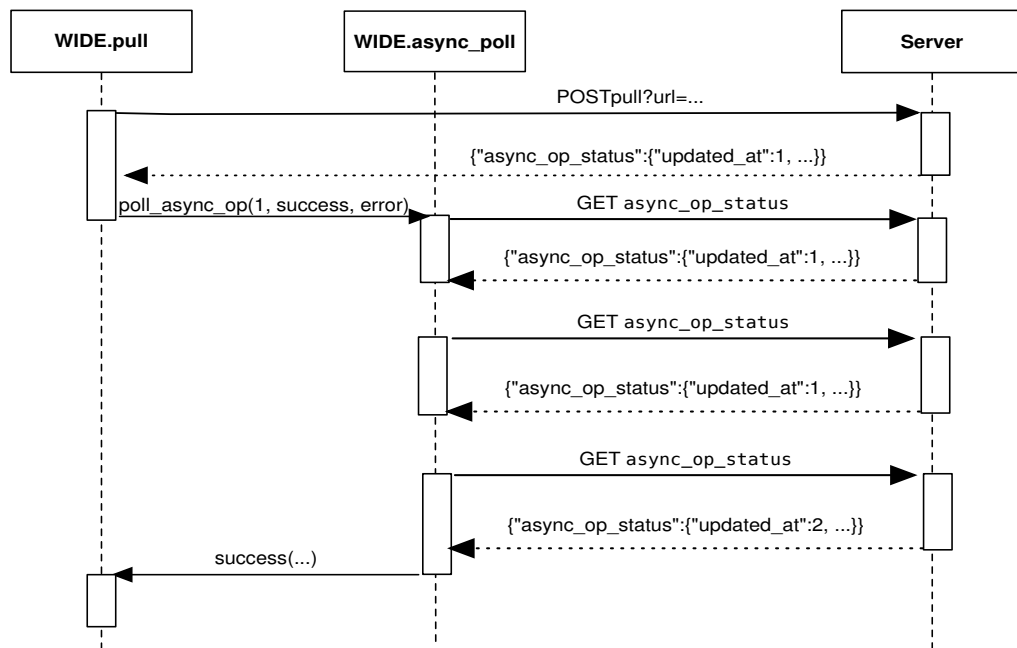
Se encuentra en el archivo `wide.async_op.js`. Es el encargado de realizar polling cuando se invoca una operación asincrónica.

Esto sucede por ejemplo cuando se hace “pull” de otro repositorio para importar cambios hechos en este. En la figura 6.2 se representa este proceso en un diagrama de secuencia.

`poll_async_op(timestamp, after_operation, error)`: inicia el polling, el cual continuará hasta que el servidor comunique un resultado con un timestamp mayor al pasado por parámetro.

Se llamará a `after_operation` en caso de que la operación haya sido exitosa, y a `error` de lo contrario.

`is_async_op_in_progress()`: devuelve `true` si hay una operación en progreso, `false` de lo contrario.



**Figura 6.2:** Diagrama de secuencia del polling realizado al hacer “pull”.

### Diffstat

Se encuentra en el archivo `wide.diffstat.js` y es responsable de brindar comportamiento y estilo unificado a todas las listas de archivos modificados dentro del repositorio.

No cuenta ni con métodos ni con atributos públicos.

## 6.4. Utilización de componentes de terceros

Todo el código JavaScript se construyó sobre jQuery [10].

Se utilizó jQuery UI [11] para todos los componentes visuales de la interfaz de usuario (botones, diálogos, etc.).

Para el árbol de exploración de directorios se utiliza jsTree [34] con un plugin propio y otras modificaciones.

El componente para la edición de código es llamado Ace [35]. Se tuvo que desarrollar una extensión para el coloreado de código C/C++. Esta fue aceptada por los desarrolladores del componente, y agregada al repositorio oficial.

Los algoritmos utilizados para la disposición de los componentes provienen de jLayout [36].



# CAPÍTULO 7

## Detalles de implementación del servidor

### Índice

7.1. Arquitectura global . . . . .	44
7.2. Servidor Web . . . . .	45
Arquitectura . . . . .	45
Modelos . . . . .	45
Vistas . . . . .	49
Controladores . . . . .	49
Arquitectura para soporte de diversos SCMs . . . . .	52
Autenticación de usuarios . . . . .	52
7.3. Tareas asincrónicas . . . . .	52
Proceso de compilación . . . . .	53
7.4. Panel de administración . . . . .	53
7.5. Proxy para acceso externo a los repositorios . . . . .	54

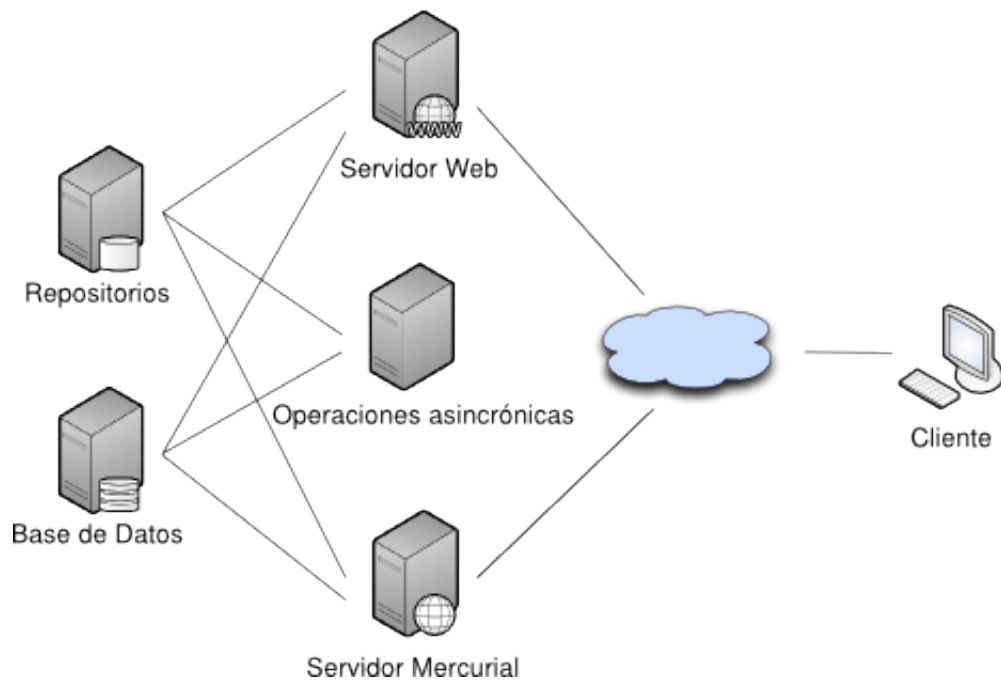
### Índice de figuras

7.1. Diagrama de relaciones entre componentes. . . . .	44
7.2. Diagrama con atributos y relaciones entre modelos. . . . .	47
7.3. Diagrama de clases (modelos). . . . .	48

7.4. Diagrama de clases (controladores).	50
--	----

## 7.1. Arquitectura global

En la figura 7.1 se pueden observar las relaciones entre los distintos componentes que conforman el sistema.



**Figura 7.1:** Diagrama de relaciones entre componentes.

El servidor Web, desarrollado en *Ruby on Rails* [8], sirve las páginas que componen a la IDE web, implementa una API RESTful [37] para acceder a los proyectos y repositorios, y también provee el panel de administración web.

El cliente web accede utilizando tecnología AJAX a la información de los repositorios a través de esta API.

También es posible interactuar externamente con los repositorios a través del servidor Mercurial, utilizando cualquier cliente. Por ejemplo *TortoiseHG* y el cliente de consola oficial.

Este servicio es provisto por un proxy, desarrollado en Python, que utiliza la base de datos para autenticar y autorizar el acceso a los repositorios.

Paralelamente a estos servicios corre otro, que ejecuta asincrónicamente tareas que requieren de un largo tiempo de ejecución, y que son encoladas por los otros servicios. Algunos ejemplos de estas tareas son el clonado de repositorios, la compilación de proyectos, etc..

El sistema puede escalar ejecutando estos servicios en servidores diferentes, siempre y cuando estos tengan acceso a la base de datos y al directorio que contiene los repositorios.

## 7.2. Servidor Web

### Arquitectura

El servidor está implementado con Ruby on Rails (RoR). El sistema se divide en modelos (implementan lógica de negocio, y son persistidos en la base de datos), vistas y controladores de acuerdo al patrón MVC.

RoR abstrae el acceso a la base de datos siguiendo el patrón ActiveRecord [38].

Durante todo el desarrollo se intentó aplicar el principio DRY (Don't Repeat Yourself) evitando la duplicación de código, y de escribir "Fat Models and Skinny Controllers". Esto quiere decir que se intenta concentrar toda la lógica de negocio en los modelos, obteniendo como consecuencia código fácil de entender, terso y breve, tanto en los controladores, como en las vistas.

### Modelos

Los modelos de la aplicación son los que contienen la lógica de negocio. Esto quiere decir que dentro de ellos se implementan las validaciones y el comportamiento necesario para el sistema.

La figura 7.2 fue generada a partir del esquema de la base de datos. Esta presenta un diagrama, similar a un diagrama de entidad-relación, en el cual se indican las relaciones entre los distintos modelos, y los atributos de cada uno.

La figura 7.3 es un diagrama de clases que incluye sólo los modelos y las clases utilitarias. Dado que todos los modelos descienden de la clase `ActiveRecord::Base`, se obvió esta relación en el diagrama. Tampoco se incluyen los métodos necesarios para leer y modificar los atributos presentes en la base de datos, ya que estos son agregados de forma dinámica en tiempo de ejecución. Se puede observar que la mayoría del comportamiento de los modelos está implementado en las clases `Project` y `Repository`.

La clase `Repository` delega la mayoría de su comportamiento al adaptador de SCM que corresponda, y mantiene un cache con información sobre el estado del repositorio.

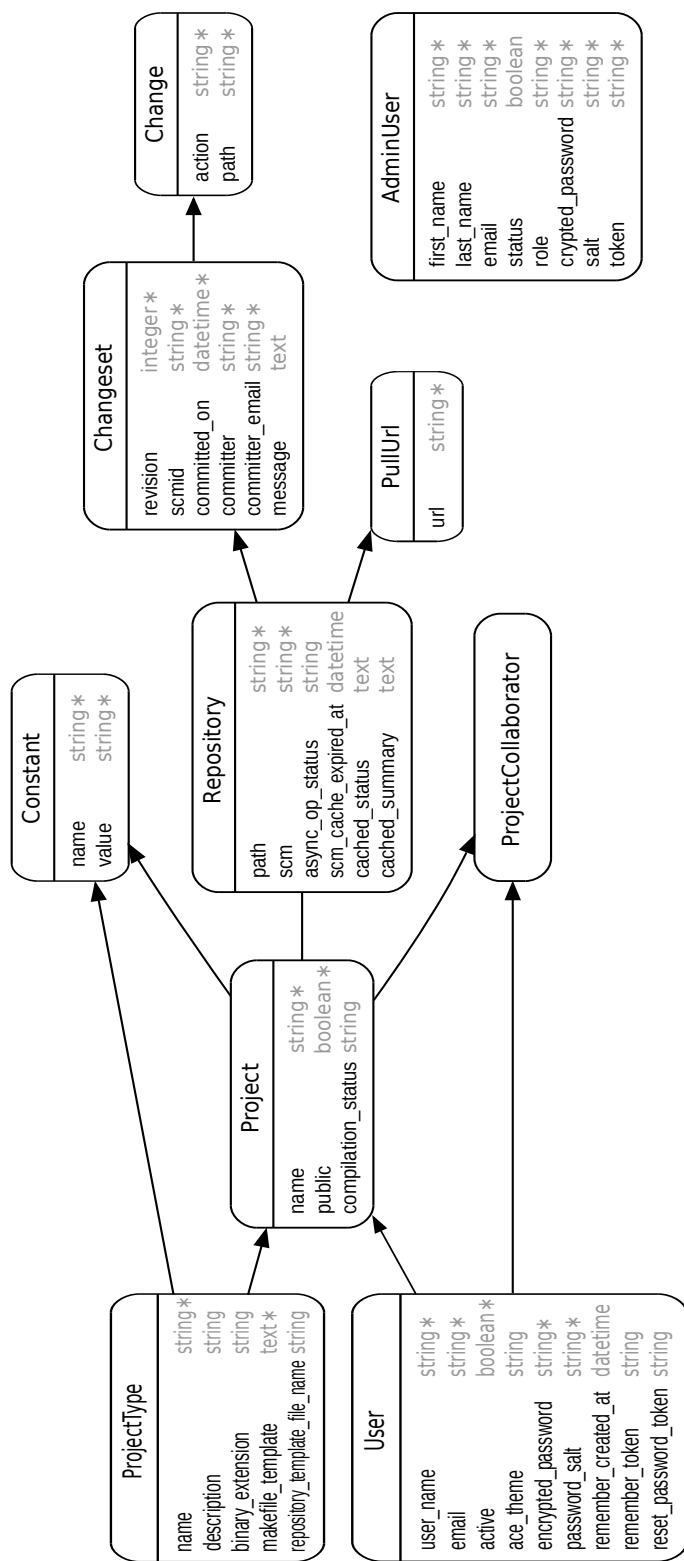


Figura 7.2: Diagrama con atributos y relaciones entre modelos.

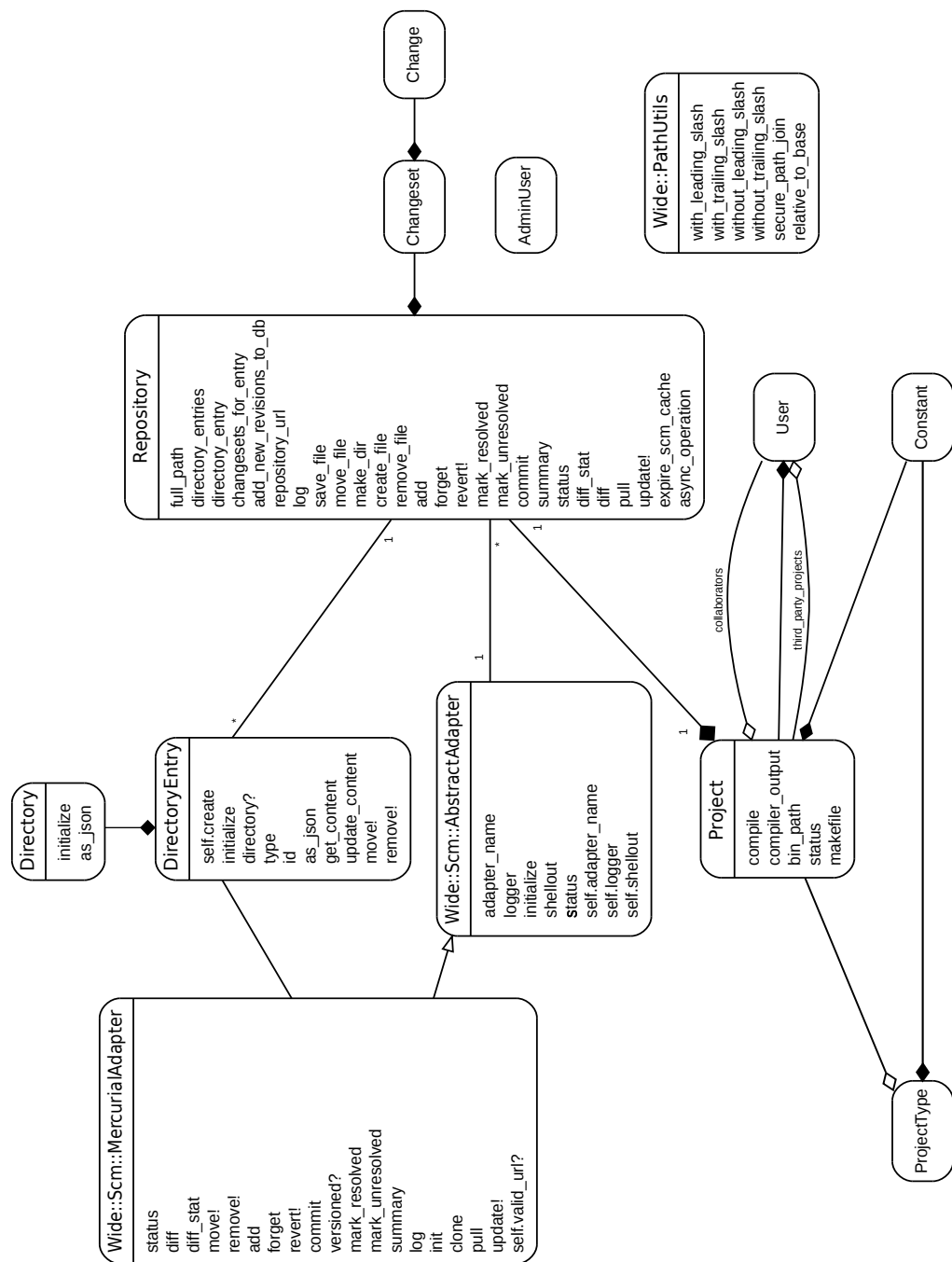


Figura 7.3: Diagrama de clases (modelos).

## Vistas

Las plantillas que conforman las vistas están escritas en un lenguaje llamado `HamI` [39], el cual tiene como objetivo ser lo más terso y claro posible. Para esto utiliza el sangrado (indentación) para delimitar el comienzo y fin de una etiqueta.

El código de las plantillas se limita a iterar colecciones y rellenar campos con valores de los modelos, delegando a ellos y a los controladores la mayoría de la lógica.

En los casos en que se detectó duplicación de código en las plantillas, se extrajeron de ellas “partials”, los cuales pueden luego ser incluidos desde más de un lugar.

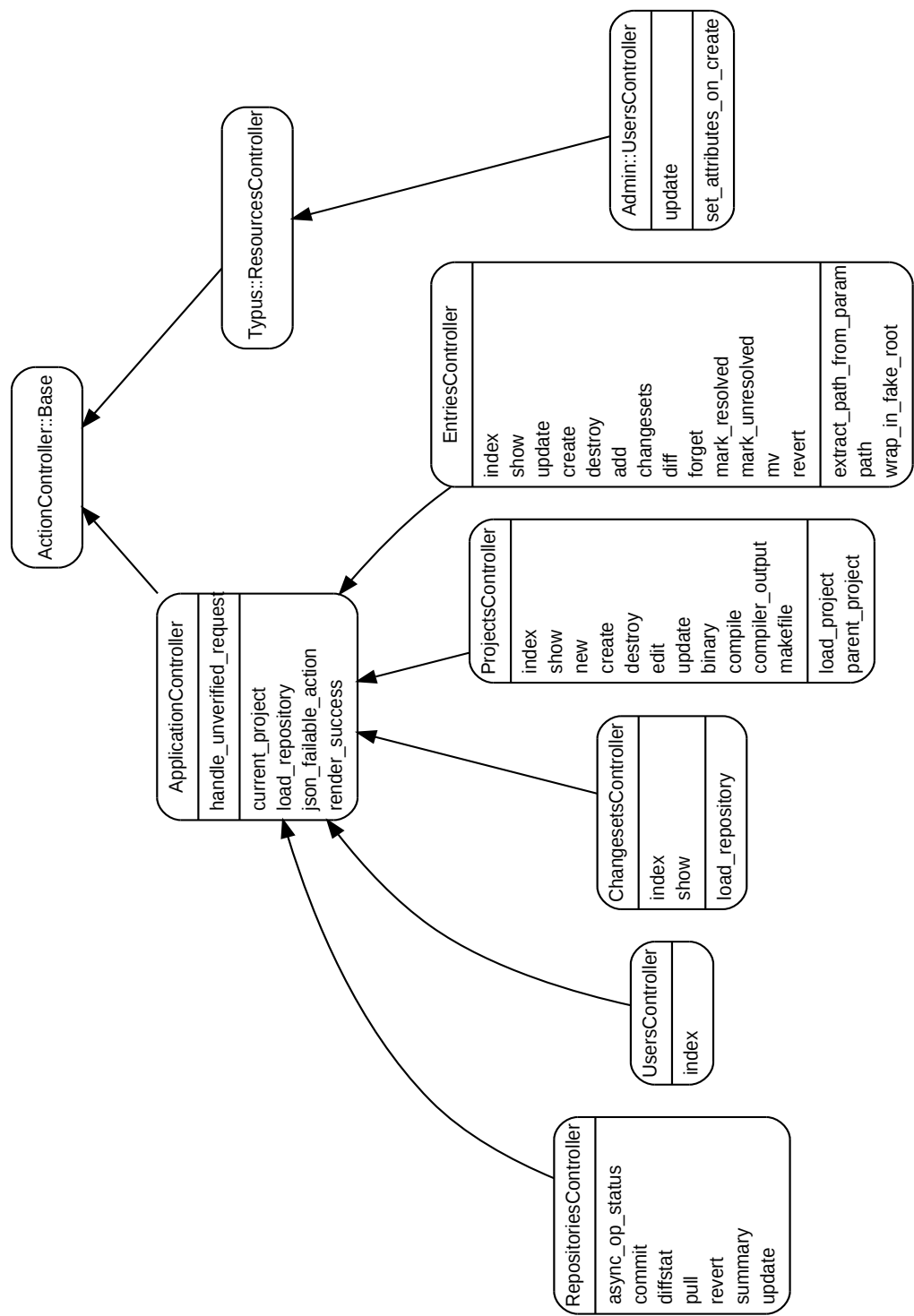
## Controladores

En la figura 7.4 se observa un diagrama de clases de los controladores.

Las clases `EntriesController` y `RepositoriesController` son las responsables de implementar la mayoría de la API REST, la cual es descrita en un documento aparte.

`ProjectsController` se encarga de las acciones relacionadas con la visualización, alta, baja y modificación de los proyectos, y también atiende los pedidos de compilación a través de la API.

En el diagrama de clase se puede observar que se evita la duplicación de código creando filtros que se ejecutan antes que los métodos correspondientes a las acciones. Estos se ocupan de procesar los parámetros recibidos, y de cargar en base a ellos las instancias de los modelos necesarias para atender los pedidos.



**Figura 7.4:** Diagrama de clases (controladores).



Método	URI	Acción
POST	/projects/:id/compile	ProjectsController#compile
GET	/projects/:id/compiler_output	ProjectsController#compiler_output
GET	/projects/:id/binary	ProjectsController#binary
GET	/projects/:id/makefile	ProjectsController#makefile
GET	/projects/:id/repository/diffstat	RepositoriesController#diffstat
GET	/projects/:id/repository/summary	RepositoriesController#summary
GET	/projects/:id/repository/async_op_status	RepositoriesController#async_op_status
POST	/projects/:id/repository/revert	RepositoriesController#revert
POST	/projects/:id/repository/update	RepositoriesController#update
POST	/projects/:id/repository/commit	RepositoriesController#commit
POST	/projects/:id/repository/pull	RepositoriesController#pull
GET	/projects/:id/repository/entries	EntriesController#index
GET	/projects/:id/repository/entries/*path	EntriesController#show
POST	/projects/:id/repository/entries/*path	EntriesController#update
PUT	/projects/:id/repository/entries/*path	EntriesController#create
DELETE	/projects/:id/repository/entries/*path	EntriesController#destroy
POST	/projects/:id/repository/entries/*path/add	EntriesController#add
POST	/projects/:id/repository/entries/*path/mark_resolved	EntriesController#mark_resolved
POST	/projects/:id/repository/entries/*path/mark_unresolved	EntriesController#mark_unresolved
POST	/projects/:id/repository/entries/*path/mv	EntriesController#mv
POST	/projects/:id/repository/entries/*path/forget	EntriesController#forget
POST	/projects/:id/repository/entries/*path/revert	EntriesController#revert
GET	/projects/:id/repository/entries/*path/diff	EntriesController#diff

**Tabla 7.1:** Mapeo de las rutas de la API a los controladores.

## Arquitectura para soporte de diversos SCMs

El soporte de SCMs se implementó de forma extensible, de manera que en un futuro sea fácil agregar soporte para otros SCM.

Se puede observar en la figura 7.4 que para soportar un SCM, se debe implementar un adaptador para el mismo. Este adaptador debe heredar de la clase `Wide::SCM::AbstractAdapter`, la cual implementa funcionalidad básica común a todos los adaptadores, tal como la ejecución y comunicación con procesos externos a través de un *pipe*.

Se desarrolló un adaptador para Mercurial, y otro llamado *Filesystem*, el cual no realiza ningún tipo de versionado.

## Autenticación de usuarios

Para la autenticación de usuarios se utilizó una solución llamada Devise [13], la cual permite incorporar fácilmente al sistema la siguiente funcionalidad:

- Registro de usuarios.
- Activación de usuarios.
- Envío de emails para permitir la recuperación de usuarios.
- Manejo de sesiones.

## 7.3. Tareas asincrónicas

La ejecución asincrónica y en segundo plano de tareas largas se logró utilizando *Delayed Job* [40]. Esta solución depende de RoR, y permite además de encolar tareas de forma manual, especificar que ciertos métodos deben ser ejecutados siempre de forma asincrónica.

Las siguientes tareas se realizan asincrónicamente:

- Compilación de proyectos.
- Clonación de repositorios externos.
- Inicialización de nuevos repositorios.
- Obtención de cambios desde repositorios externos (“pull”).
- Envío de emails.

## Proceso de compilación

Al solicitar la compilación de un proyecto, se siguen los siguientes pasos:

1. Se graba el proyecto, indicando que se está corriendo el proceso de compilación.
2. Se encola la compilación del proyecto.
3. Un trabajador de *Delayed Job* desencola la tarea.
4. Se copia el contenido del repositorio a un directorio temporal.
5. Se genera el **Makefile** correspondiente al tipo de proyecto.
6. Se corre el **Makefile**, grabando los mensajes de salida y error en un archivo, dentro de la ruta configurada para almacenar los resultados de compilación.
7. Si la compilación fue exitosa, y se generó un archivo binario llamado **binary**, se lo mueve dentro de la ruta configurada.
8. Se vuelve a grabar el proyecto, indicando si se produjeron errores durante la compilación.

## 7.4. Panel de administración

Se utilizó Typus [12] para construir el panel de administración. Typus es un framework que permite obtener un panel de administración fácilmente extensible, a partir de archivos de configuración YAML [41]. En estos se especifican tanto roles y permisos de los administradores, como los campos de cada modelo que se presentarán en los listados y en los formularios de edición.

Este framework no contaba con soporte para relaciones de tipo **has\_many :through**. Por lo tanto se implementó esta funcionalidad, y se enviaron al desarrollador del framework los parches correspondientes. Estos fueron incorporados al repositorio oficial.

Como medida de seguridad, el sistema de autenticación y autorización de usuarios del panel de administración es totalmente independiente al del resto del sistema.

## 7.5. Proxy para acceso externo a los repositorios

Para lograr que sea posible la autenticación y autorización de usuarios para el acceso externo a los repositorios utilizando clientes Mercurial, fue necesario crear un proxy, el cual actúa entre los usuarios y el servidor de repositorios incluido en mercurial (**hgweb**).

Este proxy fue implementado en Python, basado en HgRedmine [42].

# Instalación

## Índice

8.1.	Dependencias . . . . .	55
8.2.	Supuestos . . . . .	56
8.3.	Procedimiento de instalación . . . . .	56
	Preparación del entorno de despliegue . . . . .	56
	Preparación del servidor . . . . .	57
	Despliegue . . . . .	58
	Configuración del sistema . . . . .	58

En este capítulo se han las instrucciones de instalación del sistema utilizando un servidor **Apache 2** corriendo bajo **GNU/Linux**.

## 8.1. Dependencias

El servidor en el que se instale el sistema debe contar con los siguientes componentes ya instalados:

- Ruby v1.8.7.
- Python v2.6.
- Mercurial, versión mayor o igual a v1.6.3.
- Apache 2.

- `mod_xsendfile`. [25]
- Phusion Passenger. [26]
- Capistrano. [27]
- Rubygems, versión mayor o igual a v1.3.6. [28]
- SQLite3.

## 8.2. Supuestos

Para la instalación del sistema se supone que:

- Todos los componentes del sistema correrán en el mismo servidor.
- El servidor corre el sistema operativo **GNU/Linux** con el software listado en la sección 8.1 ya instalado y configurado.
- Se utilizará *Apache 2* con *Phusion Passenger* como servidores de la aplicación.
- La distribución instalada en el servidor es derivada de *Debian*.
- Se creará en el servidor un usuario llamado **wide**, el cual será usado exclusivamente por el sistema.
- La instalación se realizará desde un “entorno de despliegue”, desde el cual se cuenta con acceso remoto al servidor por *ssh*.
- El sistema se instalará en el home del usuario **wide**, el cual se encuentra bajo la ruta `/home/wide/`.
- Se ha configurado previamente en el servidor de nombres (NS) correspondiente los dominios a utilizar por el sistema.

## 8.3. Procedimiento de instalación

### Preparación del entorno de despliegue

En el entorno de despliegue (puede usarse el home de otro usuario en el mismo servidor en el que quedará instalado el sistema) seguir los siguientes pasos:

1. Obtener el código del sistema desde el repositorio almacenado en GitHub:

```
$ git clone git://github.com/gkleiman/wide.git
$ cd wide
```

2. Instalar bundler:

```
$ sudo gem install bundler
```

3. Instalar las gemas necesarias para realizar el despliegue:

```
$ bundle install
```

4. Configurar los datos de del servidor sobre el que correrá la aplicación, editando `config/deploy.rb`.

```
$ vim config/deploy.rb
```

5. Copiar clave pública de `ssh` al servidor:

```
$ scp ~/.ssh/id_rsa.pub \
  <ip_del_servidor>:/home/wide/.ssh/authorized_keys
```

## Preparación del servidor

Realizar los siguientes pasos en el servidor:

1. Crear el usuario `wide`:

```
$ sudo adduser wide --no-login
```

2. De ser necesario utilizar un proxy para acceder a sitios de Internet a través de los protocolos `http` y `https`, agregarlos en:

```
/etc/environment.
```

Por ejemplo:

```
$ echo "http_proxy=url" | sudo tee -a /etc/environment  
$ echo "https_proxy=url" | sudo tee -a etc/environment
```

3. Instalar las dependencias utilizadas por el wrapper de Mercurial:

```
$ sudo apt-get install python-bcrypt
```

4. Instalar bundler:

```
$ sudo gem install bundler
```

## Despliegue

Para hacer el despliegue se deben ejecutar los siguientes comandos desde el entorno de despliegue:

```
$ cap deploy:setup  
$ cap deploy  
$ cap deploy:migrate  
$ cap deploy:restart
```

## Configuración del sistema

### Configuración del servicio web

1. Configurar el dominio del servicio, el dominio del servicio de acceso remoto a los repositorios y la dirección de e-mail del sistema, modificando el siguiente archivo:

```
/home/wide/application/current/config/application.yml.
```

2. Copiar el archivo: `~wide/hgwide/wide.apache2.example.conf`

a

`/etc/apache2/sites-available` y modificarlo de acuerdo a la configuración del servidor.

```
$ sudo cp /home/wide/hgwide/wide.apache2.example.conf \  
          /etc/apache2/sites-available/wide  
$ sudo vim /etc/apache2/sites-available/wide
```



3. Activar el sitio:

```
$ sudo a2ensite wide
```

### Configuración del servicio de acceso externo a los repositorios

1. Copiar el archivo:

```
~wide/hgwide/hgwide.apache2.example.conf
```

a

/etc/apache2/sites-available, y modificarlo de acuerdo a la configuración del servidor.

```
$ sudo cp /home/wide/hgwide/hgwide.apache2.example.conf \
    /etc/apache2/sites-available/hgwide
$ sudo vim /etc/apache2/sites-available/hgwide
```

2. Activar el servicio:

```
$ sudo a2ensite hgwide
```

3. Reiniciar el servidor web:

```
$ sudo /etc/init.d/apache2 restart
```



# CAPÍTULO 9

## Administración y mantenimiento del sistema

### Índice

9.1. Acceso al panel web de administración . . . . .	62
9.2. Creación de nuevos administradores . . . . .	62
9.3. Activación de nuevos usuarios . . . . .	64
9.4. Creación de un nuevo tipo de proyectos . . . . .	65
Plantilla de Makefile . . . . .	66

### Índice de figuras

9.1. Primer paso para la creación del primer administrador. . .	62
9.2. Segundo paso para la creación del primer administrador. .	63
9.3. Pantalla principal del panel de administración. . . . .	63
9.4. Creación y activación de un nuevo administrador. . . . .	64
9.5. Activación de un usuario. . . . .	65
9.6. Creación de un nuevo tipo de proyecto. . . . .	66

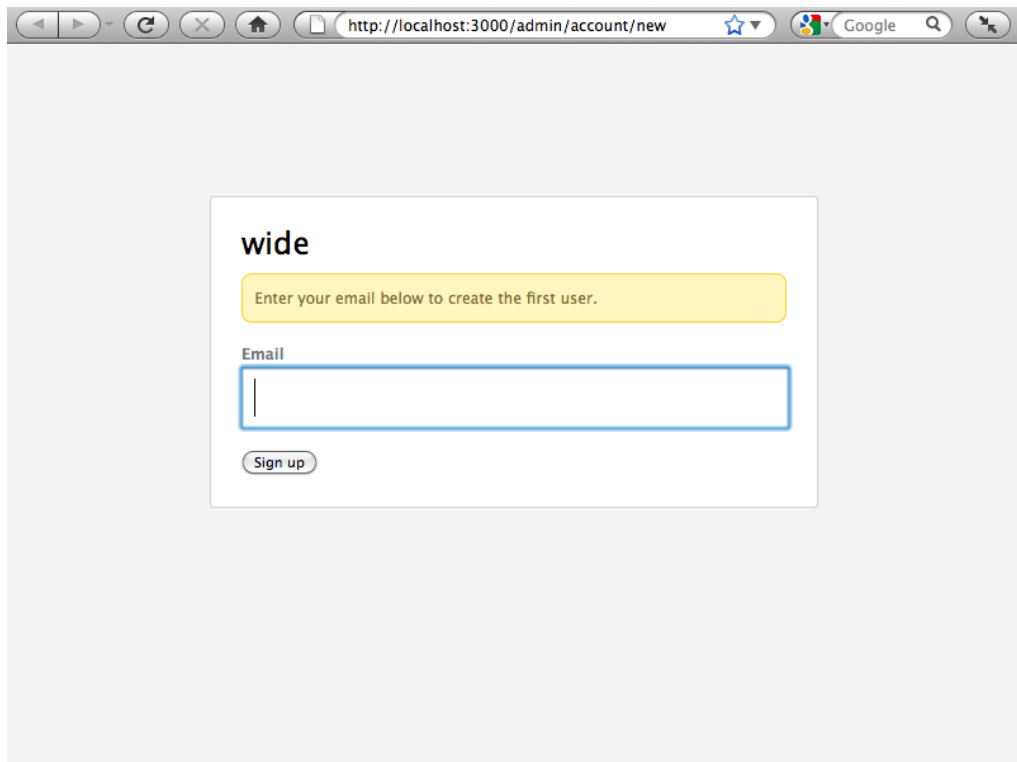
En este capítulo se proveen instrucciones de administración y mantenimiento para los administradores del sistema.

Para estas instrucciones se asume que el sistema es accesible utilizando la URL <http://wide.labi.fi.uba.ar>.

## 9.1. Acceso al panel web de administración

El sistema cuenta con un panel web de administración, accesible bajo la URL `http://wide.labi.fi.uba.ar/admin`.

Por motivos de seguridad, el panel de administración cuenta con un sistema de usuarios independiente. Al utilizar el panel por primera vez, se presentará un formulario para iniciar el proceso de creación del primer administrador, tal como se observa en las figuras 9.1 y 9.2.



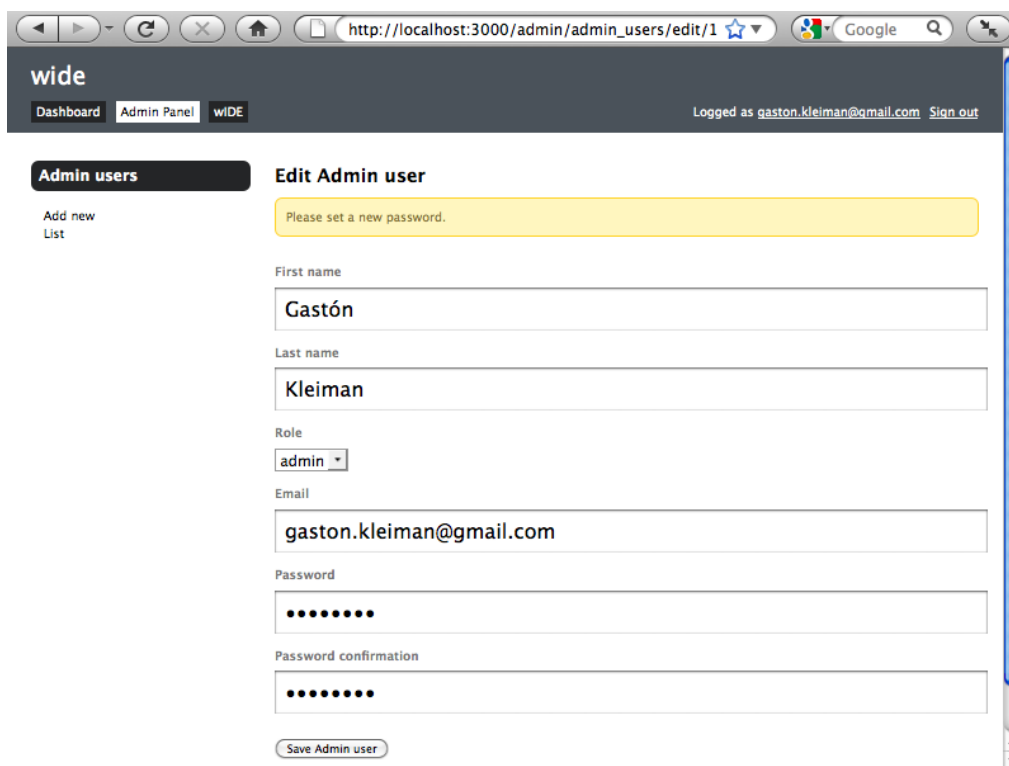
**Figura 9.1:** Primer paso para la creación del primer administrador.

Una vez creado el primer administrador, al ingresar, aparecerá la pantalla principal del panel, como en la figura 9.3.

## 9.2. Creación de nuevos administradores

Para crear un nuevo administrador se deben seguir los siguientes pasos:

1. Ingresar al panel de administración.



The screenshot shows a web browser window with the URL `http://localhost:3000/admin/admin_users/edit/1`. The page is titled "wide" and has a navigation bar with "Dashboard", "Admin Panel", and "wiDE". The user is logged in as "gaston.kleiman@gmail.com". The main content area is titled "Edit Admin user" and contains a form with the following fields:

- Please set a new password.** (Yellow highlighted box)
- First name:** Gastón
- Last name:** Kleiman
- Role:** admin (dropdown menu)
- Email:** gaston.kleiman@gmail.com
- Password:** (masked with dots)
- Password confirmation:** (masked with dots)
- Save Admin user** (button)

Figura 9.2: Segundo paso para la creación del primer administrador.

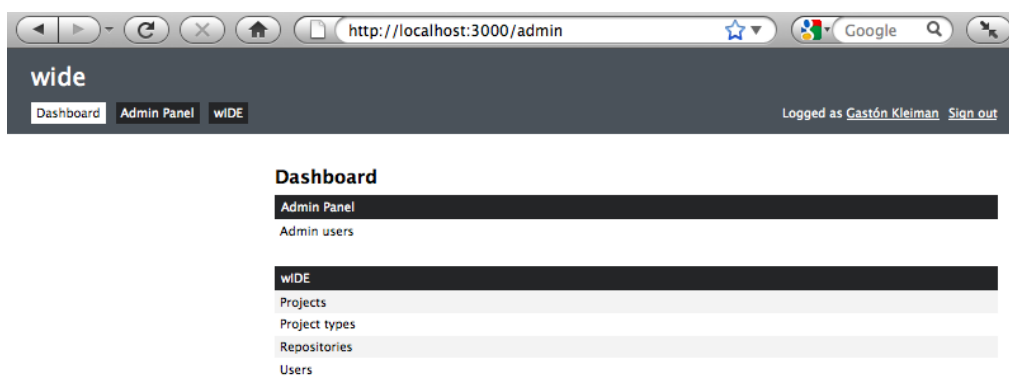
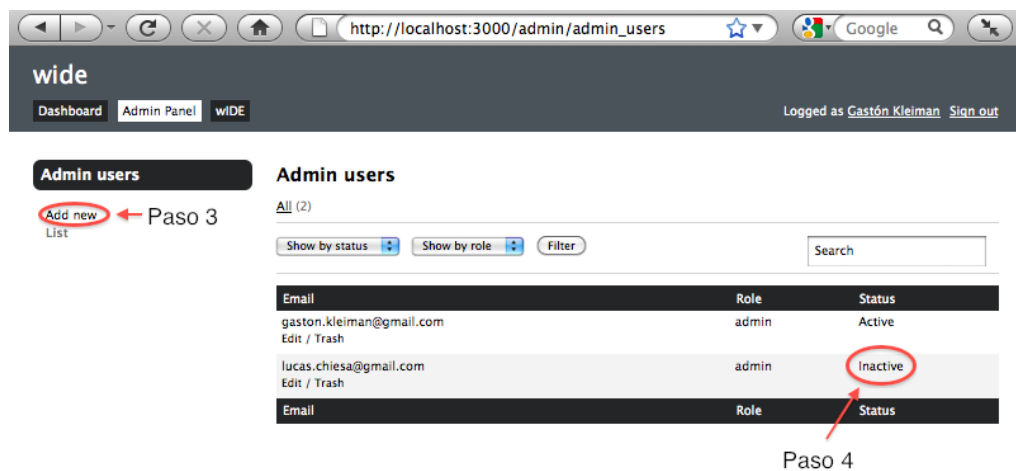


Figura 9.3: Pantalla principal del panel de administración.

2. En la pantalla principal hacer click en “*Admin Users*”.
3. Hacer click en “*Add new*”, tal como se indica en la figura 9.4.
4. Completar y enviar el formulario.
5. Activar el nuevo administrador, haciendo click en “*Inactive*”.
6. Confirmar el cambio de estado, haciendo click en “*Aceptar*” en el diálogo que aparecerá.



**Figura 9.4:** Creación y activación de un nuevo administrador.

### 9.3. Activación de nuevos usuarios

Por seguridad sólo podrán ingresar al sistema usuarios que hayan sido activados por un administrador. Por lo tanto se notificarán los nuevos registros a todos los administradores activos mediante un e-mail.

Para activar un usuario, se deben seguir los siguientes pasos:

1. Ingresar al panel de administración.
2. En la pantalla principal hacer click en “*Users*”. Se cargará un listado de los usuarios, como el de la figura 9.5.
3. Hacer click en “*Inactive*”.

4. Confirmar el cambio de estado.

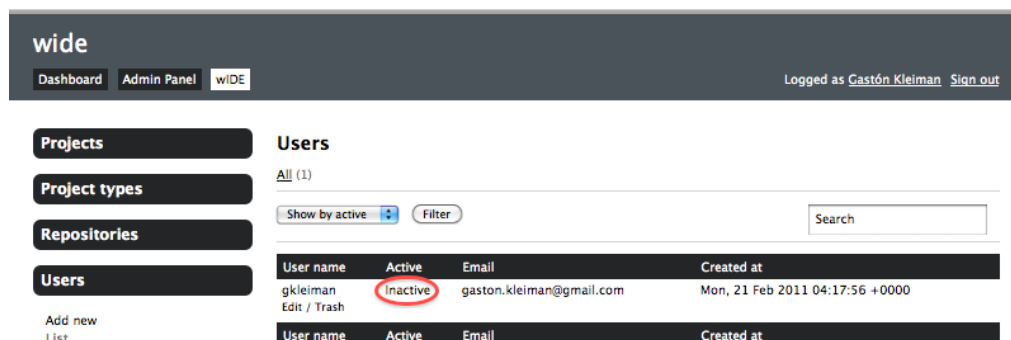


Figura 9.5: Activación de un usuario.

## 9.4. Creación de un nuevo tipo de proyectos

Para crear un nuevo tipo de proyecto, se debe acceder al formulario de creación de proyectos, siguiendo pasos análogos a los explicados anteriormente.

El formulario es similar al de la figura 9.6, y contiene los siguientes campos:

**Name:** nombre del tipo de proyecto. Cuando un usuario cree un proyecto, deberá elegir un tipo de proyecto mediante lista desplegable con el nombre de cada uno.

**Description:** descripción del tipo de proyecto. En la versión actual del sistema, este campo no es visible.

**Binary extension:** extensión que se agregará al archivo binario producido por el proceso de compilación, antes de enviárselo al usuario.

**Makefile template:** plantilla de `Makefile` a usar para compilar proyectos de este tipo.

**Repository template:** archivo comprimido en formato `tar.gz`. Al crearse un proyecto de este tipo, se extraerá el contenido de este archivo, y realizará un “*commit inicial*”.

The screenshot shows the 'New Project type' form in the 'wide' web interface. The form includes the following fields and content:

- Name:** LaTeX
- Description:** Definir el archivo a compilar en la constante TARGET
- Binary extension:** pdf
- Makefile template:**

```

TARGET=@ @constants["TARGET"] || "tp" %>

AUX := $(addsuffix /document.aux, $(SUBDIRS))
TARGET_FILES := $(patsubst %, $(TARGET).%, pdf)

# Script para compilar Latex, que compila tantas veces como sea necesario
# hasta que las referencias cruzadas queden correctas:
LATEX = pdflatex
LATEX_OPTS = --interaction nonstopmode
COMPILE_LATEX = { \
    $(LATEX) $(LATEX_OPTS) $< && \
    while grep -q \
        'No file ..\.(aux\|toc\|).\\|Rerun to get cross-references right.' \
        $(basename $<).log ; \
    do \
        $(LATEX) $(LATEX_OPTS) $<; \

```
- Repository template:** A text input field with a 'Browse...' button and a 'Remove' link.
- Create Project type:** A button at the bottom of the form.

**Figura 9.6:** Creación de un nuevo tipo de proyecto.

Una vez creado un proyecto, es posible asociar al mismo constantes, que serán agregadas a los proyectos de este tipo. Estas pueden ser modificadas en cada proyecto, y son accesibles desde las plantillas de **Makefile**, permitiendo cierto grado de configuración del proceso de compilación.

## Plantilla de Makefile

El **Makefile** utilizado para compilar cada proyecto es creado a partir de la plantilla correspondiente a su tipo de proyecto.



Esta plantilla debe ser escrita formato ERB [44]. Por lo tanto está conformada por el texto plano que irá en el `Makefile` y puede contener código Ruby embebido a través de la utilización de las siguientes etiquetas (en inglés, extraídas de la documentación original de ERB):

```
<% Ruby code -- inline with output %>
<%= Ruby expression -- replace with result %>
<%# comment -- ignored -- useful in testing %>
<%% or %%> -- replace with <% or %> respectively
```

El código embebido tendrá acceso a los siguientes objetos:

**@project:** instancia del proyecto para el cual se generará el `Makefile`. Es una instancia de un modelo `ActiveRecord`, por lo cual permite acceder a todos los atributos y relaciones listadas en la figura 7.3, a través de la API `ActiveRecord` [45].

**@constants:** diccionario que contiene una entrada por cada constante del proyecto para el que se generará el `Makefile`. Las claves del diccionario son los nombres de las constantes, mientras que sus valores son los valores de las mismas.

Es posible acceder a las constantes de un proyecto tanto a través de `project`, como de `@constant`.

Por lo tanto si el proyecto cuenta con una constante cuyo nombre es `TARGET`, y su valor es `tp`, entonces se cumplen las siguientes igualdades:

```
@project.constants.find_by_name('TARGET') == 'tp'
@constants['TARGET'] == 'tp'
```

Para que una compilación sea considerada exitosa, se debe crear en el directorio dentro del cual es invocado el `Makefile` un archivo llamado `binary`, y el proceso `make` debe finalizar con el código de retorno 0.

### Ejemplo de plantilla para la compilación de un proyecto $\text{\LaTeX}$

Si se deseara crear un tipo de proyecto que utilice `pdflatex` para generar un archivo `pdf` a partir de un documento  $\text{\LaTeX}$ , cuyo nombre se define en la constante `TARGET`, entonces la plantilla podría ser:

```
TARGET<%= @constants["TARGET"] || "tp" %>

AUX_:=_$(addsuffix_,/documento.aux,_$(SUBDIRS))
TARGET_FILES_:=_$(patsubst_,_,$(TARGET).%,_pdf)
```

```

#_Script_para_compilar_Latex,_que_compila_tantas_veces_como_sea
  _necesario
#_hasta_que_las_referencias_cruzadas_queden_correctas:
LATEX=_pdflatex
LATEX.OPTS=_--interaction_nonstopmode
COMPILELATEX=_{ _\
__$(LATEX) _$(LATEX.OPTS) _$<.&&_\
__while_grep_q_\
__'No_file_.*\.(aux|toc)\.\\Rerun_to_get_cross-references_
  right.' _\
_____$(basename_$<).log _; _\
__do_\
____$(LATEX) _$(LATEX.OPTS) _$<; _\
__done; _}

.PHONY: _all

all: _$(TARGET_FILES)

%.pdf: _%.tex
__@echo_"Generando_$@"...
__$ (COMPILELATEX)
__mv_$ (TARGET_FILES) _binary

```

# CAPÍTULO 10

## Manual del usuario

### Índice

---

10.1. Usuarios . . . . .	70
Registro . . . . .	70
Modificación de datos del usuario . . . . .	71
10.2. Listado de proyectos . . . . .	71
Creación de un proyecto . . . . .	73
Eliminación de un proyecto . . . . .	74
Duplicación de un proyecto . . . . .	74
10.3. Entorno de desarrollo . . . . .	74
Preferencias de un proyecto . . . . .	75
Estado de los archivos . . . . .	76
Creación de archivos y directorios . . . . .	76
Operaciones de control de versiones . . . . .	76
Commit de cambios realizados . . . . .	78
Visualización del historial de cambios . . . . .	79
Obtención de cambios desde otro repositorio . . . . .	80
Compilación . . . . .	80

---

### Índice de tablas

---

10.1. Iconos y sus correspondientes estados . . . . .	77
---	----

---

## Índice de figuras

---

10.1. Formulario de registro de usuarios . . . . .	71
10.2. Formulario de configuración de la cuenta de un usuario . .	72
10.3. Listado de proyectos . . . . .	72
10.4. Formulario para la creación de un nuevo proyecto . . . . .	73
10.5. Entorno de desarrollo . . . . .	75
10.6. Formulario de configuración de un proyecto . . . . .	76
10.7. Menú contextual . . . . .	77
10.8. Diálogo de “commit” . . . . .	78
10.9. Historial de cambios (changesets) . . . . .	79
10.10. Vista detalle de un cambio . . . . .	80

---

En este capítulo se proveen instrucciones de uso para los usuarios del sistema.

Se asume que el sistema es accesible bajo la URL `http://wide.labi.fi.uba.ar`.

## 10.1. Usuarios

### Registro

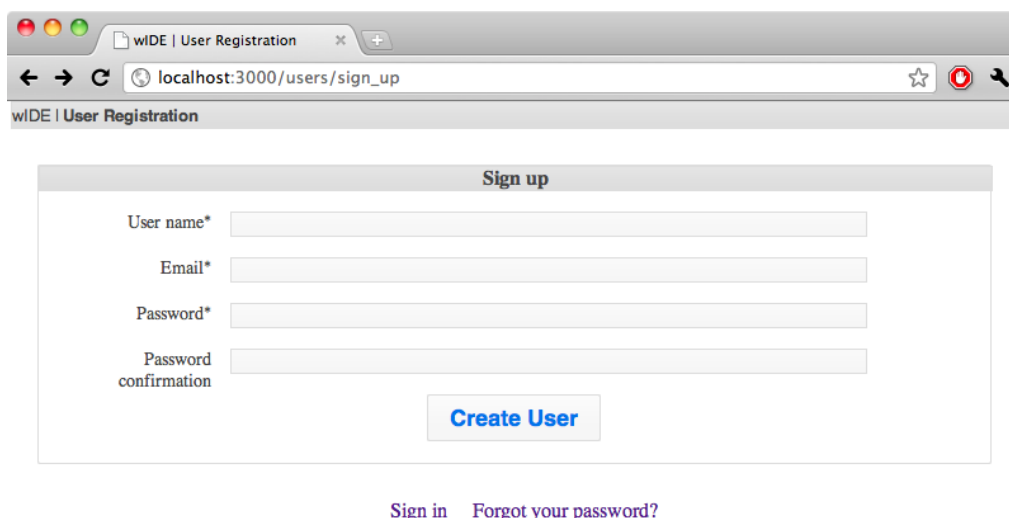
Para utilizar el sistema es necesario registrar un usuario. Esto se realiza desde el formulario de registro de usuarios, accesible al hacer click en el enlace “Sign up”, localizado en la página de raíz del sistema, debajo del formulario de ingreso.

Una vez hecho click, se presentará un formulario (Figura 10.1), en el cual se deben completar los siguientes campos:

**User name:** nombre de usuario. Puede estar compuesto sólo por caracteres alfanuméricos, guiones (normal y bajo) y puntos. No puede comenzar con un punto.

**Email:** debe ser una dirección de e-mail válida. El sistema verificará que no exista otro usuario con la misma dirección.

**Password:** debe tener como mínimo 6 caracteres.



The screenshot shows a web browser window with the title 'wIDE | User Registration'. The address bar displays 'localhost:3000/users/sign\_up'. The page content features a 'Sign up' form with the following fields: 'User name\*', 'Email\*', 'Password\*', and 'Password confirmation'. A blue 'Create User' button is positioned below the form. At the bottom of the page, there are two links: 'Sign in' and 'Forgot your password?'.

**Figura 10.1:** Formulario de registro de usuarios

Un usuario debe ser activado por un administrador antes de poder ingresar al sistema. El sistema notificará automáticamente la activación de la cuenta, mediante el envío de un e-mail a la dirección de correo electrónico asociada.

## Modificación de datos del usuario

Si se desean modificar datos o preferencias del usuario, es necesario acceder al formulario de configuración de la cuenta (Figura 10.2). Para esto se debe hacer click sobre el vínculo “Account Settings”, ubicado en la barra de navegación, a la derecha.

Desde este formulario es posible cambiar la dirección de e-mail del usuario, su contraseña, y el tema de colores utilizado por el editor de texto (“Ace theme”).

Por seguridad se requiere el ingreso de la contraseña actual para realizar cualquier cambio.

## 10.2. Listado de proyectos

Al ingresar al sistema, se presenta una pantalla (Figura 10.3) que contiene dos tablas.

**Account Information**

User name\* gkleiman

Email\* gaston.kleiman@gmail.com  
To change your avatar, create an account in [Gravatar](#) using this email address.

Ace theme Pastel on dark

Password  
Leave blank if you don't want to change it.

Password confirmation

Current password\*  
We need your current password to confirm your changes.

[Update User](#)

**Figura 10.2:** Formulario de configuración de la cuenta de un usuario

Una con proyectos propios, y otra con los ajenos a los que se tiene acceso como colaborador.

**My Projects**

Name	Project Type	Status	
<a href="#">kiss</a>	AVR C	Ready	
<a href="#">wide-doc</a>	LaTeX	Ready	
<a href="#">wide-graphics</a>	graphviz	Ready	

[Crear un proyecto](#) → [Add a New Project](#)

**Projects from others**

Owner	Name	Repository URL	
pepe	rabbitmq-c	<a href="http://hg.wide.labi.fi.uba.ar/pepe/rabbitmq-c">http://hg.wide.labi.fi.uba.ar/pepe/rabbitmq-c</a>	

**Figura 10.3:** Listado de proyectos

A través de estos listados se pueden crear, eliminar y duplicar proyectos.

## Creación de un proyecto

Haciendo click en “Add a New Project” al final de la tabla de proyectos propios, se accede al formulario para la creación de un nuevo proyecto (Figura 10.4).

The screenshot shows a web browser window with the address bar at `localhost:3000/projects/new`. The page title is "wIDE | gkleiman | New Project". The form itself is titled "New Project" and contains the following elements:

- Name\***: A text input field.
- Project type\***: A dropdown menu currently showing "LaTeX".
- Public**: An unchecked checkbox.
- Scm\***: A dropdown menu currently showing "Mercurial".
- Repository URL:**: A text input field with a note above it: "Enter a URL below only if you want to clone an external repository." Below the input field are two example URLs:  
`http[s]://[user[:pass]]@[host[:port]]/[path][:#revision]`  
`ssh://[user[:pass]]@[host[:port]]/[path][:#revision]`
- Create Project**: A blue button at the bottom right of the form.

**Figura 10.4:** Formulario para la creación de un nuevo proyecto

El formulario está compuesto por los siguientes campos:

**Name:** es el nombre a través del que se identificará al proyecto. Puede estar compuesto sólo por caracteres alfanuméricos y guiones (normal y bajo).

**Project type:** es el tipo de proyecto. Este definirá el proceso de compilación, y podrá tener constantes y una estructura de directorio, que serán copiadas al nuevo proyecto.

**Public:** si se tilda este campo, el repositorio creado será público. Esto significa que cualquier persona que conozca la URL del repositorio podrá tener acceso de lectura, aunque no sea colaborador, ni cuente con un usuario en el sistema.

Por más que el proyecto sea público, sólo el dueño y los colaboradores podrán escribir y modificar el repositorio.

**SCM:** sistema de control de versiones a utilizar.

**Repository URL:** en caso de completar este campo, el repositorio del proyecto creado será una copia del apuntado por la URL ingresada.

## Eliminación de un proyecto

Es posible eliminar un proyecto haciendo click en el tarro de basura, ubicado en el extremo derecho de la fila correspondiente al mismo (para más detalle ver la figura 10.3).

Para evitar la eliminación accidental, se abrirá un diálogo pidiéndole al usuario que confirme la acción a realizar.

## Duplicación de un proyecto

Todo proyecto propio, o en el que se sea un colaborador, puede ser duplicado. Al duplicar un proyecto, se creará uno nuevo, que contendrá un clon del repositorio del original. De esta manera se mantendrá el historial de versiones.

Esta acción se inicia al hacer click en la flecha, ubicada a la izquierda del tarro de basura (para más detalle ver la figura 10.3).

## 10.3. Entorno de desarrollo

Al hacer click en una fila del listado de proyectos propios (Figura 10.3), se accede al entorno de desarrollo (Figura 10.5).

Este está conformado por los siguientes componentes:

**Barra de navegación:** muestra la ubicación actual, y contiene enlaces a distintas páginas del sistema.

**Barra de herramientas:** contiene botones para realizar distintas operaciones.

**Árbol de exploración:** permite explorar y manipular los archivos y directorios del repositorio.

**Tabla con resultados de la compilación:** contiene una tabla con el listado de los mensajes producto del proceso de compilación.

**Paneles de edición de archivos**



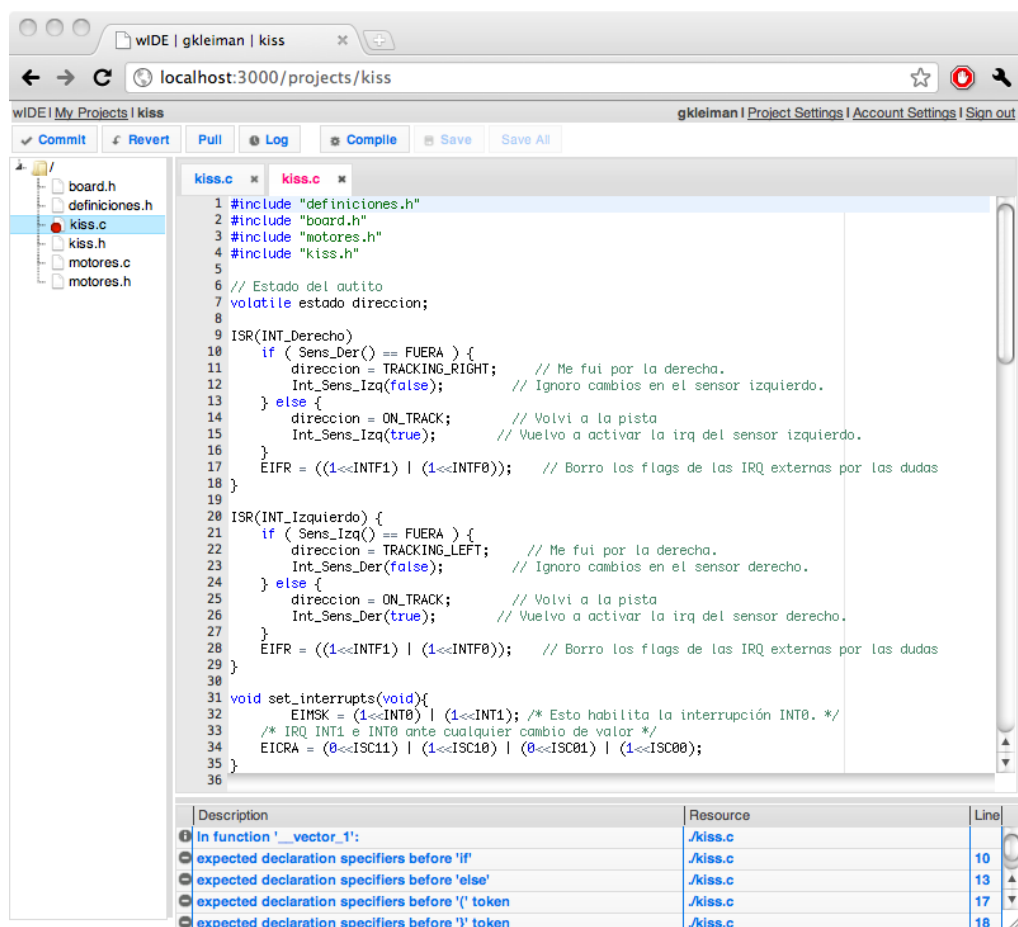


Figura 10.5: Entorno de desarrollo

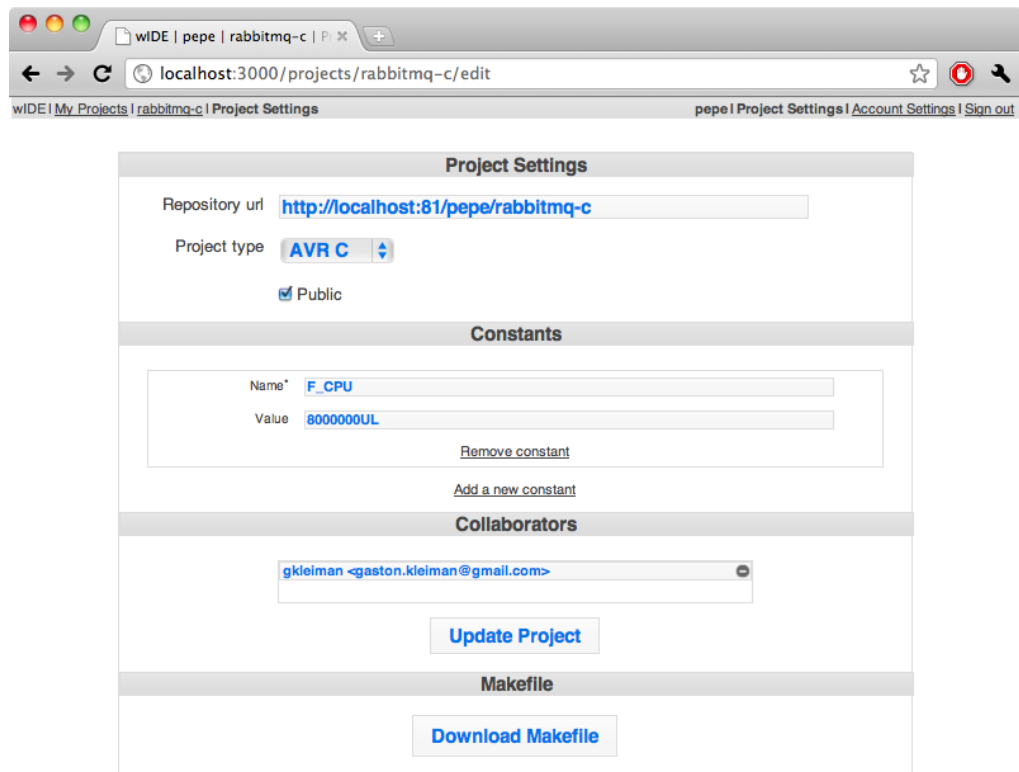
## Preferencias de un proyecto

Haciendo click en “Project Settings” en la barra de navegación, se accede a la configuración del proyecto (Figura 10.6).

En esta se encuentra la URL (“Repository url”) requerida para acceder externamente al repositorio, utilizando un cliente SCM.

A su vez es posible definir si el proyecto es público o no, administrar las constantes (utilizadas para personalizar el proceso de compilación) y los colaboradores.

Finalmente se tiene la opción de descargar el **Makefile** utilizado por el sistema para la compilación.



The screenshot shows a web browser window with the address bar at `localhost:3000/projects/rabbitmq-c/edit`. The page title is "Project Settings". The form contains the following sections:

- Repository url:** `http://localhost:81/pepe/rabbitmq-c`
- Project type:** `AVR C` (dropdown menu)
- Public:** ☒
- Constants:**
  - Name: `F_CPU`
  - Value: `8000000UL`
  - Buttons: [Remove constant](#), [Add a new constant](#)
- Collaborators:**
  - Input field: `gkleiman <gaston.kleiman@gmail.com>`
  - Button: [Update Project](#)
- Makefile:**
  - Button: [Download Makefile](#)

**Figura 10.6:** Formulario de configuración de un proyecto

## Estado de los archivos

En el árbol de exploración, los directorios son representados con una carpeta, y los archivos por una hoja.

Para indicar el estado de un archivo, se superponen imágenes sobre el mismo. En la tabla 10.1 se indica a qué estado corresponde cada imagen.

## Creación de archivos y directorios

La creación de archivos y directorios se realiza a través de un menú contextual (Figura 10.7), el cual se despliega al hacer click con el botón derecho sobre el directorio que los contendrá.

## Operaciones de control de versiones

A través del menú contextual (Figura 10.7) también es posible realizar operaciones relacionadas con el control de versiones

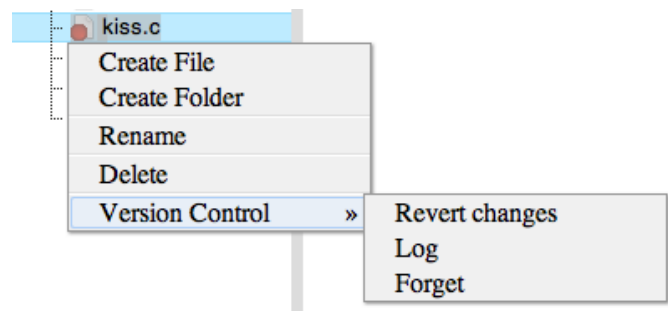


Figura 10.7: Menú contextual

A continuación se detallan las operaciones realizables, y a qué ítem del menú corresponden.

**Agregar un archivo al repositorio:** mediante la opción “Add”.

**Eliminar un archivo del repositorio:** “Forget”. No borra el archivo, sino que se deja de tenerlo en cuenta para el control de versiones.

**Descartar los cambios hechos desde el último commit:** “Revert changes”.

**Ir al historial de modificaciones de un archivo:** “Log”.

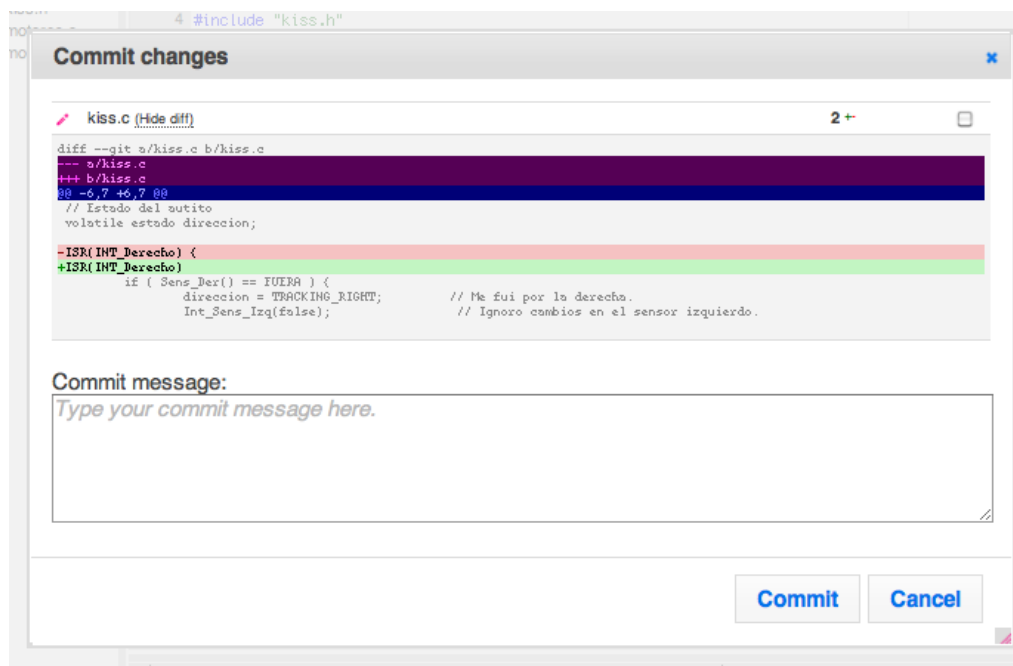
Ícono	Estado
	Agregado
	Borrado
	Ignorado
	Modificado
	Conflictos resueltos
	Con conflictos
	No versionado

Tabla 10.1: Iconos y sus correspondientes estados

Referencia de iconos y sus correspondientes estados en el sistema SCM

## Commit de cambios realizados

Cuando se hayan realizado cambios a archivos versionados, se habilitará el botón que dice “Commit”.



**Figura 10.8:** Diálogo de “commit”

Al apretar el botón se abrirá un diálogo (Figura 10.8), a través del cual es posible guardar las modificaciones.

En este se muestra una tabla con una fila por archivo cambiado. Cada fila contiene un número, correspondiente a la cantidad de líneas alteradas. Junto a este se indica mediante signos y colores la proporción entre líneas agregadas y eliminadas.

Se puede ver la diferencia entre el último commit y la versión actual de un archivo a través del enlace “View diff”.

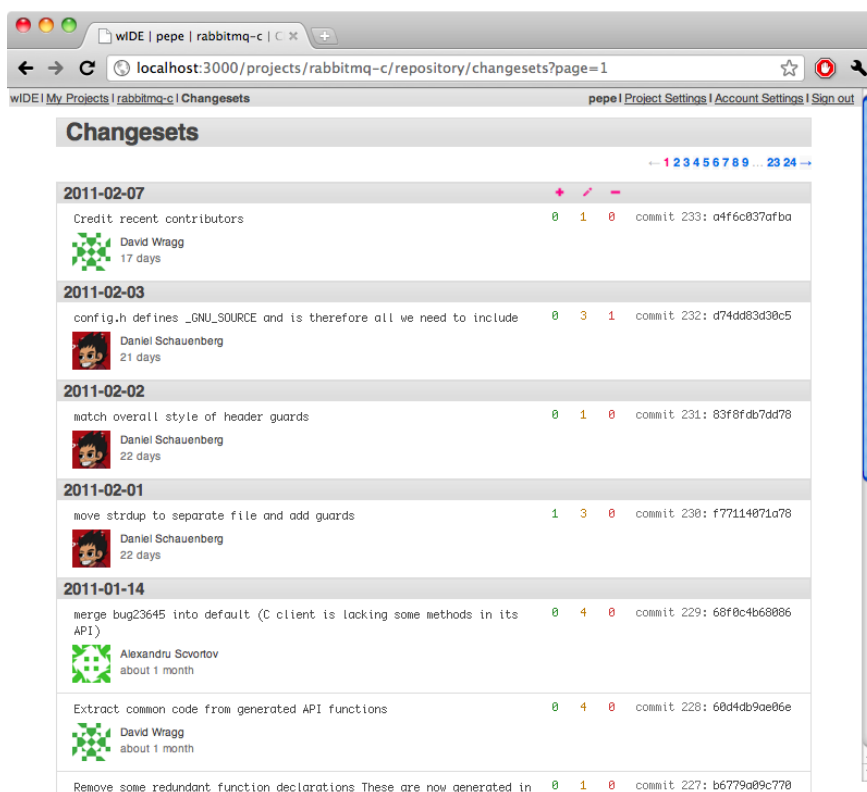
Al final de cada fila se encuentra un cuadrado, el cual se debe tildar si se desean incluir en el “commit” las modificaciones realizadas al archivo correspondiente.

Todo “commit” debe incluir un mensaje describiendo los cambios realizados.

Mediante el botón “Revert” de la barra de herramientas se abre un diálogo con aspecto y funcionalidad similares al de “Commit”, pero que permite deshacer las modificaciones en vez de guardarlas.

## Visualización del historial de cambios

Mediante el botón “Log” se accede al historial de cambios del repositorio (Figura 10.9).



**Figura 10.9:** Historial de cambios (changesets)

En este se muestra la descripción, número de revisión, identificador único, nombre del autor y su avatar, y la cantidad de archivos agregados, modificados y eliminados de cada conjunto de cambios (“changeset”)

Al hacer click en una descripción o un identificador, se abrirá una página con más detalles (Figura 10.10).

En esta se listan todos los archivos modificados, junto un resumen de la cantidad de cambios introducidos por la revisión.

A través del enlace “Update to this revision” se puede cambiar el contenido del repositorio por el correspondiente a la revisión visualizada. Se debe tener en cuenta que esta acción descarta todo cambio actual no grabado mediante un “commit”.

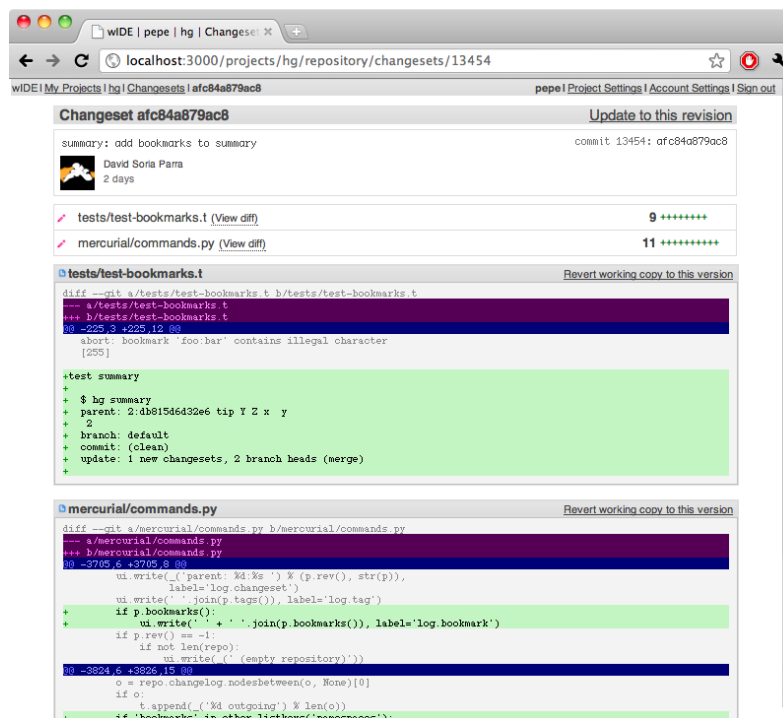


Figura 10.10: Vista detalle de un cambio

Los enlaces “Revert working copy to this version” no alteran todo el repositorio, sino sólo el archivo correspondiente.

## Obtención de cambios desde otro repositorio

El botón “Pull” abrirá un diálogo en el que se puede ingresar la URL de otro repositorio, desde el cual se descargarán y aplicarán cambios.

En el servidor esta tarea es encolada y ejecutada de forma asincrónica, por lo que el sistema notificará el inicio y fin de la misma.

## Compilación

Apretando el botón “Compile” se guardan todos los archivos editados, y se solicita al servidor la compilación del proyecto. Esta tarea no es trivial, y suele tener larga duración, por lo que también se realiza de forma asincrónica.

Una vez finalizado el proceso, el sistema notificará los resultados. Si este fue exitoso, se descargará el binario obtenido. En cambio, en caso de que

no haya terminado correctamente, se listarán los errores producidos.





## Conclusiones

El cliente se mantuvo involucrado y alcanzable durante todo el proyecto. La elección de una metodología de desarrollo ágil con iteraciones de corta duración fue apropiada, ya que permitió aprovechar esta buena predisposición, mediante la obtención de feedback frecuente,

Las estimaciones fueron correctas, ya que se pudo implementar sin demoras toda la funcionalidad planificada, e incluso agregar algunos de los cambios solicitados por el cliente.

Para la elección de los frameworks y componentes de terceros incorporados al sistema se tuvo en cuenta el tipo de licencia (aceptándose sólo software libre), su grado de madurez, y el tamaño y actividad de su comunidad de desarrollo. Esto permitió obtener soporte en los casos en que la documentación no era suficiente.

Al trabajar con algunos proyectos de software libre se encontraron algunas fallas, las cuales fueron arregladas, contribuyendo devuelta los cambios necesarios. *Mercurial*, el SCM utilizado por el “Club de Robótica” y por grandes proyectos de *Mozilla*, se encuentra entre los proyectos a los que se enviaron parches. En todos los casos la respuesta de la comunidad de desarrolladores fue positiva, incorporándolos a los repositorios oficiales [46] [47] [48] [49] [50] [51] [52] [53] [54].

El objetivo inicial del proyecto era desarrollar una IDE Web que permita compilar programas para diversas arquitecturas, teniendo en cuenta principalmente el soporte para las placas de prototipado comercializados por *mbed*. Para lograr dicho soporte es necesario utilizar una biblioteca privada publicada por ellos en la compilación de los proyectos.

Sin embargo, la falta de documentación sobre los pasos necesarios para incluir esta biblioteca en el proceso de compilación impidieron el cumpli-

miento de este objetivo.

Esto dio origen a un proyecto dentro del “Club de Robótica”, que tiene como objetivo desarrollar una plataforma libre, similar a la comercializada por *mbed*. En este se planea utilizar *wIDE* como IDE (reemplazando al “compilador online” de *mbed*), diseñar una placa propia, e implementar una biblioteca libre.

El sistema fue probado exitosamente con los siguientes tipos de proyectos:

1. C y C++
2. AVR C
3. Graphviz
4. L<sup>A</sup>T<sub>E</sub>X

Actualmente *wIDE* se encuentra desplegado en el servidor del Club, y los miembros del mismo se muestran entusiasmados, adoptándolo en sus desarrollos.

# Bibliografía

- [1] *Club de Robótica*, <http://www.clubderobotica.com.ar/>, [http://labi.fi.uba.ar/club\\_robotica.html](http://labi.fi.uba.ar/club_robotica.html) (última visita: 16/02/2011).
- [2] *Rapid Prototyping for Microcontrollers*, <http://mbed.org/> (última visita: 16/02/2011).
- [3] *Mercurial*, <http://mercurial.selenic.com/> (última visita: 16/02/2011).
- [4] *Redmine*, <http://www.redmine.org/> (última visita: 16/02/2011).
- [5] *git*, <http://git-scm.com/> (última visita: 16/02/2011).
- [6] *Vim*, <http://www.vim.org/> (última visita: 16/02/2011).
- [7] *L<sup>A</sup>T<sub>E</sub>X*, <http://www.latex-project.org/> (última visita: 16/02/2011).
- [8] *Ruby on Rails 3*, <http://rubyonrails.org/> (última visita: 16/02/2011).
- [9] *RSpec*, <http://rspec.info/> (última visita: 16/02/2011).
- [10] *jQuery v1.5*, <http://jquery.com/> (última visita: 16/02/2011).
- [11] *jQuery UI v1.8.9*, <http://jqueryui.com/> (última visita: 16/02/2011).
- [12] *Typus*, <https://github.com/fesplugas/typus/wiki> (última visita: 16/02/2011).
- [13] *Devise: Flexible authentication solution for Rails with Warden*, <https://github.com/plataformatec/devise> (última visita: 16/02/2011).
- [14] *Comparison of JavaScript-based source code editors*, [http://en.wikipedia.org/wiki/Comparison\\_of\\_JavaScript-based\\_source\\_code\\_editors](http://en.wikipedia.org/wiki/Comparison_of_JavaScript-based_source_code_editors) (última visita: 16/02/2011).

- [15] *Resque*, <https://github.com/defunkt/resque> (última visita: 16/02/2011).
- [16] *Beanstalkd*, <http://kr.github.com/beanstalkd/> (última visita: 16/02/2011).
- [17] *mercurial-server*, <http://www.lshift.net/mercurial-server.html> (última visita: 16/02/2011).
- [18] *Apache 2*, <http://httpd.apache.org/> (última visita: 16/02/2011).
- [19] *lighttpd*, <http://www.lighttpd.net/> (última visita: 16/02/2011).
- [20] *Firefox*, <http://www.mozilla.com/en-US/> (última visita: 16/02/2011).
- [21] *Google Chrome*, <http://www.google.com/chrome> (última visita: 16/02/2011).
- [22] *Safari*, <http://www.apple.com/safari/>, (última visita: 16/02/2011).
- [23] *nginx*, <http://nginx.org/> (última visita: 16/02/2011).
- [24] *SQLite*, <http://www.sqlite.org/> (última visita: 16/02/2011).
- [25] *mod\_xsendfile*, [https://tn123.org/mod\\_xsendfile/](https://tn123.org/mod_xsendfile/) (última visita: 16/02/2011).
- [26] *Phusion Passenger*, <http://www.modrails.com/install.html> (última visita: 16/02/2011).
- [27] *Capistrano*, <https://github.com/capistrano/capistrano> (última visita: 16/02/2011).
- [28] *RubyGems*, <http://rubygems.org/> (última visita: 16/02/2011).
- [29] *Bundler*, <http://gembundler.com/> (última visita: 16/02/2011).
- [30] *RequireJS*, <http://requirejs.org/>, (última visita: 16/02/2011).
- [31] Douglas Crockford, 2008, *Javascript: The Good Parts*, O'Reilly Media, Inc..
- [32] *jQuery UI CSS Framework*, <http://docs.jquery.com/UI/Theming/API> (última visita: 16/02/2011).
- [33] *JSLint*, <http://www.JSLint.com> (última visita: 16/02/2011).

- [34] *jsTree*, <http://www.jstree.com/> (última visita: 16/02/2011).
- [35] *Ace*, <http://ace.ajax.org/> (última visita: 16/02/2011).
- [36] *jLayout*, <http://www.bramstein.com/projects/jlayout/> (última visita: 16/02/2011).
- [37] *Representational State Transfer (REST)*, [http://en.wikipedia.org/wiki/Representational\\_State\\_Transfer](http://en.wikipedia.org/wiki/Representational_State_Transfer) (última visita: 16/02/2011).
- [38] Martin Fowler, 2002, *Patterns of Enterprise Application Architecture*, Addison-Wesley Professional.
- [39] *Haml*, <http://haml-lang.com/> (última visita: 16/02/2011).
- [40] *Delayed Job*, [https://github.com/collectiveidea/delayed\\_job](https://github.com/collectiveidea/delayed_job) (última visita: 16/02/2011).
- [41] *YAML: YAML Ain't Markup Language*, <http://www.yaml.org/> (última visita: 16/02/2011).
- [42] *HgRedmine*, <http://hg.9thsoft.com/repos/hgredmine> (última visita: 16/02/2011).
- [43] *JSON*, <http://www.json.org/> (última visita: 19/02/2011).
- [44] *ERB – Ruby Templating*, <http://www.ruby-doc.org/stdlib/libdoc/erb/rdoc/classes/ERB.html> (última visita 21/02/2011).
- [45] *Active Record – Object-relational mapping put on rails*, [http://api.rubyonrails.org/files/activerecord/README\\_rdoc.html](http://api.rubyonrails.org/files/activerecord/README_rdoc.html) (última visita: 21/02/2011).
- [46] *mercurial/crew: 104c9ed93fc5*, <http://hg.intevation.org/mercurial/crew/rev/104c9ed93fc5> (última visita: 24/02/2011).
- [47] *Commit bedcd128 to fesplugas's typus - GitHub*, <https://github.com/fesplugas/typus/commit/bedcd128> (última visita: 24/02/2011).
- [48] *Commit 5875997cc to fesplugas's typus - GitHub*, <https://github.com/fesplugas/typus/commit/5875997cc> (última visita: 24/02/2011).
- [49] *Commit ba4204 to ajaxorg's ace - Github*, <https://github.com/ajaxorg/ace/commit/ba4204> (última visita: 24/02/2011).

- 
- [50] *Commit 666f011 to ajaxorg's ace - Github*, <https://github.com/ajaxorg/ace/commit/666f011> (última visita: 24/02/2011).
  - [51] *Commit a04cee8dc to ajaxorg's ace - Github*, <https://github.com/ajaxorg/ace/commit/a04cee8dc> (última visita: 24/02/2011).
  - [52] *Commit eb35995e to ajaxorg's ace - Github*, <https://github.com/ajaxorg/ace/commit/eb35995e> (última visita: 24/02/2011).
  - [53] *Commit f6f97918 to ajaxorg's ace - Github*, <https://github.com/ajaxorg/ace/commit/f6f97918> (última visita: 24/02/2011).
  - [54] *Commit f804a62f to ajaxorg's ace - Github*, <https://github.com/ajaxorg/ace/commit/f804a62f> (última visita: 24/02/2011).

# Listado de código

El código del sistema se encuentra disponible en un repositorio público `git` accesible bajo la URL: <http://github.com/gkleiman/wide>, y en el CD entregado junto con esta documentación.





## Carta de aprobación del cliente