SRS

TC3002B: Desarrollo de Aplicaciones Avanzadas de Ciencias Computacionales

Group 501

TC3003B: Implementación de Redes de Área Amplia y Servicios Distribuidos

Group 502

Sylvia Fernanda Colomo Fuente
A01781983

Francisco Flamenco Andrade
A01732742

Natalia Maya Bolaños
A01781992

Shaul Zayat Askenazi
A01783240

Grupo de Desarrollo: CloudStripe Team

Establecimiento: 02/06/2025

Última Modificación: 02/06/2025

V.1.0

Bajo la instrucción de:

Victor Manuel De La Cueva

Jorge Rodríguez Ruiz

Gerardo Jesús Camacho González

Edith Carolina Arias Serna

Marco Antonio González

Octavio Navarro Hinojosa

1. Introducción

1.1 Propósito

Este documento tiene como propósito definir de manera clara y detallada los requerimientos funcionales y no funcionales del sistema web denominado MARS MIPS Generator, una aplicación que permitirá a los usuarios escribir código en lenguaje C-, compilarlo a código MIPS y visualizar o descargar el resultado directamente desde un entorno web.

El sistema está orientado a dos tipos de usuarios:

- Usuarios finales, quienes podrán interactuar con el compilador.
- Administradores, quienes tendrán acceso a funcionalidades de auditoría y gestión de cuentas.

Este SRS está destinado a desarrolladores, diseñadores, testers, y stakeholders del proyecto.

1.2 Alcance

El sistema proporcionará una plataforma web accesible que ofrezca las siguientes funcionalidades principales:

- Registro y autenticación de usuarios.
- Ingreso de código en lenguaje C- y conversión inmediata a MIPS.
- Visualización y descarga del código MIPS generado.
- Interfaz adaptativa y accesible para diferentes dispositivos.
- Registro de inicio de sesión y actividad del usuario (input/output) accesible para administradores.
- Gestión de usuarios por parte del administrador, incluyendo eliminación de cuentas.

El sistema no contempla almacenamiento de proyectos, historial de código ni ejecución en cola, dado que la compilación es inmediata y el código no se guarda en la plataforma.

1.3 Definiciones, acrónimos y abreviaturas

Término/Acrónimo	Definición
C-	Lenguaje de programación simplificado basado en C, utilizado con fines educativos.
MIPS	Microprocessor without Interlocked Pipeline Stages, lenguaje ensamblador para arquitectura RISC.
RU	Requerimiento de Usuario.
RA	Requerimiento de Administrador.
RNF	Requerimiento No Funcional.
RE	Restricción.
SRS	Software Requirements Specification (Especificación de Requerimientos de Software).

2. Descripción general

2.1 Perspectiva del producto

El sistema MARS MIPS Generator es una aplicación web que servirá como interfaz para un compilador previamente desarrollado que traduce código fuente en lenguaje C- a MIPS. Esta plataforma se ofrecerá como un servicio accesible desde navegadores web, permitiendo al usuario realizar todo el proceso de compilación sin necesidad de instalar software adicional.

El sistema se desplegará en una nube privada con infraestructura física localizada en el campus universitario, usando servicios proporcionados por OpenStack, lo que permite el aprovisionamiento dinámico de recursos, escalabilidad y control total del entorno de ejecución.

La plataforma se compone de:

- Una interfaz de usuario web (frontend).
- Un backend que gestiona sesiones, autenticación y compilación.
- Un componente de administración para la supervisión y gestión de usuarios.
- Un motor de compilación que traduce código C- a MIPS y devuelve el resultado.

2.2 Funcionalidad del producto

Las funcionalidades principales del sistema incluyen:

Funciones del Usuario:

- RU-01: Registro de cuentas.
- RU-02: Inicio de sesión.
- RU-03: Ingreso de código C- mediante un editor en línea.
- RU-04: Visualización inmediata del código MIPS resultante.
- RU-05: Descarga del código MIPS.
- RU-06: Accesibilidad (conformidad con pautas WCAG).
- RU-07: Interfaz usable y adaptativa para múltiples dispositivos.

Funciones del Administrador:

- RA-01: Visualización de logins por hora.
- RA-02: Registro detallado de actividades (input/output por usuario).
- RA-03: Eliminación de cuentas de usuario.
- (Opcional futuro): Estadísticas de uso, manejo de errores del compilador, gestión avanzada de usuarios.

2.3 Características de los usuarios

Se distinguen dos tipos de usuarios:

- **Usuarios generales**: Pueden registrarse, iniciar sesión, introducir código C-, compilar, visualizar y descargar el código en MIPS. Se espera que tengan conocimientos básicos de programación en C-.

- **Administradores**: Usuarios autorizados para monitorear el uso de la plataforma, consultar logs, gestionar cuentas. Se espera que sean personal técnico o académico con responsabilidades de gestión del entorno.

2.4 Restricciones

- RE1: El sistema debe estar alojado en una infraestructura privada de nube OpenStack.
- RE2: La ejecución de código debe realizarse de forma inmediata, sin sistema de colas.
- RE3: El código fuente no se almacena en la plataforma una vez que se genera el resultado.
- RE4: El sistema debe funcionar en navegadores modernos (Chrome, Firefox, Edge).
- R5: El proyecto debe ser desarrollado como una Aplicación Web.
- RE6: La Aplicación Web debe ser desarrollada por un equipo de 4 personas.
- RE7: El desarrollo de la Aplicación Web debe de estar completado para el viernes 6 de junio de 2025.
- RE8: Se usará GitFlow para el desarrollo.

2.5 Suposiciones y dependencias

- Se asume que los usuarios tienen conexión a Internet y un navegador compatible.

- Se asume que el motor de compilación C- a MIPS ya está desarrollado y probado antes del despliegue web.

- Se depende de la correcta configuración de servicios en la nube OpenStack, como instancias, almacenamiento y red privada.

- El sistema depende de componentes de software como servidores web, bases de datos para autenticación y almacenamiento temporal, y controladores de sesión.

3. Requerimientos específicos

3.1 Requerimientos funcionales

RU-01: Registro de cuenta

- El sistema deberá permitir a un nuevo usuario registrarse con un correo electrónico y contraseña válidos.
- Se validará que el correo no esté registrado previamente.

RU-02: Inicio de sesión

- El sistema deberá permitir a los usuarios registrados iniciar sesión mediante sus credenciales.
- Al iniciar sesión exitosamente, se redirigirá al usuario al editor de código.
- En caso de error, se mostrará un mensaje claro.

RU-03: Ingreso de código C-

- El sistema debe mostrar un editor en línea para que el usuario ingrese código fuente en C-.
- El editor debe permitir copiar y pegar el contenido fácilmente.

RU-04: Ver resultado del compilador

- El sistema deberá mostrar en pantalla el resultado de la compilación (código MIPS).
- Si hay errores en el código C-, se deberá mostrar un mensaje explicando el fallo.

RU-05: Descarga del código MIPS

- El sistema deberá permitir al usuario descargar el resultado del compilador en un archivo `.asm` o similar.

RU-06: Accesibilidad

- El sistema deberá ser navegable mediante teclado y compatible con lectores de pantalla.

RU-07: Usabilidad

- La aplicación deberá ser responsiva y funcional en pantallas de diferentes tamaños (desktop, tablet, móvil).
- El diseño deberá ser claro, intuitivo y minimalista para facilitar la interacción.

RA-01: Visualización de logins

- El sistema deberá permitir al administrador ver un log con la hora de inicio de sesión de cada usuario.
- El log mostrará al menos: hora, fecha, correo electrónico del usuario.

RA-02: Registro de actividad del usuario

- El sistema deberá registrar cada compilación realizada por un usuario, guardando:
    - Fecha y hora.
    - Usuario.
    - Código fuente introducido.
    - Salida generada (código MIPS).
- Esta información deberá ser visible para los administradores desde un panel.

RA-03: Eliminación de cuentas de usuario

- Una cuenta inactiva no podrá iniciar sesión ni acceder a ninguna funcionalidad de la plataforma.

- Por políticas de seguridad y auditoría, los datos de la cuenta (incluyendo historial de uso, compilaciones, y logs) no serán eliminados físicamente de la base de datos, sino que se conservarán en estado inactivo.

3.2 Requerimientos no funcionales

RNF-01: Rendimiento

- El sistema deberá compilar código en un tiempo no mayor a 5 segundos, para entradas de tamaño promedio (<300 líneas).

RNF-02: Seguridad

- Las contraseñas deberán ser almacenadas con hashing seguro (ej. bcrypt).
- Las sesiones deben estar protegidas con tokens seguros (ej. JWT, cookies con HttpOnly y Secure).
- Se deberá implementar protección contra ataques comunes: XSS, CSRF, inyecciones.

RNF-03: Disponibilidad

- El sistema deberá estar disponible al menos 95% del tiempo (mientras la infraestructura OpenStack esté operativa).

RNF-04: Mantenibilidad

- El código fuente deberá estar documentado y estructurado en módulos que faciliten futuras modificaciones.
- Se usará control de versiones (ej. Git) y repositorio centralizado.

RNF-05: Compatibilidad

- El sistema debe ser compatible con navegadores modernos: Chrome, Firefox, Edge.
- Debe funcionar correctamente sin necesidad de plugins.

4. Apéndices y referencias técnicas

4.1 Infraestructura del sistema

El sistema estará desplegado en una nube privada provista por la infraestructura física del campus universitario. Esta nube utiliza OpenStack como plataforma de orquestación y administración de servicios de nube. Los servicios críticos (autenticación, ejecución del compilador, bases de datos y frontend) estarán contenidos o virtualizados de acuerdo con las mejores prácticas de despliegue seguro.

- **Plataforma de nube:** OpenStack
- **Tipo de despliegue:** Nube privada en infraestructura on-premise
- **Tecnologías servidoras:** Linux (Debian Server), Nginx, servicios OpenStack (Horizon)
- **Lenguaje backend:** Python (FastAPI, Swagger).
- **Lenguaje frontend:** React.
- **Base de datos:** MongoDB
- **Control de versiones:** Git (y GitHub).

4.2 Compilador C- a MIPS

La lógica de compilación está basada en un compilador desarrollado previamente que convierte el código en lenguaje C- a instrucciones MIPS. Esta lógica será empaquetada como un servicio interno que recibirá código fuente y devolverá el resultado compilado.

- **Lenguaje fuente:** C- (subconjunto simplificado de C)
- **Lenguaje destino:** MIPS Assembly
- **Modo de compilación:** Inmediato (no por lotes)
- **Entrada:** Código fuente C-
- **Salida:** Código MIPS o mensajes de error

4.3 Consideraciones de seguridad

- Las contraseñas deben almacenarse con algoritmos seguros de hash (bcrypt o equivalente).
- Las sesiones deben manejarse con mecanismos como JWT o cookies HttpOnly y Secure.
- El sistema debe prevenir ataques comunes como:
    - **Cross-Site Scripting (XSS)**
    - **Cross-Site Request Forgery (CSRF)**
    - **Inyección de código (Command Injection)**
    - **Accesos no autorizados (Broken Access Control)**

4.4 Consideraciones de accesibilidad y usabilidad

- Se recomienda seguir las pautas de diseño inclusivo y el uso de contrastes adecuados, navegación por teclado y etiquetas semánticas en HTML.
- La interfaz debe ser responsiva y funcional en dispositivos móviles, tablets y escritorios.

4.5 Limitaciones conocidas

- No se implementará persistencia de proyectos por parte del usuario (no se guardan múltiples archivos ni historiales personales).
- El sistema se centrará en la funcionalidad directa: escribir, compilar, ver y descargar.

4.6 Tecnología de base de datos: Justificación del uso de MongoDB

Para la gestión de datos en la plataforma web se ha elegido **MongoDB**, una base de datos NoSQL orientada a documentos. Esta decisión se toma en lugar de utilizar una base de datos relacional como MySQL o PostgreSQL por las siguientes razones:

Ventajas de MongoDB para este proyecto

1. **Esquema flexible:**
   La naturaleza dinámica de los documentos JSON en MongoDB permite almacenar registros de actividad con estructuras que pueden variar entre usuarios o versiones, como:
   - Código fuente (`input`) de longitud variable
   - Salida generada (`output`) de distinta naturaleza
   - Logs de sesión con timestamps diversos
     Una base de datos relacional requeriría esquemas más rígidos, dificultando la evolución del sistema.

2. **Alta velocidad de inserción y lectura:**
   MongoDB está optimizado para registrar datos rápidamente sin necesidad de realizar transacciones complejas o mantener relaciones estrictas. Esto es ideal para registrar:
   - Ejecuciones frecuentes del compilador
   - Registros de login por hora

3. **Desacoplamiento de datos y simplicidad:**
   Al no existir relaciones complejas (no hay múltiples tablas con claves foráneas), se facilita el desarrollo y mantenimiento. Por ejemplo:
   - El historial de ejecución de un usuario puede almacenarse como un documento completo, sin necesidad de hacer joins.

4. **Escalabilidad:**
   MongoDB permite escalar horizontalmente con mayor facilidad que muchas bases de datos relacionales, lo que es ideal si el sistema crece a más usuarios o instituciones.

5. **Integración con tecnologías modernas:**
   MongoDB se integra de manera natural con aplicaciones basadas en Node.js, Express y React, tecnologías comúnmente usadas para construir este tipo de web apps.

Por qué no se utilizó una base de datos relacional

- **Sobrecoste en complejidad:** Bases como MySQL o PostgreSQL requieren una definición estricta del esquema, relaciones y normalización, lo que implica mayor complejidad para un sistema cuya estructura de datos es relativamente simple y centrada en documentos autónomos. Además, requiere más tiempo y esfuerzo diseñar un esquema para este proyecto.

- No se requieren transacciones complejas ni relaciones entre tablas.

- **Escasa variabilidad de tipos de entidades:** El sistema maneja principalmente dos entidades clave: usuarios y sus actividades. No hay necesidad de manejar muchas tablas ni relaciones complejas.

- **Escalabilidad limitada por diseño:** Las bases relacionales tradicionales no escalan horizontalmente de forma nativa sin soluciones adicionales.

4.7 Accesibilidad (WCAG)

La aplicación fue desarrollada con base en las pautas WCAG 2.1, nivel **AA**, priorizando:

- Uso adecuado de etiquetas en formularios.

- Mensajes de error comprensibles y visibles.

- Contraste suficiente entre texto y fondo.

- Estructura lógica del contenido.