ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

Asistente virtual para la comprensión de errores de programación

PROYECTO INTEGRADOR

Previo la obtención del Título de:

Ingeniero en Ciencias Computacionales

Presentado por:

Valeria Denisse Barzola Armanza

Alex Andrés Vélez Llaque

GUAYAQUIL - ECUADOR

Año: 2023

DEDICATORIA

El presente proyecto lo dedico a mi familia, mis padres, mis hermanas, quienes han sido mi mayor soporte durante toda mi carrera universitaria. A todos mis profesores quienes han enriquecido mi formación académica con su dedicación y conocimiento, y que han sido clave en mi desarrollo profesional. A todos los amigos que hice durante esta etapa, por haber sido una guía y un apoyo constante en mi camino. Especialmente a Alex, cuya dedicación y soporte han sido clave para este y muchos proyectos.

Valeria Barzola A.

DEDICATORIA

El presente proyecto lo dedico a mi familia, mis padres, mis tíos, quienes han estado siempre ahí para brindarme todo su apoyo y que nunca me falte nada. A mis amigos: Joangie, Carlos, Joseph y Jaime, quienes llegaron para hacer este pedacito de mi vida un camino mucho más ameno. Y en especial a Valeria, mi amiga, compañera de carrera, de tesis, quien estuvo siempre dispuesta a dar lo mejor de sí para cumplir nuestros objetivos en la carrera.

Alex Vélez Ll.

AGRADECIMIENTOS

Nuestro más sincero agradecimiento al profesor M.Sc. Rafael Bonilla por su guía y apoyo para el desarrollo de este proyecto y la Dra. Marisol Villacrés por su guía en la escritura del presente documento.

DECLARACIÓN EXPRESA

“Los derechos de titularidad y explotación, nos corresponde conforme al reglamento de propiedad intelectual de la institución; *Valeria Barzola Armanza y Alex Vélez Llaque* damos nuestro consentimiento para que la ESPOL realice la comunicación pública de la obra por cualquier medio con el fin de promover la consulta, difusión y uso público de la producción intelectual"

Valeria Barzola Armanza

Alex Vélez Llaque

EVALUADORES

LUCIA MARISOL VILLACRES FALCONI

Digitally signed by LUCIA MARISOL VILLACRES FALCONI
Date: 2023.02.14 17:33:42 -05'00'

Ph.D. Lucía Marisol Villacrés
PROFESOR DE LA MATERIA

Firmado electrónicamente por:
RAFAEL IGNACIO BONILLA ARMIJOS

M.Sc. Rafael Bonilla Armijos
PROFESOR TUTOR

RESUMEN

Aprender programación puede ser un proceso complicado para los principiantes, y una de las principales complicaciones es el entendimiento de los errores de compilación que se suelen encontrar a menudo. Este proyecto busca ayudar a programadores principiantes a entender los errores de programación mediante mensajes sencillos de entender, ejemplos y posibles soluciones.

En este trabajo se propuso a EVA, un asistente virtual que ayuda a la comprensión de errores de programación. Para su implementación, se utilizaron metodologías de Diseño Centrado en el Usuario. A través de entrevistas se obtuvo los errores más comunes encontrados por los estudiantes, con los cuales se construyó la primera iteración de la base de conocimientos. Adicionalmente, EVA lleva un registro de las consultas de los estudiantes, que usa para generar estadísticas que puedan ser consultadas por profesores. Con el objetivo de tener bajos costos de infraestructura, se utilizaron tecnologías *serverless* de AWS, y Firebase como capa de persistencia.

Finalmente, se realizaron talleres con 51 estudiantes y reuniones con 3 profesores de Fundamentos de Programación, para conocer sus percepciones acerca de EVA. El 91.5% de los estudiantes indicó que EVA les hubiese sido de ayuda al inicio del semestre, mientras que el 74.5% sintieron que solucionaron los errores de los talleres más rápido. Los profesores concordaron que utilizarían EVA como herramienta de apoyo, y que las estadísticas les ayudarían a conocer el rendimiento de sus estudiantes. Por lo que EVA logra agilizar el proceso de los estudiantes para encontrar soluciones a errores de programación.

Palabras Clave: Errores de programación, Diseño Centrado en el Usuario, Asistente Virtual, Aprendizaje Autónomo.

ABSTRACT

Learning programming can be a complicated process for beginners, and one of the main complications is understanding the compilation errors that are often encountered. This project aims to help beginner programmers understand programming errors through messages simple to understand, examples and possible solutions.

In this work we proposed EVA, a virtual assistant that helps to understand programming errors. User-Centered Design methodologies were used for its implementation. Interviews were conducted to obtain the most common errors encountered by students with which the first iteration of the knowledge base was built. Additionally, EVA keeps a record of student requests, which is used to generate statistics that can be consulted by teachers. To have low infrastructure costs, AWS serverless technologies were used, and Firebase was used as a persistence layer.

Finally, workshops with 51 students and meetings with 3 professors of Programming Fundamentals were held, to learn their perceptions about EVA. 91.5% of the students indicated that EVA would have been helpful to them at the beginning of the semester, while 74.5% felt that they fixed bugs in the workshops faster. The teachers agreed that they would use EVA as a support tool, and that the statistics would help them know their students' performance. In conclusion, EVA manages to speed up the process for students to find solutions to programming errors.

Keywords: *Programming errors, User-Centered Design, Virtual Assistant, Autonomous Learning.*

ÍNDICE GENERAL

ABREVIATURAS

ESPOL	Escuela Superior Politécnica del Litoral
AWS	Amazon Web Services

SIMBOLOGÍA

ms	Milisegundo
req	Requerimiento
USD	Dólar estadounidense
MB	Megabyte

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1

1. INTRODUCCIÓN

1.1 Descripción del problema

El pensamiento computacional es una habilidad fundamental para todos, no solo para los programadores. Según Mark Surman, director ejecutivo de Mozilla, el código se ha convertido en la cuarta alfabetización, por lo que es importante que todos sepan cómo funciona nuestro mundo digital [1].

Debido a la importancia para la sociedad moderna de desarrollar habilidades de programación en los estudiantes, incluso desde edades tempranas, algunos gobiernos como Australia, EE. UU, Brasil y Reino Unido, por ejemplo, han emprendido iniciativas en este ámbito con el apoyo de varias compañías [2].

Aprender programación puede ser un proceso complicado para los principiantes. Deben tener capacidades de memorización, comprensión, resolución de problemas, abstracción y pensamiento lógico, entre otras [3].

Una de las dificultades para aprender programación es el entendimiento de los errores de compilación o ejecución que suelen encontrar a menudo los programadores principiantes. Según Denny et al. [4], los errores de sintaxis es una de las barreras al aprender programación, retrasando la retroalimentación a los estudiantes acerca de la lógica del código desarrollado. De igual manera, Cechinel et al. [5] reportó que uno de los problemas más comunes de programadores principiantes es la falta de capacidad para encontrar errores.

Se realizó una encuesta a 26 estudiantes de pregrado de una materia introductoria a programación, la cual reportó que el 65.4% de los estudiantes intenta resolver los errores solamente leyendo la consola, sin embargo, el 84.6% de los estudiantes encuentran estos errores medianamente o muy complejos de entender. Adicionalmente, en la encuesta, la mayoría de los estudiantes indicaron que uno de sus errores más frecuentes eran errores de sintaxis.

Por estas complicaciones, los estudiantes usualmente requieren la ayuda del profesor o compañeros para solucionar sus errores, pero en un curso con muchos estudiantes, brindar ayuda constante a cada uno de ellos se vuelve una tarea complicada.

1.2. Justificación del problema

En el estudio realizado por Yorah Bosse et al. [6], se puede entender más a fondo las percepciones de los estudiantes, en donde varios de ellos presentan sus frustraciones respecto a errores de sintaxis de maneras como: "Todavía tengo muchos errores en cosas básicas como llaves, paréntesis y punto y coma"- E1 y "No he podido ejecutar el programa debido a algunos errores en la sintaxis que no sé cómo solucionar"- E2. Otros estudiantes se enfocaron más en expresar sus frustraciones respecto a entender estos mensajes de error de la siguiente manera: "Tuve dificultad con el lenguaje, incluso con el material complementario, me costaba ponerlo en práctica debido a que los mensajes del programa no ayudaron en nada" - E3 y "No pude interpretar los mensajes que mostraba el programa, por lo que tuve que ejecutar partes del programa por separado en otra ventana hasta que pude identificar el error" - E4.

Por ejemplo, un error común mientras se aprende a programar es el que se observa en la Figura 1.1 En el mensaje de error, la descripción principal del problema "*list index out of range*" recién aparece en la última línea, mientras que en las primeras aparecen términos como "Traceback", "*most recent call*" o "*module*" que pueden ser confusos para un programador principiante, sobre todo porque usualmente se lee de arriba a abajo y estos tecnicismos será lo primero que él lea. Además, no se presenta ningún ejemplo o ayuda que le sirva para tener una idea de cómo solucionar dicho error.\

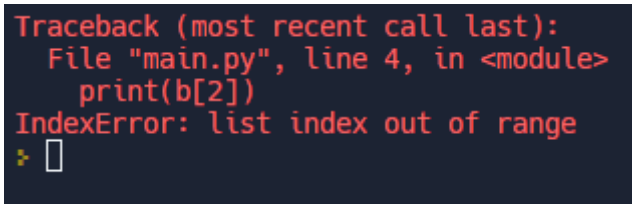


Figura 1.1 Ejemplo de error [Autoría propia]

Estudios existentes sugieren que diferentes tipos de tecnologías podrían ayudar a los principiantes en programación a manejar la complejidad de interpretar errores.

Por ejemplo, M. Hristova et al. [7] proponen una herramienta para el lenguaje de programación Java que generaba mejores mensajes de error que los compiladores existentes y proporciona sugerencias de cómo solucionar el error. Sin embargo, la

herramienta solo funciona para un lenguaje de programación e idioma específico, además de que no fue probado con estudiantes.

Por otro lado, otra forma de ayudar a los estudiantes es a través de los jueces automáticos de código como Codeforces [8] Codechef [9] y DOMjudge [10], las cuales ofrecen retroalimentación automática de la solución propuesta indicando si es correcta, tiene errores de compilación o ejecución, si es ineficiente, entre otras. Sin embargo, estas herramientas no funcionan para casos generales donde un estudiante no esté desarrollando código para solucionar un problema existente de la plataforma.

Por lo tanto, este proyecto busca cerrar las brechas descritas previamente, ayudando a programadores principiantes a entender diferentes tipos de errores mediante mensajes sencillos de entender y además brindarle ejemplos y posibles soluciones sobre código que hayan desarrollado. Además, se busca que la herramienta propuesta sea extensible a más de un lenguaje de programación.

1.3 Objetivos

Objetivo general

Diseñar un asistente virtual que a partir de un mensaje de error de programación y su código, brinde una explicación y ejemplos de cómo se puede corregir, para ayudar a los estudiantes a entender y solucionar sus errores de manera más sencilla.

Objetivos específicos

1. Generar una base de conocimiento acerca de los errores más comunes en el lenguaje de programación Python en español.
2. Implementar un módulo que analice el error ingresado por el usuario y lo relacione con un error de la base de conocimiento.
3. Implementar una interfaz gráfica que permita a los usuarios interactuar con el asistente.

1.4. Marco teórico

1.4.1 Bases teóricas

Un lenguaje de programación es un lenguaje formal que les proporciona a los seres humanos la capacidad de escribir instrucciones que las computadoras puedan ejecutar para realizar una tarea. A pesar de esto, las computadoras al solo entender bits de encendido o apagado (lenguaje de máquina) no entienden las instrucciones de un lenguaje de programación directamente. Por esto, para que una computadora pueda entender un programa escrito en un lenguaje de programación, antes debe de existir un proceso de traducción entre el lenguaje de programación y el lenguaje de máquina. Esta traducción puede llevarse a cabo a través de la compilación o la interpretación de un programa. Una vez las instrucciones estén en lenguaje de máquina, el programa puede ser ejecutado por la computadora.

Un error de programación es cualquier fallo que puede ocurrir durante la traducción o la ejecución de un programa. Estos errores pueden pertenecer a tres categorías:

1. Errores de sintaxis: Aquellos errores provocados por no seguir las reglas definidas por el lenguaje de programación. Un ejemplo de este tipo de error pueden ser errores de escritura como abrir paréntesis y no cerrarlos, poner espacios de más, omitir un punto y coma, entre otros. Estos errores impiden que el programa se ejecute.
2. Errores de ejecución: Errores que suceden mientras se ejecuta un programa y provocan su finalización de manera no controlada. Un ejemplo de este tipo de error es tratar de ejecutar operaciones no soportadas por el lenguaje de programación o las computadoras. Por ejemplo: dividir por cero, hacer referencia a archivos que faltan, llamar a funciones no válidas, entre otras.
3. Errores lógicos: Errores que son provocados cuando las instrucciones no producen el resultado deseado. Un ejemplo de este tipo de error es cuando se le asigna un valor incorrecto a una variable, causando que ejecute las operaciones indicadas con un valor incorrecto. Estos errores son los más difíciles de resolver debido a que el entorno de programación no identifica ningún comportamiento erróneo al ejecutar el programa.

El presente proyecto se enfocará en los errores de sintaxis y errores de ejecución.

1.4.2 Trabajos relacionados

1.4.2.1 Dificultades en el entendimiento de errores

Se han realizado varios estudios para comprender cómo es la interacción de los programadores principiantes con los mensajes de errores de programación. Por ejemplo, J. Prather et al. [11] encontraron inconsistencias entre los resultados de experimentos de diferentes investigadores que trataban de mejorar los mensajes de error del compilador. Ellos postularon que estas inconsistencias pueden deberse a diversas razones: (1) los estudiantes no leen los mensajes de error, (2) los investigadores se equivocan en qué deben medir, (3) los efectos de un mensaje de error mejorado son difíciles de medir, (4) los mensajes no están diseñados apropiadamente, (5) los mensajes son diseñados correctamente, pero los estudiantes no los entienden debido a un incremento en la carga cognitiva. Además, realizaron varios experimentos con más de 30 estudiantes de cursos introductorios a las ciencias computacionales donde encontraron que los estudiantes sí leían los mensajes de error del compilador, y que encontraban más útiles los mensajes mejorados que los mensajes por defecto, además que sí eran de ayuda incluso en ocasiones donde tenían una carga cognitiva más alta.

Por otro lado, la revisión literaria realizada por Brett A. Becker et al. [12], sobre los errores de compilación y por qué estos eran considerados ineficaces, proponen como resultado 10 pautas que deben implementarse para mejorar los mensajes de errores brindados por los compiladores. Algunas de las pautas expuestas son por ejemplo: (1) legibilidad, actualmente los mensajes de errores no presentan un vocabulario familiar si no que están cargados de tecnicismos, (2) carga cognitiva, se debe garantizar simplicidad pero elegancia al momento de brindar *feedback* por el bien del usuario, actualmente los mensajes de errores pueden tener redundancias que distraigan al usuario o incluso lo confunda, (3) contexto del error, es importante proveer información sobre el código del programa más relevante para ayudar a comprender y abordar el error fácilmente, actualmente algunos entornos de edición de código resaltan gráficamente la ubicación del error, lo cual les ayuda a localizar el error de manera más rápida.

Por su lado, F. Restrepo et al. [13] a través del uso del juez automático de código DOMJudge fomentaban a los estudiantes a practicar ejercicios de programación de forma continua. Esta herramienta ofrecía retroalimentación sobre la eficiencia de los programas que se desarrollaban, además de indicarles sus errores de diversos tipos. Al evaluar el desempeño de los estudiantes se encontró que sólo un 38% entregaban los programas sin errores de ningún tipo en su primer intento y un 62% necesitaban retroalimentación sobre diversos errores de la solución propuesta. Además, gracias a tener una herramienta automatizada que brindara retroalimentación, los estudiantes podían notar estos errores y corregirlos en repetidas ocasiones antes de enviar su entrega final.

1.4.2.2 Herramientas Tecnológicas

Diversos investigadores han desarrollado herramientas tecnológicas para detectar errores de sintaxis y de ejecución automáticamente, y ofrecer retroalimentación del error que sea más comprensible para los estudiantes.

Por ejemplo, M. Hristova et al. [7] propusieron una herramienta que identifica y corrige errores de programación en Java. El objetivo era que la herramienta interactiva proporcione mensajes de error más comprensibles que los compiladores existentes y que proporcione sugerencias sobre cómo solucionar el error. A pesar de esto, no fue probada en entornos reales de aprendizaje, por lo que no se pudo concluir acerca de su utilidad para estudiantes principiantes de programación.

En el estudio realizado por Pettit, R.S. et al. [14], se propone la mejora de una herramienta de evaluación de código, denominada *Athene*, para brindar mejores mensajes de errores a los estudiantes de niveles iniciales de programación. Esta herramienta estaba orientada a los concursos de programación, donde cada vez que un estudiante enviaba una solución para un problema en específico, una base de datos registra el tiempo de envío, el código del programa, la puntuación y los comentarios proporcionados. La mejora de la herramienta incluía mejores mensajes de error, haciendo estos más entendibles, sin embargo, concluyen con que mejorar los mensajes de error del compilador no hace que los estudiantes sean menos propensos a repetir los mismos errores.

Por otro lado, Serth, Sebastian et al. [15] desarrollaron *CodeOcean*, una herramienta para calificar código automáticamente de una variedad de lenguajes de programación, con el objetivo de automatizar la evaluación del código de estudiantes de MOOCs (cursos masivos en línea y abiertos). Donde para minimizar la cantidad de preguntas a los instructores, se ofrecía tres mecanismos de retroalimentación a los estudiantes: (1) anotaciones para errores en tiempo de ejecución donde se reformula el mensaje de error para ayudar a los estudiantes a encontrar la causa principal del error, (2) pistas específicas de los ejercicios y (3) la posibilidad de hacer preguntas a sus compañeros sobre su implementación.

Otro enfoque para brindar retroalimentación más detallada automáticamente ha sido a través del uso de analizadores estáticos de código. Por ejemplo, Ansgar Fehnker y Tim Blok [16] propusieron la adaptación de un analizador estático de código de Java para mostrar mensajes de retroalimentación más fáciles de entender sobre errores de conceptos que tienen usualmente los programadores principiantes. Esto se hizo a través de la adición de nuevas reglas al analizador estático PMD para encontrar errores comunes que tenían los estudiantes principiantes, y la mejora de los mensajes de advertencia de las reglas que ya existían en PMD. Los mensajes de advertencia que usaron mostraban qué parte del código generaba la advertencia, sugerencias de cómo solucionarlo, y por último una referencia bibliográfica que explicara el concepto que presumiblemente el estudiante no conocía al cometer ese error. Para comprobar su efectividad, utilizaron la herramienta adaptada con código de estudiantes principiantes y código de profesionales, encontrando que con las nuevas reglas para principiantes se encontraba un 17% más de advertencias comparado a las reglas estándar de PMD, además que de este conjunto de nuevas reglas aparecía un 164% más de advertencias en el código de principiantes que en el código hecho por profesionales. Sin embargo, en el estudio no se analizó la efectividad de la estructura de los mensajes mejorados de las advertencias.

CAPÍTULO 2

2. Metodología

2.1 Análisis

La fase de análisis inició con el desafío de escoger un nombre representativo para el sistema, optando por EVA debido a sus siglas en inglés: *Error Virtual Assistant.*

Durante esta fase se realizaron diversas actividades para definir características finales del sistema. Estas actividades involucraron a los usuarios finales del sistema como estudiantes y profesores, además de reuniones con el cliente y el tutor. A continuación, se presenta de manera detallada estas actividades.

Actividad	Objetivo	Duración
Comunicación con el cliente	Definir requerimientos funcionales y no funcionales del sistema para estudiantes y profesores.	1 hora
Reuniones con estudiantes	Entender sus entornos, complicaciones, necesidades y como se sienten respecto a los errores de programación.	3 horas
Reuniones con el tutor	Definir arquitectura del sistema	1 hora

Tabla 1 Actividades con usuarios para principios de diseño

A partir de esta información recolectada, se definieron los principios de diseño de la solución:

1. Brindar mensajes de errores más sencillos de entender.
2. Brindar ejemplos de cuándo aparecen los errores.
3. Estar integrada a un entorno de desarrollo.
4. Brindar soporte para lenguaje de programación Python.

2.1.1 Usuarios del sistema

Los usuarios que interactúan directamente con el sistema son:

a) **Estudiante**: Utiliza el sistema para obtener información sobre errores de programación.
b) **Profesor**: Monitorea el avance académico de los estudiantes a través de un *dashboard* que presenta estadísticas de las consultas de los estudiantes.

2.2 Requerimientos

Luego del análisis se levantaron los siguientes requerimientos:

2.2.1 Requerimientos funcionales

- **Explicación y ejemplos a partir de errores**
  Los estudiantes podrán ingresar al sistema algún mensaje de error que se le haya presentado en Python. El sistema deberá mostrarle una explicación en español del error y ejemplos de cuándo sucede dicho error, de tal manera que le ayude al estudiante a entenderlo.

- **Soluciones tentativas de errores**
  Los estudiantes podrán ingresar al sistema algún mensaje de error, junto con el código que generó el error en Python. El sistema podrá ofrecerle soluciones tentativas para el error, de tal manera que le ayude al estudiante a solucionarlo.

- **Historial de errores**
  Los estudiantes podrán consultar el historial de errores que hayan ingresado previamente. El historial mostrará la explicación del error, los ejemplos y las soluciones que se propusieron, de tal manera que los ayude a repasar los errores que hayan tenido previamente.

- **Consulta de errores**
  Los estudiantes podrán consultar todos los errores disponibles en el sistema con sus respectivas explicaciones, de tal manera que les sirva como referencia para aprender acerca de algún error en particular.

- **Estadísticas de errores consultados**
  Los profesores podrán consultar estadísticas de los errores ingresados en el sistema por los estudiantes a través de un *dashboard*, de tal manera que les permita conocer qué temas deben de reforzar en clases.

2.2.2 Requerimientos no funcionales

- **Entorno:**
    - La interacción de los estudiantes con el sistema deberá ser a través de una extensión para el navegador Google Chrome.
    - El despliegue del sistema en producción deberá ser de bajo o nulo costo, sin tener que preocuparse del mantenimiento de servidores.
- **Implementación:**
    - El módulo encargado de procesar los requerimientos de los estudiantes deberá ser implementado en Python.
- **Eficiencia**
    - El sistema debe ser capaz de soportar a 1000 estudiantes concurrentes.
- **Escalabilidad**
    - La arquitectura y diseño del sistema deben permitir extender el soporte a más lenguajes de programación con facilidad.

2.2.3 Alcance de la solución

La solución será una extensión para el navegador Google Chrome que será un asistente que ayude a entender los mensajes de error que se les presente a los estudiantes. La solución será escalable para expandir el soporte a nuevos lenguajes de programación y nuevas excepciones con facilidad. Sin embargo, para el alcance de este proyecto sólo se tomará en cuenta al lenguaje de programación Python y las excepciones *built-in* más comunes [17].

2.2.4 Riesgos y beneficios de la solución

Riesgos:

- **Limitaciones de infraestructura**
  Que se tengan limitaciones de infraestructura propia para alojar el sistema y se tenga que buscar alternativas gratuitas o de paga que impongan muchas restricciones.
- **Problemas de concurrencia**
  Que existan problemas de concurrencia al existir muchos usuarios conectados al mismo tiempo y exista pérdida o inconsistencia de datos por condiciones de carrera.
- **Navegador desactualizado**
  Que los estudiantes utilicen un navegador antiguo que sea incompatible con las tecnologías actuales y les impida usar el sistema.

- **Adaptación a la interfaz**
  Que los estudiantes no se adapten a la interfaz, no les agrade la experiencia de usuario, y a pesar de obtener un beneficio con la aplicación, decidan no utilizarla.

- **Errores específicos**
  Que existan errores demasiado específicos que el sistema sea incapaz de reconocer o proponer una solución.

- **Efectividad de la aplicación**
  Que la aplicación produzca malas sugerencias de solución que causen un desinterés en los estudiantes.

Beneficios:

- **Mejor comprensión de errores**
  Los estudiantes tendrían explicaciones mejoradas y ejemplos en español para entender de mejor manera los errores de programación que les aparezcan en Python.

- **Ayuda automática**
  Los estudiantes podrán obtener ayuda automáticamente sin tener que esperar a que alguien más les ayude a entender y solucionar sus errores.

- **Menor carga para profesores**
  Al tener las ayudas automáticas para los estudiantes disminuirían las consultas de errores hacia los profesores, ya que el sistema ayudaría con muchas de ellas.

- **Retroalimentación para profesores**
  Los profesores tendrán retroalimentación de los errores que más cometen los estudiantes para así identificar cuáles son los temas que más debe reforzar en clases.

2.4 Diseño de la solución

A partir de los requerimientos funcionales y no funcionales del sistema, se propuso una solución a alto nivel que se detalla a continuación.

2.4.1 Diagrama de casos de uso

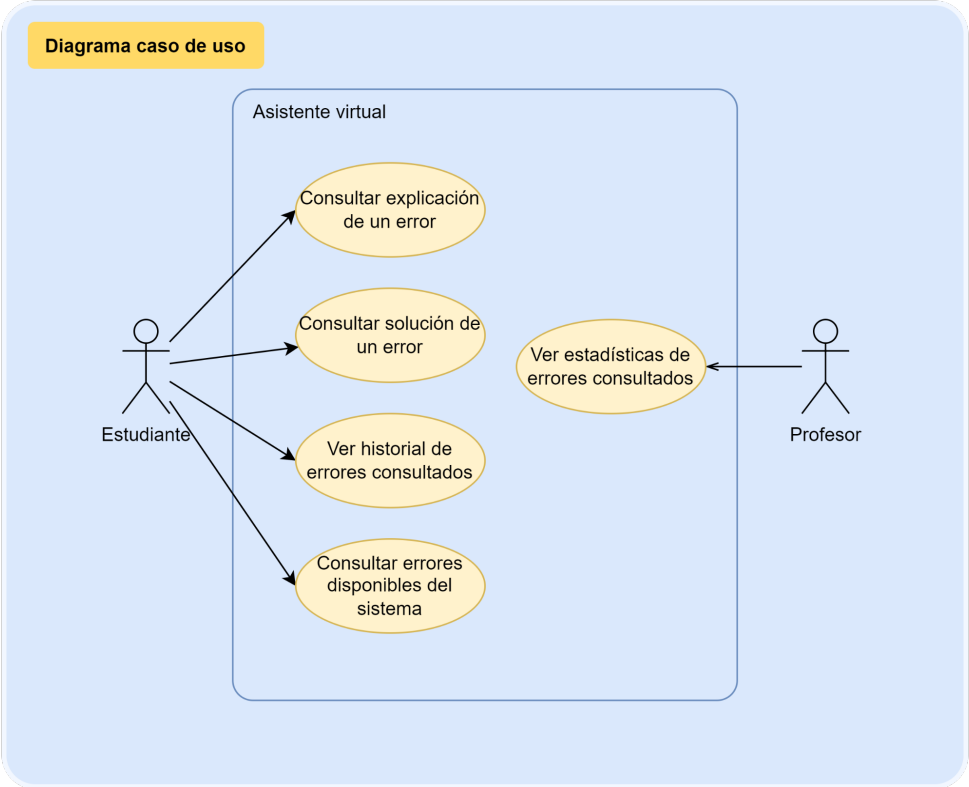


Figura 2.1 Diagrama de casos de uso [Autoría propia]

El diagrama de casos de uso mostrado en la Figura 2.1 detalla los actores con sus respectivas interacciones con el sistema. El estudiante consulta por explicaciones y soluciones de un error, puede ver un historial de errores consultados y además puede ver una lista de errores disponibles en el sistema.

Por otro lado, el profesor puede ver las estadísticas de los errores consultados por los estudiantes mediante un *dashboard*.

2.4.2 Arquitectura

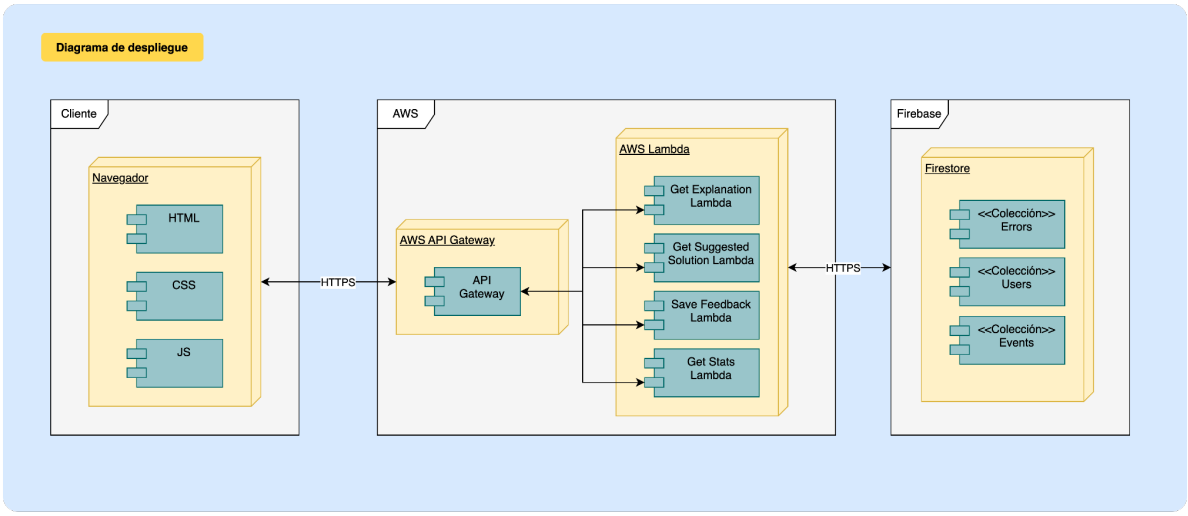


Figura 2.2 Diagrama de despliegue [Autoría propia]

En la Figura 2.2 se puede observar el diagrama de despliegue propuesto para el sistema. Se propuso una arquitectura en capas cliente servidor.

Cliente

Del lado del cliente, el único nodo que utilizaría el usuario sería el navegador, donde a través de una extensión con el navegador, se encontrarían los componentes de HTML, CSS y Javascript necesarios para renderizar la interfaz gráfica que usa el usuario para interactuar con el sistema.

Una alternativa al uso de una extensión fue crear un sitio web completo para el sistema, pero se escogió tener la interfaz en una extensión para que los estudiantes puedan consultar sus errores en la misma página que el entorno de desarrollo, y evitar que se cambien de contexto.

Servidor

Del lado del servidor, se optó por utilizar el proveedor de servicios en la nube *Amazon Web Services* (AWS) debido a la gran variedad de tecnologías que ofrece, a su modelo de precios de pago por uso, y a que es la alternativa más usada como proveedor de servicios en la nube del momento en comparación a alternativas como *Microsoft Azure* o *Google Cloud Platform* [18].

Para la implementación del servidor se tuvieron 2 alternativas. La primera era desplegar servidores web con el uso de servicios como *AWS EC2* [19] o *AWS ECS* [20], pero estas alternativas representaban un costo muy elevado al ser servicios que están ejecutándose todo el tiempo, y conlleva a una carga de trabajo mayor al dar mantenimiento a los servidores en el caso de *EC2*. Por este motivo se optó por una alternativa de servicios en la nube sin servidor o *serverless*.

Utilizando una arquitectura *serverless,* el cliente se comunicaría con el servicio *AWS API Gateway*, que es un servicio completamente administrado (sin tener que preocuparse de la infraestructura) que facilita la creación, publicación, mantenimiento y protección de una *API* a cualquier escala [21]. Esta sería la puerta de entrada a cualquier funcionalidad del sistema.

El procesamiento de los requerimientos sería a través del servicio AWS Lambda, el cual es un servicio que permite ejecutar código sin pensar en la infraestructura. Este servicio permite ahorrar costos al pagar solamente por el tiempo de informática que se utiliza, en lugar de pagar servidores que siempre estén encendidos donde gran parte del tiempo se encuentren inactivos mientras no haya consultas de los usuarios [22].

De esta manera, el API Gateway invocaría a funciones lambda dependiendo del requerimiento del cliente. Estas funciones lambda serían las encargadas de llevar a cabo toda la lógica del sistema y el procesamiento para atender los requerimientos del cliente. Por lo que, analizaría el requerimiento, consultaría la información a la base de datos, y retornaría una respuesta al cliente.

Base de datos

Finalmente, como capa de persistencia se pensó en utilizar archivos de texto plano almacenados en *buckets* de *AWS S3* [23]. Esto debido al muy bajo costo de esta alternativa, y a que no se tendría preocupaciones por el alojamiento y mantenimiento de un servidor de base de datos, además que, al ser otro servicio de AWS, facilita la integración con los demás componentes. A pesar de esto, esta alternativa presentaba algunas desventajas en cuanto a la consistencia de datos, ya que mantener actualizados los documentos podía llevar a problemas de concurrencia cuando existiesen muchos usuarios conectados al mismo tiempo. Además de que al no contar con un sistema gestor de base de datos (SGBD) la consulta y actualización de datos se volvía una tarea más compleja de ejecutar.

Por ello se optó por utilizar el proveedor en la nube Firebase con su servicio Firestore como base de datos. Esto debido a que Firebase ofrece una capa gratuita con límites amplios de hasta 1GiB de almacenamiento, y 50 mil operaciones de lectura por día [24] lo cual representaría un límite promedio de 50 operaciones de lectura por cada usuario en el pico más alto de usuarios esperados del sistema. Por lo que la capa gratuita de Firebase bastaría para soportar la carga más alta esperada de usuarios en el sistema, y con ella se tendría todos los beneficios de un SGBD ayudando a evitar problemas de concurrencia.

2.4.3 Diagrama de secuencia

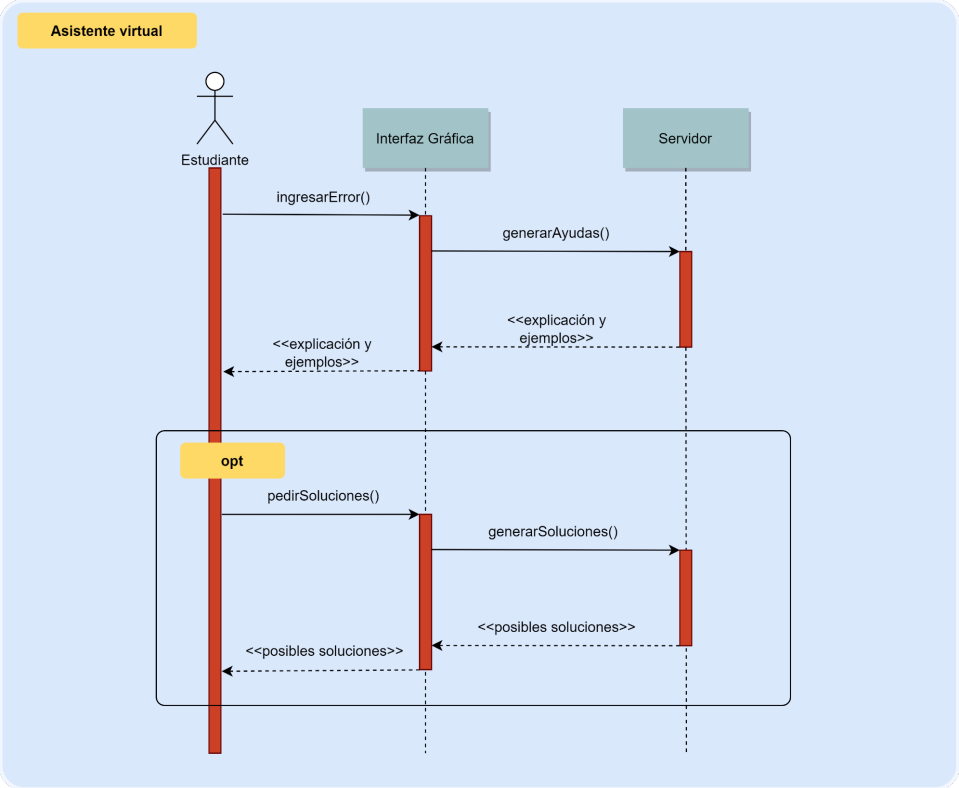


Figura 2.3 Diagrama de secuencia [Autoría propia]

En la Figura 2.3, se detalla más a profundidad el flujo de la interacción del estudiante con el sistema. El estudiante inicia ingresando un error generado por el compilador y le pide al sistema (servidor) que genere una explicación y ejemplos para dicho error. Estos son mostrados en la interfaz gráfica.

El flujo además presenta la opción de obtener una posible solución al error. El sistema analizará el código ingresado por el estudiante y a través de la interfaz gráfica le mostrará un mensaje indicando los cambios necesarios para solucionar el error.

2.4.5 Diseño de base de datos

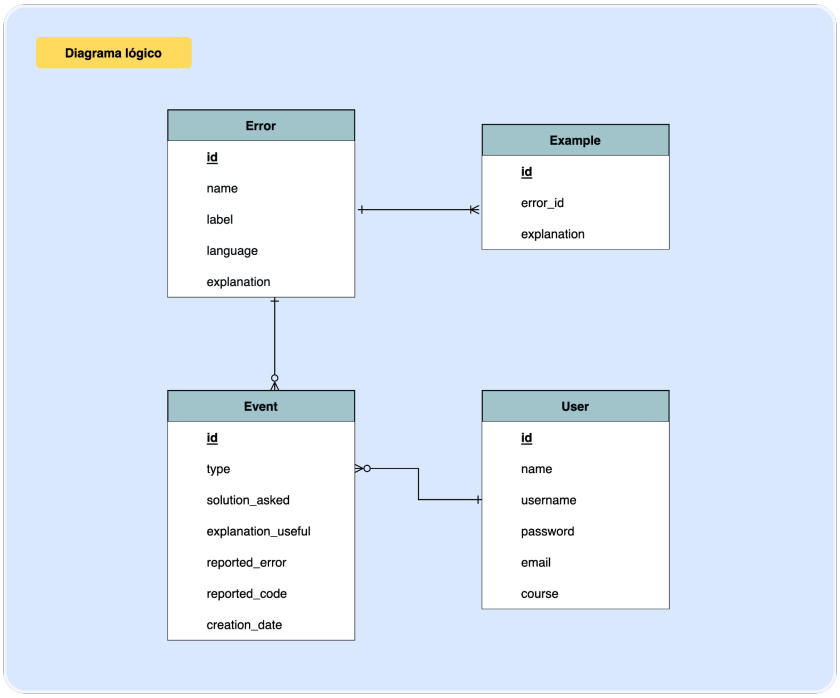


Figura 2.4 Modelo lógico de la base de datos [Autoría propia]

En la Figura 2.4 se puede observar la estructura de la base de datos del sistema. Esta está conformada por cuatro entidades principales:

1. **Error**
   Entidad principal de la base de conocimiento. Aquí se guarda un nombre técnico y un nombre legible del error, el lenguaje de programación al cual pertenece el error y su explicación. Esta explicación se guardará como texto con formato HTML.

2. **Ejemplo**
   Todos los posibles ejemplos que pueda tener un error en específico. De estos se guardará la explicación en texto con formato HTML.

3. **Usuario**
   Aquí se guardará la información para identificar a los usuarios del sistema.

4. **Evento**
   Registro de actividad de un usuario. Aquí se guardará un historial de uso del sistema. Se guardará la información relevante del evento como:

   a. Mensaje de error ingresado al sistema por el usuario.
   b. Código que genera el error ingresado al sistema por el usuario.
   c. Respuesta del estudiante sobre si le fue útil la ayuda brindada por el sistema.

2.4 Prototipo

2.4.1 Prototipo asistente virtual para estudiantes

El prototipo a continuación ejemplifica cómo los estudiantes interactuarán con el sistema.

Consulta de ayuda

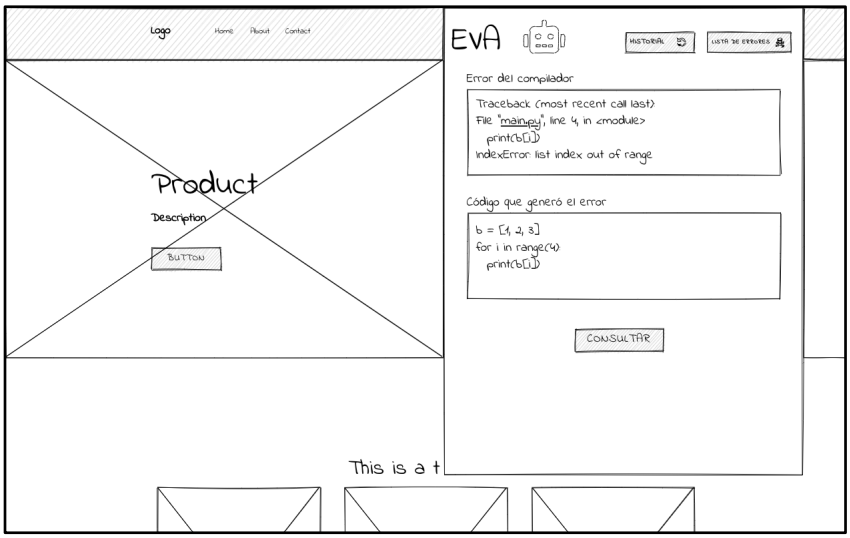


Figura 2.5 Prototipo - consulta de ayudas y ejemplos [Autoría propia]

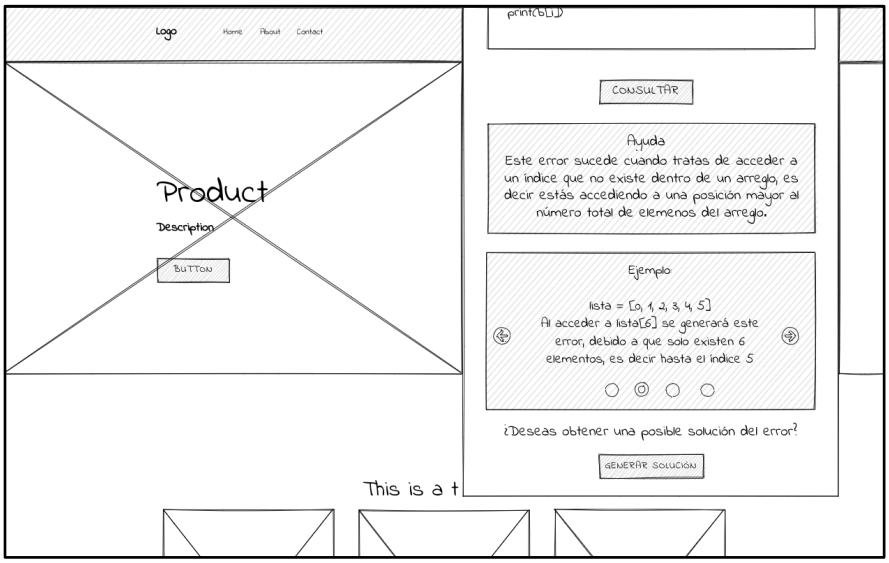


Figura 2.6 Prototipo - consulta de ayudas y ejemplos [Autoría propia]

La Figura 2.5 y 2.6 permite observar la pantalla principal del sistema. En esta se le presenta al usuario dos cajas de texto.

- Caja de texto del error: En esta el usuario ingresará el mensaje de error generado por el compilador

- Caja de texto del código: En esta el usuario ingresará el código que generó dicho error.

Una vez ingresada la información necesaria, el usuario podrá hacer clic en el botón de “consultar” para obtener un mensaje de error sencillo de entender y posibles ejemplos de cuándo ocurre dicho error, como muestra la imagen 5.2

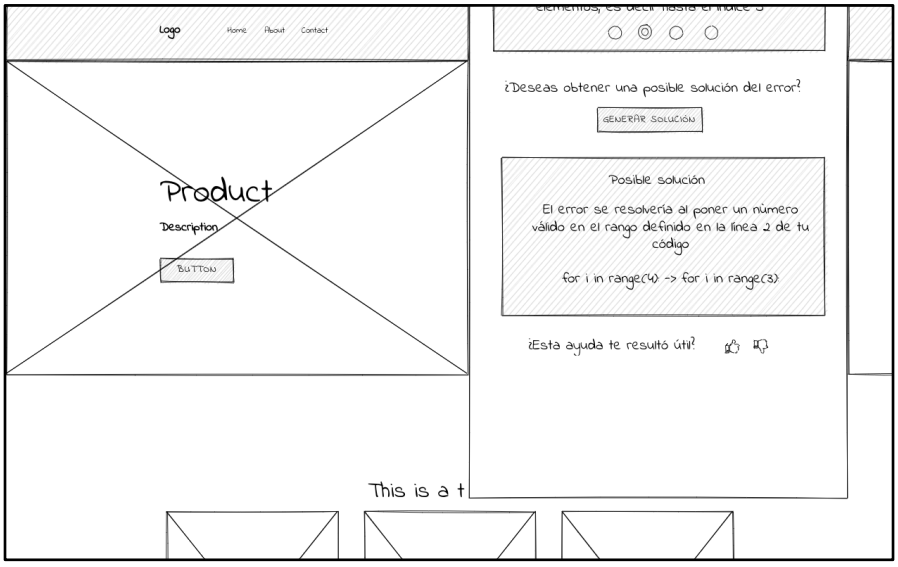


Figura 2.7 Prototipo - consulta de soluciones [Autoría propia]

En la Figura 2.7, se puede observar la última parte del flujo de consulta de ayudas para el estudiante. En esta, el usuario puede tener ayuda más personalizada al hacer clic al botón de “generar solución”, donde en base al código, el asistente tratará de decirle al estudiante qué problema en específico tiene su código, y tratará de brindarle una solución. Finalmente, el asistente le consultará al usuario si esta ayuda ha sido útil para entender o solucionar su error de programación.

Historial

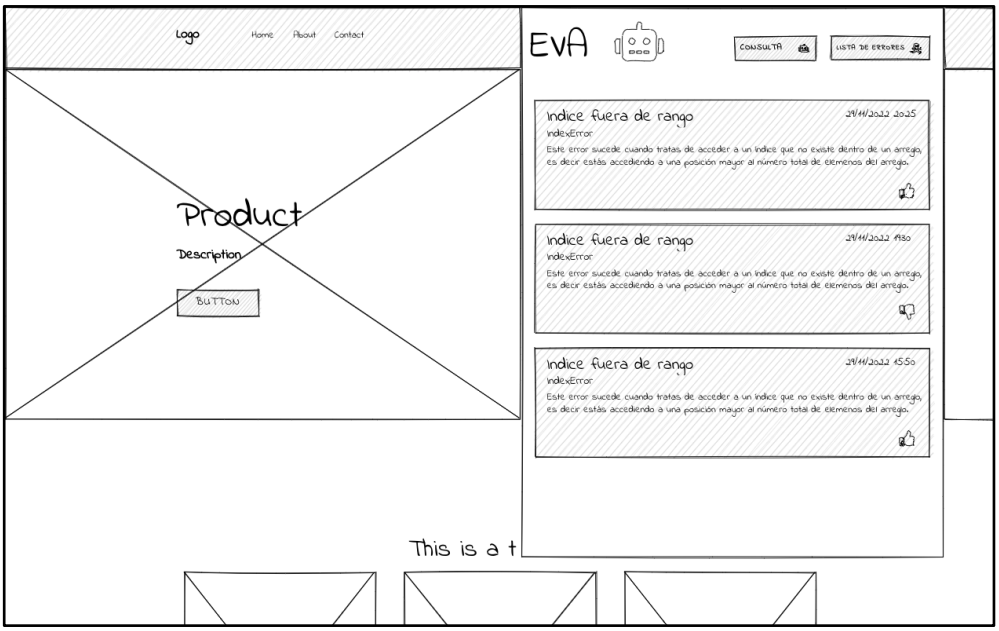


Figura 2.8 Prototipo - historial de consultas [Autoría propia]

En la Figura 2.8, podemos observar otra de las funcionalidades del sistema, la cual permite a los estudiantes revisar un historial de todas las consultas de errores que han realizado. En el historial pueden ver el nombre y la descripción del error, la fecha en la que fue consultado dicho error y además si esta ayuda les fue o no útil.

Lista de errores

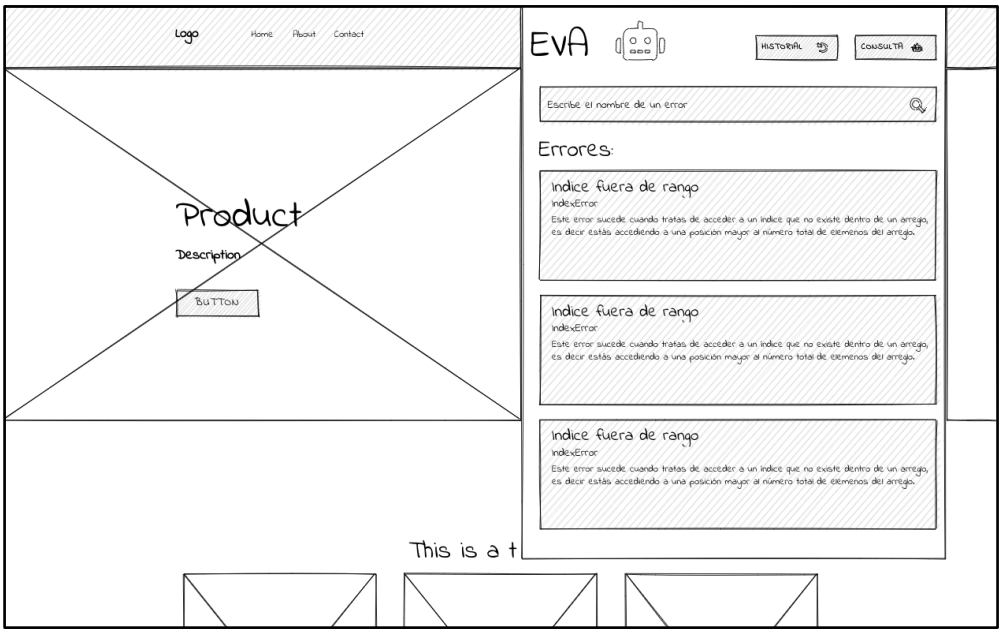


Figura 2.9 Prototipo - lista de errores disponibles [Autoría propia]

Finalmente, en la Figura 2.9 podemos observar la última funcionalidad del sistema, la cual les permite a los estudiantes revisar toda la lista de errores disponibles en el sistema en cualquier momento.

2.4.2 Prototipo sistema para profesores

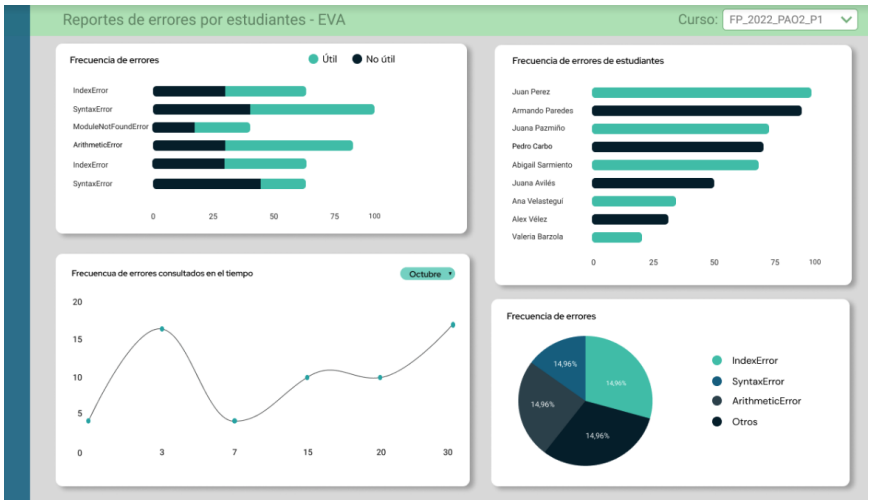


Figura 2.10 Prototipo - dashboards para profesores. [Autoría propia]

En la Figura 2.10 podemos observar el *dashboard* con el que interactuarán los profesores. En él, se les presentarán diferentes estadísticas sobre la interacción de los estudiantes con el asistente virtual, entre ellas:

- Frecuencia de errores consultados.
- Frecuencia de errores consultados por estudiante.
- Frecuencia de errores consultados en el tiempo.

2.5 Evaluación

Con el objetivo de crear un sistema de calidad, se definieron diferentes métricas de evaluación para conocer la usabilidad del sistema.

Métrica	Descripción	Usuario objetivo	Plan de reclutamiento
Efectividad de la ayuda	Mide la percepción de los estudiantes acerca de la utilidad de las ayudas proporcionadas por el sistema.	Estudiante	Se planearán talleres (30 mins) con los estudiantes del curso del tutor y/o cliente (25 ~ 60 estudiantes).
Tiempo ahorrado	Mide cuánto tiempo se demora el estudiante en solucionar un error utilizando el sistema en comparación al tiempo que se hubiese demorado si no hubiese estado utilizando el sistema.	Estudiante	Se planearán talleres (30 mins) con los estudiantes del curso del tutor y/o cliente (25 ~ 60 estudiantes).
Satisfacción del usuario	Mide el nivel de satisfacción de los usuarios respecto al sistema en general.	Estudiante	Se planearán talleres (30 mins) con los estudiantes del curso del tutor y/o cliente (25 ~ 60 estudiantes).
Efectividad de estadísticas recolectadas	Mide la percepción de los profesores acerca de la utilidad de las estadísticas proporcionadas por el *dashboard*.	Profesor	Se solicitarán reuniones con varios profesores de fundamentos de programación (2 ~ 6 profesores).

Tabla 2 Métricas de evaluación del sistema

2.5.1 Efectividad de la ayuda

A través de una escala, mide la percepción de los estudiantes acerca de la utilidad de las ayudas proporcionadas por el sistema.

2.5.2 Tiempo ahorrado

A través de una escala, mide la percepción del estudiante acerca de cuánto tiempo se ha ahorrado al utilizar el sistema para solucionar un error.

2.5.3 Satisfacción del usuario

Mide el nivel de satisfacción de los usuarios respecto al sistema. En base a esta métrica se puede conocer de manera definitiva si el sistema es de ayuda para los estudiantes al indicar si ellos encuentran útiles las ayudas y soluciones brindadas.

2.5.4 Efectividad de estadísticas recolectadas

Mide el nivel de satisfacción de los profesores respecto a las estadísticas mostradas. En base a esta métrica se puede conocer si las estadísticas son útiles para identificar que temas deben ser reforzados en clases con los estudiantes.

CAPÍTULO 3

3. Resultados y análisis

Con el objeto de medir la efectividad de la aplicación, se realizaron talleres con 51 estudiantes de Fundamentos de Programación, para conocer sus percepciones acerca de la aplicación. Además, se tuvieron reuniones individuales con 3 profesores de la materia Fundamentos de Programación para conocer su percepción de la aplicación y de las estadísticas presentadas en el *dashboard*.

3.1 Pruebas de usuarios

3.1.1 Pruebas con estudiantes

Para las pruebas con los estudiantes se realizaron talleres donde se planeaba que utilicen la aplicación en casos hipotéticos, y que al final expresen sus percepciones de la aplicación.

Los talleres se llevaron a cabo con dos paralelos, en los laboratorios de ESPOL. Durante el taller, los estudiantes iniciaron instalando el aplicativo, iniciando sesión y creando un nuevo proyecto de repl.it [25], editor de código utilizado en Fundamentos de Programación.

Una vez todo configurado, se les presentó cuatro preguntas con programas de Python que contenían errores de ejecución, dos de las cuales las debían resolver utilizando la aplicación, y las otras dos sin utilizar la aplicación. Esto con el objetivo de que los estudiantes puedan percibir las diferencias entre solucionar los errores utilizando o no la aplicación. Estas preguntas se pueden encontrar en el Apéndice A.

Al finalizar el taller, los estudiantes respondieron un formulario de satisfacción acerca del sistema, el cual contenía preguntas para medir las métricas de evaluación de: efectividad de la ayuda, tiempo ahorrado y satisfacción del usuario en general de la aplicación. Estas preguntas se pueden encontrar en el Apéndice B.

3.1.2 Pruebas con profesores

Para las pruebas con los profesores se realizaron reuniones, en las cuales primero se les presentó a los profesores la aplicación y sus casos de uso. Una vez claro el flujo de la aplicación, se les presentó un *dashboard* con varios gráficos estadísticos que describen las consultas de los estudiantes que se han realizado en la aplicación.

Luego, se les hizo varias preguntas acerca de la utilidad de la aplicación, si la usarían con sus estudiantes. Además, se les preguntó qué tan útiles consideran ellos que son los gráficos del *dashboard*, y acerca de la influencia que la información presentada por el mismo tendría en sus estrategias para dictar clases.

3.2 Resultados

3.2.1 Pruebas con estudiantes

Al analizar las respuestas de los estudiantes al formulario de satisfacción se obtuvieron los resultados presentados a continuación.

Como se observa en la Figura 3.1, el 68% de los estudiantes indicó que la aplicación le hubiese sido de ayuda al inicio del semestre, mientras que el 23.5% indicó que le hubiese sido de **mucha** ayuda al inicio del semestre.

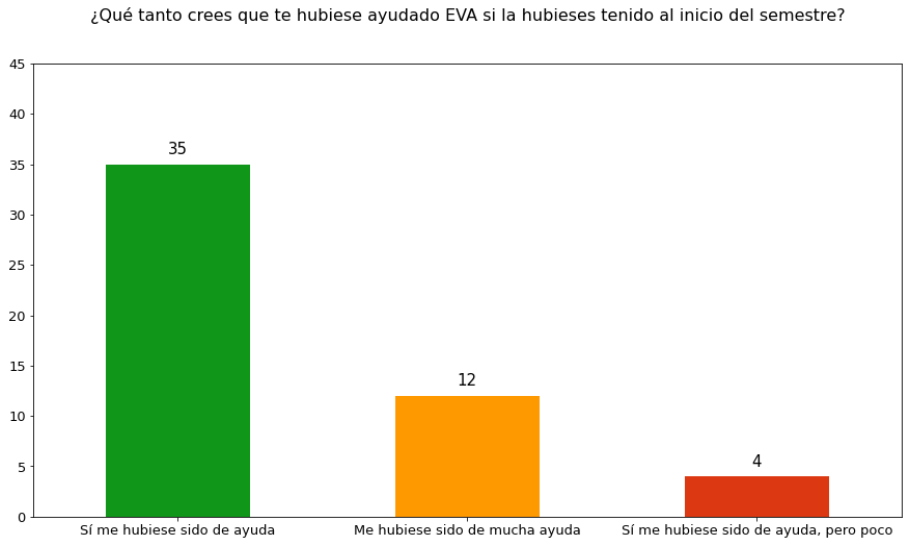


Figura 3.11 Resultados de efectividad [Autoría propia]

Por otro lado, como se observa en la Figura 3.2, la mayoría de los estudiantes (74.5%) sintieron que solucionaron los errores de los talleres más rápido utilizando la aplicación.

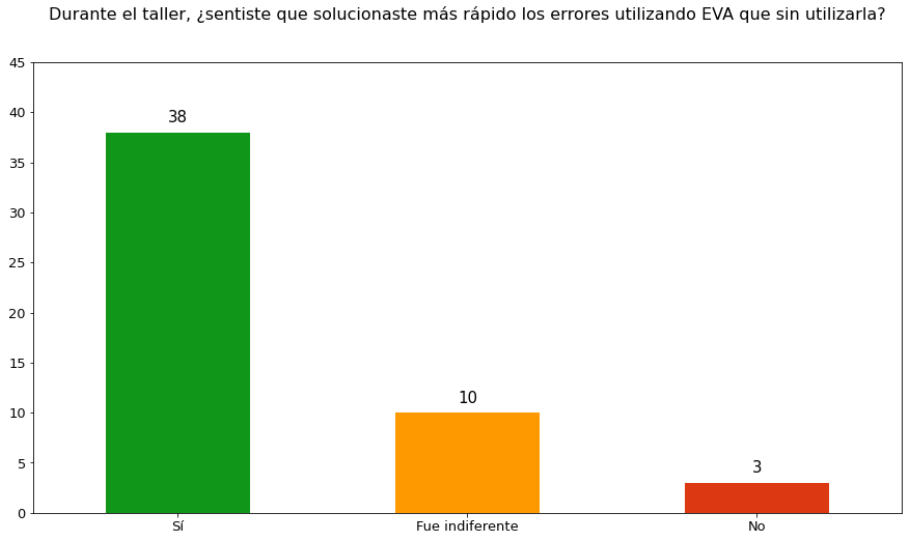


Figura 3.2 Resultados de ahorro de tiempo [Autoría propia]

Finalmente, como se observa en la Figura 3.3, el 64.7% de los estudiantes indicó que se sintió satisfecho con la aplicación, mientras que el 31.4% indicó se sintió **muy** satisfecho con la aplicación.

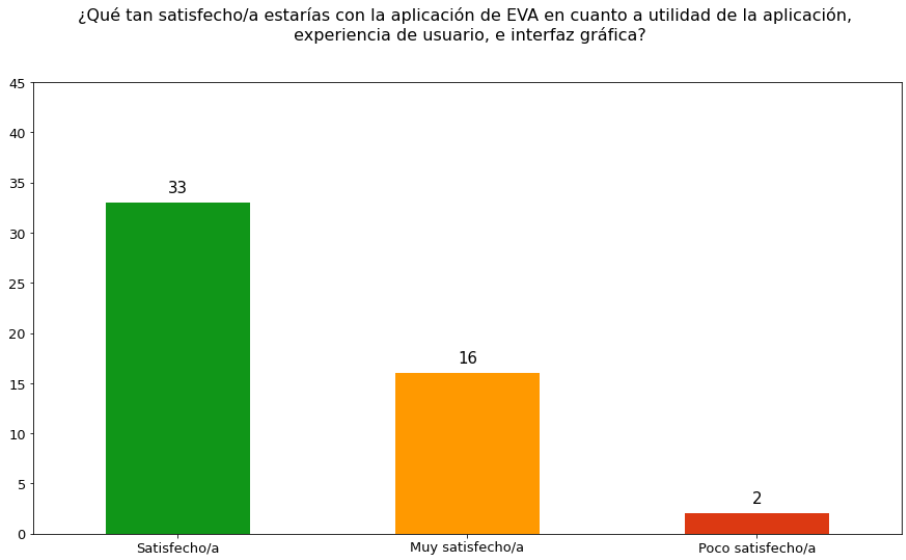


Figura 3.3 Resultados de satisfacción [Autoría propia]

3.2.2 Pruebas con profesores

Luego de las reuniones con los profesores se obtuvo lo siguiente:

- Los 3 profesores concordaron en que utilizarían EVA como una herramienta de apoyo para sus estudiantes.
- Los 3 profesores concordaron en que las estadísticas presentadas en el *dashboard* les ayudarían a conocer el rendimiento de sus estudiantes.
- Algunos profesores indicaron que les sería útil clasificar las métricas por los temas revisados durante el curso, como por ejemplo “introducción a variables”, “manejo de listas”, entre otros.
- Una profesora indicó que el sistema debería imponer más restricciones a la sección de “posible solución” para que los estudiantes se esfuercen más en entender los errores antes de pedir una posible solución.

3.3 Costos

3.3.1 Costos iniciales

Entre los costos iniciales se tienen los costos de diseño e implementación del sistema. Para ello, entre 2 personas se desarrolló el sistema en un tiempo equivalente a 30 días no consecutivos con un promedio de 4 horas diarias, teniendo un costo por hora de $4.00, se tendría:

Descripción	**Factor**
Días de desarrollo	30
Horas por día	4
Número de personas	2
Precio por hora	$4.00
Total	$960

Tabla 3 Descripción de inversión inicial del sistema

Adicionalmente se tomó en cuenta el costo inicial de subir el sistema como extensión de navegador a la tienda para que pueda ser accedida por cualquier usuario, esto tiene un costo único de $5.

Por lo tanto, los costos iniciales ascenderían a $965.

3.3.2 Costos de infraestructura

Se evaluaron los costos de infraestructura ofrecidos por los proveedores de servicio en la nube: AWS [26] y Firebase [24]. Esta evaluación se basa en los costos mensuales que se tendrían al mantener 1000 usuarios con un uso elevado de la aplicación.

El costo mensual por infraestructura en AWS sería de $8.30, mientras que por Firebase se tendría un valor mensual de $0.60, lo que resultaría en un costo mensual por infraestructura de $8.90. Los detalles del cálculo mensual por costos de infraestructura pueden encontrarse en el Apéndice C.

3.3.3 Modelo de negocio

Para hacer que el sistema sea sostenible y que pueda escalar en el tiempo, se tiene pensado manejar diversas suscripciones dependiendo de la cantidad de usuarios que utilizarían la aplicación.

Tipo de plan	Usuarios soportados	Costo mensual
Inicial	hasta 100	$1.50 x usuario
Intermedio	hasta 500	$1.00 x usuario
Premium	hasta 1000	$0.50 x usuario

Tabla 4 Modelos de subscripción mensual

El sistema podría tener como clientes a universidades, colegios, Bootcamps o cursos ofrecidos en plataformas de internet que ofrezcan cursos introductorios a programación, y no solo de Ecuador sino de la región.

CAPÍTULO 4

4. Conclusiones y recomendaciones

4.1 Conclusiones

- El sistema logró agilizar el proceso de los estudiantes para encontrar soluciones a diferentes errores de programación.
- El sistema pudo identificar correctamente la mayoría de los errores ingresados por los estudiantes lo que creó una percepción en ellos de que les habría ayudado durante sus primeras semanas aprendiendo programación.
- El sistema debe mejorar el módulo de brindar soluciones según lo reportado por los estudiantes, quienes indicaron que les gustaría que las soluciones sean más específicas.
- En general los estudiantes se sintieron satisfechos utilizando la aplicación.
- El dashboard ofrece información relevante para los profesores, dado que les ayudaría a conocer el rendimiento de sus estudiantes.

4.2 Recomendaciones

- Se recomienda utilizar técnicas como crowdsourcing para alimentar la base de conocimiento de una manera más eficiente.
- Se podría utilizar Inteligencia Artificial para lograr entender mejor la lógica de programación de los estudiantes y así brindar soluciones más precisas.
- Actualmente la aplicación está disponible solamente como extensión de Google Chrome, pero se podría extender para que sea posible utilizarla desde diferentes entornos de desarrollo comúnmente utilizados como Visual Studio Code o PyCharm.
- El proceso de copiar y pegar el error junto con el código puede llegar a causar una alta demanda cognitiva de los usuarios, que se disminuiría si la aplicación obtuviera esta información automáticamente del entorno de desarrollo.
- Actualmente el sistema solo soporta errores en el lenguaje de Python, sin embargo, la arquitectura del software está diseñada para extenderse a varios lenguajes de programación con facilidad, por lo que se recomienda expandir el soporte a más lenguajes de programación.
- Las estadísticas del dashboard podrían segmentarse por temas para reducir la carga cognitiva de los profesores.

BIBLIOGRAFÍA

[1] E. Shein, “Should everybody learn to code?,” Communications of the ACM, vol. 57, no. 2, pp. 16–18, 2014.

[2] Bosse, Yorah, y Marco Aurélio Gerosa. «Why Is Programming so Difficult to Learn?: Patterns of Difficulties Related to Programming Learning Mid-Stage». ACM SIGSOFT Software Engineering Notes, vol. 41, n.o 6, enero de 2017, pp. 1-6.

[3] Piteira, Martinha, y Carlos Costa. «Learning Computer Programming: Study of Difficulties in Learning Programming». Proceedings of the 2013 International Conference on Information Systems and Design of Communication - ISDOC ’13, ACM Press, 2013, p. 75.

[4] Denny, Paul, et al. «Understanding the Syntax Barrier for Novices». Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education - ITiCSE ’11, ACM Press, 2011, p. 208.

[5] Cechinel, et al. (2008). Desenvolvimento de Objetos de Aprendizagem para o Apoio à Disciplina de Algoritmos e Programação.

[6] Y. Bosse and M. A. Gerosa, “Why is programming so difficult to learn?,” *ACM SIGSOFT Software Engineering Notes*, vol. 41, no. 6, pp. 1–6, 2017.

[7] M. Hristova, A. Misra, M. Rutter, and R. Mercuri, “Identifying and correcting Java programming errors for introductory computer science students,” *ACM SIGCSE Bulletin*, vol. 35, no. 1, pp. 153–156, 2003.

[8] Mirzayanov, M (2015) *Codeforces*.

[9] *Codechef: Competitive programming: Participate & learn: Codechef* (2021)

[10] Eldering, J, Kinkhorst, T and Warken, P (2010) *Domjudge*, *DOMjudge*.

[11] J. Prather, R. Pettit, K. H. McMurry, A. Peters, J. Homer, N. Simone, and M. Cohen, “On novices' interaction with compiler error messages,” *Proceedings of the 2017 ACM Conference on International Computing Education Research*, 2017.

[12] Becker, B.A. *et al.* (2019) “Compiler error messages considered unhelpful,” *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education* [Preprint].

[13] F. Restrepo-Calle, J. J. Ramírez Echeverry, and F. A. González, “Continuous assessment in a computer programming course supported by a software tool,” *Computer Applications in Engineering Education*, vol. 27, no. 1, pp. 80–89, 2018.

[14] Pettit, R.S., Homer, J. and Gee, R. (2017) “Do enhanced compiler error messages help students?,” *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* [Preprint].

[15] Serth, Sebastian & Staubitz, Thomas & Teusner, Ralf & Meinel, Christoph. (2021). CodeOcean and CodeHarbor: Auto-Grader and Code Repository.

[16] Fehnker, Ansgar, y Tim Blok. «Automated Program Analysis for Novice Programmers». *Third International Conference on Higher Education Advances*, 2017.

[17] “Built-in Exceptions — Python 3.8.2 documentation,” *docs.python.org* [Online]. Disponible en: https://docs.python.org/3/library/exceptions.html

[18] “Stack Overflow Developer Survey 2022,” *Stack Overflow* [Online]. Disponible en: https://survey.stackoverflow.co/2022/#section-most-popular-technologies-cloud-platforms.

[19] “AWS | Elastic compute cloud (EC2) de capacidad modificable en la nube,” *Amazon Web Services, Inc.*, 2019 [Online]. Disponible en: https://aws.amazon.com/es/ec2/

[20] “AWS | Gestión de contenedores (ECS) compatible con los de Docker,” *Amazon Web Services, Inc* [Online]. Disponible en: https://aws.amazon.com/es/ecs (accessed Dec. 04, 2022).

[21] “Amazon API Gateway | API Management | Amazon Web Services,” *Amazon Web Services, Inc* [Online]. Disponible en: https://aws.amazon.com/es/api-gateway/

[22] “AWS | Lambda - Gestión de recursos informáticos,” *Amazon Web Services, Inc* [Online]. Disponible en: https://aws.amazon.com/es/lambda/

[23] “AWS | Almacenamiento de datos seguro en la nube (S3),” *Amazon Web Services, Inc* [Online]. Disponible en: https://aws.amazon.com/es/s3/

[24] “Firebase Pricing | Firebase,” *Firebase*, 2019 [Online]. Disponible en: https://firebase.google.com/pricing

[25] Replit, “The collaborative browser based IDE,” *replit*. [Online]. Disponible en: https://replit.com/.

[26] F. Livesey, “Pricing,” *Amazon*, 1976. [Online]. Disponible en: https://aws.amazon.com/pricing/.

APÉNDICES

Apéndice A

Preguntas de talleres con estudiantes

1. Copia y pega el siguiente código en replit, y **sin utilizar EVA** resuelve todos los errores que contenga, y pega el código corregido como respuesta a esta pregunta.

   Para el siguiente ejercicio debes descargar este archivo movies.csv y subirlo en replit. El código busca la película mejor calificada.

```
movies_name = []

movies_rates = []

with open("../movies.csv", "r") as f:
    for line in f:
        movie = line.split(",")
        movies_name.append(movie[1])
        movies_rates.append(movie[4])

    index = movies_rates.index(max(movies_rates))

    print("Pelicula con mayor calificacion: ",
movies_name[index])
```

2. Copia y pega el siguiente código en replit, y **utilizando EVA** resuelve todos los errores que contenga, y pega el código corregido como respuesta a esta pregunta.

   El código intenta añadir un nuevo estudiante a una lista.

```
students = [
    {'name': 'John', 'age': 25, 'spec': 'math'},
    {'name': 'Peter', 'age': 24, 'spec': 'math'},
    {'name': 'Anna', 'age': 23, 'spec': 'math'},
]

def add_student(student_name, student_age, student_spec):
    student = {
      'name': student_name,
      'age': student_age,
      'spec': student_spec
    }
    students.add(student)
    print('Nuevo estudiante añadido: ', student['nam'])

add_student('valeria', 19, 'lang')
```

3. Copia y pega el siguiente código en replit, y **sin utilizar EVA** resuelve todos los errores que contenga, y pega el código corregido como respuesta a esta pregunta.

   El código cuenta la cantidad de estudiantes aprobados y reprobados de una lista de calificaciones.

```
import Numpy as np

calificaciones = np.array([8, 9, 10])

resumen = {}

for calificacion in calificaciones:
  if calificacion >= 6:
    resumen["aprobados"] += 1
  else:
    resumen["reprobados"] += 1

print("Estudiantes reprobados: ", resumen["reprobados"])
print("Estudiantes aprobados: ", resumen["aprobados"])
```

4. Copia y pega el siguiente código en replit, y utilizando EVA resuelve todos los errores que contenga, y pega el código corregido como respuesta a esta pregunta.

   El código encuentra el n-ésimo número en la serie de fibonacci, serie en la que el n-ésimo número es la suma de los dos anteriores: 1, 1, 2, 3, 5... (Por ejemplo, el 5 es la suma de 3 + 2, el 3 la suma de 2 + 1 y así sucesivamente).

```
def fibonacci(n):
  if n == 1:
    return 1
  return fibonacci(n - 1) + fibonacci(n - 2)

print(f"El 5to número en la sucesión de fibonacci es:
{fibonacci(5)}")
```

Apéndice B

Preguntas formulario retroalimentación

1. ¿De qué carrera eres?
2. Del 1 al 5, ¿qué tan complicada se te ha hecho la materia fundamentos de programación?
    a. 1 - Nada
    b. 2
    c. 3
    d. 4
    e. 5 - Mucho
3. ¿Qué tanto crees que te hubiese ayudado EVA si la hubieses tenido al inicio del semestre?
    a. No me hubiese ayudado nada
    b. Sí me hubiese sido de ayuda, pero poco
    c. Sí me hubiese sido de ayuda
    d. Me hubiese sido de mucha ayuda
4. Durante el taller, ¿sentiste que solucionaste más rápido los errores utilizando EVA que sin utilizarla?
    a. Sí
    b. No
    c. Fue indiferente
5. En general, ¿qué tan satisfecho/a estarías con la aplicación de EVA en cuanto a utilidad de la aplicación, experiencia de usuario, e interfaz gráfica?
    a. Nada satisfecho/a
    b. Poco satisfecho/a
    c. Satisfecho/a
    d. Muy satisfecho/a
6. ¿Cómo crees que podríamos mejorar para hacer a EVA más útil?

Apéndice C

Detalle de costos al mes por usuarios

Se ha considerado a un usuario con uso elevado del sistema como un usuario que realice alrededor de 1000 peticiones HTTP al servidor de la aplicación en un mes.

Las ejecuciones de AWS Lambda tienen un costo según la cantidad de milisegundos (ms) de ejecución de la función y de la cantidad de memoria de la función. Las funciones lambdas utilizadas para la aplicación tienen una memoria de 512 MB, y el costo por ms de ejecución de este tipo de lambdas es de 0,0000000083 USD. Se han evaluado las ejecuciones de funciones lambda y ninguna dura más de 1 segundo de ejecución.

Con estas asunciones, los costos mensuales para 1000 usuarios serían:

Descripción	**Factor**
Costo por tiempo de ejecución	0,0000000083 USD/ms
Duración promedio por requerimiento	1000 ms/req
Requerimientos por usuarios en un mes	1000 req/usuarios
Cantidad de usuarios	1000 usuarios
Total por mes	8,30 USD

Tabla 5 Proyección de costos mensuales de AWS por mil usuarios

En cuanto a Firebase, se considerarán los costos de escritura y lectura en la base de datos. Con base a las pruebas realizadas se estima que un usuario con un uso elevado del sistema realizaría alrededor de 2500 lecturas de documentos en un mes, y 300 escrituras.

Con estas asunciones los costos mensuales para 1000 usuarios en Firebase tomando en cuenta su calculadora de precios serían:

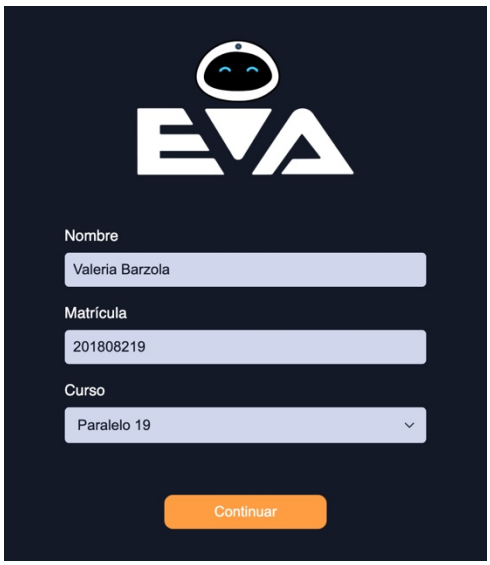
Descripción	**N operaciones**	**Costo**
Operaciones de lectura	2.500.000	0,60 USD
Operaciones de escritura	300.000	Gratis
Total por mes		0,60 USD

Tabla 6 Proyección de costos mensuales de Firebase por mil usuarios

Apéndice D

Capturas del funcionamiento del sistema

1. Registro

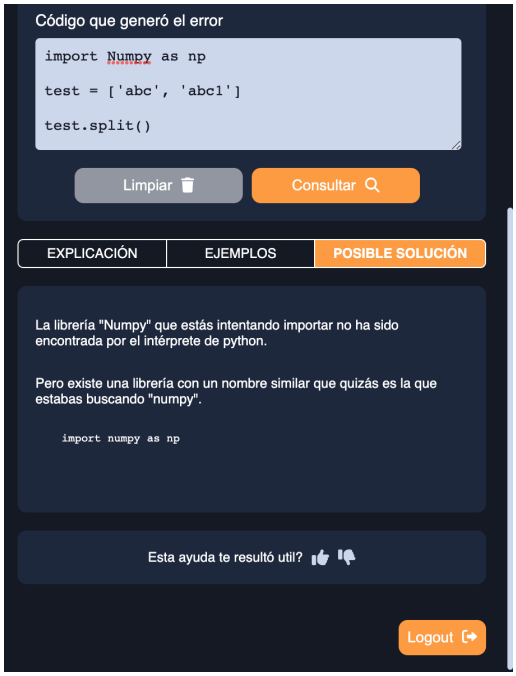


2. Ingreso error

3. Resultado de consulta de explicación

4. Consultar ejemplos

5. Resultado consulta posible solución



6. Ver historial de consultas

7. Ver errores del sistema