Evidence 4 - Math Expressions Analyzer - UI

Ana Elena Velasco García A01639866

Ana Paola Jiménez Sedano A01644532

Cesar Morán Macías A01645209

Obed Nehemías Muñoz Reynoso

Implementación de métodos computacionales

Grupo 205

Domingo 15 de junio del 2025

Evidence 4 - Math Expressions Analyzer - UI

Este proyecto consiste en una aplicación web con arquitectura cliente-servidor. Permite a los usuarios ingresar expresiones matemáticas, y valida en tiempo real si la expresión es sintácticamente correcta utilizando un Autómata de Pila (PDA) implementado en el backend con Go. El cliente está desarrollado con React.

Características Cliente (Frontend con React):

- Estilo visual dinámico con colores por tipo de carácter.
- Validación automática al escribir.
- Cambia el color según validez: verde si es válida, rojo si no.

Características Servidor (Backend con Go):

- Lee el archivo YAML con el autómata.
- Procesa cada expresión con el autómata de pila.
- Envia una respuesta indicando si es válida o no.

Conexión Cliente-Servidor

Configuración del servidor

Para habilitar la comunicación entre el cliente-servidor, primero levantamos el servidor de go en el localhost:8080 (puerto default de Gin). Los pasos que seguimos fueron:

1. Creamos el servidor con gin.default()
2. Permitimos la comunicación entre diferentes puertos (por ejemplo localhost:8080 para el backend en Go y localhost:5174 para el frontend en React) usando cors.Deafault() (middleware llamado Cross Origin Resource Sharing). Gracias a esto, el servidor puede aceptar peticiones del cliente sin bloquearlas por políticas de seguridad del navegador.

3. Definimos un endpoint GET /status para verificar que el servidor de go está bien levantado.
4. Definimos el endpoint POST /validate el cual recibirá las expresiones del front y ejecutará validateExpressions() para validarla y regresarle una respuesta al front para que cambie el color según corresponda.
5. Arrancamos el servidor con server.Run()

```
251  func main() {
252
253      buildPDA() //Reads tha automata from the yaml
254
255      server := gin.Default()
256      server.Use(cors.Default())
257      server.GET("/status", status)
258      server.POST("/validate", validateExpression)
259
260      server.Run()
261
262  }
```

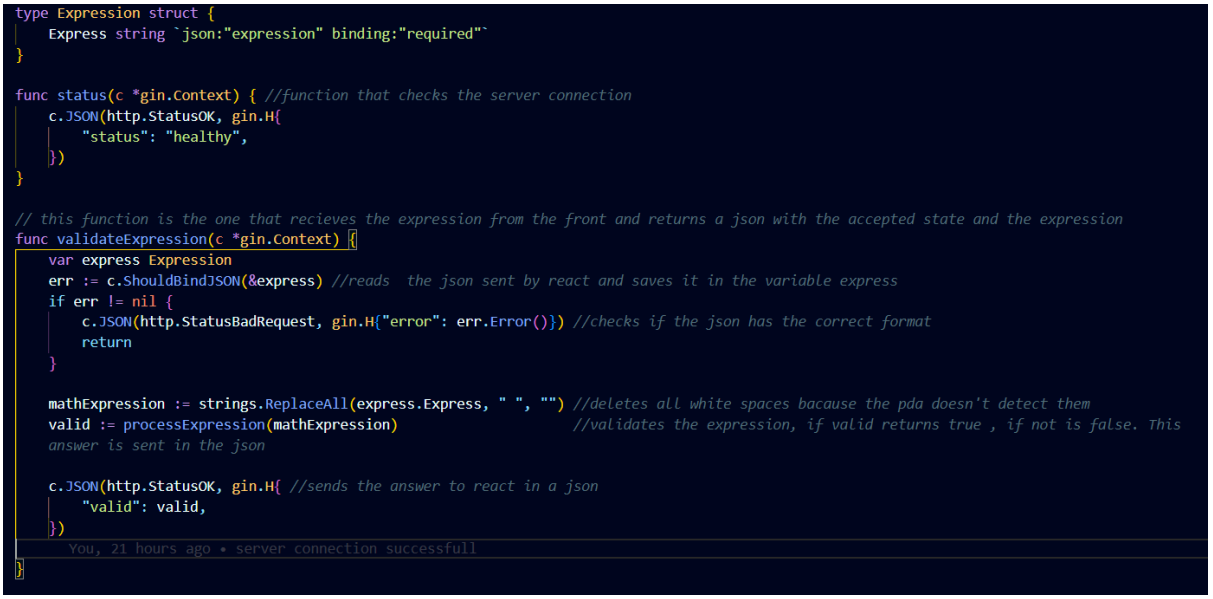
Comunicación desde el cliente

Una vez configurado el servidor, configuramos nuestro cliente para que pudiera mandar a través de una request POST la expresión matemática. Esta expresión la envía react mediante un fetch en formato json al endpoint /validate de localhost:8080. Usamos POST para que la enviara a través del cuerpo de la http request y no por la url como se haría con un GET.

```
fetch('http://localhost:8080/validate', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({ expression: expression })
})
```

Validación de la expresión en el servidor

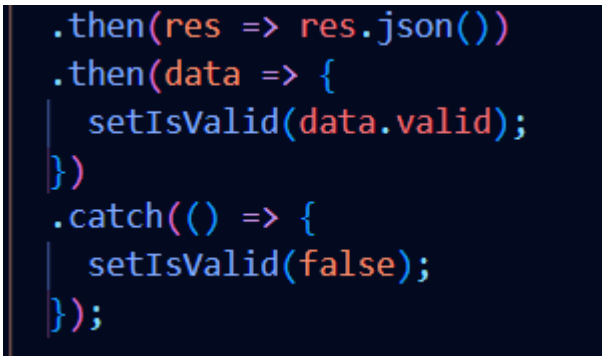
Una vez enviada la expresión, el servidor detecta que le enviaron un post en /validate, entonces llama a validateExpression(). Dentro de esta, obtiene el campo de expression del json, si no lo tiene o está mal formateado lanza error. Si está correcto, le quita los espacios en blanco y la procesa con processExpression() la cual regresa el true o false de la validación. Una vez regresado, con c.JSON() se construye la respuesta y la manda al .then() de react.



```
type Expression struct {
    Express string `json:"expression" binding:"required"`
}

func status(c *gin.Context) { //function that checks the server connection
    c.JSON(http.StatusOK, gin.H{
        "status": "healthy",
    })
}

// this function is the one that recieves the expression from the front and returns a json with the accepted state and the expression
func validateExpression(c *gin.Context) {
    var express Expression
    err := c.ShouldBindJSON(&express) //reads  the json sent by react and saves it in the variable express
    if err != nil {
        c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()}) //checks if the json has the correct format
        return
    }

    mathExpression := strings.ReplaceAll(express.Express, " ", "") //deletes all white spaces bacause the pda doesn't detect them
    valid := processExpression(mathExpression)                     //validates the expression, if valid returns true , if not is false. This
    answer is sent in the json

    c.JSON(http.StatusOK, gin.H{ //sends the answer to react in a json
        "valid": valid,
    })
       You, 21 hours ago • server connection successfull
}
```

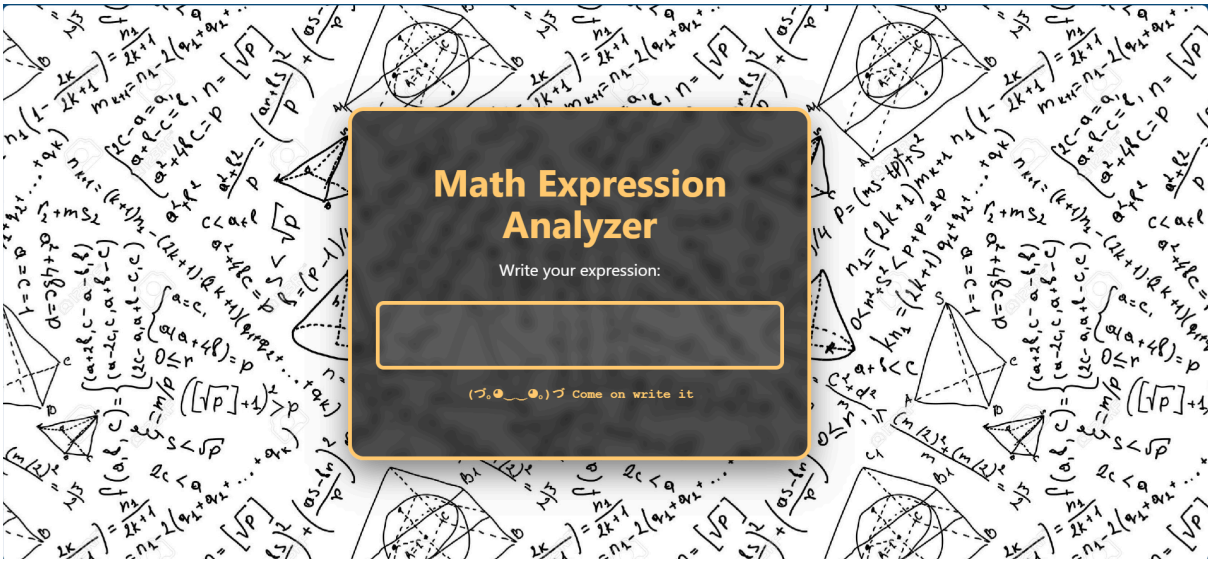
Procesamiento de la respuesta en el cliente

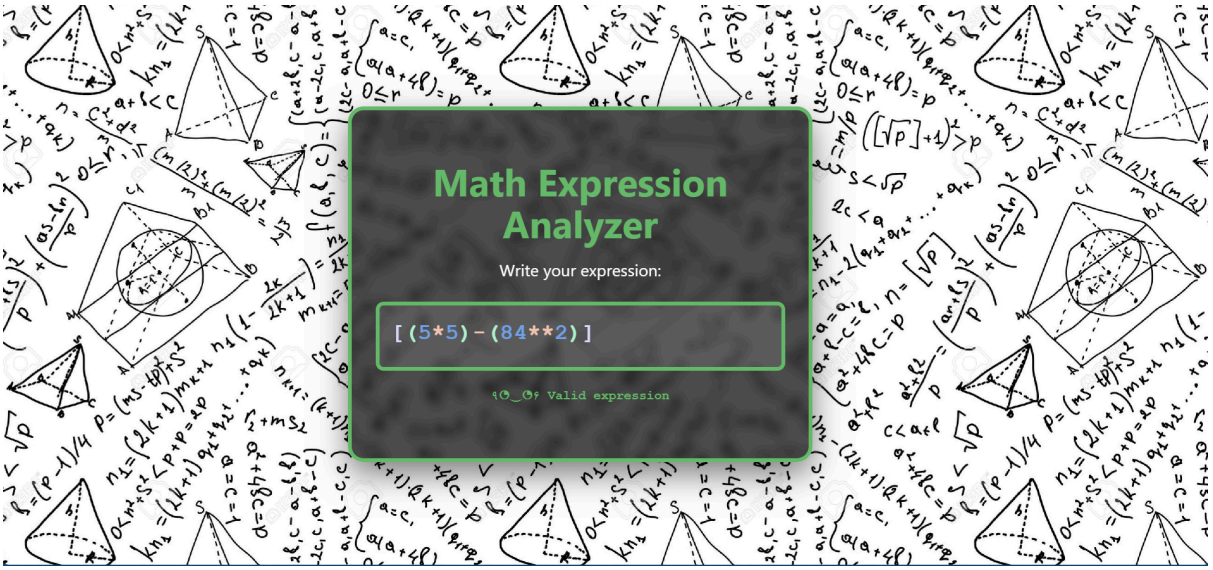
Dentro de esta serie de thens, se convierte la response de string a json y guarda el true o false de valid en el estado isValid para que en base a eso react muestre amarrillo si no hay nada, verde si es válida o roja si es inválida.

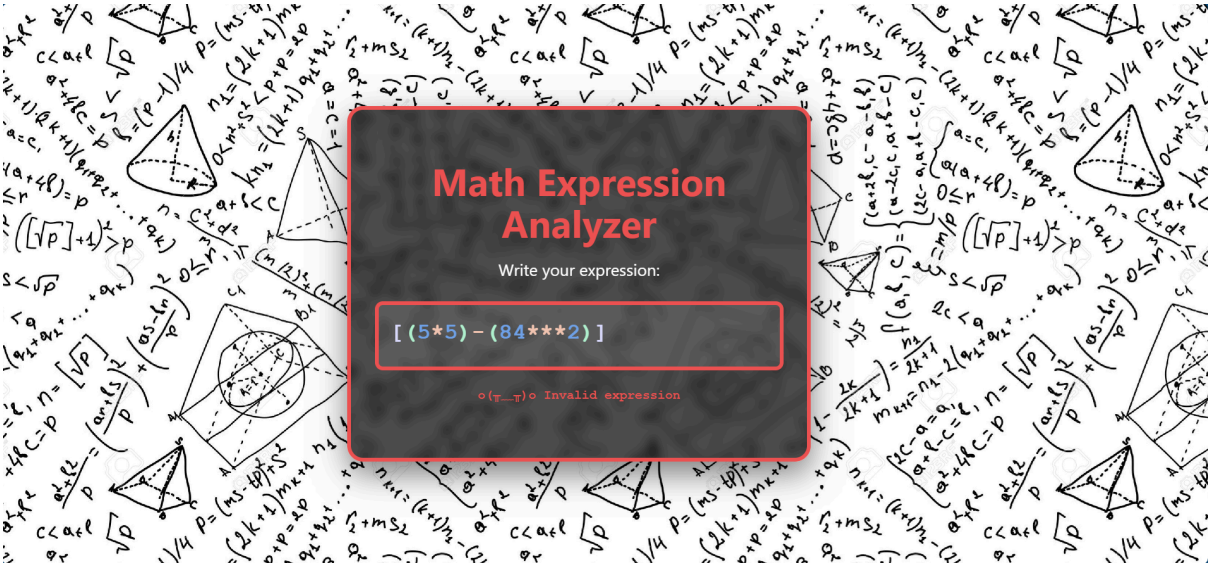


```
.then(res => res.json())
.then(data => {
  setIsValid(data.valid);
})
.catch(() => {
  setIsValid(false);
});
```

Capturas del funcionamiento:







Conclusiones:

Ana Elena:

Durante el desarrollo de este proyecto tuvimos la oportunidad de aplicar y reforzar los aprendizajes en clase como el desarrollo web, la arquitectura cliente-servidor y el análisis de expresiones matemáticas desde un enfoque tanto práctico como teórico. Trabajar con React me permitió explorar cómo una interfaz puede responder dinámicamente a lo que escribe el usuario, y cómo pequeños detalles como el cambio de colores o la validación en tiempo real pueden mejorar considerablemente la experiencia de uso.

Ana Paola:

Este proyecto me pareció muy interesante y me ayudó a comprender mejor cómo funcionan las peticiones y respuestas entre un servidor y un cliente. También me pareció muy interesante aprender un poco más sobre como crear servidores con Go y sobre cómo es posible utilizar lenguajes y tecnologías distintas para el frontend y el backend. Esto es posible gracias a que existen protocolos como HTTP(S), que estandarizan la comunicación y permiten integrar las tecnologías adecuadas y más óptimas según su propósito.. Además, me gustó aprender cómo cambiar en diferentes colores lo ingresado en el input, lo cual me ayudó a darme una mejor idea de cómo funcionan los editores de texto o IDEs, que resaltan la sintaxis en tiempo real.

César:

Este proyecto me permitió comprender mejor la estructura y validación de expresiones matemáticas, ya que pude visualizar cómo se analizan en tiempo real a través de una interfaz dinámica, al desarrollar la parte visual con React aprendimos en conjunto el cómo podíamos resaltar los elementos de la expresión como paréntesis, corchetes, números, operadores, etc.

teniendo así una gran experiencia de usuario que facilita la comprensión de la expresión escrita, fue interesante integrar esta lógica con una interfaz amigable ya que pese a que el proceso detrás es algo complicado, es importante que el usuario final tenga una facilidad de uso impecable.