PROYECTO EDD

FASE 2

Manual de Técnico

Oswaldo Antonio Choc Cuters
Estructuras de datos
Sección A
201901844

Descripción

Se realizo mediante Visual Studio Code (IDE) la creación de una api teniendo la función del backend (servidor) realizado en Golang juntamente con Fiber y la funcionalidad del frontend (cliente) en react y node.js en esta api se aplicó el uso de estructuras para poder organizar de mejor forma la información ingresada desde el cliente y así mismo ir accediendo a los métodos y funciones de cada estructura para poder trabajar de una forma más ordenada.

Es importante tener en cuenta que en el presente manual se hace mención del funcionamiento de cada clase, objeto o variable y va dirigido para cualquier persona con cierto conocimiento básico de programación.

Objetivos

Objetivos generales:

- Explicar el proceso por el cual se diseñó el programa.
- Rectificar el proceso realizado.

Objetivos específicos:

- Describir las herramientas utilizadas para el diseño y desarrollo del programa.
- Definir cada clase, método y funciones que presenta el programa.
- Definir la lógica que se utilizó para la realización del programa.
- Explicar los conceptos de estructuras de datos que se implementaron

Especificación técnica:

Se solicitó a la creación como estudiante de estructuras de datos, una aplicación para llevar control de todos los estudiantes tutores como los que desean llevar las tutorías.

Lógica del programa:

Tomando él cuenta el tema de estructura de datos se implementó varias de ellas para almacenar la información.

Estructuras utilizadas:

- **Árbol B**: Manejo de asignación de cursos de tutores.
- **Tabla Hash**: Manejo de usuarios en el sistema, tutores y estudiantes.
- **Grafos**: Manejo de relación entre cursos.
- **Árbol de Merkle**: Control de libros certificados.
- **Decodificación y Codificación**: Manejo de PDF’s de tutores.
- **Encriptación**: Contraseñas de los usuarios

Desarrollo del programa:

Librerías utilizadas:

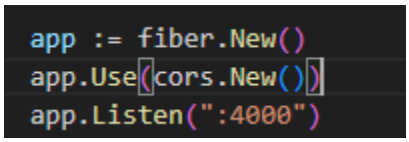
- fiber/v2
- Bootstrap
- Node.js
- Crypto/sha256
- Encoding/hex}
- Graphviz

Desarrollo:

Como se pudo explicar en los objetivos de este manual se explicará el conocimiento de como se desarrolló esta api en el área del backend lo que también es conocido como servidor. Esta api tiene integrado conceptos básicos de trabajar con un cliente-servidor y además de trabajar con estructuras de datos donde se estará almacenando la información que ingrese el cliente.

Conexión del Backend

Como se comentó anteriormente se trabajó con Fiber en el entorno del backend lo cual Fiber es un marco web Go rápido, liviano y flexible, inspirado en el marco web Express creado sobre Fast HTTP, el motor HTTP más rápido para Go, que facilita la creación de aplicaciones web con una sobrecarga mínima. Está diseñado para ser simple y fácil de usar, centrándose en el alto rendimiento y la simultaneidad



En las tres líneas de código que se muestran se logra observar la inicialización de Fiber y los cors que sirven para no tener problemas al momento de conectarse con el frontend y así mismo la última línea de código indica que se estará escuchando el servidor en el puerto 4000.

Login

En el Login o inicio de sesión se programó tres posibles escenarios los cuales son:

- Administrador
- Tutor
- Estudiante

Para poder comenzar con la api en funcionamiento adecuado, tendrá que ingresar primero el administrador una vez ingresado se podrán logear los tutores o estudiantes.

En sesión: **ADMINISTRADOR**:

Para poder ingresar como administrador el programa solo podrá reconocer las siguientes credenciales:

Usuario: Admin_201901844
Contraseña: admin

Una vez ingresado con estas credenciales se podrá realizar las cargas de los estudiantes y tutores para poder almacenarlos en el sistema. La funcionalidad del Login tiene el siguiente formato.

```
func Validar(c *fiber.Ctx) error {
    var usuario peticiones.PeticionLogin
    listaSimple = &ArbolB.ListaSimple{Inicio: nil, Longitud: 0}
    c.BodyParser(&usuario)
    if usuario.UserName == "ADMIN_201901844" {
        if usuario.Password == "admin" {
            return c.JSON(&fiber.Map{
                "status":  200,
                "message": "Credenciales correctas",
                "rol":     1,
            })
        }
    } else {
        if usuario.Tutor {
            arbolTutor.Buscar(usuario.UserName, listaSimple)
            if listaSimple.Longitud > 0 {
                if listaSimple.Inicio.Tutor.Valor.Password == SHA256(usuario.Password) {
                    fmt.Println("entro")
                    return c.JSON(&fiber.Map{
                        "status":  200,
                        "message": "Credenciales correctas",
                        "rol":     2,
                    })
                }
            }
        } else {
            if tablaAlumnos.Buscar(usuario.UserName, SHA256(usuario.Password)) {
                return c.JSON(&fiber.Map{
                    "status":  200,
                    "message": "Credenciales correctas",
                    "rol":     3,
                })
            }
        }
    }
    return c.JSON(&fiber.Map{
        "status":  400,
        "message": "Credenciales incorrectas",
        "rol":     0,
    })
}
```

La funcionalidad de Fiber mostrar un mensaje de retorno lo cual será leído por el frontend y mostrará las respectivas páginas donde podrán ingresar el administrador, tutor o estudiante.

Archivos de entrada:

El administrador podrá ingresar a los tutores, estudiantes y cursos mediante un archivo de entrada los cuales para los tutores y estudiantes es un archivo de tipo (.csv) y para los cursos un formato tipo (.json) como se muestra a continuación.

A	B	C	D
carnet	nombre	curso	password
202103105	Luis Daniel Castellanos Betancourt	772	tutor1
202103206	Kewin Maslovy Patzán Tzún	781	tutor1
202103252	Kevin Estuardo Del Cid Quezada	775	tutor1
202106003	Jhonatan Alexander Aguilar Reyes	771	tutor1
202109750	Luis Antonio Castro Padilla	770	tutor1
202110180	Juan Carlos González Valdez	772	tutor1

Archivo tutores

carnet	nombre	password	Curso 1	Curso 2	Curso 3
202110509	Mario Ernesto Marroquin Perez	2021!@	775	773	781
202110553	Andrea Jazmin Rodriguez Citalan	2021!@	770	980	785
202111563	Jose Pablo Menard Pimentel	2021!@	774	781	285
202111849	Sergio Andrés Larios Fajardo	2021!@	281	286	975
202113580	Andrés Alejandro Agosto Méndez	2021!@	980	280	281

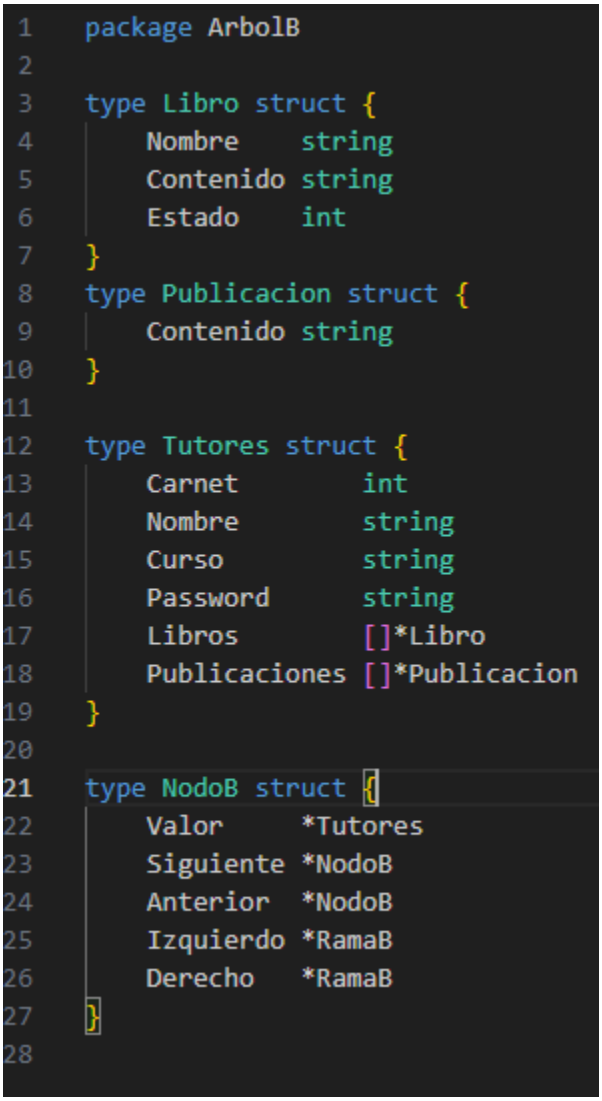
Archivo estudiantes


```
{
    "Cursos": [
        {
            "Codigo": "0771",
            "Post": ["0781", "0780", "0789"]
        },
        {
            "Codigo": "0771",
            "Post": ["0781", "0780", "0789"]
        },
        {
            "Codigo": "0771",
            "Post": ["0781", "0780", "0789"]
        },
        {
            "Codigo": "0771",
            "Post": ["0781", "0780", "0789"]
        }
    ]
}
```


Archivo cursos

Tutores:

Los tutores serán ingresados en la estructura de datos de un Árbol B los cuales tienen las siguientes estructuras.



```go
package ArbolB

type Libro struct {
    Nombre    string
    Contenido string
    Estado    int
}
type Publicacion struct {
    Contenido string
}

type Tutores struct {
    Carnet        int
    Nombre        string
    Curso         string
    Password      string
    Libros        []*Libro
    Publicaciones []*Publicacion
}

type NodoB struct {
    Valor     *Tutores
    Siguiente *NodoB
    Anterior  *NodoB
    Izquierdo *RamaB
    Derecho   *RamaB
}

```

Según la información que va a ser ingresada en el archivo de entrada desde el cliente se podrá almacenar en el Árbol B el cual en el cual desde el frontend se programó para poder leer cada línea del archivo de entrada y poder mandar esa información y desde el backend almacenarla.

```javascript
const uploadFileTutor = (event) => {
  const file = event.target.files[0];
  const reader = new FileReader();

  reader.onload = (event) => {
    const content = event.target.result;
    const parsedData = parseCSV(content);
    parsedData.map(async (row) => {
      if (row.length > 1) {
        const response = await fetch(
          "http://localhost:4000/registrar-tutor",
          {
            method: "POST",
            headers: {
              "Content-Type": "application/json",
            },
            body: JSON.stringify({
              Carnet: parseInt(row[0]),
              Nombre: row[1],
              Curso: row[2],
              Password: row[3],
            }),
          }
        );

        const result = await response.json();
      }
    });
  };
```

Parte del frontend

```
app.Post("/registrar-tutor", RegistrarTutor)
```

```go
func RegistrarTutor(c *fiber.Ctx) error {
    var tutor peticiones.PeticionRegistroTutor
    c.BodyParser(&tutor)
    //fmt.Println(tutor)
    arbolTutor.Insertar(tutor.Carnet, tutor.Nombre, tutor.Curso, SHA256(tutor.Password))
    return c.JSON(&fiber.Map{
        "status": 200,
    })
}
```

Parte del backend

Estudiantes:

Los tutores serán ingresados en la estructura de datos de la tabla hash los cuales tienen las siguientes estructuras.

```go
package tablahash

type Persona struct {
    Carnet   int
    Nombre   string
    Password string
    Cursos   []string
}

type NodoHash struct {
    Llave   int
    Persona *Persona
}

```

Según la información que va a ser ingresada en el archivo de entrada desde el cliente se podrá almacenar en la tabla hash el cual en el cual desde el frontend se programó para poder leer cada línea del archivo de entrada y poder mandar esa información y desde el backend almacenarla.

```go
app.Post("/registrar-alumno", RegistrarAlumno)
```

```go
func RegistrarAlumno(c *fiber.Ctx) error {
    var alumno peticiones.PeticionRegistroAlumno
    c.BodyParser(&alumno)
    //fmt.Println(alumno)
    tablaAlumnos.Insertar(alumno.Carnet, alumno.Nombre, SHA256(alumno.Password), alumno.Cursos)
    return c.JSON(&fiber.Map{
        "status": 200,
    })
}
```

Parte del backend

```
const uploadFileAlumno = (event) => {
  const file = event.target.files[0];
  const reader = new FileReader();

  reader.onload = (event) => {
    const content = event.target.result;
    const parsedData = parseCSV(content);
    console.log(parsedData);
    parsedData.map(async (row) => {
      if (row.length > 1) {
        const response = await fetch(
          "http://localhost:4000/registrar-alumno",
          {
            method: "POST",
            headers: {
              "Content-Type": "application/json",
            },
            body: JSON.stringify({
              Carnet: parseInt(row[0]),
              Nombre: row[1],
              Password: row[2],
              Cursos: [row[3], row[4], row[5]],
            }),
          }
        );
        const result = await response.json();
      }
    });
  };

  reader.onerror = (error) => {
    console.error("Error al leer el archivo:", error);
  };

  reader.readAsText(file);
};

const UploadCursos = (e) => {···
};
```

```
const parseCSV = (csvContent) => {
  const rows = csvContent.split("\n");
  const encabezado = rows.slice(1);
  const sinRetorno = encabezado.map((row) => row.trim());
  const data = sinRetorno.map((row) => row.split(";"));
  return data;
};
```

Parte del frontend

Una vez ingresado los alumnos a la tabla hash el contenido de esta se manda a la parte del frontend donde se podrá visualizar de una mejor forma.

```go
func TablaAlumnos(c *fiber.Ctx) error {
    return c.JSON(&fiber.Map{
        "status":  200,
        "Arreglo": tablaAlumnos.ConvertirArreglo(),
    })
}
```

Parte del backend

```jsx
const [alumnosRegistrados, SetAlumnosRegistrados] = useState([]);
useEffect(() => {
    async function peticion() {
        const response = await fetch("http://localhost:4000/tabla-alumnos");
        const result = await response.json();
        SetAlumnosRegistrados(result.Arreglo);
    }
    peticion();
}, []);
```

```jsx
<div id="contenedorTabla">
    <table>
        <thead>
            <tr>
                <th scope="col">#</th>
                <th scope="col">Posicion</th>
                <th scope="col">Carnet </th>
                <th scope="col">Nombre </th>
                <th scope="col">Password </th>
                <th scope="col">Cursos </th>
            </tr>
        </thead>
        <tbody>

            {alumnosRegistrados.map((element, j) => {
                if (element.Id_Cliente != "") {
                    return (
                        <>
                            <tr key={"alum" + j}>
                                <th scope="row">{j + 1}</th>
                                <td>{element.Llave}</td>
                                <td>{element.Persona.Carnet}</td>
                                <td>{element.Persona.Nombre}</td>
                                <td>{element.Persona.Password}</td>
                                <td>{element.Persona.Cursos[0]+", "+element.Persona.Cursos[1]+", "+element.Persona.Cursos[2]}</td>
                            </tr>
                        </>
                    );
                }
            })}

        </tbody>
    </table>
</div>
```

Parte del frontend

Curso:

Los tutores serán ingresados como datos de un grafo los cuales tienen las siguientes estructuras.

```go
package Grafo

type NodoListaAdyacencia struct {
    Siguiente *NodoListaAdyacencia
    Abajo     *NodoListaAdyacencia
    Valor     string
}
```

Según la información que va a ser ingresada en el archivo de entrada desde el cliente se podrá almacenar en el grafo el cual en el cual desde el frontend se programó para poder leer cada línea del archivo de entrada y poder mandar esa información y desde el backend almacenarla.

```go
app.Post("/registrar-cursos", RegistrarCursos)
```

```go
func RegistrarCursos(c *fiber.Ctx) error {
    var cursito peticiones.PeticionCursos
    c.BodyParser(&cursito)
    //fmt.Println(cursito)
    for _, curso := range cursito.Cursos {
        if len(curso.Post) > 0 {
            for j := 0; j < len(curso.Post); j++ {
                grafoCursos.InsertarValores(curso.Codigo, curso.Post[j])
            }
        } else {
            grafoCursos.InsertarValores("ECYS", curso.Codigo)
        }
    }
    return c.JSON(&fiber.Map{
        "status": 200,
    })
}
```

Parte del backend

```
const UploadCursos = (e) => {
  var reader = new FileReader();
  reader.onload = async (e) => {
    var obj = JSON.parse(e.target.result);
    console.log(obj);
    const response = await fetch("http://localhost:4000/registrar-cursos", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({
        Cursos: obj.Cursos,
      }),
    });
  };
  reader.readAsText(e.target.files[0]);
};


const parseCSV = (csvContent) => {
  const rows = csvContent.split("\n");
  const encabezado = rows.slice(1);
  const sinRetorno = encabezado.map((row) => row.trim());
  const data = sinRetorno.map((row) => row.split(";"));
  return data;
};
```

```
const parseCSV = (csvContent) => {
  const rows = csvContent.split("\n");
  const encabezado = rows.slice(1);
  const sinRetorno = encabezado.map((row) => row.trim());
  const data = sinRetorno.map((row) => row.split(";"));
  return data;
};
```

Parte del frontend

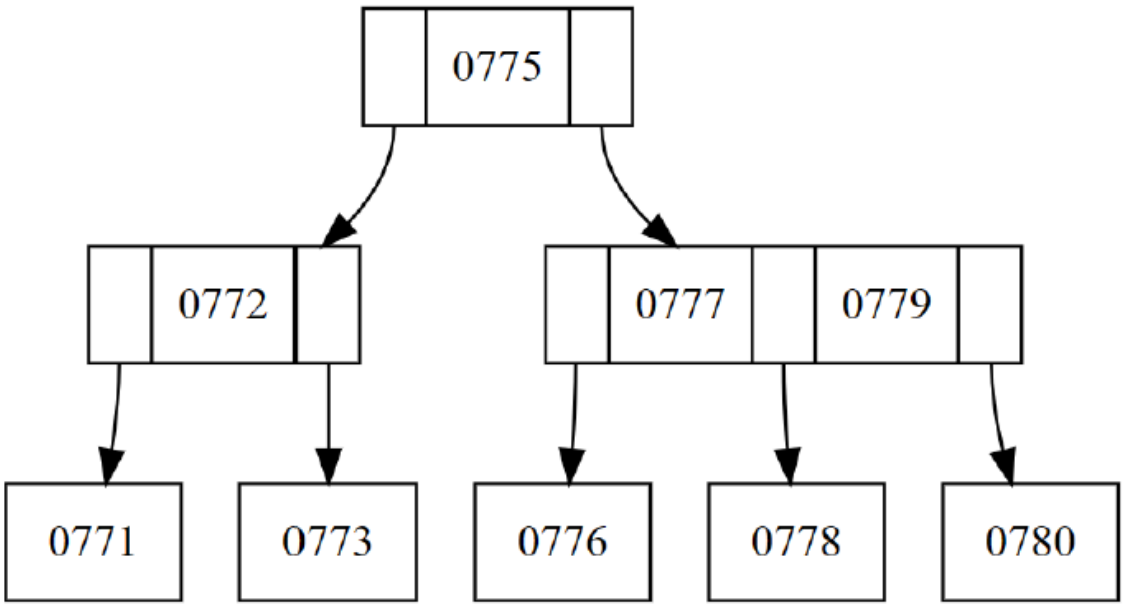
Reportes:

Como administrador tendrá la opción de mostrar los reportes de la información que fue almacenada en los archivos de entrada los cuales son los siguientes:

- Reporte Árbol B
- Reporte Grafo
- Reporte Árbol Merkle

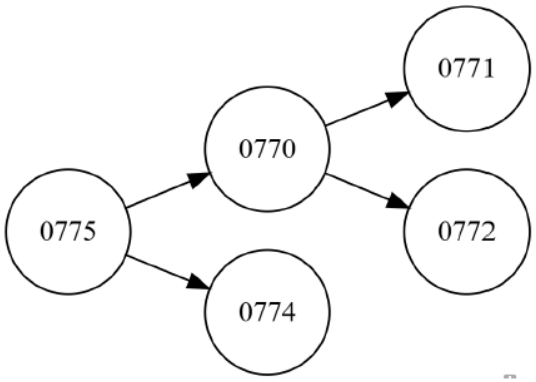
Reporte ArbolB:

En este reporte se mostrará la información de los tutores que fueron almacenado y deberá de tener el siguiente formato creado con Graphviz mostrando el código del curso para mejor visualización.



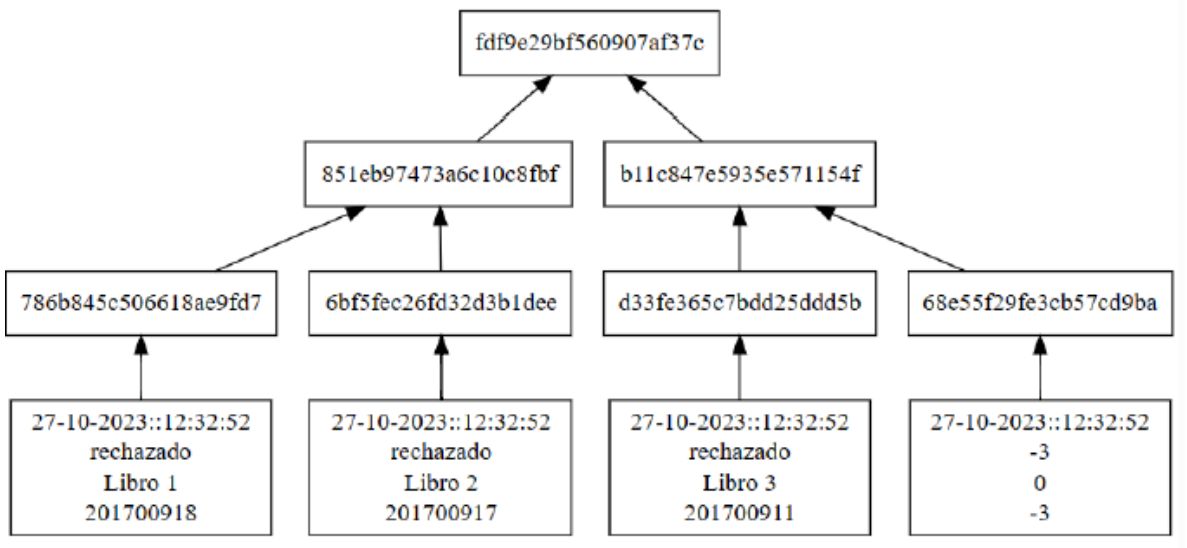
Reporte Grafo:

Para la realización del reporte se usará la lista de adyacencia para partir del grafo, para ello se mostrará el código del curso y su post-requisito de la siguiente manera:



Reporte Árbol Merkle:

Para la realización del reporte se hará de la siguiente manera, mostrando únicamente las cadenas resultantes de las funciones hash a excepción de la última línea que muestra los datos que se utilizó para esas funciones.



Funciones para la creación de los archivos de los reportes.

```go
package GenerarArchivos

import (
    "fmt"
    "os"
    "os/exec"
)

func CrearArchivo(nombre_archivo string) {
    var _, err = os.Stat(nombre_archivo)

    if os.IsNotExist(err) {
        var file, err = os.Create(nombre_archivo)
        if err != nil {
            return
        }
        defer file.Close()
    }
    fmt.Println("Archivo generado exitosamente")
}

func EscribirArchivo(contenido string, nombre_archivo string) {
    var file, err = os.OpenFile(nombre_archivo, os.O_RDWR, 0644)
    if err != nil {
        return
    }
    defer file.Close()
    _, err = file.WriteString(contenido)
    if err != nil {
        return
    }
    err = file.Sync()
    if err != nil {
        return
    }
    fmt.Println("Archivo guardado correctamente")
}

func Ejecutar(nombre_imagen string, archivo string) {
    path, _ := exec.LookPath("dot")
    cmd, _ := exec.Command(path, "-Tjpg", archivo).Output()
    mode := 0777
    _ = os.WriteFile(nombre_imagen, cmd, os.FileMode(mode))
    //dot lista.dot -Tjpg lista.jpg -o
}
```

Aceptar o rechazar libros:

El Administrador podrá aceptar o rechazar libros que los tutores carguen, por lo cual debe realizar una interfaz donde pueda ver los libros, una vez que el libro sea aceptado o rechazado, se debe tener un control de seguridad, se utilizará el árbol de Merkel el cual es una estructura fiable para conservar la integridad de datos, para realizar este árbol se realizara el método SHA-3 para las funciones de hash criptográfico, partirá del siguiente información desde los nodos hojas, sin importar si se acepta o rechaza, por lo cual se debe considerar lo siguiente:

La información que contendrá cada nodo hoja para realizar dicho árbol es la siguiente:

- Fecha y hora de Acción (se tomará la información de la fecha de su computadora, tomando en cuenta el siguiente formato (DD-MM-YY::HH:MM:SS)
- Acción realizada (Rechazado o Aceptado)
- Nombre del libro
- Tutor que subió el libro

En sesión: **TUTOR**:

Para poder iniciar sesión como tutor se deberá en el programa como usuario el número de carnet del tutor y la contraseña según los archivos de entrada ingresados por el administrador donde se encuentra la contraseña correspondiente de cada tutor, por ejemplo:

carnet	nombre	curso	password
202103105	Luis Daniel Castellanos Betancourt	772	tutor1

- Usuario: 202103105
- Contraseña: tutor1

Para realizar las validaciones correspondientes se busca dentro de los valores del árbol B que se ingresaron por el administrador y se validan si el carnet ingresado existe y si coincide con la contraseña del mismo. Si cumple se mostrar el menú de tutor.

Subir libro:

La opción de subir libro se guardará en formato de archivo pdf y así como se trabajó con los demás archivos de entrada primeramente se leerá la información almacenada directamente desde el frontend y luego se mandará esa información en este caso el archivo al backend y se deberá de almacenar el libro a la estructura de datos del árbol B del tutor que fue logeado.

```go
func (a *ArbolB) GuardarLibro(raiz *NodoB, nombre string, contenido string, carnet int) {
    if raiz != nil {
        aux := raiz
        for aux != nil {
            if aux.Izquierdo != nil {
                a.GuardarLibro(aux.Izquierdo.Primero, nombre, contenido, carnet)
            }
            if aux.Valor.Carnet == carnet {
                raiz.Valor.Libros = append(raiz.Valor.Libros, &Libro{Nombre: nombre, Contenido: contenido, Estado: 1})
                fmt.Println("Registre el libro")
                return
            }
            if aux.Siguiente == nil {
                if aux.Derecho != nil {
                    a.GuardarLibro(aux.Derecho.Primero, nombre, contenido, carnet)
                }
            }
            aux = aux.Siguiente
        }
    }
}
```

```
app.Post("/registrar-libro", GuardarLibro)
```

```
func GuardarLibro(c *fiber.Ctx) error {
    var libro peticiones.PeticionLibro
    c.BodyParser(&libro)
    //fmt.Println(libro)
    arbolTutor.GuardarLibro(arbolTutor.Raiz.Primero, libro.Nombre, libro.Contenido, libro.Carnet)
    return c.JSON(&fiber.Map{
        "status": 200,
    })
}
```

Parte del backend

```
function CargarLibros() {
    const [contenidoPDF, setContenidoPDF] = useState("");
    const uploadFileTutor = (event) => {
      const file = event.target.files[0];
      const reader = new FileReader();

      reader.onload = async (event) => {
        const content = event.target.result;
        //console.log(content);
        setContenidoPDF(content);
        const valorLocal = localStorage.getItem("user");
        const response = await fetch("http://localhost:4000/registrar-libro", {
          method: "POST",
          headers: {
            "Content-Type": "application/json",
          },
          body: JSON.stringify({
            Carnet: parseInt(valorLocal),
            Nombre: "Libro1",
            Contenido: content,
          }),
        });

        const result = await response.json();
      };

      reader.onerror = (error) => {
        console.error("Error al leer el archivo:", error);
      };

      reader.readAsDataURL(file);
    };
```

Parte del frontend

Subir Publicación:

La opción de subir publicación el tutor tendrá la opción de poder escribir en un texarea la publicación que sea de su interés y así mismo subir dicha publicación y guardarla en sus datos la publicación que fue escrita

y se deberá de almacenar la publicación a la estructura de datos del árbol B del tutor que fue logeado.

```go
func (a *ArbolB) GuardarPublicacion(raiz *NodoB, contenido string, carnet int) {
    if raiz != nil {
        aux := raiz
        for aux != nil {
            if aux.Izquierdo != nil {
                a.GuardarPublicacion(aux.Izquierdo.Primero, contenido, carnet)
            }
            if aux.Valor.Carnet == carnet {
                raiz.Valor.Publicaciones = append(raiz.Valor.Publicaciones, &Publicacion{Contenido: contenido})
                fmt.Println("Registre la publicación")
                return
            }
            if aux.Siguiente == nil {
                if aux.Derecho != nil {
                    a.GuardarPublicacion(aux.Derecho.Primero, contenido, carnet)
                }
            }
            aux = aux.Siguiente
        }
    }
}
```

```go
app.Post("/registrar-publicacion", GuardarPublicacion)
```

```go
func GuardarPublicacion(c *fiber.Ctx) error {
    var publicacion peticiones.PeticionPublicacion
    c.BodyParser(&publicacion)
    fmt.Println(publicacion)
    arbolTutor.GuardarPublicacion(arbolTutor.Raiz.Primero, publicacion.Contenido, publicacion.Carnet)
    return c.JSON(&fiber.Map{
        "status": 200,
    })
}
```

Parte del backend

```javascript
const [publicacion, setPublicacion] = useState("");

const MandarPublicacion = async (event) => {
  event.preventDefault();
  const valorLocal = localStorage.getItem("user");
  const response = await fetch("http://localhost:4000/registrar-publicacion", {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify({
      Carnet: parseInt(valorLocal),
      Contenido: publicacion,
    }),
  });
};
```

Parte del frontend

En sesión: **Estudiante**:

Para poder iniciar sesión como tutor se deberá en el programa como usuario el número de carnet del estudiante y la contraseña según los archivos de entrada ingresados por el administrador donde se encuentra la contraseña correspondiente de cada tutor, por ejemplo:

202110509	Mario Ernesto Marroquin Perez	2021!@	775	773	781

- Usuario: 202110509
- Contraseña: 2021!@

Para realizar las validaciones correspondientes se busca dentro de los valores de la tabla hash que se ingresaron por el administrador y se validan si el carnet ingresado existe y si coincide con la contraseña del mismo. Si cumple se mostrar el menú de estudiante.

Para el menú de estudiante se podrá visualizar los datos almacenados del mismo y los datos de los tutores que hayan hecho alguna publicación o hayan subido un libro de la siguiente manera:

Ver libros:
Aquí el estudiante podrá ver los libros que sus tutores han subido y ya fueron aceptados por el administrador. Aquí el estudiante podrá ver cada uno de ellos, al estar codificados en Base64.

Ver Publicaciones:
Aquí el estudiante podrá ver las publicaciones que sus tutores han realizado. Aquí el estudiante podrá ver cada uno de ellos.