Manual Tecnico

USAC PENSUM

Lenguajes Formales y de programación
Douglas Xavier Santiago Soto Mejia

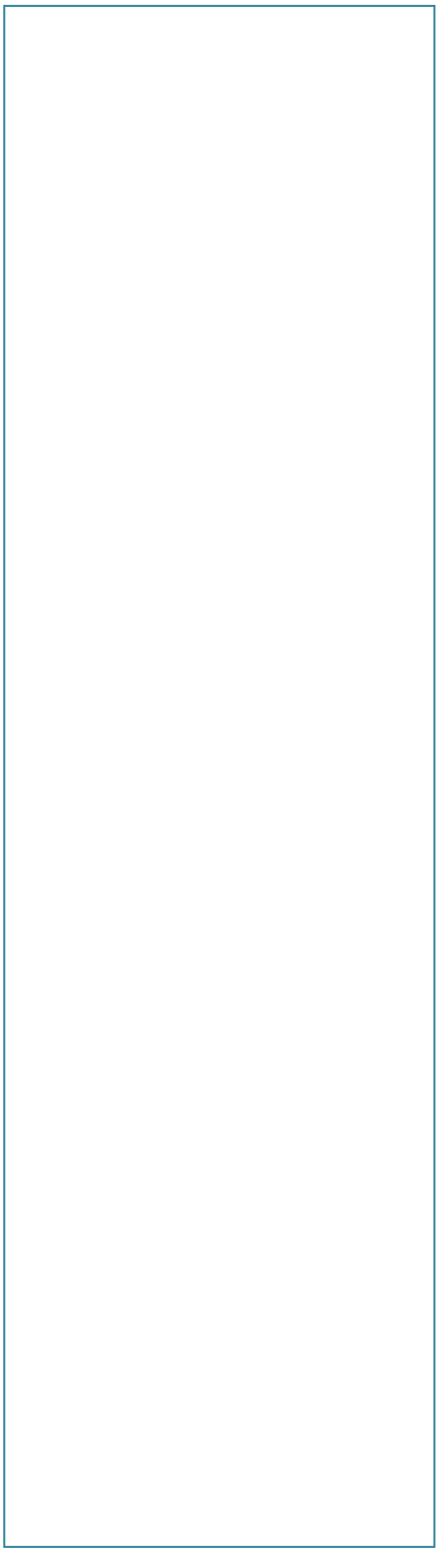


Tabla de contenido

Manual Tecnico

Introducción

A través del lenguaje de programación TypeScript, se realiza un servidor web que simula la obtención de un diversos pensums para diversas carreras en la universidad, los cuales pasaran por un análisis léxico previo.

Objetivo

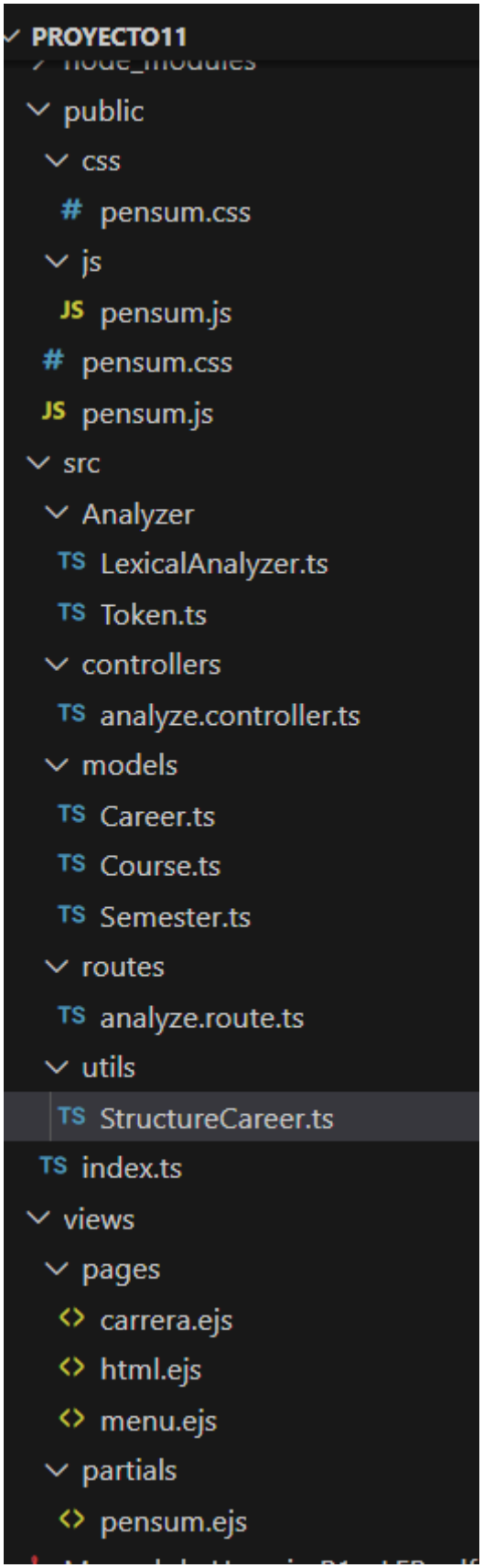
El objetivo de la aplicación es analizar los cursos y los semestres contenidos en un archivo, una vez hecho esto, pasaremos a la fase del analizador léxico, que, si ejecuta sin ningún error, mostrara en un HTML el pensum ingresado.

Contenido

Requerimientos

- Computadora Portatil.
- Windows 7 o superior.
- 4 de RAM minimo.
- Netbeans 8.2 o superior.
- Sistema operativo Windows

Carpetas


PROYECTO11
public
css
pensum.css
js
pensum.js
pensum.css
pensum.js
src
Analyzer
LexicalAnalyzer.ts
Token.ts
controllers
analyze.controller.ts
models
Career.ts
Course.ts
Semester.ts
routes
analyze.route.ts
utils
StructureCareer.ts
index.ts
views
pages
carrera.ejs
html.ejs
menu.ejs
partials
pensum.ejs

Analyzer

Tokens

Codigo para el declaramiento de tokens

```
enum Type {
    UNKNOW,
    KEY_OPEN,
    KEY_CLOSE,
    COR_OPEN,
    COR_CLOSE,
    PAR_OPEN,
    PAR_CLOSE,
    SEMICOLON,
    EQUAL,
    RESERVERD_WORD,
    NUMBER,
    STRING,
    COLON,
    ASSIGN,
    COMA
}

class Token {

    private row: number;
    private column: number;
    private lexeme: string;
    private typeToken: Type;
    private typeTokenString: string;

    constructor(typeToken: Type, lexeme: string, row: number, column: number) {
        this.typeToken = typeToken;
        this.typeTokenString = Type[typeToken];
        this.lexeme = lexeme;
        this.row = row;
        this.column = column;
    }

    public getRow(): number {
        return this.row;
    }
```

```
    public getRow(): number {
        return this.row;
    }

    public getColumn(): number {
        return this.column;
    }

    public getLexeme(): string {
        return this.lexeme;
    }

    public getTypeToken(): Type {
        return this.typeToken;
    }

    public getTypeTokenString(): string {
        return this.typeTokenString;
    }

    public setRow(row: number): void {
        this.row = row;
    }

    public setColumn(column: number): void {
        this.column = column;
    }

    public setLexeme(lexeme: string): void {
        this.lexeme = lexeme;
    }

    public setTypeToken(typeToken: Type): void {
        this.typeToken = typeToken;
        this.typeTokenString = Type[typeToken];
```

```
    public setTypeToken(typeToken: Type): void {
        this.typeToken = typeToken;
        this.typeTokenString = Type[typeToken];
    }

    public setTypeTokenString(typeTokenString: string): void {
        this.typeTokenString = typeTokenString;
    }
}

export { Token, Type };
```

LexicalAnalyzer

Codigo del analizador lexico.

```
import { Token, Type } from "./Token";

class LexicalAnalyzer {
    private row: number;
    private column: number;
    private auxChar: string;
    private state: number;
    private tokenList: Token[];
    private errorList: Token[];
    private reservedWords: string[];

    constructor() {
        this.row = 1;
        this.column = 1;
        this.auxChar = '';
        this.state = 0;
        this.tokenList = [];
        this.errorList = [];
        this.reservedWords = ["Carrera", "Semestre", "Curso", "Nombre", "Area", "Prerequisitos"];
    }

    public scanner(input: string): Token[] {
        input += "#";
        let char: string;

        for (let i = 0; i < input.length; i++) {
            char = input[i];

            switch (this.state) {
                case 0:
                    if (/[a-zA-Z]/.test(char)) {
                        this.state = 1;
                        this.addCharacter(char);
                    } else if (char === '"') {
                        this.state = 2;
                        this.addCharacter(char);
                    } else if (/\d/.test(char)) {
```

```
                this.addCharacter(char);
            } else {
                switch (char) {
                    case ':': this.state = 6; break;
                    case '=': this.state = 7; break;
                    case ';': this.state = 8; break;
                    case '{': this.state = 9; break;
                    case '}': this.state = 10; break;
                    case '[': this.state = 11; break;
                    case ']': this.state = 12; break;
                    case '(': this.state = 13; break;
                    case ')': this.state = 14; break;
                    case ',': this.state = 15; break;
                    case ' ':
                    case '\t':
                        this.column += (char === '\t') ? 4 : 1;
                        continue;
                    case '\n':
                    case '\r':
                        this.row++;
                        this.column = 1;
                        continue;
                    case '#':
                        if (i === input.length - 1) console.log("Analisis léxico finalizado.");
                        continue;
                    default:
                        this.addError(Type.UNKNOW, char, this.row, this.column++);
                        continue;
                }
                this.addCharacter(char);
            }
            break;
```

```
    case 1:
        if (/[a-zA-Z0-9]/.test(char)) {
            this.addCharacter(char);
        } else {
            const lexeme = this.auxChar;
            const type = this.reservedWords.includes(lexeme)
                ? Type.RESERVERD_WORD
                : Type.UNKNOW;
            this.addToken(type, lexeme, this.row, this.column - lexeme.length);
            this.clean();
            i--;
        }
        break;

    case 2:
        if (char === '"') {
            this.addCharacter(char);
            this.state = 4;
        } else {
            this.addCharacter(char);
            this.state = 3;
        }
        break;

    case 3:
        if (char === '"') {
            this.addCharacter(char);
            this.state = 4;
        } else {
            this.addCharacter(char);
        }
        break;

    case 4:
```

```
        case 4:
            this.addToken(Type.STRING, this.auxChar, this.row, this.column - this.auxChar.length);
            this.clean();
            i--;
            break;

        case 5:
            if (/\d/.test(char)) {
                this.addCharacter(char);
            } else {
                this.addToken(Type.NUMBER, this.auxChar, this.row, this.column - this.auxChar.length);
                this.clean();
                i--;
            }
            break;

        case 6:
            this.addToken(Type.COLON, this.auxChar, this.row, this.column - 1);
            this.clean(); i--; break;

        case 7:
            this.addToken(Type.EQUAL, this.auxChar, this.row, this.column - 1);
            this.clean(); i--; break;

        case 8:
            this.addToken(Type.SEMICOLON, this.auxChar, this.row, this.column - 1);
            this.clean(); i--; break;

        case 9:
            this.addToken(Type.COR_OPEN, this.auxChar, this.row, this.column - 1);
            this.clean(); i--; break;

        case 10:
            this.addToken(Type.COR_CLOSE, this.auxChar, this.row, this.column - 1);
```

```
public scanner(input: string): Token[] {
                this.addToken(Type.COR_CLOSE, this.auxChar, this.row, this.column - 1);
                this.clean(); i--; break;

            case 11:
                this.addToken(Type.KEY_OPEN, this.auxChar, this.row, this.column - 1);
                this.clean(); i--; break;

            case 12:
                this.addToken(Type.KEY_CLOSE, this.auxChar, this.row, this.column - 1);
                this.clean(); i--; break;

            case 13:
                this.addToken(Type.PAR_OPEN, this.auxChar, this.row, this.column - 1);
                this.clean(); i--; break;

            case 14:
                this.addToken(Type.PAR_CLOSE, this.auxChar, this.row, this.column - 1);
                this.clean(); i--; break;

            case 15:
                this.addToken(Type.COMA, this.auxChar, this.row, this.column - 1);
                this.clean(); i--; break;
        }
    }

    return this.tokenList;
}

private addCharacter(char: string): void {
    this.auxChar += char;
    this.column++;
}

private clean(): void {
    this.auxChar = '';
```

```
        }

        return this.tokenList;
    }

    private addCharacter(char: string): void {
        this.auxChar += char;
        this.column++;
    }

    private clean(): void {
        this.auxChar = '';
        this.state = 0;
    }

    private addToken(type: Type, lexeme: string, row: number, column: number): void {
        this.tokenList.push(new Token(type, lexeme, row, column));
    }

    private addError(type: Type, lexeme: string, row: number, column: number): void {
        this.errorList.push(new Token(type, lexeme, row, column));
    }

    public getErrorList(): Token[] {
        return this.errorList;
    }

    public getTokenList(): Token[] {
        return this.tokenList;
    }
}

export { LexicalAnalyzer };
```

Analyzers

Analyze.controller

Codigo utilizado para controlador.

```
import { Request, Response } from "express";
import { LexicalAnalyzer } from "../Analyzer/LexicalAnalyzer";
import { Career } from "../models/Career";
import { getCareers } from "../utils/StructureCareer";


export const analyze = (req: Request, res: Response) => {

    const input = req.body;
    let lexicalAnalyzer: LexicalAnalyzer = new LexicalAnalyzer();

    lexicalAnalyzer.scanner(input);

    let tokenList = lexicalAnalyzer.scanner(input);
    let errorList = lexicalAnalyzer.getErrorList();
    let careers: Career[] = getCareers(tokenList);


    res.json({
        "tokens": tokenList,
        "errors": errorList,
        "careers": careers
    });
}

export const home = (req: Request, res: Response) => {

    res.render('pages/menu');
}

export const pensum = (req: Request, res: Response) => {

    const id = req.params.id;
    res.render('pages/carrera', {id});
}
```

Analyze.route

Codigo utilizado para la ruta a llamar.

```
import { Router } from "express";
import { analyze, home, pensum } from "../controllers/analyze.controller";

const analyzeRouter = Router();

analyzeRouter.get('/', home);
analyzeRouter.post('/analyze', analyze);
analyzeRouter.get('/pensum/:id', pensum);

export default analyzeRouter;
```

pages

index.ts

codigo principal para definir la ruta del servidor.

```
import express from 'express';
import analyzeRouter from './routes/analyze.route';

const app = express();
const PORT = 4000;

app.set('view engine', 'ejs');

app.use(express.static('public'));
app.use(express.text());
app.use(analyzeRouter);

app.listen(PORT, () => {
    console.log(`el servidor es http://localhost:${PORT}`);
});
```

Package.json

codigo que contiene la ruta

```
{
  "name": "proyecto11",
  "version": "1.0.0",
  "main": "index.js",
  ▷Debug
  "scripts": {
    "dev": "nodemon --exec ts-node src/index.ts"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "devDependencies": {
    "@types/express": "^5.0.3",
    "@types/node": "^24.0.1",
    "nodemon": "^3.1.10",
    "ts-node": "^10.9.2",
    "typescript": "^5.8.3"
  },
  "dependencies": {
    "ejs": "^3.1.10",
    "express": "^5.1.0"
  }
}
```

Menu.ejs

codigo que contiene el html de la pagina principal.

```html
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title>Pensum USAC - Analizador Léxico</title>
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0-beta3/css/all.min.css" />
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.65.2/codemirror.min.css" />
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.65.2/theme/dracula.min.css" />
  <style>
    body { background-color: #f0f0f0; font-family: Arial, sans-serif; margin: 0; }
    header { background-color: #006d77; color: white; padding: 1rem; text-align: center; }
    nav { background-color: #023047; padding: 0.5rem; display: flex; justify-content: space-around; }
    nav a { color: white; text-decoration: none; padding: 0.5rem 1rem; }
    nav a:hover { background-color: #219ebc; }
    .container { display: flex; flex-wrap: wrap; padding: 1rem; gap: 1rem; }
    .editor-section, .results-section { background-color: white; border-radius: 8px; padding: 1rem; box-shadow: 0 2px 4px rgba(0,0,0,0.1)
    .editor-section { flex: 2; min-width: 300px; }
    .results-section { flex: 3; min-width: 300px; display: flex; flex-direction: column; gap: 1rem; }
    .CodeMirror { height: 300px; }
    table { width: 100%; border-collapse: collapse; margin-top: 1rem; }
    th, td { padding: 0.5rem; border: 1px solid #ccc; text-align: left; }
    th { background-color: #023047; color: white; }
    .error-row { color: #d00000; font-weight: bold; }
    .file-options { display: flex; gap: 0.5rem; margin-top: 1rem; }
    button { background-color: #219ebc; color: white; border: none; padding: 0.5rem 1rem; border-radius: 4px; cursor: pointer; }
    button:hover { background-color: #0077b6; }
    .hidden { display: none; }
  </style>
</head>
<body>
  <header>
    <h1>Pensum Interactivo USAC</h1>
    <p>Analizador Léxico - Proyecto LFP 2025</p>
  </header>

  <nav>
```

```
  <nav>
    <a href="/">Inicio</a>
    <a href="#" id="error-report-link">Reporte de Errores</a>
    <a href="#" id="manual-tecnico-link">Manual Tecnico</a>
    <a href="#" id="manual-usuario-link">Manual de Usuario</a>
  </nav>

  <div class="container">
    <div class="editor-section">
      <h2>Editor de Código (.plfp)</h2>
      <textarea id="code-editor">Carrera: "Ingeniería en Sistemas" [
    Semestre: 01 {
        Curso: 101 {
            Nombre: "Introducción a la Programación";
            Area: 01;
            Prerequisitos: (101);
        }
    }
]</textarea>

      <div class="file-options">
        <button id="clear-editor">Limpiar</button>
        <button id="load-file">Cargar Archivo</button>
        <input type="file" id="file-input" accept=".plfp" style="display: none;" />
      </div>
      <button id="analyze-button">Analizar</button>
    </div>

    <div class="align-items-center" id="pensums"></div>

    <div class="results-section">
      <div class="tokens-container">
        <h2>Tabla de Tokens</h2>
        <table id="tokens-table">
```

```
      <div class="tokens-container">
        <table id="tokens-table">
          <thead>
            <tr>
              <th>No.</th>
              <th>Fila</th>
              <th>Columna</th>
              <th>Lexema</th>
              <th>Token</th>
            </tr>
          </thead>
          <tbody></tbody>
        </table>
      </div>

      <div class="errors-container hidden" id="error-report-container">
        <h2>Errores Léxicos</h2>
        <table id="errors-table">
          <thead>
            <tr>
              <th>No.</th>
              <th>Fila</th>
              <th>Columna</th>
              <th>Caracter</th>
              <th>Descripción</th>
            </tr>
          </thead>
          <tbody></tbody>
        </table>
      </div>
    </div>
  </div>
```

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.65.2/codemirror.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.65.2/mode/clike/clike.min.js"></script>
<script>
  const editor = CodeMirror.fromTextArea(document.getElementById('code-editor'), {
    lineNumbers: true,
    mode: 'text/x-csrc',
    theme: 'dracula',
    indentUnit: 4,
    tabSize: 4,
    lineWrapping: true
  });

  document.addEventListener('DOMContentLoaded', () => {
    const clearButton = document.getElementById('clear-editor');
    const loadButton = document.getElementById('load-file');
    const fileInput = document.getElementById('file-input');
    const analyzeButton = document.getElementById('analyze-button');
    const tokensTable = document.getElementById('tokens-table').getElementsByTagName('tbody')[0];
    const errorsTable = document.getElementById('errors-table').getElementsByTagName('tbody')[0];
    const errorReportLink = document.getElementById('error-report-link');
    const errorReportContainer = document.getElementById('error-report-container');
    const pensums = document.getElementById('pensums');

    clearButton.addEventListener('click', () => editor.setValue(''));
    loadButton.addEventListener('click', () => fileInput.click());

    fileInput.addEventListener('change', (event) => {
      const file = event.target.files[0];
      if (file) {
        const reader = new FileReader();
        reader.onload = (e) => editor.setValue(e.target.result);
        reader.readAsText(file);
      }
    });
```

```
errorReportLink.addEventListener('click', (e) => {
  e.preventDefault();
  errorReportContainer.classList.toggle('hidden');
});

analyzeButton.addEventListener('click', async () => {
  const content = editor.getValue().trim();
  if (!content) {
    alert('Que vas a analizar? si no hay nada');
    return;
  }

  try {
    const response = await fetch('/analyze', {
      method: 'POST',
      headers: { 'Content-Type': 'text/plain' },
      body: content
    });

    const data = await response.json();
    pensums.innerHTML = '';

    tokensTable.innerHTML = '';
    errorsTable.innerHTML = '';

    if (data.tokens?.length > 0) {
      data.tokens.forEach((token, index) => {
        const row = tokensTable.insertRow();
        row.insertCell(0).textContent = index + 1;
        row.insertCell(1).textContent = token.row;
        row.insertCell(2).textContent = token.column;
        row.insertCell(3).textContent = token.lexeme;
```

```
        row.insertCell(2).textContent = token.column;
        row.insertCell(3).textContent = token.lexeme;
        row.insertCell(4).textContent = token.typeTokenString || token.type;
      });
    } else {
      const row = tokensTable.insertRow();
      row.insertCell(0).textContent = 'Sin tokens encontrados';
      row.cells[0].colSpan = 5;
    }

    if (data.errors?.length > 0) {
      errorReportContainer.classList.remove('hidden');
      data.errors.forEach((error, index) => {
        const row = errorsTable.insertRow();
        row.className = 'error-row';
        row.insertCell(0).textContent = index + 1;
        row.insertCell(1).textContent = error.row;
        row.insertCell(2).textContent = error.column;
        row.insertCell(3).textContent = error.lexeme || ' ';
        row.insertCell(4).textContent = error.description || 'Desconocido';
      });
    } else {
      errorReportContainer.classList.add('hidden');
      alert('Analisis completado, libre de errores');

      console.log(data.careers);

      data.careers.forEach(( _, index) =>{
        pensums.innerHTML += `<a class="btn btn-success btn_user" href="${'/pensum/' + (index + 1)}" target="_blank">${'Pensum: ' + (in
      });
```

```
            pensums.innerHTML += `<a class="btn btn-success btn_user" href="${'/pensum/' + (index + 1)}" target="_blank
          });

          data.careers.forEach((career, index) =>{
              localStorage.setItem(`pensum${index+1}`, JSON.stringify(career.html));

          });

        }

      } catch (err) {
        console.error(err);
        alert(`ERROR ${err.message}`);
      }
    });
  });
 </script>
</body>
</html>
```

Carrera.ejs

Codigo de la pagina Html del pensum.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.65.2/codemirror.min.css" />
    <link rel="stylesheet" href="/css/pensum.css">
    <title>Tu pensum we</title>
</head>
<body>
    <h1>Este es tu pensum</h1>
    <div class="container_user">
        <%- include('../partials/pensum', {id}) %>
    </div>
<script src="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.65.2/codemirror.min.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.65.2/mode/clike/clike.min.js"></script>
    <script src="/js/pensum.js"></script>
</body>
</html>
```

models

careers

Codigo para las carreras.

```
import { Semester } from "./Semester";
export class Career{

    private name: string;
    private semesters: Semester[];
    private html: string;

    constructor(name: string){
        this.name = name;
        this.semesters = [];
        this.html='';

    }

    getName(): string{
        return this.name;
    }

    addSemester(semester: Semester){
        this.semesters.push(semester);
    }

    getSemesters(): Semester[]{
        return this.semesters;
    }

    generateHtml(){
        this.html = `
        <h1> Ingenieria: ${this.name} </h1>

        <table id= "${this.name}" class="table_pensum table table-striped">
        <thead>
        <tr>
${this.semesters.map((item) =>{
      return `<th Semestre NO.${item.getNum()} </th>`
}).join('\n')}
        </tr>
```

```
  ${this.semesters.map((item) =>{
        return `<th Semestre NO.${item.getNum()} </th>`
  }).join('\n')}
          </tr>
          </thead>
  <tbody>
          `;
  for(let i =0; i<6; i++){
      this.html +=`<tr>`;

      for(const semester of this.semesters){
          this.html += `${semester.getHtml()[i]}`;
      }
      this.html += `</tr>\n`;
  }

  this.html += `
  </tbody>
  </table>
  `;
      }

      getHtml(): string{
          return this.html;
      }
  }
```

Course

Codigo para los cursos.

```
export class Course {

    private code: string;
    private name: string;
    private area: number;
    private prerequisites: string[];
    private html: string;

    constructor(code: string){
        this.code = code;
        this.name ='';
        this.area = 0;
        this.prerequisites =[];
        this.html= '';
    }

    getCode(): string {
        return this.code
    }

    setName(name: string){
        this.name= name;
    }

    getName(): string {
        return this.name
    }

    setArea(area: number){
        this.area=area;
    }

    getArea(): number{
        return this.area
    }

    addPrerequisites(code: string){
```

```
    getArea(): number{
        return this.area
    }

    addPrerequisites(code: string){
        this.prerequisites.push(code);
    }

    getPrerequisites(): string[]{
        return this.prerequisites;
    }

    generateHtml(){
        this.html=`
        <div id= "${this.code}">
        <span class="codigo">${this.code}</span>
        <span>${this.name}</span>
        <span class="pre">
            ${this.prerequisites.map((item) =>{
               return `<p>${item}</p>`
            }).join('\n\t')}
        </span>
        </div>
        `;

    }

    getHtml(): string{
        return this.html;
    }

}
```

Semester

Codigo para elos semestres.

```
import {Course} from "./Course";
export class Semester{

    private num: number;
    private courses: Course[]
    private html: string[];

    constructor(num: number){
        this.num=num;
        this.courses=[];
        this.html = [];
    }

    getNum(): number{
        return this.num;
    }

    addCourse(course: Course){
        this.courses.push(course);
    }

    getCourses(): Course[]{
        return this.courses;
    }

    generateHtml(){
        for(let i=0; i<6; i++){
            this.html[i]= `
            <td>
${this.getCourseArea(i+1)}

            </td>
            `;
        }
    }
    private getCourseArea(area: number): string {
        let htmlCourse = '';
```

```
    generateHtml(){
        for(let i=0; i<6; i++){
            this.html[i]= `
            <td>
${this.getCourseArea(i+1)}

            </td>
            `;
        }
    }
    private getCourseArea(area: number): string {
        let htmlCourse = '';
        for(const course of this.courses){
            if(course.getArea()==area){
                htmlCourse += `${course.getHtml()}\n`;
            }
        }

        return htmlCourse;
    }

    getHtml(): string[]{
        return this.html;
    }

}
```

partials

Codigo que se utiliza para el html carreras

```
1   <div id="pensum" class="<%= id  %>"></div>
```

public

pensum.css

Codigo para decorar.

```css
.container_user{
    text-align: center;
    margin: 2rem 1rem 2rem 1rem;
}

.table_pensum{
    width: 100%;
    max-height: 100vh;
}

.table_pensum th {
    border: 2px solid gray;
}

.table_pensum td {
    padding: 1.5rem 0 1.5rem 0;
    border: 2px solid gray;
}

.table_pensum p{
    margin: 0;
}

.table_pensum div{
    margin: 2%;
    padding: 4%;
    width: 60%;
    max-width: 60%;
    text-align: center;
    position: relative;
    background-color: aliceblue;

}

.pre_curso{
    background-color: rgb(232, 118, 19) !important;
}
```

```css
.table_pensum div{
    padding: 4%;
    width: 60%;
    max-width: 60%;
    text-align: center;
    position: relative;
    background-color: aliceblue;

}

.pre_curso{
    background-color: rgb(232, 118, 19) !important;
}

.codigo{
    position: absolute;
    top: 0px;
    left: 0px;
    width: 15%;
    height: 100%;
    background-color: blue;
}

.pre {
    position: absolute;
    top: 0px;
    right: 0px;
    width: 15%;
    height: 100%;
    background-color: aqua;
}

.pre_front {
    font-size: 12px;
}
```

pensum.js

Codigo para js.

```
document.addEventListener('DOMContentLoaded', () =>{

    const pensum = document.getElementById('pensum');

    let id = pensum.classList.item(0);

    let carrera = JSON.parse(localStorage.getItem(`pensum${id}`));

    if(carrera !== null){
        pensum.innerHTML = carrera;

        let cursos = pensum.querySelectorAll('div');

        const limpiar = () =>{
            for(const curso of cursos){
               curso.classList.remove('pre_curso');
            }
        }
        const marcarCursos= (pres) =>{
            if(pres.length !==0){
                for(const pre of pres){
                    const pre_curso = document.getElementById(pre.innerText);

                    pre_curso.classList.add('pre_curso');

                    marcarCursos(pre_curso.children[2].children);
                }

                return;
            }
        }

        const getCurso = (event) =>{
          limpiar();

          const curso = event.currentTarget;
          curso.classList.add('pre_curso');
```

```
        }
    }

    const getCurso = (event) =>{
      limpiar();

      const curso = event.currentTarget;
      curso.classList.add('pre_curso');

      console.log(curso);

      const pre = curso.children[2].children;

      console.log(pre);

      if(pre.length ===0)  alert('Este curso no tiene prerequisitos')

        marcarCursos(pre);
    }
    for(const curso of cursos){
        curso.addEventListener('click', getCurso);

        let pre = curso.children[2].children;

        if(pre.length >3){
            curso.children[2].classList.add('pre_font');
        }
    }
}
});
```

StructureCareer

Codigo para estructuras las carreras

```
import { Token, Type } from "../Analyzer/Token";
import { Career } from "../models/Career";
import { Course } from "../models/Course";
import { Semester } from "../models/Semester";

export const getCareers = (tokens: Token[]): Career[] => {
    let careers: Career[] = [];


    let flags: boolean[]= [false, false, false, false, false];
    let career: Career;
    let semester: Semester;
    let course: Course;

    tokens.forEach((token: Token, index: number) =>{

            if(token.getLexeme() == 'Carrera'){
              flags[0]=true;
            }
            if(flags[0] && token.getTypeToken()==Type.STRING){
career= new Career(token.getLexeme().slice(1, token.getLexeme().length - 1));
flags[0] = false;
            }

             if(token.getLexeme() == 'Semestre'){
              flags[1]=true;
            }
            if(flags[1] && token.getTypeToken()==Type.NUMBER){
semester= new Semester(Number(token.getLexeme()));
flags[1] = false;
            }

            if(token.getLexeme() == 'Curso'){
              flags[2]=true;
            }
            if(flags[2]){
             if(!flags[3] && !flags[4] && token.getTypeToken()== Type.NUMBER){
```

```
flags[1] = false;
            }

            if(token.getLexeme() == 'Curso'){
              flags[2]=true;
            }
            if(flags[2]){
             if(!flags[3] && !flags[4] && token.getTypeToken()== Type.NUMBER){
                course = new Course(token.getLexeme());
             }else if (!flags[4] && token.getTypeToken() == Type.NUMBER){
                course.setArea(Number(token.getLexeme()));
                flags[3] = false;
             }else if(token.getTypeToken() == Type.NUMBER){
                course.addPrerequisites(token.getLexeme());
             }

             if(token.getTypeToken() == Type.STRING){
               course.setName(token.getLexeme().slice(1, token.getLexeme().length -1));
             }

             if (token.getLexeme() == 'Area'){

                flags[3] = true;
             }
             if(token.getTypeToken()== Type.PAR_OPEN){
                flags[4] = true;
             }
             if(token.getTypeToken()== Type.PAR_CLOSE){
                flags[4] = false;
             }

             if (token.getTypeToken()== Type.COR_CLOSE){
                course.generateHtml();
                semester.addCourse(course);
```

```
            }
            if(token.getTypeToken()== Type.PAR_OPEN){
                flags[4] = true;
            }
            if(token.getTypeToken()== Type.PAR_CLOSE){
                flags[4] = false;
            }

            if (token.getTypeToken()== Type.COR_CLOSE){
                course.generateHtml();
                semester.addCourse(course);
                flags[2] = false;

                flags[1] = tokens[index +1].getLexeme() != 'curso' ? true : false;
            }
           }
           if (flags[1] && token.getTypeToken() == Type.COR_CLOSE){
            semester.generateHtml();
            career.addSemester(semester);
            flags[1]= false;
           }

           if(token.getTypeToken() == Type.KEY_CLOSE){
            career.generateHtml();
            careers.push(career);
           }



    });

    return careers;
}
```

Recomendaciones

Ahora conociendo las funcionalidades del programa, se recomienda al usuario seguir las indicaciones dadas anteriormente, con el fin de evitar un mal funcionamiento del programa que provoque el cierre inmediato del mismo.