Instituto Pío IX Casa Salesiana

Desarrollo de Aplicaciones Informáticas

Tema: Creación de proyecto con `NextJS`

Docente: Ing. Pablo Morandi
Ayudante: Matías Marchesi
Curso: 5to año
Especialidad: Informática

Redactado por: Ing. Pablo Morandi
Versión: v1.0

---

0. Instalación de Node.js

Next.js se ejecuta sobre **Node.js**, por lo que primero debemos instalarlo (en caso de no tenerlo).

Descargar e Instalar en Windows

1. Ir a la página oficial:
   👉 https://nodejs.org/
2. Descargar la **versión LTS (Long Term Support)** (más estable y recomendada).
3. Ejecutar el instalador y dejar las opciones por defecto.
4. Una vez instalado, abrir la **terminal de Windows (CMD o PowerShell)** y comprobar:

```
node -v
npm -v
```

Deberíamos ver algo como:

```
v20.11.1   ← versión de Node.js
10.2.4     ← versión de npm
```

> **Nota:** `npm` (Node Package Manager) y `npx` (Node Package Execute) se instalan automáticamente con Node.js.

---

1. Creación del Proyecto Next.js

Instalación desde Cero

Documentación oficial de Next.js:
👉 https://nextjs.org/docs

Para crear nuestro proyecto de Next.js utilizamos el comando oficial de creación:

```
npx create-next-app@latest mi-proyecto-nextjs
```

Durante la instalación, Next.js nos preguntará:

```
✔ Would you like to use TypeScript? › No
✔ Would you like to use ESLint? › Yes
✔ Would you like to use Tailwind CSS? › No
✔ Would you like to use `src/` directory? › Yes
✔ Would you like to use App Router? › Yes
✔ Would you like to customize the default import alias? › Yes
```

Explicación de las opciones:

- **TypeScript**: Por ahora usamos JavaScript puro
- **ESLint**: Herramienta que nos ayuda a escribir código más limpio
- **Tailwind CSS**: Framework de CSS (lo veremos más adelante)
- **src/ directory**: Organiza mejor nuestros archivos
- **App Router**: El nuevo sistema de rutas de Next.js (recomendado)

Comandos Post-Instalación

```
# Navegar al proyecto
cd mi-proyecto-nextjs

# Instalar dependencias
npm install
```

```
# Ejecutar el servidor de desarrollo
npm run dev
```

---

2. Entendiendo el package.json

El `package.json` es el archivo más importante de nuestro proyecto. Es como el **DNI** de la aplicación.

Estructura del package.json

```
{
  "name": "mi-proyecto-nextjs",
  "version": "0.1.0",
  "private": true,
  "scripts": {
    "dev": "next dev",
    "build": "next build",
    "start": "next start",
    "lint": "next lint"
  },
  "dependencies": {
    "react": "^18",
    "react-dom": "^18",
    "next": "14.0.0"
  },
  "devDependencies": {
    "eslint": "^8",
    "eslint-config-next": "14.0.0"
  }
}
```

node_modules

Es una carpeta que se genera al instalar dependencias con `npm install`.
Contiene **todas las librerías necesarias** para que nuestro proyecto funcione.

Ejemplo de contenido:

```
node_modules/
├── next/
├── react/
├── react-dom/
└── … muchas más
```

> ⚠ Importante: nunca se sube `node_modules` a GitHub porque es muy pesada.
> Por eso en el archivo `.gitignore` aparece la línea:

```
node_modules/
```

Secciones Importantes

Información del proyecto:

- **name**: Nombre de nuestro proyecto
- **version**: Versión actual (seguimos versionado semántico)
- **private**: true significa que no se puede publicar en npm

Scripts disponibles:

- **dev**: Inicia el servidor de desarrollo
- **build**: Construye la aplicación para producción
- **start**: Ejecuta la aplicación en modo producción
- **lint**: Revisa el código en busca de errores

Dependencias:

- **dependencies**: Librerías necesarias para que funcione la aplicación
- **devDependencies**: Herramientas solo para desarrollo

¿Por qué es importante?

```
# Al ejecutar npm install, lee este archivo
npm install

# Al ejecutar npm run dev, ejecuta el script "dev"
npm run dev

# Al agregar librerías, se actualiza automáticamente
npm install axios
```

---

3. Estructura de Carpetas y Archivos

Después de crear el proyecto, tenemos esta estructura:

```
mi-proyecto-nextjs/
├── .next/                  # Archivos generados automáticamente
├── node_modules/           # Librerías instaladas
├── public/                 # Archivos estáticos (imágenes, etc.)
├── src/                    # Nuestro código fuente
│   └── app/
│       ├── favicon.ico
│       ├── globals.css
│       ├── layout.js       # Layout principal
│       └── page.js         # Página de inicio
├── .eslintrc.json         # Configuración de ESLint
├── .gitignore             # Archivos que Git debe ignorar
├── next.config.js         # Configuración de Next.js
├── package.json           # Información del proyecto
└── README.md              # Documentación del proyecto
```

Archivos Clave

src/app/page.js - Página principal (ruta `/`):

```
export default function Home() {
  return (
    <main>
      <h1>Página de Inicio</h1>
    </main>
  );
}
```

src/app/layout.js - Layout que envuelve todas las páginas:

```
export default function RootLayout({ children }) {
  return (
    <html lang="es">
      <body>
        {children}
      </body>
    </html>
  );
}
```

Sistema de Rutas

Next.js usa **routing basado en carpetas**:

```
src/app/
├── page.js          → /
├── login/
│   └── page.js      → /login
└── home/
    └── page.js      → /home
```

---

4. Creando Páginas Básicas

Vamos a crear dos páginas fundamentales para nuestra aplicación.

Página Home

Creamos `src/app/home/page.js`:

```
export default function HomePage() {
    return (
        <div>
            <h1>Bienvenido al Sistema</h1>
            <p>Esta es la página principal de nuestra aplicación</p>
        </div>
    );
}
```

Página Login

Creamos `src/app/login/page.js`:

```
export default function LoginPage() {
    return (
        <div>
            <h1>Iniciar Sesión</h1>
            <p>Ingresa tus credenciales para acceder al sistema</p>
            ...
            /* Componente de Formulario, por ejemplo */
            ...
        </div>
    );
}
```

Otras Convenciones de Nomenclatura

Archivos de componentes:

```
Title.js          ☑ PascalCase
UserCard.js       ☑ PascalCase
NavigationBar.js  ☑ PascalCase

title.js          ❌ Evitar
user-card.js      ❌ Evitar
```

Variables y funciones:

```
// ☑ camelCase para variables y funciones
const userName = 'Juan';
const getUserData = () => { };

// ☑ PascalCase solo para componentes
const UserProfile = () => { }; // Función "flecha"
function UserProfile () {};  // Función tradicional
```

Constantes:

```
// ☑ UPPER_CASE para constantes
const API_URL = 'https://api.example.com';
const MAX_USERS = 100;
```

Estructura Final del Proyecto

Finalmente, la estructura del proyecto deberia tener un aspecto como este:

```
mi-proyecto-nextjs/
├── src/
│   └── app/
│       ├── components/
│       │   └── Title.js         # Componente reutilizable
│       ├── home/
│       │   └── page.js          # Página Home con componente
│       ├── login/
│       │   └── page.js          # Página Login con componente
│       ├── layout.js            # Layout principal
│       └── page.js              # Página de inicio (/)
├── package.json                 # Configuración del proyecto
├── next.config.js              # Configuración de Next.js
└── README.md                   # Documentación
```

Rutas Disponibles

- `/` - Página de inicio (landing)
- `/home` - Página principal del sistema
- `/login` - Página de autenticación

Comandos Útiles para el Desarrollo

```
# Iniciar servidor de desarrollo
npm run dev

# Construir para producción
npm run build

# Iniciar en modo producción
npm start

# Instalar nueva dependencia
npm install nombre-paquete

# Ver información del proyecto
npm info
```
