LearnWeb

Miguel Arroyo Calvo

Arquitectura MVC

Con Symfony

- Controller (Intermediario entre Vista y Modelo)
- Entity (Modelo, tablas de una BBDD)
- Repository (Acceso a datos)

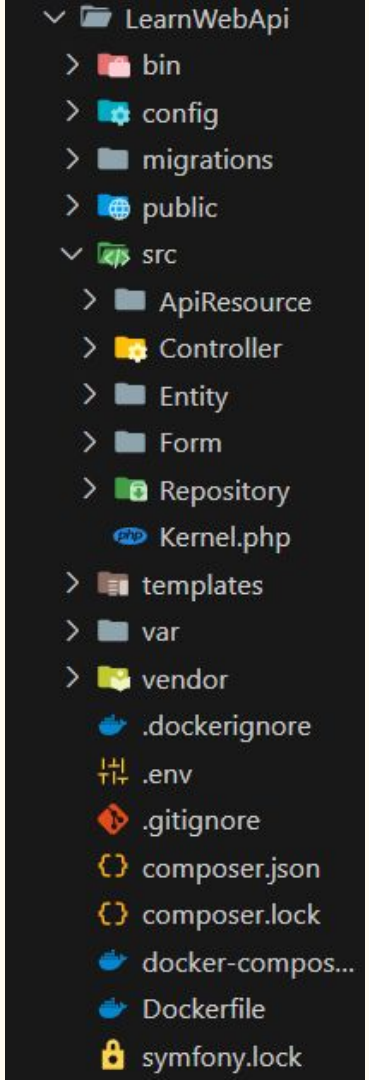
Estructura

src/

|-- Controller/

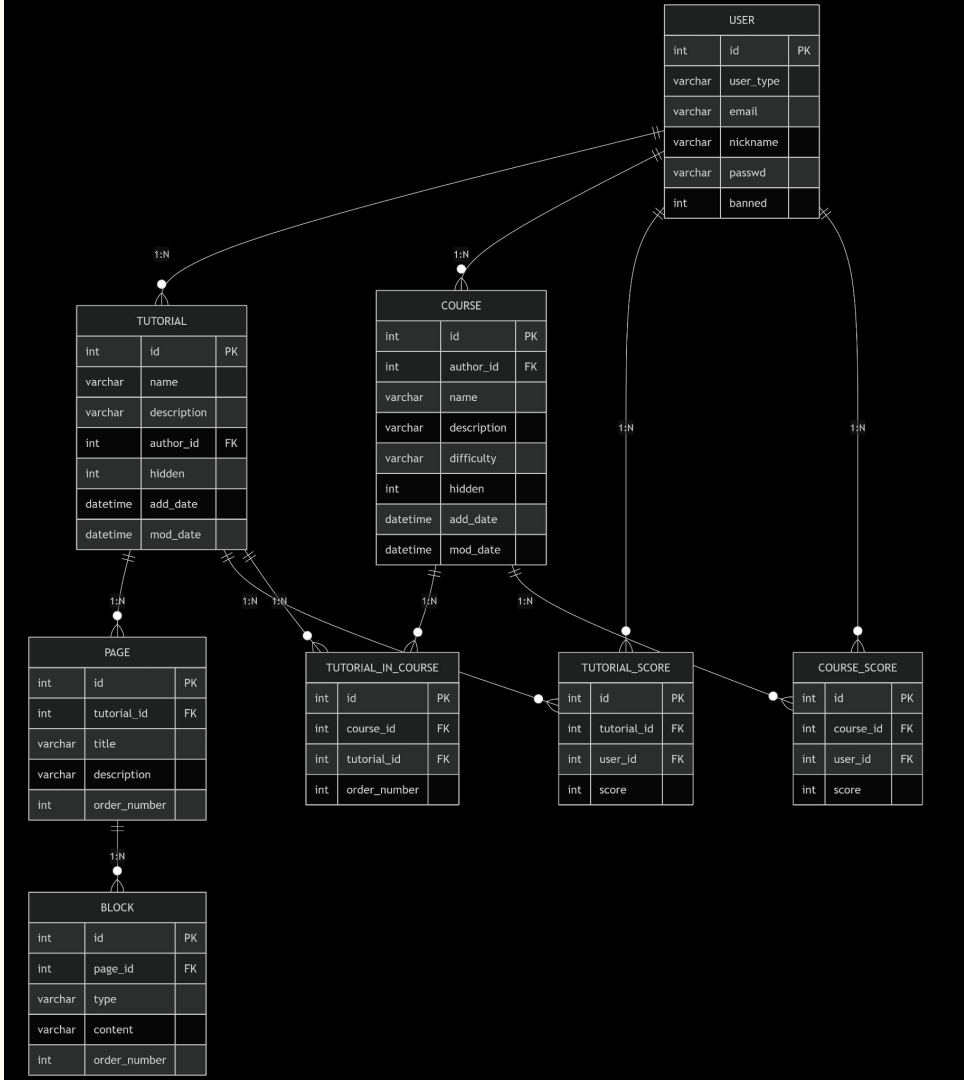
|-- Entity/

|-- Repository/



Base de datos LearnWeb

- Relacional
- Cardinalidades 1:N (OneToMany)
  - Según la configuración del proyecto, funciona en el puerto 3306
  - Un ejemplo de relación entre tablas es: los usuarios pueden crear uno o varios tutoriales, pero cada tutorial solo tiene un autor



Tipos de usuario

Administradores

ROLE_ADMIN

- Gestionan tutoriales
- Gestionan cursos
- Banean / desbanean usuarios
- Registran a otros administradores

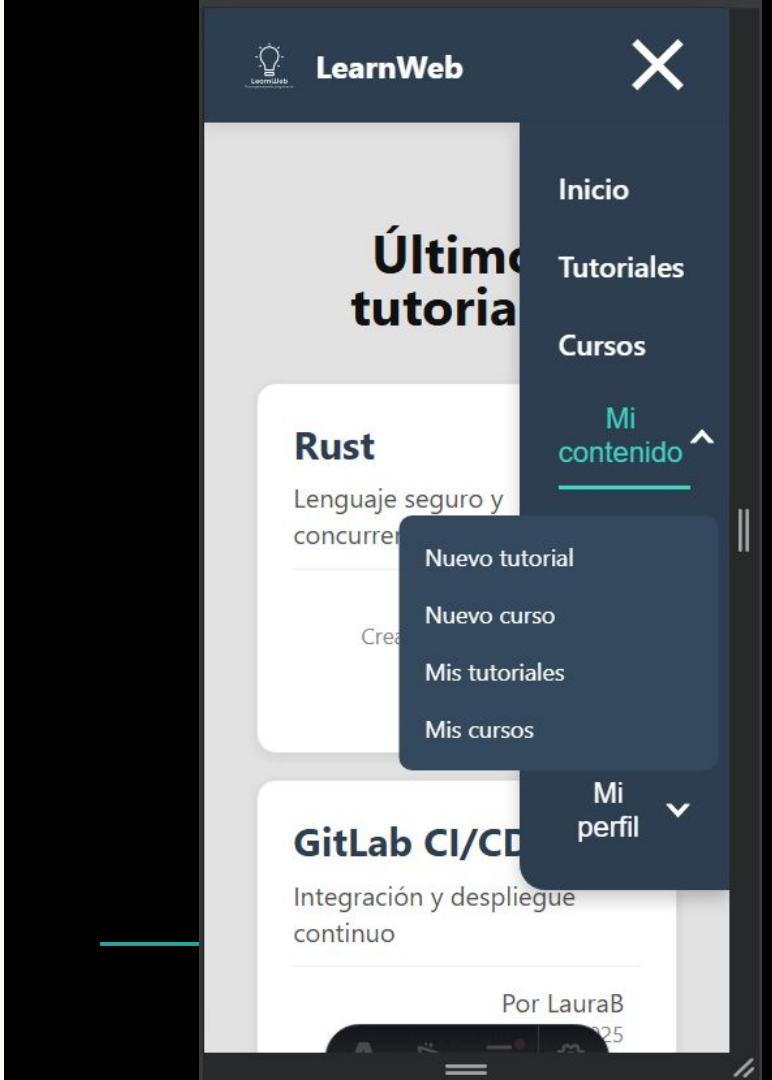
Estudiantes o usuarios corrientes

ROLE LEARNER

- Crean tutoriales y cursos
- Editan y ocultan/muestran los suyos propios
- Puntúan tutoriales y cursos

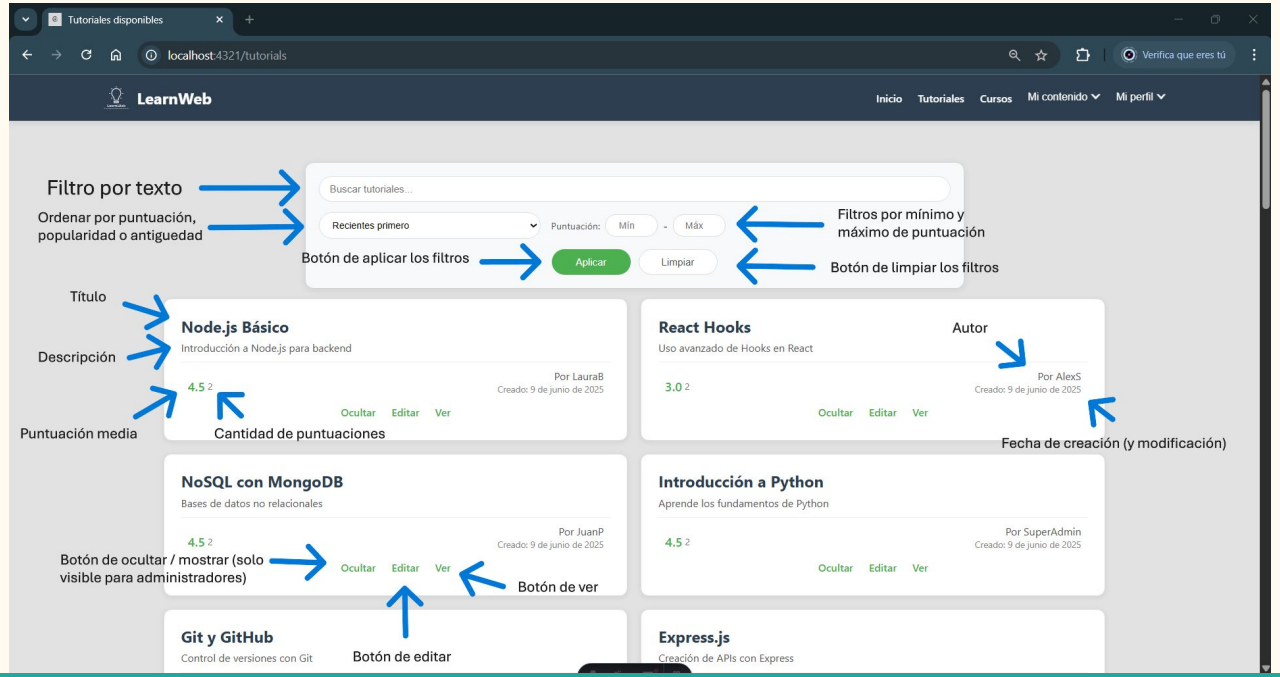
Web responsive

- media queries con max-width
- width: auto o %
- Menú hamburguesa para pantallas más pequeñas
- display: flex, flex-direction: row o column, flex-wrap: wrap, justify-content: center
- padding, margin y gap



Diseño: muestra de entidades

Para mostrar entidades tales como tutoriales utilizamos tarjetas o cajas, que son contenedores dentro de una lista.



Validación en formularios

- Patrones para email
- Tipo de datos
- Comprobaciones a la BBDD
- Longitud mínima o máxima de caracteres
- Confirmaciones con sweetalert2

```
const user = await getUserByLogin(nickname, password);
```

```
const emailPattern = /^[a-zA-Z0-9._%+-]+@(gmail\.com|hotmail\.com)$/;
```

```
if(!emailPattern.test(email)){
    errors = true;
    Swal.fire({
        title: 'El email introducido no es válido',
    });
}
```

```
Swal.fire({
    title: '¿Seguro?',
    text: '¿Seguro de querer registrarte con estos datos?',
    showDenyButton: true,
    confirmButtonText: 'Si',
    denyButtonText: 'No',
}).then(async (result) => {
```

```
if (typeof nickname !== "string" || nickname.length < 1) {
  Swal.fire("Please enter your nickname. ");
  errors = true;
}
```

Comunicación asíncrona con el servidor

- async await
- comunicación con el backend
- respuesta en json
- try catch

```
}).then(async (result) => {
    // Si le damos a que si
    if(result.isConfirmed) {
        // Registramos al administrador
        const user_created = await register(nickname, email, passwd);
```

```
const get_jwt = async (endpoint, token) => {
  const response = await fetch(`${apiUrl}${endpoint}`, {
    method: "GET",
    headers: {
      Authorization: `Bearer ${token}`,
      accept: "application/json",
      "Content-Type": "application/json",
    },
  });
  if (!response.ok) {
    throw new Error(
      `Error en la petición GET al endpoint: ${endpoint} (${response.status} ${response.statusText})`
    );
  }
  const data = await response.json();
  return data;
};
```

```
if (notForBannedUsers.some((route) => request.url.includes(route))) {
  try {
    const isBanned = await checkUserBan(decoded["id"], token);
    if (isBanned) {
      return redirect("/banned", 302);
    }
  } catch (banError) {
    console.error("Ban check failed:", banError);
    return next();
  }
}
```

Symfony y la BBDD

- Symfony se comunica con la BBDD
- Realiza consultas gracias al Repository
- Obtiene datos de manera rápida gracias a la API del backend, que accede a los datos o los modifica gracias a los getters o setters de las entidades

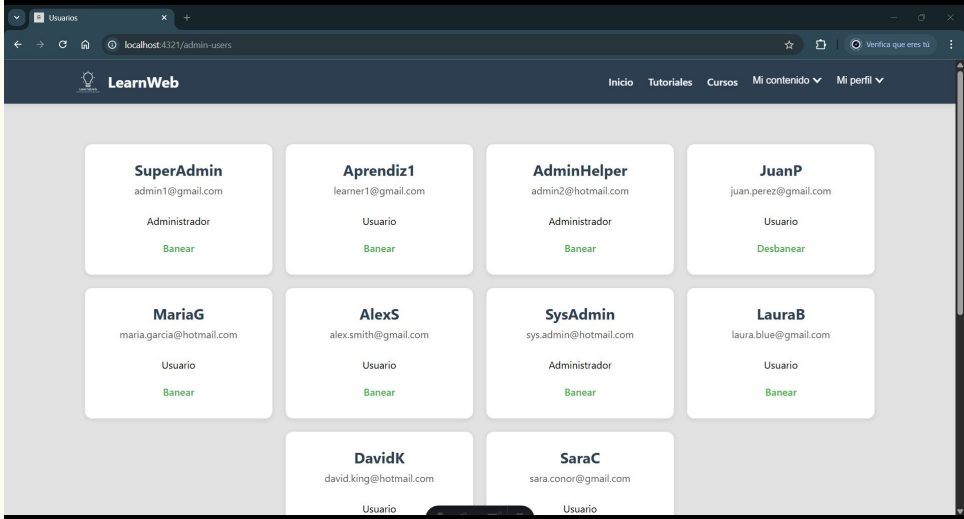
```
DATABASE_URL="mysql://root:root@localhost:3306/LearnWeb"
```

```
1 reference | 0 overrides
public function getEmail(): ?string
{
    return $this->email;
}

2 references | 0 overrides
public function setEmail(string $email): static
{
    $this->email = $email;

    return $this;
}
```

```
2 references | 0 overrides
public function validateUser($nickname): ?User
{
    return $this->createQueryBuilder(alias: 'u')
        ->where(predicates: 'u.nickname = :nickname')
        ->setParameter(key: 'nickname', value: $nickname)
        ->getQuery()
        ->getOneOrNullResult()
        ;
}
```

Gestión de usuarios

Los administradores (ROLE_ADMIN) pueden banear o desbanear usuarios, por mal uso de la aplicación web.



Contraseñas encriptadas en en backend

Las contraseñas de los usuarios que vamos a añadir son encriptadas con encriptación bcrypt para mayor seguridad

Cuando necesitamos comprobar la contraseña (p.e: login) encriptamos la introducida y la comparamos con la que tenemos en la BBDD

```php
#[Route(path: '/api/users/register/{nickname}/{email}/{passwd}', name: 'app_user_register', methods: ['GET', 'POST'])]
6 references | 0 overrides
public function register(SerializerInterface $serializerInterface, Request $request, UserPasswordHasherInterface $passwordHasher,
{
    $user = new User();
    $user->setUserType(userType: 'ROLE_LEARNER');
    $user->setEmail(email: $email);
    $user->setPasswd(passwd: $passwordHasher->hashPassword(user: $user, plainPassword: $passwd));
    $user->setNickname(nickname: $nickname);
    $user->setBanned(banned: False);

    $entityManager->persist(object: $user);
    $entityManager->flush();

    $response = $serializerInterface->serialize(data: $user, format: 'json');
    return new JsonResponse(data: $response, status: 200, headers: [
        'Content-Type' => 'application/json'
    ], json: true);
}
```

```php
$factory = new PasswordHasherFactory(passwordHashers: [
    'common' => ['algorithm' => 'bcrypt']
]);
$hasher = $factory->getPasswordHasher(user: 'common');
if($hasher->verify(hashedPassword: $user->getPasswd(), plainPassword: $passwd)){
    $payload = [
        'id' => $user->getId(),
        'nickname' => $user->getNickname(),
        'role' => $user->getUserType()
    ];
    //$user->setUserIdentifier($user->getNickname());
    $token = $jwtManager->createFromPayload(user: $user, payload: $payload);
    $userInfo = ['id' => $user->getId(), 'nickname' => $user->getNickname(), 'role' => $user->getUserType(), 'token' => $token];
    $response = $serializerInterface->serialize(data: $userInfo, format: 'json');
    #$response = $serializerInterface->serialize($token, 'json');
    return new JsonResponse(data: $response, status: 200, headers: [
        'Content-Type' => 'application/json',
        //'token' => $token
    ], json: true);
```

Token, sesiones y cookies

Al iniciar sesión creamos y guardamos un token en las cookies bajo la identificación jwt_token.

Obtenemos el token de las cookies cuando necesitamos usarlo en alguna petición al backend.

```
$token = $jwtManager->createFromPayload(user: $user, payload: $payload);
```

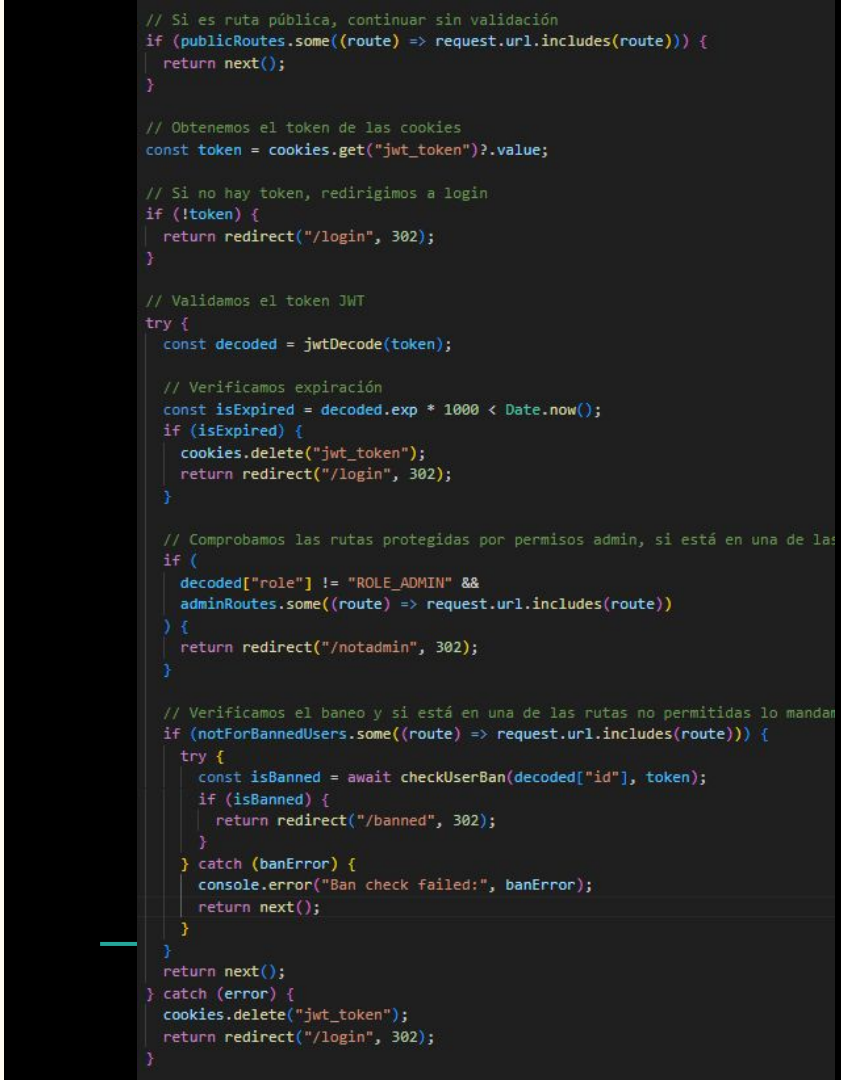
```
const user = await getUserByLogin(nickname, password);
// Si existe
if (user != null) {
  // Obtiene su token que nos ha proporcionado la API
  const token = (user)['token'];
  console.log(user);
  // Fecha de creación del token
  const d = new Date();
  d.setTime(d.getTime() + (7200*1000));
  // Guardamos el token en las cookies bajo el nombre de jwt_token, con vistas a expirar en 1 hora
  document.cookie = `jwt_token=${token}; max-age=3600; path=/;`;
```

```
export function getToken() {
  const cookies = document.cookie;

  const jwtCookie = cookies
    .split("; ")
    .find((cookie) => cookie.startsWith("jwt_token="));

  if (jwtCookie) {
    return jwtCookie.split("=")[1];
  }

  return null;
}
```

```
const get_jwt = async (endpoint, token) => {
  const response = await fetch(`${apiUrl}${endpoint}`, {
    method: "GET",
    headers: {
      Authorization: `Bearer ${token}`,
      accept: "application/json",
      "Content-Type": "application/json",
    },
```

Restricciones de acceso en la web

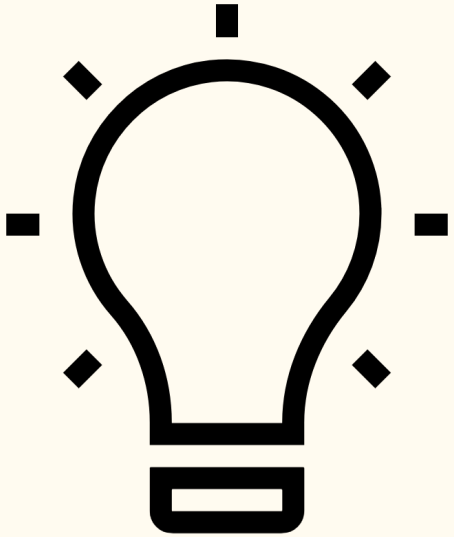
Tenemos un middleware que con el token de la sesión comprueba los roles del usuario o su estado de baneo y bloquea su acceso a ciertas partes.



```
// Si es ruta pública, continuar sin validación
if (publicRoutes.some((route) => request.url.includes(route))) {
  return next();
}

// Obtenemos el token de las cookies
const token = cookies.get("jwt_token")?.value;

// Si no hay token, redirigimos a login
if (!token) {
  return redirect("/login", 302);
}

// Validamos el token JWT
try {
  const decoded = jwtDecode(token);

  // Verificamos expiración
  const isExpired = decoded.exp * 1000 < Date.now();
  if (isExpired) {
    cookies.delete("jwt_token");
    return redirect("/login", 302);
  }

  // Comprobamos las rutas protegidas por permisos admin, si está en una de las
  if (
    decoded["role"] != "ROLE_ADMIN" &&
    adminRoutes.some((route) => request.url.includes(route))
  ) {
    return redirect("/notadmin", 302);
  }

  // Verificamos el baneo y si está en una de las rutas no permitidas lo mandam
  if (notForBannedUsers.some((route) => request.url.includes(route))) {
    try {
      const isBanned = await checkUserBan(decoded["id"], token);
      if (isBanned) {
        return redirect("/banned", 302);
      }
    } catch (banError) {
      console.error("Ban check failed:", banError);
      return next();
    }
  }
  return next();
} catch (error) {
  cookies.delete("jwt_token");
  return redirect("/login", 302);
}
```

Posibles mejoras a futuro

- Patrón para la contraseña
- Confirmación de cuenta con correo electrónico
- Nuevo tipo de usuario: profesor. Crear tutoriales y cursos sería exclusivo de profesores. Pueden promocionar de estudiante a profesor rellenando un formulario y enviando CV, los administradores revisan y confirman o niegan la solicitud.
- Los usuarios se pueden apuntar a cursos y completarlos al aprobar exámenes tipo test que crean los profesores.



LearnWeb

Cursos para aprender programación

Gracias por ver