Guía de Implementación: Login en Examen

Archivos a Copiar/Pegar

FRONTEND (Angular)

Archivos a crear/copiar:

1. `src/app/pages/login/login.ts`
   - Componente principal con la lógica del login
   - Variables: `username`, `password`, `error`
   - Método: `login()` con validaciones
2. `src/app/pages/login/login.html`
   - Formulario con inputs para usuario y contraseña
   - Botón de envío
   - Visualización de errores
3. `src/app/pages/login/login.css`
   - Estilos del formulario (opcional, pero recomendado)
4. `src/app/services/login.service.ts`
   - Servicio que maneja las peticiones HTTP/FETCH
   - Métodos: `login()`, `verifyToken()`
   - Gestión de tokens en localStorage
5. `src/app/interfaces/user.ts`
   - Interfaz TypeScript para el usuario
   - Propiedades: `name`, `password`, `admin`, `token`

Archivos a modificar:

- `src/app/routes.ts`
  - Importar el componente `Login`
  - Agregar ruta `/login`

BACKEND/API (Node.js + Express)

Archivos a crear/copiar:

1. `routes/users.js`
   - Rutas para:
     - `POST /login` - Login de usuario

        - `GET /verify/:token` - Verificación de token
        - `POST /logout/:userId` - Logout de usuario
        - Otras rutas CRUD de usuarios (GET, POST, PUT, DELETE)

2. **`services/user-service.js`**
    - Métodos principales:
        - `comprobarCredenciales(usuario)` - Valida credenciales y genera token
        - `generateToken()` - Crea un token aleatorio
        - `getByToken(token)` - Busca usuario por token
        - `logout(userId)` - Limpia el token del usuario
        - Otros métodos CRUD: `get()`, `getById()`, `post()`, `put()`, `delete()`

3. **`models/UserModel.js`**
    - Schema/modelo de MongoDB para usuarios
    - Campos: `name`, `password`, `token`, `admin`

Archivos a verificar:

- **`app.js`** - Asegurarse de que incluye la ruta de usuarios
- **`package.json`** - Verificar dependencias necesarias (express, mongodb, etc.)

Configuración de la BD (MongoDB)

Estructura de la colección `users`

```
{
  _id: ObjectId,
  name: String,         // nombre de usuario
  password: String,     // contraseña (en texto plano en este caso)
  admin: Boolean,       // permisos de administrador
  token: String         // token generado al hacer login
}
```

Datos de conexión

```
//MarcosDB:mpwd@localhost:6969
mongodb: Database: carstore;
Collection: users;
```

Código Principal a Copiar

Frontend - Componente Login (login.ts)

```
import { Component } from "@angular/core";
import { Router } from "@angular/router";
import { FormsModule } from "@angular/forms";
import { LoginService } from "../../services/login.service";

@Component({
  selector: "app-login",
  imports: [FormsModule],
  templateUrl: "./login.html",
  styleUrl: "./login.css",
})
export class Login {
  username: string = "";
  password: string = "";
  error: string = "";

  constructor(
    private authService: LoginService,
    private router: Router,
  ) {}

  async login() {
    try {
      this.error = "";
      if (!this.username || !this.password) {
        this.error = "Por favor completa todos los campos";
        return;
      }

      await this.authService.login(this.username, this.password);
      this.router.navigate(["/"]);
    } catch (err) {
      this.error = "Usuario o contraseña incorrectos";
    }
  }
}
```

Frontend - Servicio (login.service.ts)

```
import { Injectable } from "@angular/core";

@Injectable({
  providedIn: "root",
})
export class LoginService {
  private url = "http://localhost:3000/users";

  async login(name: string, password: string) {
    try {
      const response = await fetch(this.url + "/login", {
```

```
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({
          name: name,
          password: password,
        }),
      });

      if (response.status !== 200 && response.status !== 201) {
        throw new Error("Invalid credentials");
      }

      const user = await response.json();

      // Guardar solo el token en localStorage
      if (user && user.token) {
        localStorage.setItem("token", user.token);
      }

      return user;
    } catch (error) {
      console.error("Login error:", error);
      throw error;
    }
  }

  async verifyToken(token: string) {
    try {
      const response = await fetch(this.url + `/verify/${token}`);

      if (response.status !== 200 && response.status !== 201) {
        return null;
      }

      const data = await response.json();
      return data.valid ? data.user : null;
    } catch (error) {
      console.error("Verification error:", error);
      return null;
    }
  }

  getToken(): string | null {
    return localStorage.getItem("token");
  }

  logout() {
    localStorage.removeItem("token");
  }
}
```

Frontend - Template (login.html)

```
<div class="login-container">
  <form (ngSubmit)="login()">
    <h1>Login</h1>
    <input
      type="text"
      [(ngModel)]="username"
      name="username"
      placeholder="Usuario"
      required
    />
    <input
      type="password"
      [(ngModel)]="password"
      name="password"
      placeholder="Contraseña"
      required
    />
    <button type="submit">Entrar</button>
    <div *ngIf="error" class="error">{{ error }}</div>
  </form>
</div>
```

Frontend - Interfaz (user.ts)

```
export interface User {
  _id?: string;
  name: string;
  password: string;
  admin: boolean;
  token?: string;
}
```

Frontend - Rutas (routes.ts)

```
import { Routes } from "@angular/router";
import { Home } from "./pages/home/home";
import { Login } from "./pages/login/login";
// ... otros imports

const routeConfig: Routes = [
  {
    path: "login",
    component: Login,
    title: "Login",
  },
  {
```

```
    path: "",
    component: Home,
    title: "Home page",
  },
  // ... otras rutas
];

export default routeConfig;
```

Backend - Rutas (routes/users.js)

```
var express = require("express");
var router = express.Router();
let UserService = require("../services/user-service");

router.get("/", async function (req, res, next) {
  const arrayUsers = await UserService.get();
  res.status(200).json(arrayUsers);
});

router.get("/:id", async function (req, res, next) {
  const user = await UserService.getById(req.params.id);
  res.status(200).json(user);
});

router.get("/verify/:token", async function (req, res, next) {
  const user = await UserService.getByToken(req.params.token);
  if (user) {
    res.status(200).json({ valid: true, user: user });
  } else {
    res.status(401).json({ valid: false, user: null });
  }
});

router.post("/", async function (req, res, next) {
  await UserService.post(req.body);
  res.status(201).json(true);
});

router.post("/login", async function (req, res, next) {
  const user = await UserService.comprobarCredenciales(req.body);
  if (user) {
    res.status(200).json(user);
  } else {
    res.status(401).json({ error: "Invalid credentials" });
  }
});

router.post("/logout/:userId", async function (req, res, next) {
  const result = await UserService.logout(req.params.userId);
  if (result.modifiedCount === 1) {
```

```
    res.status(200).json({ success: true });
  } else {
    res.status(404).json({ error: "User not found" });
  }
});

router.put("/:id", async function (req, res, next) {
  const user = await UserService.put(
    req.params.id,
    req.body.name,
    req.body.password,
  );
  res.status(200).json(user);
});

router.delete("/:id", async function (req, res, next) {
  const result = await UserService.delete(req.params.id);
  res.status(200).json(result);
});

module.exports = router;
```

Backend - Servicio (services/user-service.js)

```
const User = require("../models/UserModel");
const { MongoClient, ObjectId } = require("mongodb");
const uri = `mongodb://MarcosDB:mpwd@localhost:6969`;
const client = new MongoClient(uri);

class UserService {
  static async get() {
    try {
      await client.connect();
      const database = client.db("carstore");
      const user = database.collection("users");
      const result = await user.find().toArray();
      return result;
    } catch (error) {
      console.error("Error getting users:", error);
    } finally {
      await client.close();
    }
  }

  static async getById(id) {
    try {
      await client.connect();
      const database = client.db("carstore");
      const user = database.collection("users");
      const filter = { _id: new ObjectId(id) };
      const result = await user.findOne(filter);
```

```
      return result;
    } catch (error) {
      console.error("Error getting user by ID:", error);
    } finally {
      await client.close();
    }
  }

  static async getByToken(token) {
    try {
      await client.connect();
      const database = client.db("carstore");
      const user = database.collection("users");
      const filter = { token: token };
      const result = await user.findOne(filter);
      return result;
    } catch (error) {
      console.error("Error getting user by token:", error);
    } finally {
      await client.close();
    }
  }

  static async comprobarCredenciales(usuario) {
    try {
      await client.connect();
      const database = client.db("carstore");
      const user = database.collection("users");
      const filter = { name: usuario.name, password: usuario.password };
      const result = await user.findOne(filter);

      if (result) {
        const newToken = this.generateToken();
        await user.updateOne(
          { _id: result._id },
          { $set: { token: newToken } },
        );
        result.token = newToken;
      }

      return result;
    } catch (error) {
      console.error("Error checking credentials:", error);
    } finally {
      await client.close();
    }
  }

  static async logout(userId) {
    try {
      await client.connect();
      const database = client.db("carstore");
      const user = database.collection("users");
      const filter = { _id: new ObjectId(userId) };
```

```
      const result = await user.updateOne(filter, { $unset: { token: "" } });
      return result;
    } catch (error) {
      console.error("Error logging out:", error);
    } finally {
      await client.close();
    }
  }

  static async post(userData) {
    try {
      await client.connect();
      const database = client.db("carstore");
      const user = database.collection("users");
      await user.insertOne(userData);
      return true;
    } catch (error) {
      console.error("Error creating user:", error);
    } finally {
      await client.close();
    }
  }

  static async put(id, name, password) {
    try {
      await client.connect();
      const database = client.db("carstore");
      const user = database.collection("users");
      const filter = { _id: new ObjectId(id) };
      const update = { $set: { name: name, password: password } };
      const result = await user.findOneAndUpdate(filter, update, {
        returnDocument: "after",
      });
      return result.value;
    } catch (error) {
      console.error("Error updating user:", error);
    } finally {
      await client.close();
    }
  }

  static async delete(id) {
    try {
      await client.connect();
      const database = client.db("carstore");
      const user = database.collection("users");
      const filter = { _id: new ObjectId(id) };
      const result = await user.deleteOne(filter);
      return result;
    } catch (error) {
      console.error("Error deleting user:", error);
    } finally {
      await client.close();
    }
```

```
  }

  static generateToken() {
    return Math.random().toString(36).substr(2) + Date.now().toString(36);
  }
}

module.exports = UserService;
```

Checklist de Implementación

Frontend

- [ ] Crear carpeta `/src/app/pages/login/`
- [ ] Copiar `login.ts` (componente)
- [ ] Copiar `login.html` (template)
- [ ] Copiar `login.css` (estilos, opcional)
- [ ] Crear `/src/app/services/login.service.ts`
- [ ] Crear `/src/app/interfaces/user.ts`
- [ ] Actualizar `/src/app/routes.ts` con ruta `/login`
- [ ] Verificar importaciones en `app.ts` (si aplica)

Backend

- [ ] Crear carpeta `/routes/` (si no existe)
- [ ] Copiar `routes/users.js`
- [ ] Crear carpeta `/services/` (si no existe)
- [ ] Copiar `services/user-service.js`
- [ ] Crear carpeta `/models/` (si no existe)
- [ ] Copiar `models/UserModel.js`
- [ ] Actualizar `app.js` para incluir la ruta de usuarios
- [ ] Verificar `package.json` con dependencias necesarias

Base de Datos

- [ ] Verificar MongoDB corriendo en `localhost:6969`
- [ ] Crear base de datos `carstore`
- [ ] Crear colección `users`
- [ ] Insertar datos de prueba (usuarios de ejemplo)

Pruebas

- [ ] Probar login con credenciales correctas
- [ ] Probar login con credenciales incorrectas
- [ ] Verificar token en localStorage
- [ ] Probar verificación de token
- [ ] Probar logout

Variables Importantes

Variable	Valor	Ubicación
API URL	`http://localhost:3000`	login.service.ts
API Usuarios	`/users`	login.service.ts
MongoDB URI	`mongodb://MarcosDB:mpwd@localhost:6969`	user-service.js
Base de Datos	`carstore`	user-service.js
Puerto API	`3000`	`bin/www`

Notas Importantes

- **Token**: Se genera automáticamente en el backend y se guarda en `localStorage` en el frontend
- **Validaciones**: El frontend valida que los campos no estén vacíos
- **Credenciales**: Las contraseñas se envían en texto plano (en un examen está bien, en producción usar hash)
- **CORS**: Si tienes problemas de CORS, asegúrate de permitir `http://localhost:4200` en el backend
- **Puerto Angular**: Por defecto es `4200`, asegúrate que sea accesible