Documentación TelAhorro.

**1. Introducción**

Este proyecto es una aplicación web desarrollada en **Angular** que tiene como objetivo la gestión financiera personal. Permite a los usuarios gestionar sus transacciones (ingresos y gastos), configurar metas de ahorro y consultar su historial financiero. La aplicación está dirigida a usuarios con diferentes roles, como administradores, profesores y alumnos (si se aplica en un contexto de aprendizaje). La gestión de las finanzas se realiza de manera sencilla mediante un menú lateral que incluye opciones como "Transacciones", "Ahorros" y "Configuración".

**1.1 Características principales del proyecto:**

* **Transacciones**: Los usuarios pueden registrar ingresos y gastos, visualizando un resumen con el dinero disponible, el historial de transacciones y las categorías de ingresos/gastos.
* **Ahorros**: Los usuarios pueden crear metas de ahorro, añadir dinero a las metas y visualizar el progreso mediante barras de progreso.
* **Configuración de usuario**: Los usuarios pueden gestionar su información personal, incluyendo el nombre de usuario y la contraseña.
* **Autenticación de usuarios**: La aplicación tiene un sistema de inicio de sesión y registro de usuarios. Se conecta a una API desarrollada en **Node.js** para gestionar los datos de los usuarios, control de sesiones y persistencia de información en **MongoDB**.

**1.2 Funciones y rendimiento esperado**

La aplicación debe ser capaz de gestionar las siguientes funciones:

1. **Gestión de transacciones**:
   * Los usuarios pueden agregar ingresos y gastos, con la validación de fechas hasta el día actual y la categorización de las transacciones.
   * Los datos de las transacciones (fecha, concepto, categoría, importe) se guardan y se muestran en una tabla.
   * El "dinero disponible" se actualiza automáticamente al agregar nuevas transacciones.
2. **Gestión de metas de ahorro**:
   * Los usuarios pueden definir metas de ahorro (nombre de la meta, cantidad a ahorrar y fecha límite).
   * El progreso de las metas se visualiza mediante una barra de progreso que se actualiza cuando el usuario añade dinero a la meta.
3. **Gestión de configuración de usuario**:
   * Los usuarios pueden cambiar su nombre de usuario y contraseña mediante un formulario de configuración.
4. **Autenticación**:
   * Los usuarios pueden registrarse e iniciar sesión en la aplicación.
   * El sistema de autenticación valida los datos mediante una API backend, que emite un token JWT para mantener la sesión activa.
5. **Interfaz de usuario**:
   * La aplicación tiene un diseño moderno y fácil de usar con un menú lateral desplegable y componentes bien organizados.

**1.3 Objetivos**

Los objetivos de este proyecto son:

1. Desarrollar una plataforma de gestión financiera personal utilizando Angular como framework principal para el frontend.
2. Gestionar las transacciones y ahorros de los usuarios con una estructura clara que permita ver los ingresos, gastos y metas de ahorro.
3. Implementar un sistema de autenticación utilizando JSON Web Tokens (JWT) para asegurar las sesiones de los usuarios.
4. Proporcionar una interfaz accesible y moderna con la posibilidad de cambiar información personal del usuario (como el nombre de usuario y la contraseña).
5. Persistencia de datos mediante el uso de una API backend desarrollada en Node.js, con datos almacenados en MongoDB.

**1.4. Modelado de la solución**

**1.4.1 Recursos humanos**

La plataforma estará compuesta por usuarios con roles diferenciados:

* **Administrativo**: Será responsable de la gestión de los datos de los usuarios, tales como la creación y eliminación de cuentas.
* **Profesor/Usuario**: Los usuarios (profesores o estudiantes) podrán acceder a las funcionalidades de gestión de sus transacciones, metas de ahorro y configuración personal.
* **Desarrollador de la aplicación**: El desarrollador se encargará del desarrollo y mantenimiento de la plataforma, incluyendo la implementación de la interfaz, conexión con la API backend y pruebas.
  + 1. **Recursos de hardware**

La aplicación requiere los siguientes recursos hardware:

* **Ordenador o dispositivo con navegador web**: Los usuarios pueden acceder a la aplicación desde cualquier dispositivo con un navegador moderno.
* **Servidor backendd**: En caso de utilizar una base de datos y servidor propio para la API, se requerirá un servidor donde se ejecute la aplicación backend (Node.js) y la base de datos (MongoDB).

**1.4.3. Recursos software**

Los recursos software necesarios incluyen:

* **Angular**: Framework utilizado para desarrollar la aplicación frontend.
* **Node.js**: Plataforma de backend que ejecuta la API.
* **MongoDB**: Sistema gestor de bases de datos NoSQL donde se almacenan los datos de los usuarios, transacciones y metas de ahorro.
* **HTML, CSS y JavaScript**: Tecnologías estándar utilizadas para la implementación del diseño y la lógica del frontend.
* **Postman**: Herramienta utilizada para probar la API backend y asegurar que las respuestas de las solicitudes HTTP sean correctas.

**1.5 Estructura de la aplicación**

**1.5.1 Componentes principales:**

* **Cabecera**:
  + Menú lateral con las opciones: Transacciones, Ahorros y Perfil.
  + Menú desplegable que se activa al pasar el ratón por encima.
* **Transacciones**:
  + Visualización de la información de saldo disponible, últimos ingresos y gastos.
  + Formularios para agregar ingresos y gastos.
  + Tabla para mostrar las transacciones con columnas de fecha, concepto, categoría e importe.
  + Actualización del saldo disponible automáticamente al agregar una transacción.
* **Ahorros**:
  + Formulario para crear metas de ahorro (nombre, cantidad, fecha límite).
  + Visualización del progreso de cada meta de ahorro con una barra de progreso.
  + Formulario para añadir dinero a las metas de ahorro.
* **Perfil**:
  + Datos personales del usuario (nombre, apellidos y nombre de usuario).
  + Botón cerrar sesión.
* **Login y Registro**:
  + Formulario para que los usuarios puedan crear una cuenta o iniciar sesión en la plataforma.

**2. Documentación Técnica**

**2.1 Especificación de Requisitos**

**2.1.1 Requisitos Funcionales**

* Sistema de autenticación con registro e inicio de sesión.
* Menú lateral desplegable con enlaces a transacciones, ahorros y perfil.
* Módulo de transacciones con gestión de ingresos y gastos.
* Módulo de ahorros con creación de metas y actualización de progreso.
* Información del perfil de usuario y botón de cerrar sesión

**2.1.2 Requisitos No Funcionales**

* Uso de Angular para el frontend y Node.js con MongoDB para la API.
* Validaciones en formularios para restringir fechas y campos obligatorios.
* Interfaz responsiva y amigable.
* Seguridad en el almacenamiento de credenciales.

**2.1.3 Arquitectura del Software**

* **Frontend**: Angular con estructura modular y uso de servicios para la comunicación con la API.
* **Backend**: Node.js con Express, gestionando autenticación y operaciones en MongoDB.
* **Base de Datos**: MongoDB para almacenamiento de usuarios, transacciones y metas de ahorro.
* **Comunicación**: API REST con endpoints para gestión de usuarios, transacciones y ahorros.

**2.2 Diseño del Sistema**

**2.2.1 Modelo de Base de Datos (MongoDB)**

* **Usuarios**: nombre, apellidos, nombre de usuario, contraseña y fecha de nacimiento.
* **Transacciones**: fecha, concepto, categoría, importe, tipo (ingreso/gasto), usuario.
* **Metas de ahorro**: nombre, cantidad objetivo, fecha límite, cantidad ahorrada, usuario.

**2.2.2 Código Fuente Documentado**

Se ha utilizado **Compodoc** para generar documentación del código Angular.

**2.2.3 Pruebas y Casos de Prueba**

* **Pruebas de Integración**: Verificación de la comunicación entre frontend y backend.
* **Pruebas de Aceptación**: Validaciones en Postman para los endpoints de la API.

**2.2.4 Guía de Despliegue**

1. Clonar el repositorio.
2. Instalar dependencias (npm install).
3. Configurar la API en environment.ts.
4. Ejecutar la aplicación (ng serve).

**2.3 Documentación de Usuario**

**2.3.1 Manual de Usuario**

* **Inicio de sesión**: Ingresar usuario y contraseña registrados.
* **Registro**: Completar formulario con datos personales.
* **Gestión de transacciones**: Registrar ingresos y gastos, visualizar historial.
* **Gestión de ahorros**: Crear metas y actualizar progreso.
* **Configuración**: Modificar usuario y contraseña.

**2.3.2 Guía de Instalación**

* **Requisitos**: Node.js, Angular CLI, MongoDB.

**2.4 Documentación de Gestión y Mantenimiento**

**2.4.1 Plan de Proyecto**

* **Alcance**: Desarrollo de una aplicación para gestión financiera personal.
* **Metodología**: Agile con sprints iterativos.

**2.4.2 Control de Versiones**

* Uso de **GitHub**.

**2.4.3 Documentación de Seguridad**

* **Autenticación**: Hash de contraseñas con bcrypt.
* **Protección de Datos**: Uso de JWT para autenticación segura.
* **Validaciones**: Restricciones en formularios y API para evitar ataques de inyección.