PRESENTACIÓN E ÍNDICE

INTRODUCCIÓN

La idea inicial es desarrollar una app que de forma sencilla permita tomar nota y llevar un seguimiento de cualquier objetivo, reto o nuevo habito de vida que se quiera comenzar o mantener, para poder controlar las mejoras realizadas y crear mayor adherencia.

El objetivo principal es crear una aplicación para móvil que permita registrar y visualizar los avances de forma rápida, pudiendo además compartirlo con amigos y familiares y otros usuarios de la app. El objetivo secundario es realizar está aplicación hibrida empleando tecnologías y frameworks que no se han visto en el curso en combinación con los que si se han visto para reforzar los conocimientos adquiridos.

DISEÑO

Dentro de este apartado tenemos los requerimientos de la app, los requisitos, la planificación..

TECNOLOGÍAS Y HERRAMIENTAS

Las tecnologías y herramientas empleadas son, básicamente, angular para el front, ionic y material design para la vista, la base de datos es mongodb, el servidor es nodejs, concretamente express y capacitor nos permite obtener la apk y acceder a funcionalidades nativas, añadir información al archivo AndroidManifest… Para el desarrollo se ha empleado visual studio code y github para el control de versiones. Trello para la organización del tiempo, las tareas, etc y la suite de Microsoft office para la memoria y la presentación, algunos diagramas...

Con todas estas tecnologías se ha desarrollado una aplicación híbrida, que, tal y como se puede leer, se basa en el paradigma “write once, run everywhere”. Las aplicaciones híbridas, a diferencia de las nativas, son aquellas capaces de funcionar en distintos sistemas operativos móviles. Entre ellos: Android, iOS y Windows Phone. De esta manera, una misma app puede utilizarse en cualquier smartphone, tablet, escritorio… indistintamente de su marca o fabricante.

Aunque no aprovechan todo el rendimiento que ofrece cada sistema operativo y son algo más lentas que las apps nativas, teniendo en cuenta lo requerimientos y funcionalidades de la app que se ha desarrollado es más que suficiente.

REQUISITOS

Para poder hacer una buena planificación se ha redactado una lista de requisitos, funcionales y no funcionales, que podría resumirse de la siguiente forma:

Funcionales: CRUD de usuarios, ítems y amigos, consulta de información y registro, inicio y cierre de sesión

No Funcionales: Ajustarse a las 40 horas de desarrollo estimadas, que sea una apliación más o menos segura y que el diseño sea moderno e intuitivo, lo más ajustado posible a los mockups desarrollados.

GESTIÓN DEL PROYECTO

El paradigma elegido es el desarrollo en cascada con retroalimentación, que admite la posibilidad de hacer iteraciones, es decir, durante las modificaciones que se hacen en el mantenimiento se puede ver, por ejemplo, la necesidad de cambiar algo en el diseño, lo cual significa que se harán los cambios necesarios en la codificación y se tendrán que realizar de nuevo las pruebas, es decir, si se tiene que volver a una de las etapas anteriores al mantenimiento hay que recorrer de nuevo el resto de las etapas. Los motivos principales para su elección han sido que la planificación es muy sencilla, la calidad del producto resultante es alta, permite retroceder en cualquier momento si fuera necesario y es el más sencillo de utilizar cuando el desarrollador tiene poca experiencia.

Por otra parte, dentro del bloque de Codificación, se ha determinado usar la metodología de desarrollo SCRUM. Esta metodología se adapta a los problemas que van surgiendo; y se define y cambia tras cada sprint, lo cual permite mantener una estimación más adecuada de las tareas realizadas y pendientes, realizando pequeños ajustes constantes a los requisitos y estimaciones

DIAGRAMAS

Uno de los diagramas que se ha hecho para el diseño de la app es el de los casos de uso, describiendo todas las posibles interacciones entre el usuario del sistema y la aplicación con el fin de alcanzar un objetivo.

De esta manera se proporcionan diversos escenarios que indican la sucesión de respuestas surgidas ante las acciones iniciadas por los usuarios sobre el sistema.

Esta especificación de casos de uso facilita la comprensión del sistema y expresa la intención con la que el usuario interactuará con la aplicación. Complementándose con los requisitos anteriormente descritos, establece una firme base a la hora de comenzar con la fase de diseño.

Se ha creado actor para usuarios que no han creado una cuenta o no han iniciado sesión todavía y otro para los usuarios que han iniciado sesión.

GUÍA DE ESTILOS

Muestra de la guía de estilos. Se han buscado colores y formas modernos, que transmitan energía y optimismo y que a su vez no resulten cargantes ya que la idea es que el usuario utilice la aplicación prácticamente a diario.

FRONTEND

En este caso el frontend de la aplicación se basa en Angular, con typescript, Ionic, que tiene sus propias etiquetas html y para los estilos se ha utilizado Sass, además de seguir los patrones marcados por Material Design y sus componentes.

ARQUITECTURA FRONT Y ANGULAR

Aquí tenemos dos imágenes, la primera de ellas es un esquema de como se relacionan entre sí las distintas partes de la aplicación, Ionic y Angular nos proporcionan las vistas, componentes y servicios, y capacitor (no cordova) el acceso a los componentes y funcionalidades nativas del dispositivo. Todo esto se transpila a js y css y se empaqueta en una apk para poder instalar la aplicación en diferentes dispositivos.

Por otro lado, centrándonos en Angular, su arquitectura es MVVM, ya que el modelo tiene mucha relación con la vista por el concepto de data binding en doble sentido, es decir, desde el modelo podemos modificar la vista y desde la vista podemos modificar el modelo.

COMPONENTES

Un componente controla un espacio de la pantalla, que se denomina vista. Es, en realidad, una clase de JavaScript (ES6) con el decorador @Component que incluye las propiedades y métodos disponibles para su template. Se debe incluir en esta clase todo lo referente al controlador de la vista y abstraer todos los demás métodos en servicios que serán inyectados posteriormente. Cada componente contiene una combinación de un archivo .html con un .ts y algunas veces un .css para crear un elemento con características propias tanto de comportamiento como de apariencia que se puede mostrar.

Esto es muy útil para evitar repetir código y para tener un código mas mantenible y fácil de escalar, ya que al tener archivos mas cortos es mas simple detectar fallos y corregirlos.

SERVICIOS

un servicio es un proveedor de datos, que mantiene la lógica de acceso a los mismos y la operativa relacionada con el negocio y las cosas que se hacen con los datos dentro de una aplicación. Los servicios serán consumidos por los componentes, que delegarán en ellos la responsabilidad de acceder a la información y la realización de operaciones con los datos.

En este proyecto se han utilizado principalmente para invocar a un servidor HTTP para consumir una API. También es el mecanismo para compartir funcionalidad entre componentes ya que estas clases son perfectamente instanciables desde cualquier otro fichero que las importe.

La mayoría de métodos incluidos retornan un objeto Observable. Mediante el Observable es posible suscribirse a eventos que nos permiten hacer cosas cuando cambia lo que se esté observando, es una de las mejores formas de optimizar una aplicación, ya que aumenta su rendimiento.

Para este proyecto se ha separado la lógica de negocio en 3 servicios independientes en función de con qué elemento estén relacionados (usuarios, ítems o amigos), un servicio de utilidades para tener fácilmente accesibles métodos o datos que se requieren a lo largo de toda la aplicación y, por último, un servicio dedicado exclusivamente a la autenticación de los usuarios mediante un token que es llamado antes de realizar cualquier otro método de los demás servicios. Estos servicios después se instancian desde los demás componentes del proyecto

GUARDS

Los Guards son, de alguna manera, middlewares que se ejecutan antes de cargar una ruta y determinan si se puede cargar dicha ruta o no. Se ejecutan de manera intermedia antes de determinadas acciones, en función de si devuelven true o false se carga una ruta u otra.

Una de las ventajas de utilizarlos, es que al no cargar la ruta evitamos que los usuarios vean una interfaz a la que no tienen acceso.

Estos componentes permiten estructurar el código de la aplicación de una manera más organizada y donde la lógica de la ruta está en la ruta en sí y la lógica de permisos y acceso a recursos se encuentra aislada en los guards.

En este caso se han utilizado para evitar que un usuario que no está loggeado pueda acceder a determinadas rutas, y por otro lado, para que una vez está el usuario loggeado sea redirigido a Home y no al formulario de inicio de sesión si cierra la aplicación temporalmente o la deja en segundo plano sin cerrar la sesión.

PÁGINAS Y ENRUTAMIENTO

Una de las responsabilidades de las que se hace cargo el front es la de procesar las rutas y determinar cuál será la vista que se deba mostrar en cada dirección. En este caso concreto la aplicación tiene tres páginas, registro, inicio de sesión y section, que sería el contenedor en el que se vuelca el resto de secciones de la app y se cargan el resto de los componentes. Para evitar que el usuario se descargue la definición de los componentes antes de visitarlos e incluso componentes de rutas que quizá nunca visite, se ha decidido implementar Lazy Loading, que permite mejorar el rendimiento de la aplicación por una cuestión de peso y velocidad.

la navegacion dentro de cada una de las páginas la controla un archivo especifico para cada una de ellas, cuyo objetivo es diferir la descarga de las rutas no visitadas y para ello se empaqueta cada ruta en un **bundle**.

Es decir, cuando se active la ruta section, por ejemplo, entonces se le transfiera el control al módulo SectionPageModule mediante una instrucción asíncrona. De esta forma se consiguen dos cosas: por un lado al no usarse ningún componente explícito no hay que importarlo; por otro lado la descarga del módulo que resuelva la ruta se ejecutará en segundo plano y sólo si el usuario decide visitar la ruta.

Para cargar los componentes que corresponden con cada página se emplea un componente llamado <router-outlet> y actúa como un contenedor dinámico, incrustando el componente adecuado apara cada ruta.

Además, para que los hijos aparezcan dónde deben, hay que usar un segundo <router-outlet> dentro de la vista del componente padre

FORMULARIOS Y SEGURIDAD

Se han usado formularios reactivos, que realizan un seguimiento del valor, el estado de cambio y la validez de los datos. Puede llevar datos precargados o no. Esto permite, por ejemplo, controlar fácilmente si se debe habilitar o deshabilitar algún campo o botón, además de validar los datos fácilmente antes de sanitizarlos por completo en la API, lo cual aporta una capa de seguridad extra al sistema. Por otro lado, también facilita el feedback al usuario final si hay algún error en el formulario, ya que la validación particular para cada control permite informar al usuario del fallo concreto.

BACKEND

El front-end muestra la información, pero es el back-end quien realmente trabaja con esta y filtra todas las acciones, enruta los accesos y vigila que todo fluya como tenga que fluir evitando accesos no permitidos según el usuario. En el back se ha construido un API REST usando para ello la tecnología de NodeJS y MongoDB.

BASE DE DATOS

Para la base de datos se ha elegido mongoDB , que guarda los datos en un formato muy similar a json, además de ofrecer un modelo de datos flexible. Se ha optado por esta solución a causa de la restricción de horas para la realización de este proyecto, ya que brinda la facilidad de escalar y hacer crecer el sistema de una manera muy sencilla. Se han creado tres tipos de documentos, que se identifican con el username del usuario al que pertenecen.

SEGURIDAD

La seguridad es un tema de suma importancia en la actualidad. En nuestro caso, al no permitir pagos online ni exigir identificación mediante algún documento oficial, podemos afirmar que no se guardan datos extremadamente sensibles de los usuarios y, por tanto, es poco probable que haya vulnerabilidades de seguridad graves. Aun así, se han implementado unas cuantas prácticas para garantizar un mínimo de confianza:

* Se ha filtrado y saneado la entrada datos, usando mongo-sanitize, un módulo que evita que cualquier palabra que comience con ‘$’ sea introducida en una query de MongoDB
* Se ha evitado el uso de $where ya que es uno de los operadores menos seguros de usar, debido a que, aunque se filtren datos es difícil evitar que se introduzca un 0 o un true como valor en determinadas querys
* Se han encriptado las contraseñas para mayor seguridad con la ayuda de bcrypt, una librería de encriptación que, básicamente, es una función de hashing de contraseñas, basado en cifrado Blowfish.
* Se ha implementado una autenticación mediante token. La firma de un token JWT se construye de tal forma que vamos a poder verificar que el remitente es quien dice ser (dificultando un ataque MITM)
* Se ha limitado el tamaño del cuerpo de las peticiones. Al igual que se controla el tipo de los datos de entrada, también se ha limitado su tamaño máximo para evitar ataques DoS.

MIDDLEWARES

Los middleware son funciones que tienen acceso al objeto de solicitud (req), al objeto de respuesta (res) y a la siguiente función de middleware en el ciclo de solicitud/respuestas de la aplicación. Se han usado varios middlewares de terceros y se han desarrollado dos, uno para verificar que los tokens de las distintas peticiones eran válidos y otra para verificar que el usuario y el email no estaban en uso antes de insertar un nuevo usuario en la base de datos.

PRUEBAS

Para elaborar test que permitan validar el comportamiento de la API se ha utilizado Postman. Postman es una herramienta que permite realizar peticiones a APIs y generar colecciones de peticiones que nos permitan probarlas de manera rápida y sencilla.

CONCLUSIONES

Por último, las conclusiones del desarrollo de la aplicación son las siguientes:

Dentro de las posibles mejoras, incluiría mejorar el responsive, traduciría la aplicación a más idiomas..

Las principales dificultades han sido la gestión de tiempo, que es muy limitado y muy difícil de estimar debido a la falta de experiencia. La gran variedad de resoluciones de pantallas existentes, el uso de tecnologías no usadas a lo largo del curso y problemas de compatibilidad entre versiones, librerías, etc. Los principales logros es que el diseño es muy parecido al mockup planteado, se han puesto en practica conocimientos adquiridos tanto en el primer como el segundo curso del grado y además se han aprendido nuevas tecnologías de forma autodidacta.

Como resultado tenemos una app hibrida y funcional con un diseño intuitivo.

FIN

A continuación os voy a mostrar un poco los resultados en una pequeña demo y a continuación podéis realizar las preguntas que consideréis oportunas. Muchas gracias por la atención.