UNIMATE

MEMORIA DEL PROYECTO

LABORATORIO INTEGRADO DEL SOFTWARE - 2024

Bruno Morán Vivanco 1636272
Carolina Buil Garcia 1632912
Daniel García Nilo 1633613
Hugo García Benjumea 1600866
Martí Bruix Fernandez 1600987
Oriol Alarcón Fontanet 1564525
Oriol Puertas Martín 1636385
Roman Ardanuy Hamill 1639402

Índice

1. El contrato

En el desarrollo de UNImate, nos centramos en cumplir con los puntos clave del contrato. En concreto, llegamos a completar hasta el 10. A continuación, se describe cómo abordamos y completamos cada una de estas partes:

1.1. Objetivos del cinco

Perfiles de Usuario:

- **Estado**: Completado
- **Descripción**: Implementamos el registro e inicio de sesión para el acceso a la aplicación. Una vez dentro, los estudiantes pueden personalizar sus perfiles añadiendo una foto de perfil, un color de encabezado, sus estudios, lugar de residencia y una descripción personal.
- **Equipo principal**: Hugo y Román se encargaron de esta funcionalidad.

Búsqueda de Información:

- **Estado**: Completado
- **Descripción**: Desarrollamos una lista interactiva que permite a los usuarios buscar y visualizar información detallada sobre universidades, facultades y grados. Esta funcionalidad incluye valoraciones de las universidades por parte de los usuarios, con un sistema de hasta 5 estrellas.
- **Equipo principal**: Martí y Oriol Alarcón lideraron el desarrollo de esta funcionalidad.

Foro:

- **Estado**: Completado
- **Descripción**: Implementamos un foro donde los usuarios pueden publicar, comentar y reaccionar a diversas temáticas relacionadas con la vida universitaria. También pueden filtrar por categorías.
- **Equipo principal**: Bruno y Oriol Puertas fueron responsables de la implementación del foro.

Front-end: Daniel y yo creamos el diseño y la implementación del front-end para todas estas pantallas y funcionalidades.

1.2. Objetivos del siete

Red de Amigos:

- **Estado**: Completado
- **Descripción**: Esta funcionalidad permite a los usuarios conectar y mantener una red de amigos dentro de la aplicación. Los estudiantes pueden enviar y

aceptar solicitudes de amistad, ver los perfiles de sus amigos y sus posts en el foro.
- **Equipo**: Hugo y Román trabajaron en la implementación de la red de amigos.

Mapa Interactivo:

- **Estado**: Completado
- **Descripción**: Desarrollamos un mapa interactivo que muestra la ubicación de universidades y facultades. Los usuarios pueden navegar fácilmente por el mapa para encontrar la información que necesitan.
- **Equipo**: Martí y Oriol Alarcón fueron responsables de esta funcionalidad.

Chat:

- **Estado**: Completado
- **Descripción**: Implementamos un sistema de chat que permite la comunicación en tiempo real entre los usuarios, incluyendo tanto chats privados con amigos como chats grupales. Al añadir tus estudios, se te incluye automáticamente en el chat grupal correspondiente a tu grado y facultad, de los que puedes salir si lo deseas.
- **Equipo**: Bruno y Oriol Puertas desarrollaron el chat privado, y a partir de este, Hugo y Román desarrollaron el chat grupal.

Front-end: Dani y yo diseñamos y desarrollamos todas las pantallas de estas funcionalidades.

1.3. Objetivos del diez

Recomendador de Perfiles:

- **Estado**: Completado
- **Descripción**: Esta funcionalidad utiliza un algoritmo para recomendar perfiles de otros usuarios basándose en intereses y preferencias compartidas en un formulario. De los perfiles que se te muestran en pantalla, puedes deslizar para ver más opciones o visitar su perfil y enviar una solicitud de amistad.
- **Equipo**: Martí lideró el desarrollo de esta funcionalidad.

Verificación de Matrícula:

- **Estado**: Completado
- **Descripción**: Implementamos un sistema de verificación de matrícula en el registro de los usuarios, asegurando que solo los estudiantes universitarios puedan acceder.
- **Equipo**: Oriol Alarcón fue responsable de esta funcionalidad.

Front-end: Daniel y yo creamos el diseño y la implementación del front-end para todas estas pantallas y funcionalidades.

En resumen, hemos cumplido satisfactoriamente con las funcionalidades clave especificadas en los puntos 5, 7 y 10 del contrato. Creemos que el proyecto se encuentra en la franja de nota más alta (9-10) debido al cumplimiento total de los requisitos, la superación de desafíos técnicos y la calidad del producto final entregado.

2. Metodología del desarrollo del proyecto

Para el desarrollo de la aplicación UNImate, utilizamos una metodología Agile adaptada a las necesidades específicas del proyecto. Esta metodología nos permitió ser flexibles y responder rápidamente a cualquier cambio surgiera durante el desarrollo. Nos organizamos en tres grandes sprints según cada nota del contrato. A continuación, se detalla cómo implementamos esta metodología y cómo se distribuyeron las tareas entre los miembros del equipo.

2.1. Sprint 1: Inicio del proyecto y objetivos del 5

El primer sprint se inició el día 4 de marzo, día en que el equipo se reunió por primera vez. En este sprint nos centramos en alcanzar los requisitos necesarios para obtener una nota de 5. Durante la primera mitad del sprint, el objetivo principal era establecer las bases fundamentales de la aplicación, realizando la documentación de los requisitos y elaborando el contrato.

Una vez tuvimos claros los objetivos, durante la segunda mitad del Sprint, organizamos al equipo en pares específicos para cada funcionalidad, ya que eran independientes entre sí, de la siguiente manera:

- **Perfiles de usuario:** Hugo García y Román Ardanuy
- **Búsqueda de universidades, facultades y grados por lista:** Martí Bruix y Oriol Alarcón
- **Foro:** Bruno Morán y Oriol Puertas
- **Prototipo de la app y diseño de pantallas:** Carolina Buil y Daniel García

Esta organización permitió que cada equipo se concentrará en una funcionalidad específica, asegurando un desarrollo eficiente y enfocado. Al final de este sprint, todas las funcionalidades planificadas se completaron con éxito o estaban a algunos requisitos de ser completadas, cumpliendo con los objetivos del primer Sprint.

Fechas de reuniones durante Sprint 1:

- 4 de marzo: Inicio del sprint, primera reunión y preparación presentación.
- 6 de marzo: Presentación e inicio toma de requisitos.
- 11 de marzo: Redactado de documentos de visión y especificaciones..
- 15 de marzo: Revisión de progreso.
- 18 de marzo: Aprendizaje de tecnologías y redacción del contrato.
- 20 de marzo: Evaluación intermedia. Creación proyecto y BD.
- 2 de abril: Revisión de progreso.
- 3 de abril: Revisión de progreso.
- 8 de abril: Evaluación y preparación para la presentación.
- 10 de abril: Cierre del sprint y revisión final de tareas pendientes.

2.2. Sprint 2: Del 10 de abril al 10 de mayo

El segundo sprint se centró en finalizar detalles pendientes del primer sprint y comenzar a trabajar en las funcionalidades necesarias para alcanzar la nota del 7.

Dado el éxito de las parejas formadas en el primer sprint, decidimos mantener la misma estructura, asignando nuevas tareas:

- Red de amigos: Hugo García y Román Ardanuy
- Mapa interactivo: Martí Bruix y Oriol Alarcón
- Creación del chat: Bruno Morán y Oriol Puertas
- Implementación de diseños de pantallas: Daniel García y Carolina Buil

Durante este sprint, la primera funcionalidad en completarse fue el mapa interactivo. Esto permitió que Martí Bruix y Oriol Alarcón comenzaran a trabajar en las funcionalidades previstas para alcanzar una nota de 10.

Fechas de reuniones durante Sprint 2:

- 10 de abril: Inicio del sprint, planificación y asignación de tareas.
- 15 de abril: Revisión del progreso.
- 17 de abril: Revisión del progreso.
- 22 de abril: Evaluación intermedia y reasignación de recursos según progreso.
- 24 de abril: Revisión de progreso.
- 29 de abril: Revisión de progreso.
- 3 de mayo: Evaluación intermedia y reasignación de recursos según progreso.
- 6 de mayo: Revisión de progreso y preparar presentación.
- 8 de mayo: Presentación y evaluación final de tareas antes del cierre del sprint.
- 10 de mayo: Cierre del sprint y revisión final de tareas pendientes.

2.3. Sprint 3: Del 10 de mayo al 2 de junio

En el tercer y último sprint, nuestro equipo se enfocó en completar todas las funcionalidades necesarias para alcanzar la máxima nota del contrato, así como en realizar la documentación final y llevar a cabo pruebas exhaustivas del sistema. Estas tareas se dividieron en “dos fases”.

Durante la primera fase, nos dedicamos a finalizar los requisitos pendientes del sprint 2 y a desarrollar las funcionalidades correspondientes al 10 del contrato. En la segunda fase, nos centramos en el testing y en la elaboración de la documentación para la entrega final.

Dado que la creación del chat requirió más tiempo del previsto, realizamos ajustes en la asignación de tareas y reorganizamos al equipo de la siguiente manera:

- **Chat grupal:** Hugo García y Román Ardanuy
- **Chat privado:** Oriol Puertas y Bruno Morán
- **Recomendador de perfiles:** Martí Bruix
- **Verificación de matrícula:** Oriol Alarcón
- **Finalizar Front-end:** Carolina Buil y Daniel García

Fechas de reuniones durante Sprint 3:

- ➢ 10 de mayo: Inicio del sprint, planificación y asignación de tareas.
- ➢ 13 de mayo: Revisión de progreso.
- ➢ 15 de mayo: Evaluación intermedia y preparación para la fase de testing.
- ➢ 18 de mayo: Revisión de progreso.
- ➢ 20 de mayo: Revisión de progreso.
- ➢ 22 de mayo: Evaluación final del contrato en clase y redistribución de tareas de presentación y testing.
- ➢ 25 de mayo: Preparar presentación final y vídeo promocional.
- ➢ 27 de mayo: Presentación final
- ➢ 30 de mayo: Revisión final antes del cierre del Sprint y del proyecto.

El miércoles 22 de mayo, junto al profesor Rodolfo, dimos por completadas todas las funcionalidades y pantallas requeridas para alcanzar la máxima calificación, asegurando que todos los requisitos se habían cumplido satisfactoriamente.

Gracias a esta reorganización y al esfuerzo coordinado de todo el equipo, logramos cumplir con todos los objetivos del sprint, asegurando la calidad y funcionalidad del proyecto en su totalidad.

3. Captura de Requisitos de UNImate

Para asegurar que la aplicación UNImate cumpliera con las expectativas y necesidades de los usuarios, realizamos un formulario dirigido a los usuarios finales de la app para tomar los requisitos del proyecto. Este proceso fue fundamental para definir las funcionalidades y características clave que debía tener la plataforma.

A continuación, se detalla cómo se capturaron estos requisitos y se presentan los cinco requisitos principales, tanto funcionales como no funcionales.

3.1. Proceso de captura de requisitos

Al inicio del proyecto, uno de los primeros pasos fue entender las necesidades y deseos de los usuarios potenciales. Mediante la creación y distribución de un formulario dirigido a estudiantes universitarios, realizamos preguntas clave sobre las funcionalidades deseadas, el tipo de información que les interesaría y cómo preferían interactuar con la plataforma.

Link al formulario	https://forms.gle/YPWdL81AT9u4QGu16

La distribución del formulario resultó en 45 respuestas, proporcionando una base sólida de datos sobre las expectativas y demandas de los usuarios. Con esta información en mano, Carolina Buil y Daniel García comenzaron a redactar los documentos de visión y especificaciones[1] del proyecto.

3.2. Requisitos principales

A continuación, se detallan los cinco requisitos principales de la aplicación UNImate, incluyendo tanto requisitos funcionales como no funcionales:

- **Requisito funcional RF-05**

La plataforma debía permitir a los usuarios crear y personalizar sus perfiles, añadiendo información personal, intereses y fotos de perfil. Esta funcionalidad es esencial para personalizar la experiencia del usuario y facilitar la conexión entre estudiantes con intereses similares.

ID: RF-05	Tipo: Funcional	Versión: 1.1
Título: Personalización del Perfil		

[1] Los documentos mencionados se encuentran en el zip bajo esos mismos nombres.

Descripción: La plataforma ha de permitir a los usuarios personalizar su perfil con información adicional como intereses y preferencias, así como añadir fotos de perfil y editar la información personal, para crear una experiencia más personal y única para cada usuario.
Relaciones: RF-07
Comentarios: Sin comentarios.

- **Requisito funcional RF-29**

Se implementó un foro donde los estudiantes pueden publicar, comentar y reaccionar a mensajes sobre diversos temas académicos y sociales. Este foro es fundamental para fomentar la interacción y el intercambio de información entre estudiantes, creando una comunidad activa dentro de la plataforma.

ID: RF-29	Tipo: Funcional	Versión: 1.0
Título: Añadir publicación		
Descripción: Un usuario ha de poder añadir/publicar una nueva publicación en el foro.		
Relaciones: RF-11, RF-12, RF-30, RF-31, RF-34, RNF-9		
Comentarios: Las publicaciones han de estar diferenciadas por una etiqueta donde se vea el nombre de la categoría.		

- **Requisito funcional RF-18**

La plataforma debía permitir el envío y recepción de mensajes en tiempo real, tanto en chats privados como en grupos. Esta funcionalidad facilita la comunicación directa entre los estudiantes.

ID: RF-18	Tipo: Funcional	Versión: 1.3
Título: Chat privado		
Descripción: Envío y recepción de mensajes por un chat en tiempo real entre dos usuarios que son amigos.		
Relaciones: RF-13		
Comentarios: Los usuarios han de ser amigos para habilitar el chat privado.		

- **Requisito funcional RF-27**

UNImate debía incluir un algoritmo que recomiende perfiles de otros estudiantes con intereses similares. Esta funcionalidad ayuda a los estudiantes a encontrar y

conectar con compañeros que comparten sus intereses, mejorando la cohesión social.

ID: RF-27	Tipo: Funcional	Versión: 1.1
Título: Recomendar perfiles		
Descripción: A partir dels intereses de los usuarios mostrar perfiles recomendados con intereses similares.		
Relaciones: RF-26, RF-28		
Comentarios: Habría que automatizar utilitazando p.e. IA.		

- **Requisito no funcional RNF-05**

La plataforma debía tener una interfaz de usuario intuitiva, con navegación clara y fácil de usar, garantizando una experiencia fluida para todos los usuarios. Un diseño intuitivo es crucial para la adopción y uso continuo de la plataforma por parte de los estudiantes.

ID: RNF-05	Tipo: No Funcional	Versión: 1.0
Título: Usabilidad		
Descripción: La plataforma ha de poner un fuerte énfasis en la usabilidad y la experiencia del usuario, con un diseño intuitivo y una navegación clara y sencilla, para garantizar que los usuarios puedan utilizar el sistema de manera eficaz y eficiente sin necesidad de formación adicional.		
Relaciones: RNF-02		
Comentarios: Sin comentarios.		

Los cinco requisitos principales descritos aseguran que UNImate no solo cumple con las expectativas de los usuarios, sino que también proporciona una plataforma robusta y fácil de usar que enriquece la experiencia universitaria.

4. Control de Versiones

Pasemos ahora al desarrollo. Utilizamos GitHub como plataforma principal para el control de versiones. Esta herramienta nos permitió gestionar el código fuente de manera efectiva, facilitando la colaboración entre los miembros del equipo.Este enfoque nos permitió mantener un código base estable mientras continuamos desarrollando nuevas funcionalidades.

4.1. Política de ramas

En nuestro flujo de trabajo, empleamos dos ramas principales: master y develop.

- **Rama master:** Esta es la rama principal y estable que contiene la versión de producción del código. Solo se fusionan en esta rama los cambios que han sido completamente probados y aprobados. Mantener esta rama limpia y funcional es crucial, ya que representa el estado final del proyecto.

- **Rama develop:** Esta rama sirve como la rama principal de desarrollo. Todas las nuevas funcionalidades y correcciones de errores se desarrollan en ramas separadas creadas a partir de develop. Una vez que el trabajo en estas ramas está completo, se abre un pull request para fusionar los cambios de vuelta a develop. Esta rama permite que todos los desarrolladores trabajen en paralelo sin interrumpir el flujo de trabajo de los demás.

Una vez que una funcionalidad está lista, se abre un pull request para fusionar la rama a develop. Cada pull request es revisado por el responsable designado Hugo García, quien se encarga de verificar el código, realizar pruebas adicionales y asegurarse de que no haya conflictos con otros cambios recientes. Si Hugo es el autor del pull request entonces Martí Bruix era el encargado.

Este enfoque estructurado para el control de versiones permitió mantener la estabilidad del código base mientras se desarrollaban nuevas funcionalidades de manera efectiva.

Link al repositorio	https://github.com/Carolbg28/UNImate/tree/master

5. Problemas encontrados

5.1. Aprender desde cero las tecnologías

Uno de los mayores desafíos que enfrentamos durante el desarrollo de UNImate fue aprender a utilizar tecnologías completamente nuevas para la mayoría de los miembros del equipo: React y Firebase. Este proceso de aprendizaje no fue fácil pero pudimos llevarlo a cabo con un rotundo éxito.

5.2. Planificador en Asana

Al inicio, Asana fue nuestra herramienta elegida para la planificación del proyecto por su capacidad para organizar tareas y visualizar cronogramas. Sin embargo, Asana solo te ofrecía su uso durante 30 días de forma gratuita y no nos dimos

cuenta. Al finalizar el periodo de prueba, perdimos acceso a estas funciones, lo que nos dejó sin acceso a nuestro planificador.

Para superar este obstáculo, decidimos trasladar nuestra planificación[2] a Canva. Aunque Canva es una herramienta de diseño gráfico y no de gestión de proyectos, su flexibilidad nos permitió crear un documento claro y estructurado dónde pudimos tener de nuevo una planificación ordenada y sin límite de tiempo.

5.3. Bloqueo de la Base de Datos

La base de datos de Firebase permite un máximo de 50.000 operaciones de lectura al día de forma gratuita, cantidad que no debería haber sido un problema mientras desarrollamos la aplicación. Sin embargo, hubo un día en clase que mientras estábamos avanzando con el desarrollo se llegó a ese límite y se bloqueó el acceso a la base de datos, por lo que no pudimos seguir probando la aplicación en todo el día.

Entonces intentamos descubrir cuál era el motivo por el que se había llegado al límite y encontramos que en una parte del código se había generado por error un bucle infinito de lecturas a la base de datos. Corregimos este error y al día siguiente pudimos comprobar que el número de lecturas había vuelto a la normalidad. A partir de entonces tuvimos más cuidado a la hora de hacer las lecturas y no volvimos a tener el problema de que se nos bloquease la base de datos.

5.4. La versión de Expo Go

La versión de Expo Go se actualizó de la versión 50 a la 51 y esto nos generó un gran problema, al abrir la aplicación y pulsar a cualquier lugar esta se cerraba automáticamente. Por lo tanto, no podíamos hacer nada con la aplicación.

Después de mucha investigación encontramos el error y la solución en este Issue de GitHub (https://github.com/expo/expo/issues/28618). Básicamente se trataba de un error de Expo después de la última actualización y para solucionarlo tuvimos que seguir este hilo y seguir las recomendaciones. Al final pudimos arreglarlo añadiendo lo siguiente al primer archivo de la aplicación, aunque esta dependencia no se usará explícitamente en ningún punto:

```
// Fix Expo Go crash: https://github.com/expo/expo/issues/28618
import "react-native-reanimated";
```

[2] El planificador se encuentra en el archivo ZIP con el nombre ‘Planificador Unimate’.

6. Aplicación de las 7 Propiedades de Crystal Clear

Al inicio del curso, se presentaron las 7 propiedades del método Crystal Clear como una guía para el desarrollo de software ágil. Durante el desarrollo de UNImate, hemos implementado estas propiedades de diversas maneras, asegurando que se reflejarán en nuestro día a día. A continuación, se explica cómo cada una de estas propiedades se integró en nuestro proyecto.

6.1. Frequent Delivery

Desde el inicio, adoptamos una metodología de desarrollo ágil dividida en tres sprints principales con varios controles intermedios. Cada sprint tenía objetivos claros y entregables específicos. Esto nos permitió entregar incrementos funcionales de la aplicación regularmente, asegurando que el proyecto avanzara de manera constante y que la responsable del proyecto, Carolina, pudiera ver el progreso y proporcionar retroalimentación continua.

6.2. Reflective Improvement

La mejora continua fue una parte esencial de nuestro proceso. Al final de cada reunión y Sprint, realizamos retrospectivas para evaluar lo que funcionó bien y lo que necesitaba mejorar. Este enfoque reflexivo nos permitió ajustar nuestras estrategias y prácticas para optimizar el flujo de trabajo y la colaboración dentro del equipo.

6.3. Osmotic Communication

Para mantener una comunicación fluida y constante, utilizamos varias herramientas y métodos. Creamos un grupo de WhatsApp para la comunicación instantánea, realizamos reuniones semanales (dos, tres o cuatro veces por semana) y utilizamos Trello para la gestión de tareas. Esta comunicación osmótica facilitó el intercambio de información y la rápida resolución de problemas, asegurando que todos los miembros del equipo estuvieran siempre informados sobre el estado del proyecto y las tareas pendientes.

Link Trello	https://trello.com/invite/b/eM768rnB/ATTI94e2e62a12b073ab5f6f1c06e667ad46AFE055A3/unimate

6.4. Personal Safety

Fomentamos un ambiente de trabajo seguro y de apoyo donde todos los miembros del equipo se sintieron cómodos compartiendo sus ideas, preocupaciones y

sugerencias. La toma de decisiones fue colaborativa, y cada miembro tuvo la oportunidad de contribuir al proyecto de manera significativa. Esta cultura de seguridad personal promovió la innovación y la resolución de problemas creativa, ya que todos sabían que sus contribuciones eran valoradas y respetadas.

6.5. Focus

Mantener el enfoque fue crucial para el éxito del proyecto. Cada miembro del equipo tenía roles y responsabilidades claramente definidos, lo que ayudó a mantener el enfoque en sus tareas específicas. Durante las reuniones, priorizamos los requisitos más importantes y aseguramos que los recursos se asignaran de manera eficiente. Esta claridad en los roles y prioridades permitió que cada miembro del equipo se concentrará en su trabajo sin distracciones innecesarias.

6.6. Easy Access to Expert Users

Desde el inicio, involucramos a usuarios potenciales en el proceso de desarrollo. El formulario distribuido entre estudiantes universitarios proporcionó información valiosa sobre las necesidades y preferencias de los usuarios. Además, como la aplicación iba destinada a universitarios nosotros mismos podíamos probar las funcionalidades implementadas por los compañeros del equipo o podíamos pedir a gente de clase que probara la app.

6.7. Technical Environment with Automated Tests, Configuration Management, and Frequent Integration

Para asegurar la calidad y estabilidad del código, implementamos un buen control de versiones, testing y una integración continua frecuente. Utilizamos GitHub para el control de versiones e implementamos pruebas unitarias y de exploración para identificar y corregir errores de manera temprana, asegurando que cada entrega fuera de alta calidad.

7. Secciones destacadas del código realizado

7.1. Chat

chat-messages.jsx

Este fragmento de código es una parte crucial de la funcionalidad del chat dentro de la aplicación. Aquí hay algunas partes que se podrían destacar de dentro del código:

- **Recuperación de datos del chat:** Las primeras líneas del código se encargan de recuperar los datos del chat desde Firebase. Esto es esencial para cualquier funcionalidad de chat, ya que necesitas obtener los datos del chat para mostrarlos al usuario.

```
const chatDocRef = doc(FIRESTORE_DB, "chats", chatId);
const chatDoc = await getDoc(chatDocRef);
if (!chatDoc.exists()) {
  console.log("No such document!");
  return;
}
const chatData = chatDoc.data();
```

- **Diferenciación entre chats grupales e individuales:** Este fragmento de código distingue entre un chat grupal y un chat individual. Esta característica es vital ya que hace que nuestra app pueda soportar chats grupales.

```
if (chatData.isGroup) {
  setIsGroupChat(true);
  setOtherUserData(null);
  setChatImage(chatData.groupImage);
  setChatName(chatData.name);
} else {
  const otherUserId = chatData.users.find((id) => id !== user.id);
  const otherUserDoc = await getDoc(
    doc(FIRESTORE_DB, "users", otherUserId)
  );

  setIsGroupChat(false);
  const otherUserData = otherUserDoc.data();
  setOtherUserData(otherUserData);
  setChatImage(otherUserData.profilePhotoUrl);
  setChatName(otherUserData.username);
}
```

- **Preparación de la consulta de mensajes:** Al final del fragmento, preparamos una consulta para obtener los mensajes del chat. Esta consulta se utilizará para mostrar mensajes del usuario.

```
const messagesQuery = query(
  collection(chatDocRef, "messages"),
```

7.2. Verificación de matrícula

verify-enrollment

Este fragmento de código es importante para hacer el registro, es un componente del registro la cual consiste en coger una fotografía de la matrícula para verificar si

los datos que has puesto en el registro són los que corresponden con la matrícula universitaria.

- Para poder hacer la verificación se requiere coger una fotografía de la galería, esta función es para coger la fotografía. Se ha hecho con la librería llamada “expo-image-picker”, una vez seleccionada se pasa la imagen a la función performOCR.

```
const pickImageCamera = async () => {
  let result = await ImagePicker.launchImageLibraryAsync({
    mediaTypes: ImagePicker.MediaTypeOptions.Images,
    allowsEditing: true,
    base64: true,
    allowsMultipleSelection: false,
  });
  if (!result.cancelled) {
    if (!result.assets || !result.assets[0]) return;

    performOCR(result.assets[0]);
```

- La función performOCR es la que se encarga en transformar la imagen en una cadena de texto. Transformamos la imagen gracias a la api image-to-text que es una herramienta llamada OCR la cual es la encargada de extraer el texto.

```
fetch("https://api.apilayer.com/image_to_text/upload", requestOptions)
  .then((response) => response.json())
```

- Dentro de esta función normalizamos el texto procesado de la api. Eliminamos todos los espacios, todo el texto se convierte en minúscula.

```
setExtractedText(result["all_text"]);
const normalize = (str) =>
  str
    .toLowerCase()
    .replace(/\s+/g, "")
    .replace(/[^a-záéíóúñ0-9]/gi, "");
```

- Después de extraer el texto y normalizar, haces la gestión de “error”, la cual es la comprobación de varios elementos, se mira si el nombre puesto en el registro está dentro del resultado del OCR, también se comprueba si es actual la matrícula, esto lo miramos si dentro del texto hay el elemento 2023 o 2024. También se mira si en la cadena de texto hay algún fragmento que ponga matrícula. Todo esto si no se encuentra se le envía un error u otro a la persona que realiza el registro.

```
const nameComponents = fullName
  .split(" ")
  .map((name) => normalize(name));
const allComponentsPresent = nameComponents.every((name) =>
  normalizedResult.includes(name)
);
const containsYear =
  normalizedResult.includes("2023") ||
  normalizedResult.includes("2024");
const containsMatricula =
  normalizedResult.includes("matricula") ||
  normalizedResult.includes("matrícula") ||
  normalizedResult.includes("enrollment");
```

```
if (allComponentsPresent && containsYear && containsMatricula) {
  onVerification(true);
} else {
  if (!allComponentsPresent) return onVerification(false, 1);
  if (!containsYear) return onVerification(false, 2);
  if (!containsMatricula) return onVerification(false, 3);
  onVerification(false);
}
```

8. Testing

Nuestra principal estrategia de testing durante el desarrollo de la aplicación fue el **Exploratory Testing**. No obstante, también añadimos diferentes **Unit Tests** para comprobar el funcionamiento de componentes específicos utilizando el framework Jest.

8.1. Exploratory Testing

Este tipo de testing nos permitió simular cómo podría interactuar un usuario al navegar por nuestra aplicación, ayudándonos a detectar la mayor cantidad de errores posible.

Cada equipo realizaba pruebas exploratorias a medida que se desarrollaban nuevas funcionalidades, asegurándose de que las diferentes partes de la aplicación se acoplaban correctamente. Cada vez que encontrábamos un fallo, lo comunicabamos por Whatsapp para realizar un seguimiento y mantener a todo el grupo informado sobre su resolución y el avance en el desarrollo del proyecto.

Ejemplo en el Foro:

- **Primera versión**: Las pruebas iniciales en el foro consistían en crear publicaciones, verificando que se guardaran en la base de datos y se mostrarán correctamente en la sección del foro.
- **Implementación de categorías**: Al añadir categorías a las publicaciones, repetimos las pruebas anteriores, verificando que también se guardará la categoría elegida por el usuario. Además, comprobamos qué ocurría si el usuario no seleccionaba ninguna categoría, lo que nos llevó a encontrar otro error. Lo solucionamos añadiendo un aviso que indicaba la necesidad de

escoger una categoría y asegurando que la publicación no se guardara en la base de datos si no se seleccionaba una categoría.

Pruebas adicionales en las publicaciones:

- **Visualización de publicaciones**: Comprobamos que las nuevas publicaciones se mostraban correctamente con los datos recuperados de la base de datos.
- **Reacciones a publicaciones**: Al implementar las reacciones, verificamos que las reacciones comenzaban vacías, que el usuario solo podía reaccionar una vez y que no se permitía exceder el límite de cinco reacciones.
- **Filtrado de publicaciones**: Comprobamos que al seleccionar una categoría específica, los posts se filtraban correctamente.

En conclusión, en todas las partes del proyecto, al verificar que una funcionalidad era correcta, procedíamos a testear la siguiente, asegurándonos simultáneamente de que los pasos anteriores seguían siendo correctos. Esto garantizaba que las nuevas funcionalidades se integraran correctamente con el resto de la aplicación, manteniendo la cohesión y funcionalidad del sistema completo.

8.2. Unit Testing

A continuación se muestran dos Unit tests que se han realizado. El primero, para comprobar el funcionamiento de la verificación de matrícula, con verify-enrollment.test.jsx. El segundo, para comprobar que se muestre bien el perfil de la universidad, con University_profile.test.jsx:

Verify-enrollment.test.jsx

1. En el primer test case que se mira en el test es si el botón llamado “coge imagen de la matrícula” se renderiza correctamente del código verify-enrollment.jsx.

```
describe('Enrollment', () => {
  it('renders correctly', () => {
    const { getByText } = render(<Enrollment fullName="Test User" onVerification={() => {}} />);
    expect(getByText('Coge imagen de la matrícula')).toBeTruthy();
  });
```

2. En el segundo test case se realiza la comprobación de una vez presionado el botón de “coge imagen de la matrícula” ver si se muestra el modal (Este modal es una notificación en pantalla que da las indicaciones). Se simula la acción de pulsar el botón con “fireEvent.press(button)” y se comprueba que se abre el modal porque se espera que dentro ponga “Aquí están las indicaciones: ”

```
it('opens modal on button click', () => {
  const { getByText } = render(<Enrollment fullName="Test User" onVerification={() => {}} />);
  const button = getByText('Coge imagen de la matrícula');
  fireEvent.press(button);
  expect(getByText('Aquí están las indicaciones:')).toBeTruthy();
});
```

3. En el tercer test case se asegura que cuando el usuario interactúa con el modal y acepta la acción de selección de imagen, la aplicación responde correctamente llamando a la función "imagePicker.launchImageLibrary Async". Esto garantiza que la funcionalidad de selección de imágenes esté integrada y funcione según lo esperado.

```
it('calls ImagePicker on accept', async () => {
  const { getByText } = render(<Enrollment fullName="Test User" onVerification={() => {}} />);
  const button = getByText('Coge imagen de la matrícula');
  fireEvent.press(button);
  const acceptButton = getByText('Aceptar');
  await act(async () => {
    fireEvent.press(acceptButton);
  });
  expect(ImagePicker.launchImageLibraryAsync).toHaveBeenCalled();
});
```

University_profile.test.jsx

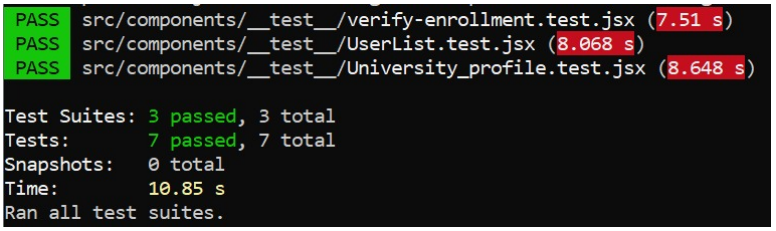
1. El primer test case asegura que se muestre toda la información necesaria de la universidad en su perfil, como el nombre, el país, la provincia, la descripción y las facultades.

```
const { getByText, getByRole } = render(<UniversityProfile id="testId" />);

await waitFor(() => {
  expect(getByText('Faculty 1')).toBeTruthy();
  expect(getByText('Faculty 2')).toBeTruthy();
});

expect(getByText('Test University')).toBeTruthy();
expect(getByText('Test Country, Test Province')).toBeTruthy();
expect(getByText('Test Description')).toBeTruthy();
```

2. El segundo test case comprueba que en el caso de que se intente acceder a una universidad que no tiene todos los campos rellenados en la base de datos se muestre el mensaje de "Información no disponible".

```
getDoc.mockResolvedValueOnce({
  exists: () => true,
  data: () => ({
    available: false,
  }),
});
const { getByText } = render(<UniversityProfile id="testId" />);
await waitFor(() => {
  expect(getByText('Información no disponible')).toBeTruthy();
});
```

Finalmente al realizar los tests vemos que estos pasan correctamente:



9. Conclusiones

En resumen, hemos desarrollado UNImate cumpliendo con todos los requisitos clave establecidos, desde la personalización de perfiles y el foro de discusión, hasta el sistema de chats y el recomendador de perfiles. Utilizando metodologías ágiles, hemos implementado y probado rigurosamente cada funcionalidad, asegurando una integración fluida y una experiencia de usuario óptima. Gracias a la colaboración eficiente y la resolución de problemas, entregamos un producto final robusto y de alta calidad.

Gracias al trabajo en equipo y a una buena comunicación, logramos entregar un producto final a las expectativas iniciales. Este proyecto nos ha permitido crecer como desarrolladores y como miembros de un proyecto SW. Estamos orgullosos de lo que hemos creado y confiamos en que UNImate algún día mejorará la vida universitaria de muchos estudiantes.