



Powered by: Alberto Ucendo Martínez

ÍNDICE

I. Introducción

- a. Presentación del proyecto
- b. Objetivos del proyecto
- c. Justificación del proyecto

II. Análisis de requerimientos

- a. Identificación de necesidades y requerimientos
- b. Identificación de público
- c. Estudio de mercado y competencia

III. Diseño y planificación

- a. Definición de la arquitectura del proyecto
- b. Diseño de la interfaz de usuario
- c. Planificación de las tareas y los recursos necesarios

IV. Implementación y pruebas

- a. Desarrollo de las funcionalidades del proyecto
- b. Pruebas unitarias y de integración
- c. Corrección de errores y optimización del rendimiento

V. Documentación

- a. Documentación técnica
- b. Documentación de usuario
- c. Manual de instalación y configuración

VI. Mantenimiento y evolución

- a. Plan de mantenimiento y soporte
- b. Identificación de posibles mejoras y evolución del proyecto

VII. Conclusiones

- a. Evaluación del proyecto
- b. Cumplimiento de objetivos y requisitos
- c. Lecciones aprendidas y recomendaciones para futuros proyectos

VIII. Bibliografía y referencias

- a. Fuentes utilizadas en el proyecto
- b. Referencias y enlaces de interés

I. Introducción

Presentación del proyecto

En un mundo digital saturado de aplicaciones dispersas, **AllOne** surge como una solución innovadora y anónima que **centraliza la comunicación y la productividad** en una única plataforma web. Actualmente combina:

- Mensajería instantánea (individual y grupal).
- Chatbot con IA integrada para asistencia inteligente.

Objetivos del proyecto

Objetivo general

Unificar múltiples servicios y herramientas digitales en una única **plataforma integrada**, eliminando la necesidad de alternar entre aplicaciones y **optimizando la experiencia del usuario** mediante inteligencia artificial manteniendo la posibilidad al usuario de mantenerse lo mas anónimo posible.

Objetivos específicos

- Centralizar aplicaciones clave (mensajería, IA, ...)
- Reducir la fragmentación digital (evitando que el usuario deba cambiar constantemente entre pestañas o dispositivos)
- Garantizar acceso rápido desde cualquier dispositivo (smartphone/PC)
- Minimizar el tiempo de aprendizaje (interfaz intuitiva y adaptable)

Justificación del proyecto

En la era digital, los usuarios están **sobrecargados de aplicaciones** (mensajería, redes sociales, asistentes de IA, etc.), en las cuales siempre tienes que ofrecer una inmensa cantidad de datos, lo que genera:

- **Frustración** por tener que cambiar constantemente entre plataformas.
- **Pérdida de tiempo** al repetir acciones en diferentes entornos.
- **Fragmentación de datos** (información dispersa en múltiples servicios).
- **Saturación cognitiva** debido a la diversidad de interfaces y funciones no integradas.
- **Desconfianza** hacia el aplicativo por la cantidad de datos y permisos que se solicitan.

AllOne surge como respuesta a este problema, ofreciendo una solución **unificada, intuitiva y potenciada por IA** que simplifica la interacción digital.

II. Análisis de requerimientos

Identificación de necesidades y requerimientos

Necesidades del usuario

- **Fragmentación de aplicaciones:** Los usuarios utilizan en promedio **8-12 apps distintas** diariamente (mensajería, redes, herramientas de trabajo).
- **Pérdida de productividad:** Cambiar entre apps consume **hasta 1.5 horas al día** (estudios de *RescueTime*).
- **Saturación de notificaciones:** Alertas no centralizadas generan **estrés digital**.
- **Falta de integración:** No existe interoperabilidad real entre determinados servicios.
- **Anonimato:** Hay una gran cantidad de usuarios que quieren mantenerse en el anonimato de manera sencilla y rápida.

Requerimientos funcionales y no funcionales

- Chat unificado.
- Aprendizaje contextual con IA
- Escalable.
- Interfaz intuitiva.

Identificación de publico

AllOne está dirigido principalmente a **profesionales digitales, estudiantes y equipos empresariales** (18-45 años) que buscan **simplificar su vida digital** o **mas privacidad** unificando mensajería y asistencia con IA en una sola plataforma. Nuestros usuarios clave incluyen:

- **Trabajadores remotos** que necesitan centralizar comunicación y gestión de proyectos.
- **Estudiantes** que organizan contactos de estudio.
- **Freelancers** que manejan múltiples clientes sin saturación de apps.
- **Empresas pequeñas** que buscan reducir costos en software disperso.

También cubrimos a **usuarios ocasionales** (como adultos mayores) que valoran simplicidad. El 87% de los encuestados prefieren soluciones todo-en-uno. AllOne resuelve su principal dolor: **la fatiga de alternar entre apps, ofreciendo un espacio integrado, inteligente y personalizable.**

Estudio de mercado y competencia

El mercado global de productividad y comunicación, valorado en USD \$96 mil millones, muestra un crecimiento acelerado (12% anual), con demanda creciente por soluciones unificadas con IA. AllOne compite en un espacio donde dominan herramientas fragmentadas como Slack (mensajería empresarial), Notion (organización) y asistentes de IA independientes, pero ninguna ofrece una integración completa entre mensajería y asistente inteligente en una sola plataforma.

Análisis de competencia revela que:

- **WhatsApp/Telegram** carecen de integración con productividad.
- **Slack/Microsoft Teams** no tienen IA contextual avanzada.
- **Notion/ClickUp** no incluyen mensajería en tiempo real.

La oportunidad clave de AllOne radica en **unificar estas funcionalidades** con un enfoque centrado en experiencia de usuario, personalización y automatización inteligente, aprovechando que el 62% de usuarios pagaría por esta solución integral (datos McKinsey 2023). La ventaja competitiva es clara: ofrecer **todo en uno** con un diseño intuitivo y adaptable a distintos perfiles (empresas, freelancers, estudiantes).

III. Diseño y planificación

Definición de la arquitectura del proyecto

AllOne utiliza una **arquitectura basada en microservicios** para garantizar escalabilidad y flexibilidad. El sistema se compone de:

- **Frontend:** Aplicación web (HTML/CSS/JS).
- **Backend:** Servicios independientes (Java/SpringBoot) que se comunican mediante APIs.
- **Base de datos:** MySQL (datos estructurados).
- **IA integrada:** DeepSeek para procesamiento de lenguaje natural.

Esta estructura permite actualizar, escalar o añadir funciones (como nuevos módulos de productividad) sin afectar el sistema completo, asegurando alto rendimiento y adaptabilidad.

Diseño de la interfaz de usuario

AllOne apuesta por una interfaz **minimalista y unificada** que prioriza la facilidad de uso en su plataforma todo-en-uno. Combina:

- **Estructura modular** área de trabajo adaptable
- **Diseño coherente** entre web y móvil (misma paleta de colores, iconos y tipografía)
- **Flujos integrados** como intercalar chats entre tus amistades y la IA
- **Asistente de IA accesible** desde cualquier pantalla

Planificación de las tareas y los recursos necesarios

El proyecto se desarrollará en **tres fases principales** con sus respectivos hitos:

1. **Desarrollo FrontEnd (1 mes y medio):** HTML, CSS y JS.
2. **Desarrollo BackEnd (1 mes):** SpringBoot con Java.
3. **Documentación refactorización de código/pruebas (medio mes aprox)**

Y como recursos he utilizado:

- Java
- HTML
- CSS basico
- Bootstrap
- JS
- API DeekSeek

IV. Implementación y pruebas

Desarrollo de las funcionalidades del proyecto

El desarrollo se centra en dos pilares principales: mensajería unificada (chat en tiempo real) y asistente de IA (procesamiento de comandos por texto y aprendizaje de preferencias).

Las pruebas incluyen:

- Validación técnica (unitarias y de integración)
- Test de usabilidad con usuarios reales
- Pruebas de carga para garantizar escalabilidad

Tecnologías clave: Java (backend), Bootstrap/HTML/CSS/JS (frontend), MySQL (datos), y modelos de DeepSeek para IA. El proceso sigue un enfoque ágil con iteraciones quincenales para lanzamientos progresivos y ajustes continuos.

Pruebas unitarias y de integración

1. Pruebas Unitarias

- **Objetivo:** Verificar que cada componente individual funciona correctamente.
- **Herramientas:** IntelliJ.
- **Ejemplos clave:**
 - Mensajería: Envío/recepción de mensajes.
 - IA: Reconocimiento preciso de comandos de voz/texto.
 - Autenticación: Validación de usuarios y tokens.

2. Pruebas de Integración

- **Objetivo:** Asegurar que los módulos funcionan juntos sin errores.
- **Herramientas:** Postman.
- **Ejemplos clave:**
 - Cuando un usuario inicia sesión se cargan sus chats o amistades/solicitudes pendientes.

Corrección de errores y optimización del rendimiento

Durante la fase de corrección, nos enfocamos en **errores críticos** (como fallos en comandos de IA o sincronización de datos) y **cuellos de botella** (lentitud en búsquedas o carga de mensajes). Las soluciones incluyen:

- **Parches inmediatos** para errores funcionales.
- **Pruebas comparativas** (A/B testing) para validar mejoras.

V. Documentación

Documentación técnica

Para la parte de registro tenemos en el frontend acceso al endpoint por POST donde se envían los datos por JSON y responde con una respuesta exitosa si todo fue bien o si algo paso:

```
const avatar = avatarInput.files[0];

const formData = new FormData();
formData.append('nombre', nombre);
formData.append('email', email);
formData.append('username', username);
formData.append('password', password);
formData.append('password2', password2);
formData.append('avatar', avatar);

fetch('http://localhost:8080/api/v1/auth/register', {
  method: 'POST',
  body: formData,
})
  .then(response => {
    if (!response.ok) {
      return response.json().then(errorData => {
        throw errorData;
      });
    }
    return response.json();
  })
  .then(data => {
    console.log('Registro exitoso:', data);
    mostrarMensaje('Registro exitoso', 'exito');
  })
  .catch(errorData => {
    console.error('Error en el registro:', errorData);
    mostrarErrores(errorData);
  });
};
```

y en el backend el código sería el siguiente, donde le indico que recibire una imagen indicándole el mediatype en el consumes, además hago validaciones para cada campo y recibo un DTO personalizado, el cual luego instancio un objeto de Usuario con los datos del DTO y ese objeto lo almaceno ya en la DB:

```
@Transactional no usages ± prueba9885
@PostMapping(value = "/api/v1/auth/register", consumes = MediaType.MULTIPART_FORM_DATA_VALUE)
public ResponseEntity<Map<String, String>> crearUsuario(@Valid @ModelAttribute UserRegisterDTO registroDTO, WebRequest request) {

    // Validar contraseñas (esto no está cubierto por tu ExceptionHandler)
    if (!registroDTO.getPassword().equals(registroDTO.getPassword2())) {
        return ResponseEntity.badRequest().body(Map.of( k1: "error", v1: "Las contraseñas no coinciden"));
    }

    // Guardar la imagen
    String nombreArchivo = guardarFotos(registroDTO.getAvatar());

    // Establecer el nombre del archivo en el request (para limpieza en caso de error)
    request.setAttribute( name: "nombreArchivo", nombreArchivo, WebRequest.SCOPE_REQUEST);

    // Intentar guardar el usuario (deja que el ExceptionHandler maneje DataIntegrityViolationException)
    Usuario usuario = Usuario.builder()
        .nombre(registroDTO.getNombre())
        .username(registroDTO.getUsername())
        .password(passwordEncoder.encode(registroDTO.getPassword()))
        .avatar(nombreArchivo)
        .email(registroDTO.getEmail())
        .build();

    userRepository.save(usuario); // Si falla, se lanzará DataIntegrityViolationException

    return ResponseEntity.status(HttpStatus.CREATED)
        .body(Map.of( k1: "message", v1: "Usuario creado exitosamente"));
}
```

En la parte del login en el frontend es casi lo mismo, acceso por POST a otro endpoint donde envío los datos por JSON, donde esa petición devolverá un token, y ese token lo añado como cookie al navegador:


```

fetch('http://localhost:8080/api/v1/auth/login', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify(data),
})
.then(response => {
  if (!response.ok) {
    return response.json().then(errorData => {
      throw errorData; // Lanzar los errores para manejarlos en el catch
    });
  }
  return response.json();
})
.then(data => {
  console.log('Login exitoso:', data);
  // Redirigir al usuario o realizar otras acciones
  document.cookie = `token=${data.token}; path=/; max-age=86400`; // Expira en 1 día
  window.location.href = "home.html";
})
.catch(errorData => {
  console.error('Error en el login:', errorData);
  if (errorData.message) {
    // Si hay un mensaje de error, mostrarlo
    mostrarMensaje(errorData.message, 'error');
  } else {
    // Mostrar un mensaje genérico de error
    mostrarMensaje('Error en el login', 'error');
  }
});

```

Y en la parte del backend recibimos un DTO de login donde generamos un token, y en base a esa autenticación, obtenemos al usuario, para enviar en la respuesta su id, username y el propio token generado, donde en el frontend ese token es el que añadimos en las cookies:

```

@PostMapping("/api/v1/auth/login") no usages ± prueba9865
public ResponseEntity<> crearTokenUsuario(@RequestBody @Valid LoginRequestDTO loginRequestDTO) {
  // Validamos al usuario en Spring (hacemos login manualmente)
  UsernamePasswordAuthenticationToken userPassAuthToken =
    new UsernamePasswordAuthenticationToken(loginRequestDTO.getUsername(), loginRequestDTO.getPassword());

  Authentication auth = authenticationManager.authenticate(userPassAuthToken); // Puede lanzar BadCredentialsException

  // Obtenemos el usuario autenticado
  Usuario user = (Usuario) auth.getPrincipal();

  // Generamos un token con los datos del usuario
  String token = this.tokenProvider.generateToken(auth);

  // Devolvemos un código 200 con el username y token JWT
  return ResponseEntity.ok(new LoginResponseDTO(user.getId(), user.getUsername(), token));
}

```

En la parte de Cerrar Sesión del usuario, el código es bastante simple, una simple función que es llamada cuando pulso en el botón, setea la cookie a nada y le pone que expira en una fecha diferente a la actual, en este caso uso la del 1 de enero de 1970, y después redirijo al propio index para que el usuario se vuelva a logear si quiere:

```
// Función para manejar el logout
function handleLogout() {
    // Elimina la cookie del token
    document.cookie = 'token=; Path=/; Expires=Thu, 01 Jan 1970 00:00:01 GMT;';
    // Redirige al login
    window.location.href = '/index.html';
}
```

Para la parte del perfil, concretamente en la de eliminar la cuenta, el botón hace una llamada a esta función donde hace una petición por el método DELETE a un determinado endpoint y si se elimina correctamente, redirige al index inicial:

```
function confirmarEliminacion() {
    if (!confirm('¿Estás seguro de que quieres eliminar tu cuenta? Esta acción no se puede deshacer.')) {
        return;
    }

    // Construimos la URL usando currentUserId
    const url = `http://localhost:8080/api/v1/usuario/delete/${currentUserId}`;

    fetch(url, {
        method: 'DELETE',
        headers: {
            'Authorization': `Bearer ${token}`,
            'Content-Type': 'application/json'
        }
    })
    .then(response => {
        if (!response.ok) {
            return response.json().then(err => Promise.reject(err));
        }
        // Si todo va bien, redirigimos
        window.location.href = 'index.html';
    })
    .catch(err => {
        console.error('Error eliminando la cuenta:', err);
        alert('Ha ocurrido un error al eliminar tu cuenta. Inténtalo de nuevo más tarde.');
```

En la parte de backend no hay mucha complicación, recibo un id, se lo paso a una función de mi servicio que se encarga primero de eliminar el avatar, para que en el servidor se elimine su foto, y después ya elimino el usuario de la DB, y en el caso de que la imagen no exista o haya habido algún error, quiero que el usuario aun así se elimine también, por eso en el catch también lo elimino:

```
@DeleteMapping("/api/v1/usuario/delete/{usuarioId}") no usages prueba9865
public ResponseEntity<?> eliminarUsuario(@PathVariable Long usuarioId){
    return usuarioService.eliminarUsuario(usuarioId);
}
```

```

public ResponseEntity<Map<String,String>> eliminarUsuario(Long usuarioId) {
    // Verificar si el usuario existe
    Optional<Usuario> usuarioOptional = usuarioRepository.findById(usuarioId);
    if(!usuarioOptional.isPresent()) {
        return ResponseEntity.ok(Map.of( k1: "error", v1: "El ID no existe"));
    }

    Usuario usuario = usuarioOptional.get();

    try {
        // Eliminar el avatar si existe
        String avatarPath = usuario.getAvatar();
        if(avatarPath != null && !avatarPath.isEmpty()) {
            // Extraer el nombre del archivo de la URL
            String fileName = avatarPath.substring( beginIndex: avatarPath.lastIndexOf( str: "/" ) + 1 );

            // Construir la ruta completa en el servidor
            Path filePath = Paths.get( first: "uploads/avatars" ).resolve(fileName).toAbsolutePath();

            // Eliminar el archivo
            Files.deleteIfExists(filePath);
        }

        // Eliminar el usuario
        usuarioRepository.deleteById(usuarioId);

        return ResponseEntity.ok(Map.of( k1: "success", v1: "Usuario eliminado correctamente"));
    } catch (IOException e) {
        // Si hay error al eliminar el archivo, igual eliminamos el usuario
        usuarioRepository.deleteById(usuarioId);
        return ResponseEntity.ok(Map.of( k1: "warning", v1: "Usuario eliminado pero no se pudo borrar su avatar: " + e.getMessage()));
    }
}

```

En la parte de editar es algo mas compleja, ya que mi idea era que el usuario pueda editar los campos que el quiera, no que este obligado a editar todo o a enviar todo para que la edicion se complete, en este caso instancio un objeto formdata, el cual si en uno de los campos tiene contenido, le añado a ese objeto el valor de ese determinado campo, asi sucesivamente con todos, luego ese objeto “fd” lo envio al endpoint por PUT (es cierto que quizia deberia usar PATCH ya que no siempre voy a enviar todos los datos, podria ser mas optimo usarlo), si todo fue bien muestro un mensaje de confirmacion, seteo una nueva cookie con los nuevos valores del usuario y vuelvo a mostrar la imagen en el caso de que haya sido cambiada:

```

const fd = new FormData();

// Solo agregar campos que están siendo editados
if (editandoNombre) fd.append('nombre', nombre);
if (editandoEmail) fd.append('email', email);
if (editandoUsername) fd.append('username', username);
if (cambiandoPassword) {
  fd.append('antiguaPassword', antiguaPassword);
  fd.append('password', password);
}
if (cambiandoAvatar) fd.append('avatar', avatarInput.files[0]);

fd.append('tipo', 'local');

const response = await fetch(`http://localhost:8080/api/v1/usuario/edit/${currentUserId}`, {
  method: 'PUT',
  headers: {
    'Authorization': `Bearer ${token}`
  },
  body: fd
});

if (!response.ok) {
  const errorData = await response.json();
  throw new Error(errorData.message || 'Error al actualizar');
}

const data = await response.json();

// Actualizar el token si viene en la respuesta
if (data.token) {
  document.cookie = `token=${data.token}; path=/; max-age=${24 * 60 * 60}`;
  token = data.token;
}

// Actualizar avatar si hubo cambio
if (data.avatarUrl) {
  const imagenAvatar = document.getElementById('imagenAvatar');
  imagenAvatar.style.backgroundImage = `url('${data.avatarUrl}?t=${new Date().getTime()}')`;
}

mostrarMensaje(data.success || "Cambios guardados correctamente", "exito");
resetFormToLockedState();
await initializeUser();
} catch (error) {

```

En el backend recibo el id del usuario en la URL, en el body recibo el nuevo usuario (en DTO) y la propia petición y su respectiva autenticación, hago ciertas validaciones como casi siempre para verificar si la imagen está en la petición o si el nombre o el email están enviados correctamente, luego actualizo el usuario con el método del servicio el cual obtiene el usuario actual, y en base a ese usuario vamos seteando sus respectivos valores en función de si le he pasado contenido a algún determinado campo, después al final lo guardo en la DB, y al final del controlador genero ese nuevo token el cual recibía en el frontend y lo seteaba de nuevo, y lo envío en la respuesta junto a esa posible futura nueva imagen, en el caso de que no haya nueva imagen, envía la que ya había antes:

```
@PutMapping(value = "/api/v1/usuario/edit/{usuarioId}", consumes = MediaType.MULTIPART_FORM_DATA_VALUE) no usages prueba9865
public ResponseEntity<?> actualizarUsuarioParcial(
    @PathVariable Long usuarioId,
    @ModelAttribute UsuarioEditDTO dto,
    BindingResult result,
    WebRequest request,
    Authentication authentication) {

    // Manejar la imagen si viene en la petición
    String nombreArchivo = null;
    if (dto.getAvatar() != null && !dto.getAvatar().isEmpty()) {
        nombreArchivo = guardarFotos(dto.getAvatar());
        request.setAttribute("nombreArchivo", nombreArchivo, WebRequest.SCOPE_REQUEST);
    }

    // Validar solo campos que vienen en el DTO
    if (dto.getNombre() != null) {
        if (dto.getNombre().isBlank()) {
            return ResponseEntity.badRequest().body(Map.of("error", "El nombre no puede estar vacío"));
        }
    }

    if (dto.getEmail() != null) {
        // Validar formato email
        if (!dto.getEmail().matches(regex: "^[A-Za-z0-9+_.-]+@[A-Za-z0-9.-]+\\.([A-Za-z]{2,6}$)")) {
            return ResponseEntity.badRequest().body(Map.of("error", "Formato de email inválido"));
        }
    }

    // Actualizar usuario (solo campos proporcionados)
    Usuario usuarioActualizado = usuarioService.actualizarUsuarioParcial(usuarioId, dto, nombreArchivo);

    // Generar nuevo token
    String nuevoToken = jwtTokenProvider.generateToken(authentication);

    // Construir respuesta
    Map<String, Object> response = new HashMap<>();
```

```

public Usuario actualizarUsuarioParcial(Long usuarioId, UsuarioEditDTO dto, String nombreArchivo) {
    Usuario usuario = usuarioRepository.findById(usuarioId)
        .orElseThrow(() -> new EntityNotFoundException("Usuario no encontrado"));

    // Actualizar solo los campos que vienen en el DTO
    if (dto.getNombre() != null) {
        usuario.setNombre(dto.getNombre());
    }

    if (dto.getEmail() != null) {
        usuario.setEmail(dto.getEmail());
    }

    if (dto.getUsername() != null) {
        usuario.setUsername(dto.getUsername());
    }

    if (nombreArchivo != null) {
        usuario.setAvatar(nombreArchivo);
    }

    // Manejo especial para contraseña
    if (dto.getPassword() != null && !dto.getPassword().isEmpty()) {
        if (!passwordEncoder.matches(dto.getAntiguaPassword(), usuario.getPassword())) {
            throw new SecurityException("La contraseña actual no es correcta");
        }
        usuario.setPassword(passwordEncoder.encode(dto.getPassword()));
    }

    return usuarioRepository.save(usuario);
}

```

```

// Construir respuesta
Map<String, Object> response = new HashMap<>();
response.put("success", "Usuario actualizado correctamente");
response.put("token", nuevoToken);

if (nombreArchivo != null) {
    response.put("avatarUrl", "http://localhost:8080/uploads/avatars/" + usuarioActualizado.getAvatar());
}

return ResponseEntity.ok(response);
}

```

En la parte del chat normal, cuando voy a enviar el mensaje es una simple funcion que es llamada desde el boton de enviar, donde hace una peticion por POST hacia un endpoint y lo envio por JSON de nuevo, ademas creo el nuevo div que almacenara el contenido del mensaje, indicandole que el mensaje es mio, asi lo represento de otro color, a diferencia de si el mensaje es “their”, le añado la hora y la paleta de comandos de edicion y borrado de mensaje:

```
try {
  const res = await fetch(
    `http://localhost:8080/api/user/${currentUser}/${currentContactId}/messages`,
    {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(dto)
    }
  );
  if (!res.ok) throw new Error(res.status);
  const nuevo = await res.json();

  // --- CREACIÓN DEL DIV COMO EN loadChatMessages ---
  const div = document.createElement('div');
  div.className = 'message mine';
  div.dataset.id = nuevo.id;

  // Contenido
  const content = document.createElement('p');
  content.textContent = nuevo.contenido;
  div.appendChild(content);

  // Hora
  const time = document.createElement('span');
  time.className = 'message-time';
  time.textContent = new Date(nuevo.createdAt).toLocaleTimeString();
  div.appendChild(time);

  // Actions container
  const actions = document.createElement('div');
  actions.className = 'message-actions';

  // Edit button
  const editBtn = document.createElement('div');
  editBtn.className = 'edit-btn';
  editBtn.title = 'Editar mensaje';
  const editIcon = document.createElement('i');
  editIcon.className = 'fas fa-pencil-alt';
  editBtn.appendChild(editIcon);
```

Ademas otro punto importante del proyecto en el frontend son las variables “currentUserId” donde hago una peticion a un endpoint cocreto al cual siempre tengo acceso al ID del usuario actual que decodrea el JWT que envio y construye una respuesta con los datos del usuario, ademas de su ID, por eso luego ese ID se lo paso a la variable “currentUserId”, un dato importante para que no guarde cache en cada peticion es enviar la cabecera en todo momento “Cache-Control: ‘no-cache’”:

```
// Función para obtener y establecer el ID del usuario
const initializeUser = async () => {
  try {
    const response = await fetch("http://localhost:8080/decode-jwt", {
      method: "GET",
      headers: {
        'Authorization': `Bearer ${token}`,
        'Cache-Control': 'no-cache',
        'Content-Type': 'application/json'
      },
      credentials: 'include'
    });

    if (!response.ok) {
      throw new Error(await response.text());
    }

    const data = await response.json();
    currentUserId = data.id;
    console.log("Usuario ID obtenido:", currentUserId);
  }
}
```

```
public ResponseEntity<?> decodeJwt(@RequestHeader(value = "Authorization", required = false) String authHeader) {
    return ResponseEntity.status(HttpStatus.UNAUTHORIZED)
        .body(Map.of(k1: "error", v1: "Authorization header missing or invalid"));
}

String jwtToken = authHeader.substring(beginIndex: 7); // Eliminar "Bearer "

// Verificar que el token no esté vacío
if (jwtToken.isEmpty()) {
    return ResponseEntity.status(HttpStatus.UNAUTHORIZED)
        .body(Map.of(k1: "error", v1: "Token is empty"));
}

SecretKey key = jwtTokenProvider.claveFirma();

// Decodificar y verificar el token
Claims claims = Jwts.parser() JwtParserBuilder
    .verifyWith(key)
    .build() JwtParser
    .parseSignedClaims(jwtToken) Jws<Claims>
    .getPayload();

// Verificar claims esenciales
if (claims.get("id") == null) {
    return ResponseEntity.status(HttpStatus.UNAUTHORIZED)
        .body(Map.of(k1: "error", v1: "Invalid token claims"));
}

// Construir respuesta
Map<String, Object> response = new HashMap<>();
response.put("avatar", "http://localhost:8080/uploads/avatars/" + claims.get(s: "avatar", String.class));
response.put("id", claims.get("id"));
response.put("nombre", claims.get("name"));
response.put("email", claims.get("email"));
response.put("username", claims.get("username"));

return ResponseEntity.ok(response);
} catch (ExpiredJwtException e) {
```


Y la variable “currentContactId” la seteo en este punto, donde inicio el chat:

```
async function startChatWithContact(contacto) {
  currentContactId = contacto.id;

  // Detener el intervalo anterior si existe
  if (chatRefreshInterval) {
    clearInterval(chatRefreshInterval);
  }

  // Cargar mensajes inmediatamente
  await loadChatMessages();

  // Configurar intervalo para refrescar mensajes cada 5 segundos
  chatRefreshInterval = setInterval(async () => {
    await loadChatMessages();
  }, 5000); // 5000 ms = 5 segundos

  document.querySelector('.chat-list-view').classList.remove('active');
  document.querySelector('.chat-detail-view').classList.add('active');

  const chatHeader = document.querySelector('.chat-header');
  const avatarUrl = contacto.avatar
    ? `http://localhost:8080/uploads/avatars/${contacto.avatar}`
    : `https://www.gravatar.com/avatar/default?s=200`;
  chatHeader.querySelector('.profile-pic').src = avatarUrl;
  chatHeader.querySelector('.chat-name').textContent = contacto.username || 'Usuario sin nombre';
}
```

Y esta funcion es llamada en cada contacto que itera en esta funcion:

```
// Función para renderizar la lista de contactos
function renderContactList(contactos) {
  const chatList = document.getElementById("chatList");
  chatList.innerHTML = '';

  if (!contactos || contactos.length === 0) {
    chatList.innerHTML = `
    <div class="no-contacts">
      No tienes contactos aún. Agrega amigos para chatear.
    </div>
  `;
    return;
  }

  contactos.forEach(contacto => {
    const contactItem = document.createElement('div');
    contactItem.className = 'chat-item';
    contactItem.innerHTML = `
    
      class="chat-avatar">
    <div class="chat-info">
      <span class="chat-name">${contacto.nombre}</span>
      <span class="chat-last-msg">@${contacto.username}</span>
    </div>
  `;

    // Evento para iniciar chat
    contactItem.addEventListener('click', () => {
      startChatWithContact(contacto);
    });

    chatList.appendChild(contactItem);
  });
}
```

Y la lista de contactos la carga de un endpoint que tengo concreto que me muestra todos los contactos que tiene un determinado usuario, estas 3 funciones son las encargadas de que la parte del chat con el contacto funcione perfectamente y tenga acceso en todo momento a la información de los 2 usuarios, tanto del emisor como del remitente:

```
// Función para cargar los contactos del usuario
async function loadUserContacts() {
  if (!currentUserId) {
    console.error("ID de usuario no disponible");
    return;
  }

  try {
    const chatList = document.getElementById("chatList");
    chatList.innerHTML = '<div class="loading">Cargando contactos...</div>';

    console.log(currentUserId)

    const response = await fetch(`http://localhost:8080/usuarios/${currentUserId}/contactos`);

    if (!response.ok) {
      throw new Error(`Error HTTP: ${response.status}`);
    }

    const contactos = await response.json();
    console.log(contactos)
    renderContactList(contactos);
  } catch (error) {
    console.error("Error al cargar contactos:", error);
    const chatList = document.getElementById("chatList");
    chatList.innerHTML = '<div class="error-msg">Error al cargar contactos</div>';
  }
}
```

En el backend recibo el ID del usuario actual y el ID del contacto hacia el que envío el mensaje, además del propio mensaje en el body de la petición, todo se realiza mediante el método del servicio el cual obtiene el usuario y el contacto, y crea un objeto mensaje con los datos correspondientes (el usuario y el contacto entre otros y el tipo de mensaje, ya que en el futuro la idea es poder enviar videos, imágenes y más cosas a parte de texto) y lo guarda en la DB:

```
@PostMapping("/api/user/{usuarioId}/{contactoId}/messages") no usages prueba9865
public ResponseEntity<Mensaje> postMensaje(
    @PathVariable Long usuarioId,
    @RequestBody MensajeDTO dto) {
    Mensaje creado = service.enviarMensaje(usuarioId, dto);
    return ResponseEntity.status(HttpStatus.CREATED).body(creado);
}
```

```

public Mensaje enviarMensaje(Long usuarioId, MensajeDTO dto) { 1 usage  prueba9865
    Usuario u = userRepo.findById(dto.getUsuarioId())
        .orElseThrow(() -> new EntityNotFoundException("Usuario/emisor no existe"));
    Usuario c = userRepo.findById(dto.getContactoId())
        .orElseThrow(() -> new EntityNotFoundException("Remitente no existente"));

    Mensaje m = Mensaje.builder()
        .usuario(u)
        .contacto(c)
        .contenido(dto.getContenido())
        .tipo(dto.getTipo())
        .build();
    return repo.save(m);
}

```

En la parte de borrar el mensaje llamo a una funcion flecha que hace una simple peticion DELETE aun endpoint enviando el ID del mensaje a borrar:

```

deleteBtn.addEventListener('click', async e => {
    e.stopPropagation();
    if (confirm("¿Estás seguro que quieres eliminar este mensaje?")) {
        try {
            const delRes = await fetch(
                `http://localhost:8080/api/messages/${nuevo.id}`,
                { method: 'DELETE' }
            );
            if (!delRes.ok) throw new Error(delRes.status);
            div.remove();
        } catch (err) {
            console.error('Error eliminando mensaje:', err);
            alert('No se pudo eliminar el mensaje.');
```

Y en la parte del backend tengo el controlador al que llamo a una funcion de mi servicio la cual envia el ID que recibe, obtiene el mensaje y lo borra:

```

@DeleteMapping("/api/messages/{messageId}") no usages  prueba9865
public ResponseEntity<?> eliminarMensaje(@PathVariable Long messageId){
    return service.eliminarMensaje(messageId);
}

```

```

public ResponseEntity<Map<String,String>> eliminarMensaje(Long messageId) { 1 usage  prueba9865
    // Verificar si el usuario existe
    Optional<Mensaje> mensajeOptional = repo.findById(messageId);
    if(!mensajeOptional.isPresent()) {
        return ResponseEntity.ok(Map.of( k1: "error", v1: "El ID no existe"));
    }

    Mensaje mensaje = mensajeOptional.get();

    repo.deleteById(messageId);
    return ResponseEntity.ok(Map.of( k1: "success", v1: "Mensaje eliminado correctamente"));
}

```

En la parte de editar el mensaje tenemos una llamada a una funcion que tengo creada, a la cual le paso el mensaje y el contenido de todo el parrafo donde se almacena el mensaje (a nivel de HTML) y esa funcion me abre un modal donde me sale una zona para escribir y un boton de guardar y cancelar, cuando el doy a guardar hace una peticion a un endpoint usando PATCH donde le envio solamente el mensaje nuevo por el que quiero sustituir y actualizo el DOM para que el cambio se vea instantaneamente aplicado:

```

// Botón Editar
const editBtn = document.createElement('div');
editBtn.className = 'edit-btn';
editBtn.title = 'Editar mensaje';
const editIcon = document.createElement('i');
editIcon.className = 'fas fa-pencil-alt';
editBtn.appendChild(editIcon);
editBtn.addEventListener('click', (e) => {
    e.stopPropagation();
    openEditModal(m, content); // Pasamos el mensaje y el elemento del contenido
});
actions.appendChild(editBtn);

```

```

function openEditModal(message, contentElement) {
  const closeBtn = document.querySelector('.close-modal');
  const cancelBtn = document.getElementById('cancelEdit');
  const saveBtn = document.getElementById('saveEdit');

  // Rellena el textarea con el mensaje actual
  textarea.value = message.contenido;
  modal.style.display = 'flex';

  // Cierra el modal
  const closeModal = () => {
    modal.style.display = 'none';
  };

  // Event listeners
  closeBtn.onclick = closeModal;
  cancelBtn.onclick = closeModal;

  saveBtn.onclick = async () => {
    const nuevoTexto = textarea.value.trim();
    if (nuevoTexto !== '' && nuevoTexto !== message.contenido) {
      try {
        const patchRes = await fetch(`http://localhost:8080/api/messages/${message.id}`, {
          method: 'PATCH',
          headers: { 'Content-Type': 'application/json' },
          body: JSON.stringify({ contenido: nuevoTexto })
        });
        if (!patchRes.ok) throw new Error(patchRes.status);

        // Actualiza tanto el DOM como el objeto original
        contentElement.textContent = nuevoTexto;
        message.contenido = nuevoTexto;

        closeModal();
      } catch (err) {
        console.error('Error editando mensaje:', err);
        alert('No se pudo editar el mensaje.');
```

A nivel de backend recibo el ID del mensaje, obtengo el mensaje completo, le seteo el contenido por el nuevo que envio en el cuerpo de la peticion, y lo guardo en la DB:

```
@PatchMapping("/api/messages/{messageId}") no usages prueba9865
public ResponseEntity<Mensaje> updateContenido(
    @PathVariable Long messageId,
    @RequestBody ActualizarMensajeDTO request
) {
    // Llama al servicio para actualizar solo el contenido
    Mensaje updated = service.updateContenido(messageId, request.getContenido());
    return ResponseEntity.ok(updated);
}
```

```
public Mensaje updateContenido(Long id, String nuevoContenido) { 1 usage prueba9865
    Mensaje msg = repo.findById(id)
        .orElseThrow(() -> new EntityNotFoundException("Mensaje no encontrado: " + id));
    msg.setContenido(nuevoContenido);
    return repo.save(msg);
}
```

Y por ultimo en la parte de la IA simplemente llamo a una funcion cuando pulso en el boton de enviar, y hago una peticion POST hacia un endpoint en mi servidor, ademas pasandole el ID de usuario actual, para asignarle a ese usuario el mensaje que envie:

```

async function sendAIMessage() {
  const input = document.getElementById('aiMessageInput');
  const message = input.value.trim();

  if (!message) return;

  // Mostrar el mensaje del usuario
  addAIMessage(message, 'sent');
  input.value = '';

  try {
    // Llamada a tu API
    const response = await fetch(`http://localhost:8080/api/chat/${currentUser.id}`, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ contenido: message })
    });

    if (!response.ok) {
      throw new Error(`Error en la respuesta: ${response.status}`);
    }

    // Obtener el texto completo de la respuesta
    let text = await response.text();

    // Eliminar todas las apariciones de "***Respuesta:***"
    text = text.replace(/\*\*Respuesta:\*\*/g, '').trim();

    // Mostrar la respuesta de la IA en el chat
    addAIMessage(text, 'received');
  } catch (err) {
    console.error('Error enviando el mensaje:', err);
    addAIMessage('¡Ups! Hubo un error al comunicarme con el servidor.', 'received');
  }
}

```

En el backend simplemente recibo un DTO donde esta el mensaje que le envío a la IA y el ID del usuario, luego en el servicio guardo el mensaje con su respectivo usuario en la DB, luego hago una petición hacia una URL la cual es la API para poder procesar una respuesta (es importante en la API indicarle que rol quieres que tenga el asistente virtual, dependiendo del rol que le des, te responderá a unas cosas u a otras), la URL y la Key la obtengo de mi archivo .properties, una vez tenga la respuesta la retorno, y esa es la respuesta que muestro desde el frontend:

```

@RestController  no usages  prueba9865
@RequestMapping("/api")
@CrossOrigin(origins = "*")
public class IAController {

  private final OpenRouterService svc; 2 usages

  public IAController(OpenRouterService svc) { this.svc = svc; }

  @PostMapping(path = "/chat/{usuarioId}", consumes = MediaType.APPLICATION_JSON_VALUE)  no usages  prueba9865
  public ResponseEntity<String> chat(
    @RequestBody ChatRequestDTO req, // Usa el DTO en lugar de la entidad
    @PathVariable Long usuarioId
  ) {
    return ResponseEntity.ok(svc.chat(req.getContenido(), usuarioId));
  }
}

```



```

public String chat(String userMessage, Long usuarioId) { 1 usage 1 prueba9865
    // 1. Obtener el usuario completo desde la base de datos
    Usuario usuario = usuarioRepository.findById(usuarioId)
        .orElseThrow(() -> new RuntimeException("Usuario no encontrado"));

    // 2. Crear y guardar el mensaje en tu base de datos
    ChatIA mensaje = ChatIA.builder()
        .contenido(userMessage)
        .usuario(usuario)
        .build();
    chatIARepository.save(mensaje);

    // 3. Llamar a la API de IA (tu lógica existente)
    HttpHeaders headers = new HttpHeaders();
    headers.setBearerAuth(apiKey);
    headers.setContentType(MediaType.APPLICATION_JSON);

    Map<String, Object> body = Map.of(
        k1: "model", v1: "deepseek/deepseek-r1:free",
        k2: "messages", List.of(
            Map.of(k1: "role", v1: "system", k2: "content", v2: "Eres un asistente servicial."),
            Map.of(k1: "role", v1: "user", k2: "content", userMessage)
        )
    );

    HttpEntity<Map<String, Object>> req = new HttpEntity<>(body, headers);
    ResponseEntity<Map> resp = rt.postForEntity(apiUrl, req, Map.class);

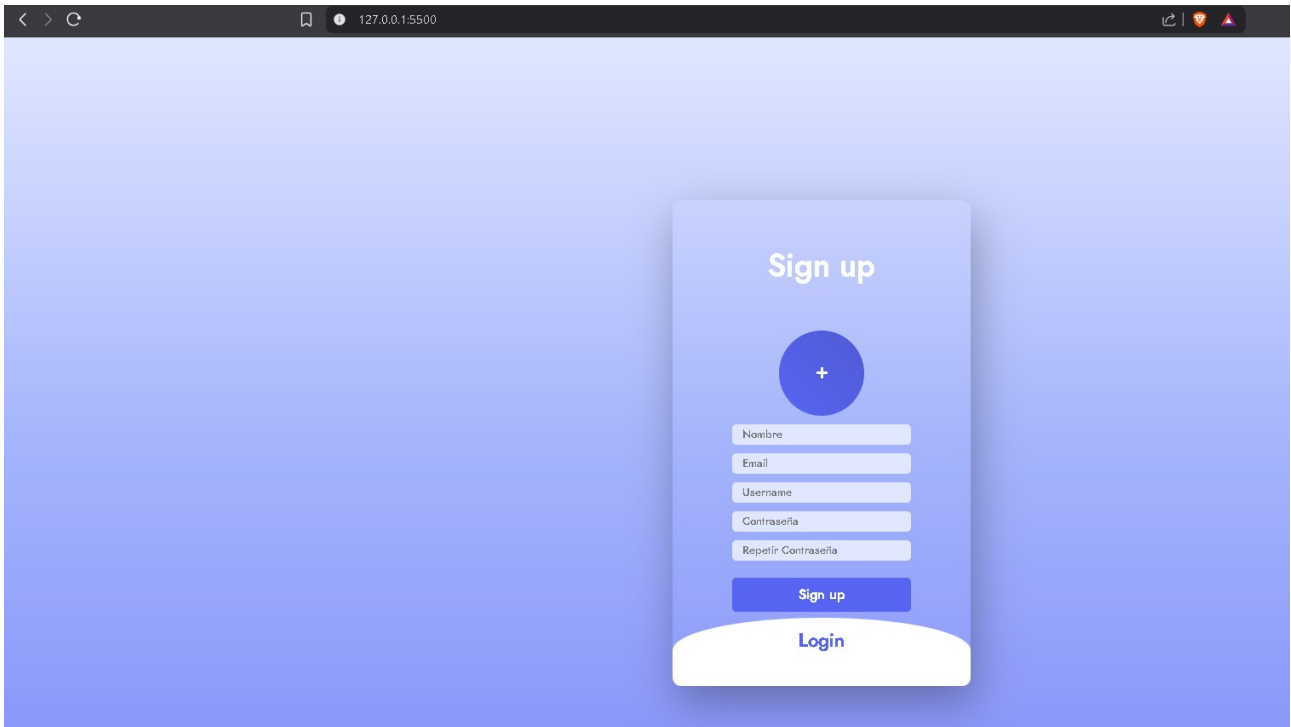
    // Procesar respuesta...
    List<?> choices = (List<?>) resp.getBody().get("choices");
    Map<?, ?> first = (Map<?, ?>) choices.get(0);
    Map<?, ?> message = (Map<?, ?>) first.get("message");

    return (String) message.get("content");
}

```

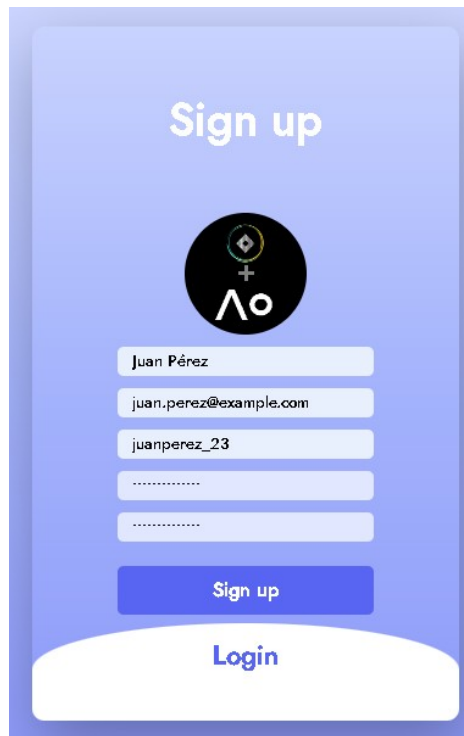

Documentacion de usuario

El flujo normal de un usuario seria el siguiente, primero el usuario ve el siguiente panel:

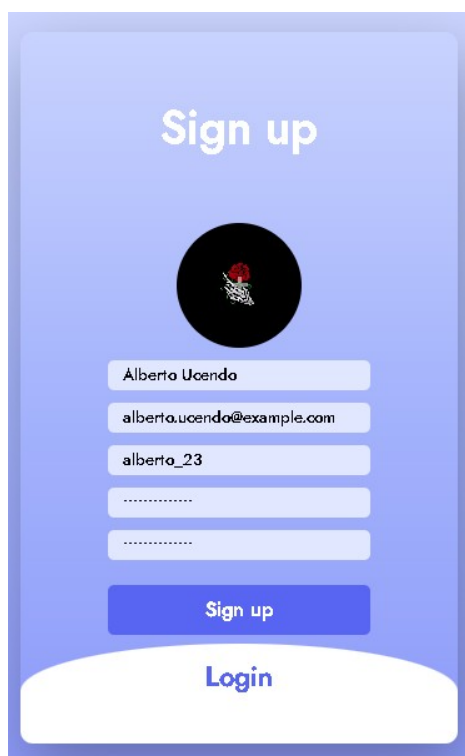


A screenshot of a web browser displaying a 'Sign up' form. The browser's address bar shows '127.0.0.1:5500'. The form is centered on a blue gradient background. It features a white rounded rectangle containing the title 'Sign up' in bold. Below the title is a blue circular icon with a white plus sign. The form includes five input fields: 'Nombre', 'Email', 'Username', 'Contraseña', and 'Repetir Contraseña'. A blue 'Sign up' button is positioned below the input fields, and a white 'Login' button is at the bottom of the white container.


El usuario se registra (registramos en este caso 2 usuarios para pruebas):



A screenshot of the 'Sign up' form with user data entered. The form is centered on a blue gradient background. It features a black circular icon with a white plus sign and the text 'Ao'. The input fields contain the following text: 'Juan Pérez', 'juan.perez@example.com', 'juanperez_23', and two empty fields with dots indicating a password. A blue 'Sign up' button is positioned below the input fields, and a white 'Login' button is at the bottom of the white container.



Sign up



Alberto Uendo

alberto.ucendo@example.com

alberto_23

.....

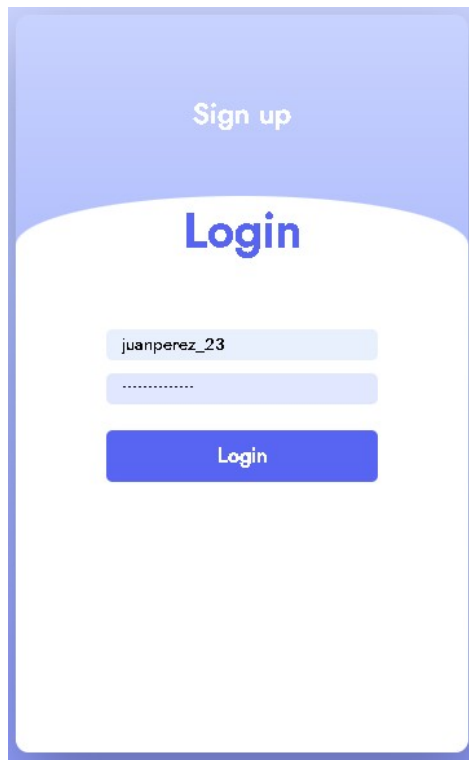
.....

Sign up

Login

This is a sign-up form with a light blue background. At the top, it says 'Sign up' in white. Below that is a circular profile picture placeholder with a red flower icon. There are five input fields: a name field with 'Alberto Uendo', an email field with 'alberto.ucendo@example.com', a username field with 'alberto_23', and two password fields with dots. A blue 'Sign up' button is below the password fields, and a white 'Login' button is at the bottom.

Despues nos logueamos y podremos ver el dashboard principal:



Sign up

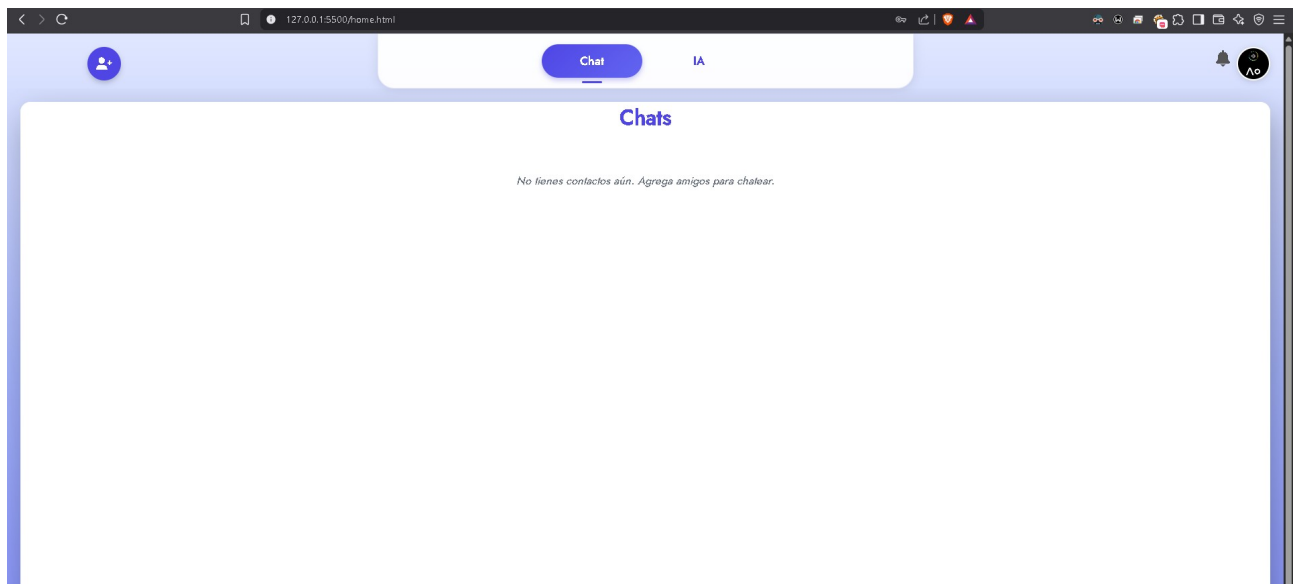
Login

juanperez_23

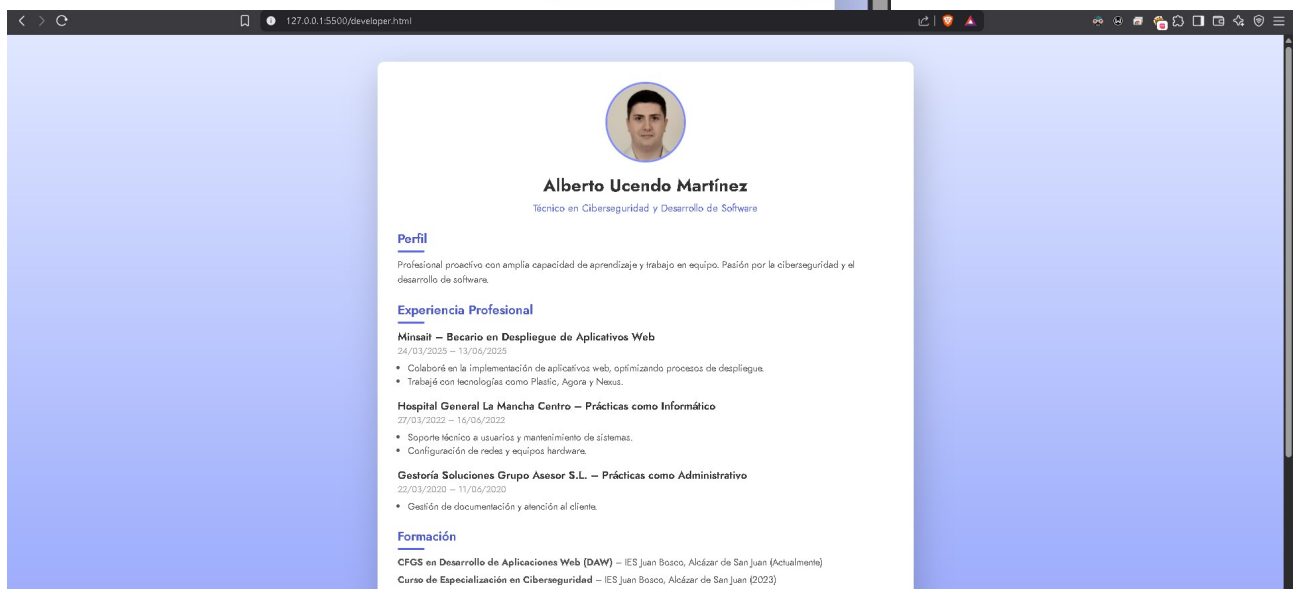
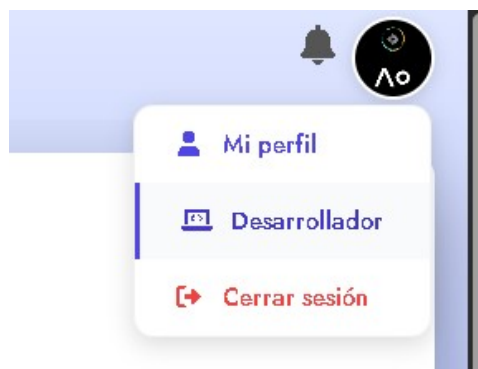
.....

Login

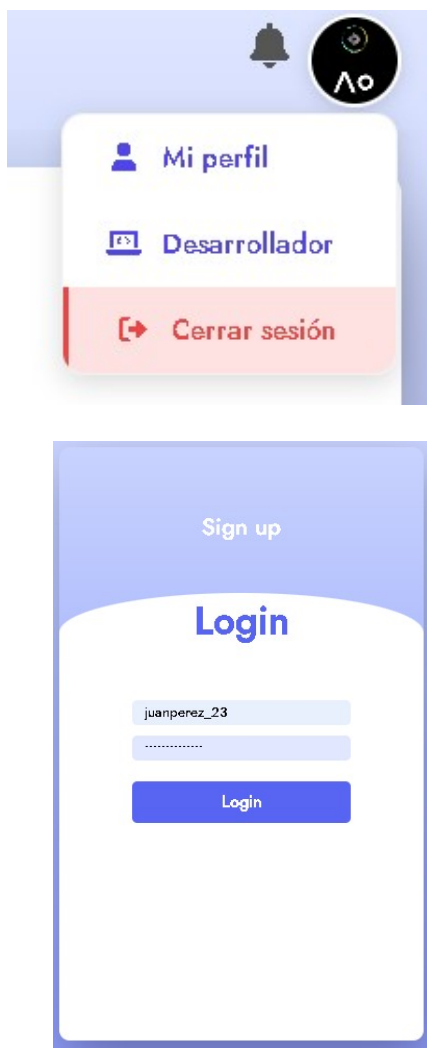
This is a login form with a light blue background. At the top, it says 'Sign up' in white. Below that is a white rounded rectangle containing the word 'Login' in blue. There are two input fields: a username field with 'juanperez_23' and a password field with dots. A blue 'Login' button is below the password field.



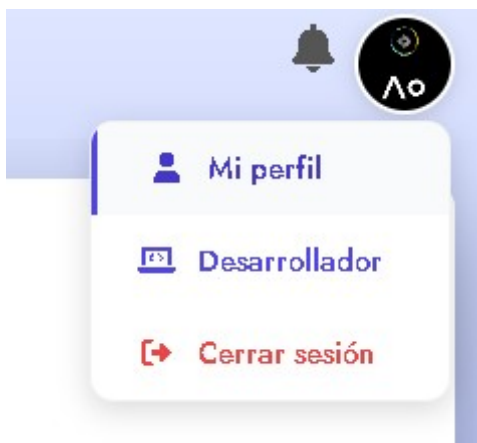
Primero de todo si hacemos hover sobre nuestra foto veremos una serie de opciones, por ejemplo si nos vamos a “Desarrollador” veremos el CV del creador del aplicativo:

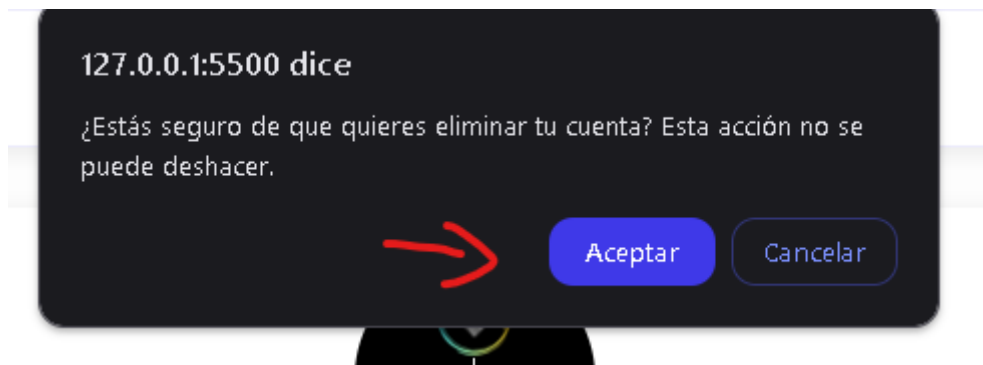
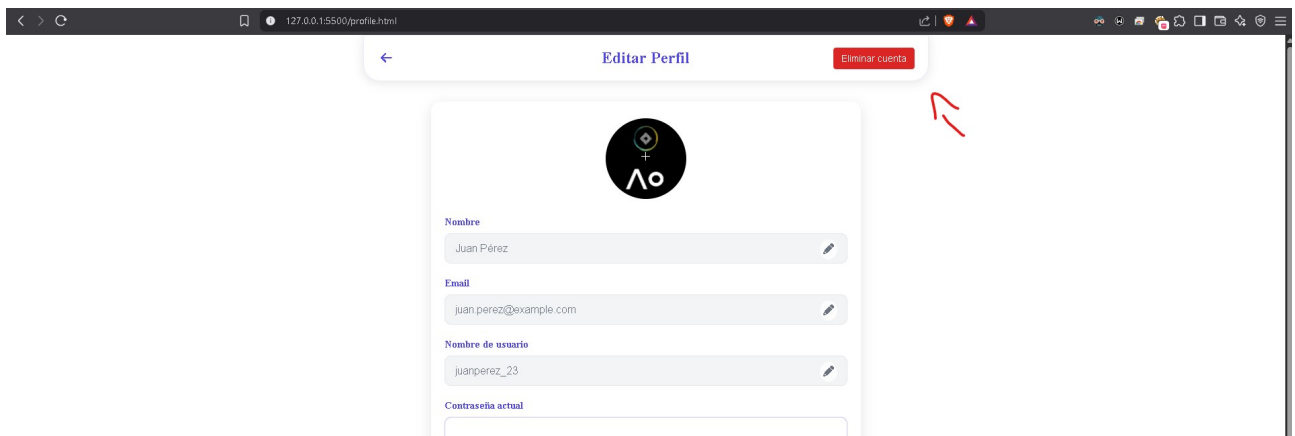


Si le damos a cerrar sesion podremos eliminar la cookie sin problema, nos redirige al panel de login de nuevo:

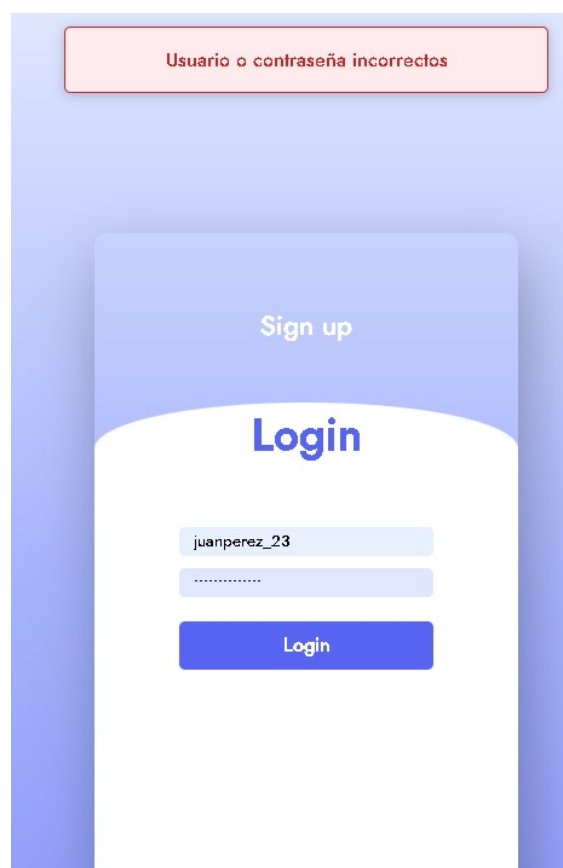


Y si nos vamos a la opcion de “Mi perfil” podremos ver todo nuestros datos por si queremos eliminar nuestra cuenta por ejemplo:

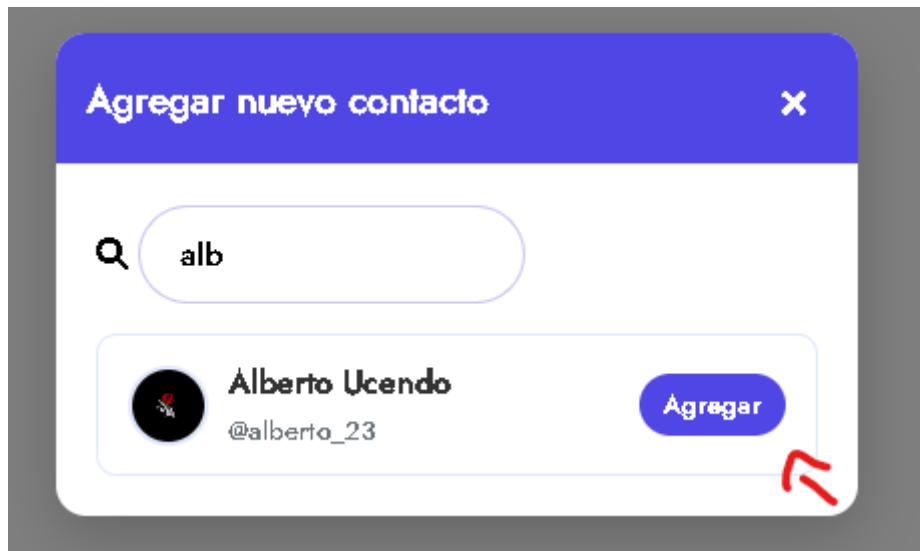
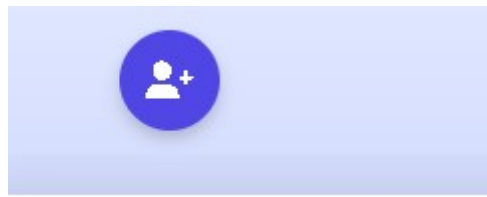




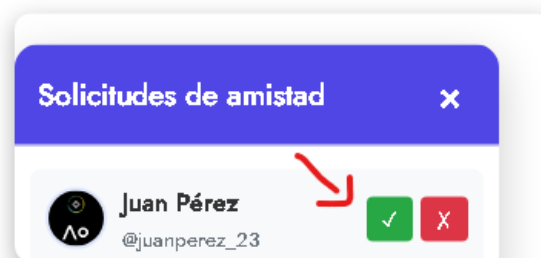
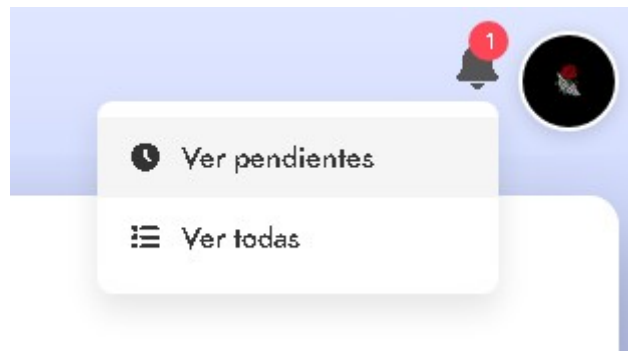
Y me redirige al panel principal, donde si nos volvemos a intentar logear, nos va a dar error ya que la cuenta no existe:

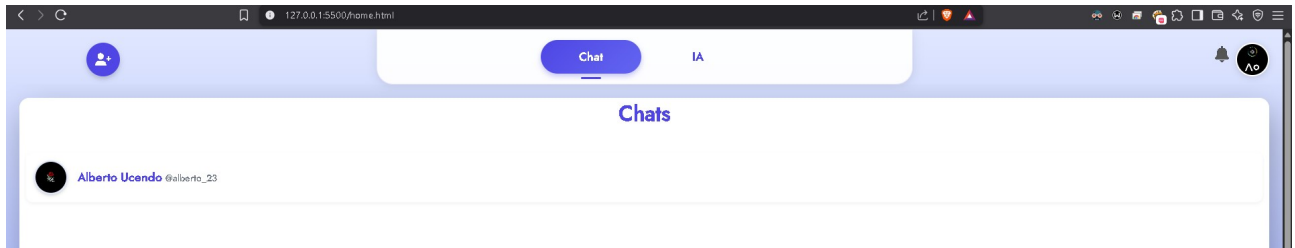
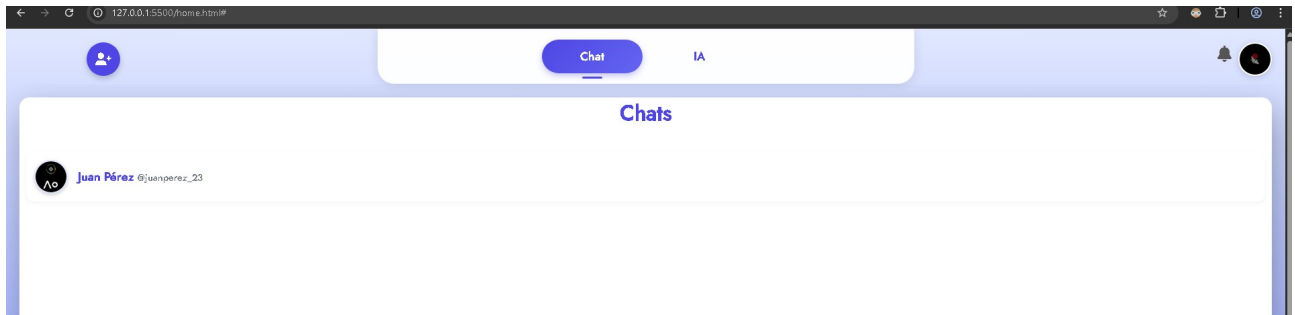


Si le damos al boton de agregar a contactos, podemos buscar el usuario al que queramos y darle al boton de “Agregar”:

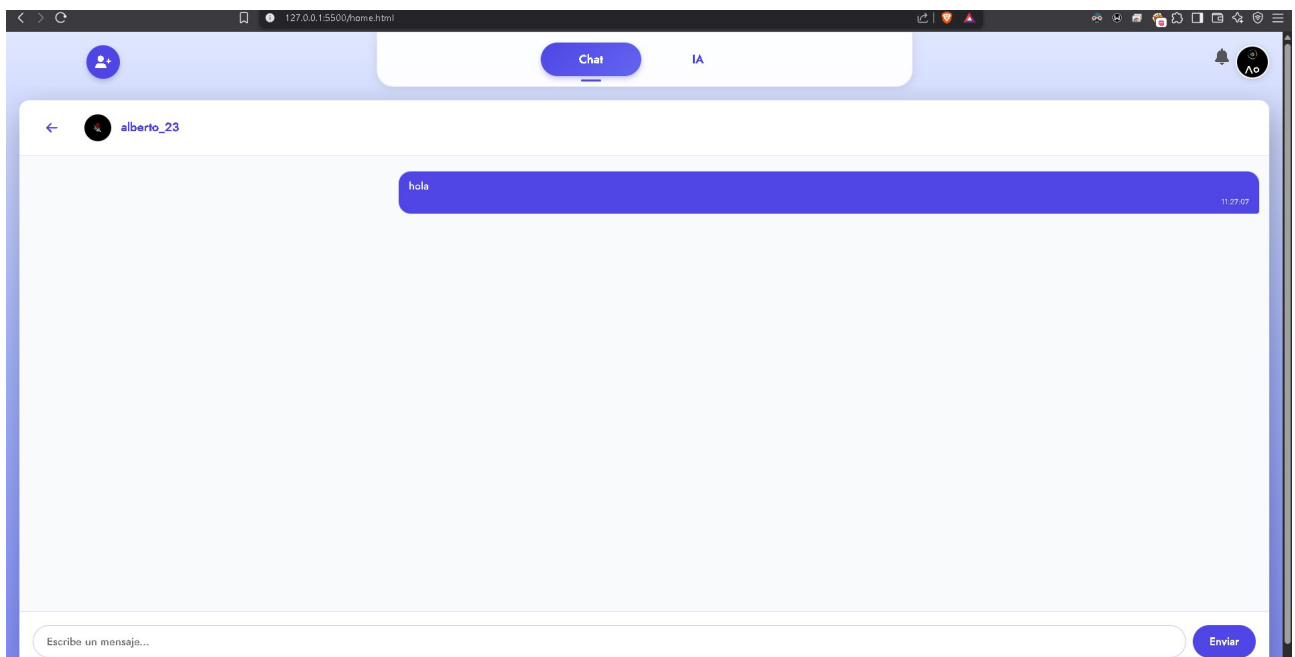


Si nos vamos al otro usuario, veremos que en la campana sale un “1”, donde si nos vamos a las solicitudes pendientes, podremos aceptar o rechazar la solicitud, en este caso la aceptamos, y si recargamos la pagina en ambos usuarios, podremos ver los chats ya disponibles para poder escribir:

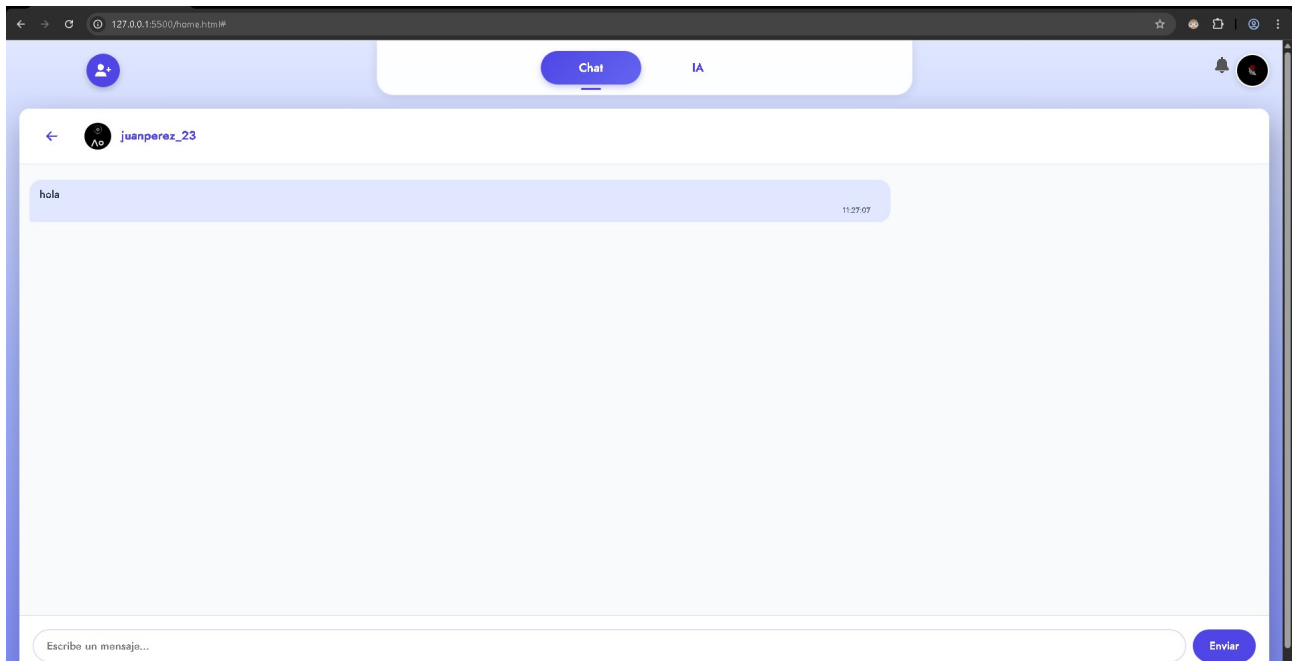




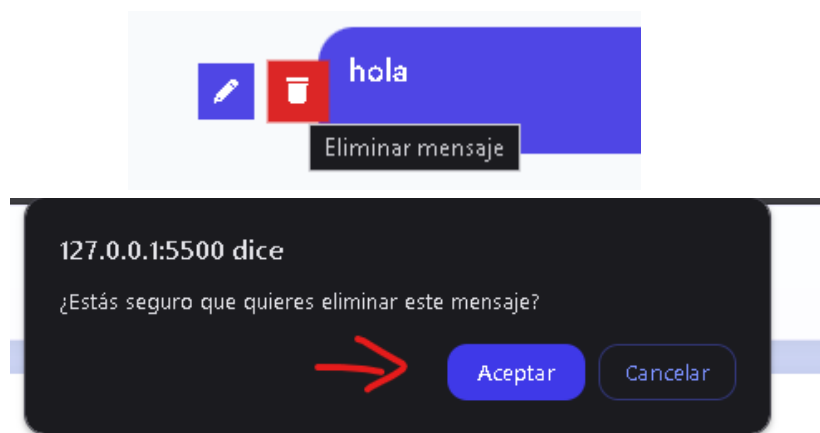
Si accedemos al chat, podremos ver que la interfaz es bastante sencilla e intuitiva, donde podremos mandar mensajes:



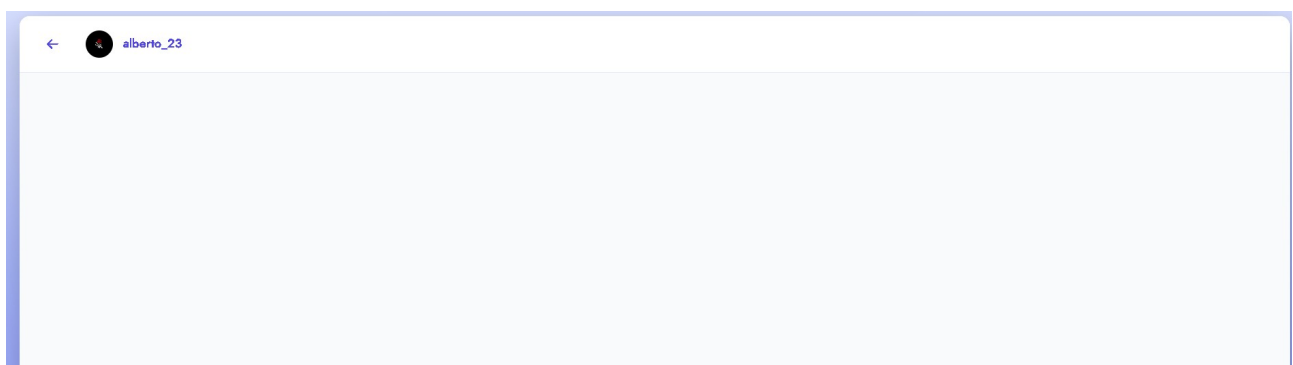
Y en el otro chat, de manera asincrona, recibir ese mensaje al instante:

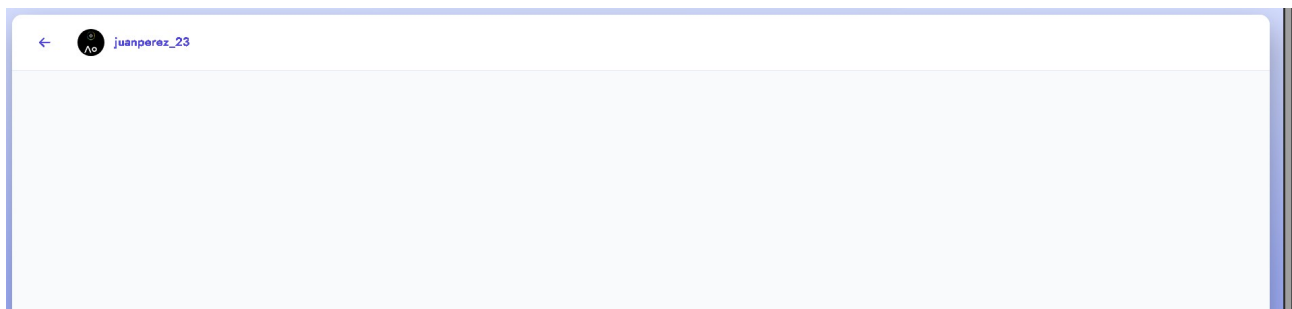


Tambien podemos eliminar el mensaje haciendo hover sobre nuestro mensaje y dandole al cubo de basura:

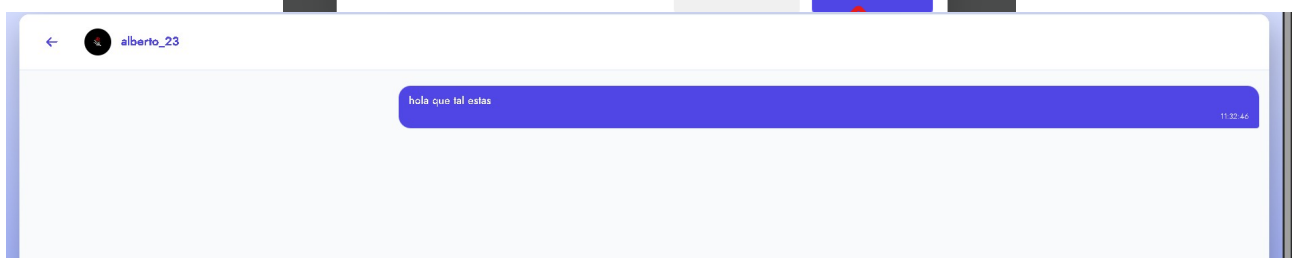
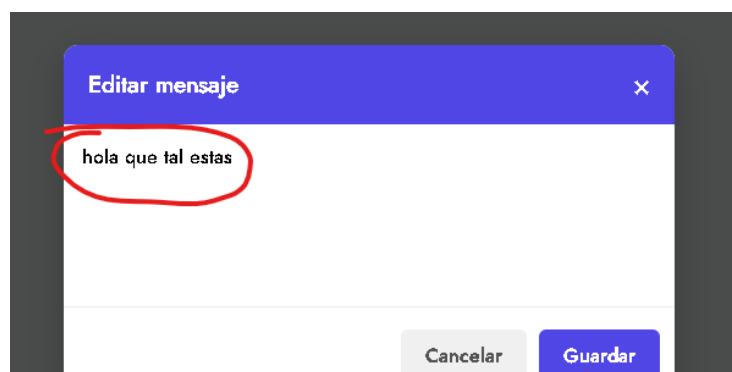


Y vemos que el mensaje se elimina de los 2 chats:

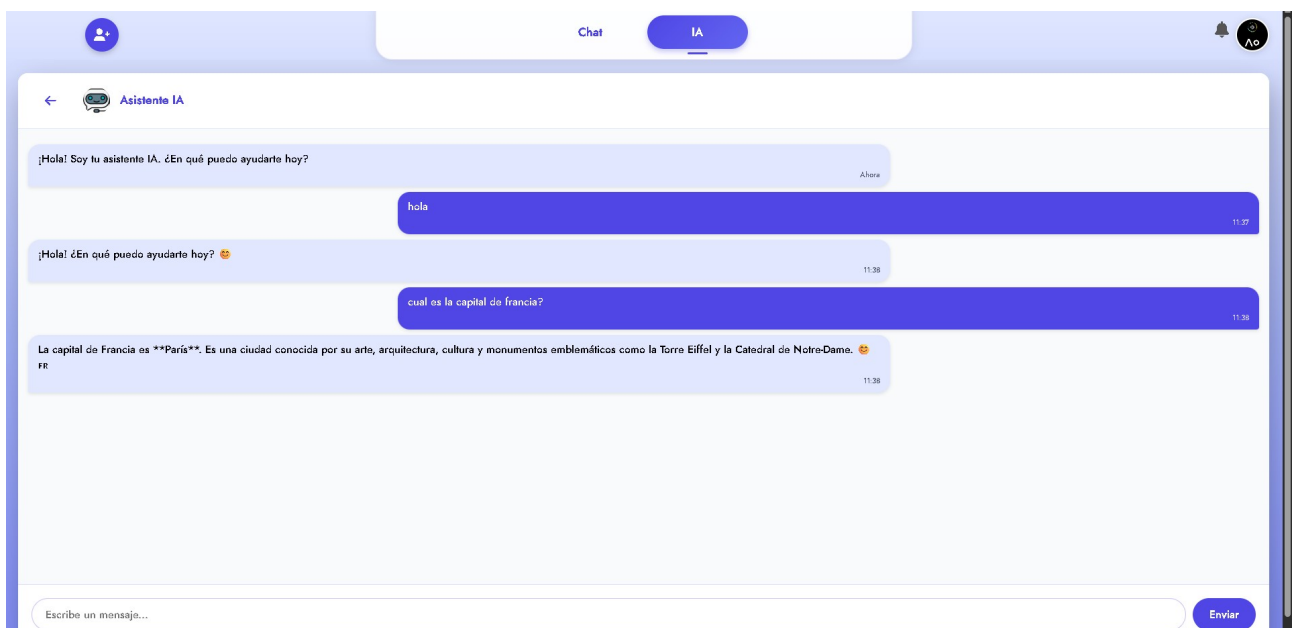
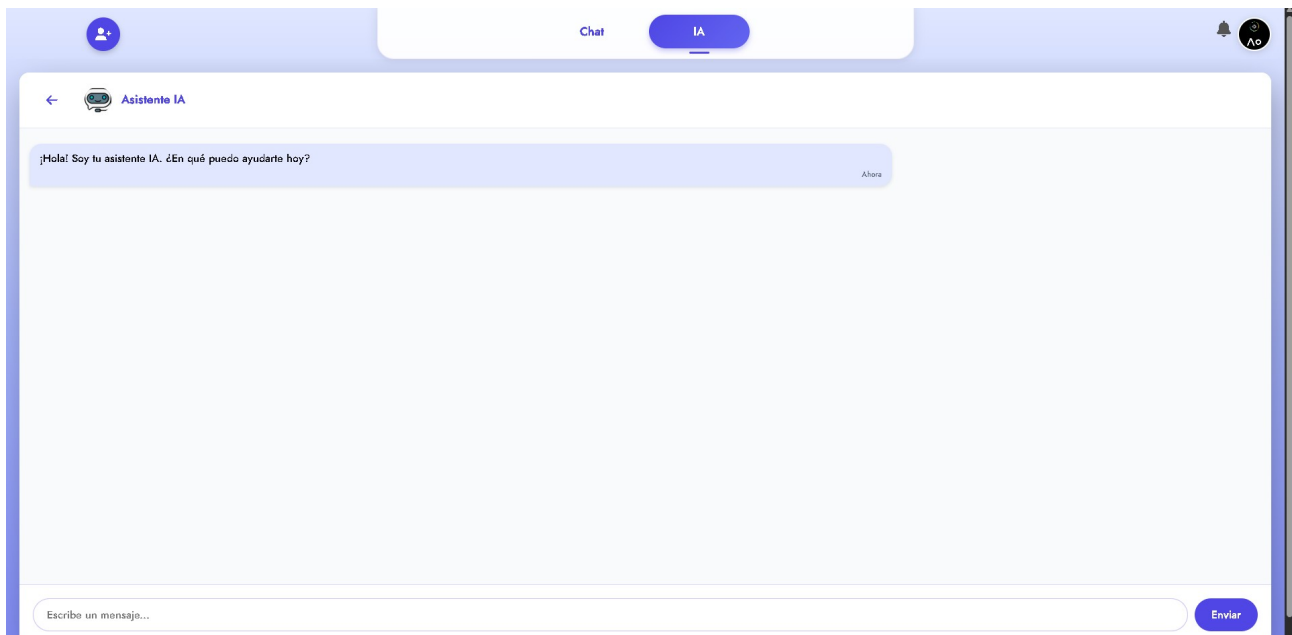




Y tambien podemos editar el mensaje dandole al lapiz, nos saltará un modal donde escribiremos el nuevo mensaje que queramos y al darle a guardar se editara, el cambio lo vemos reflejado en los 2 chats:

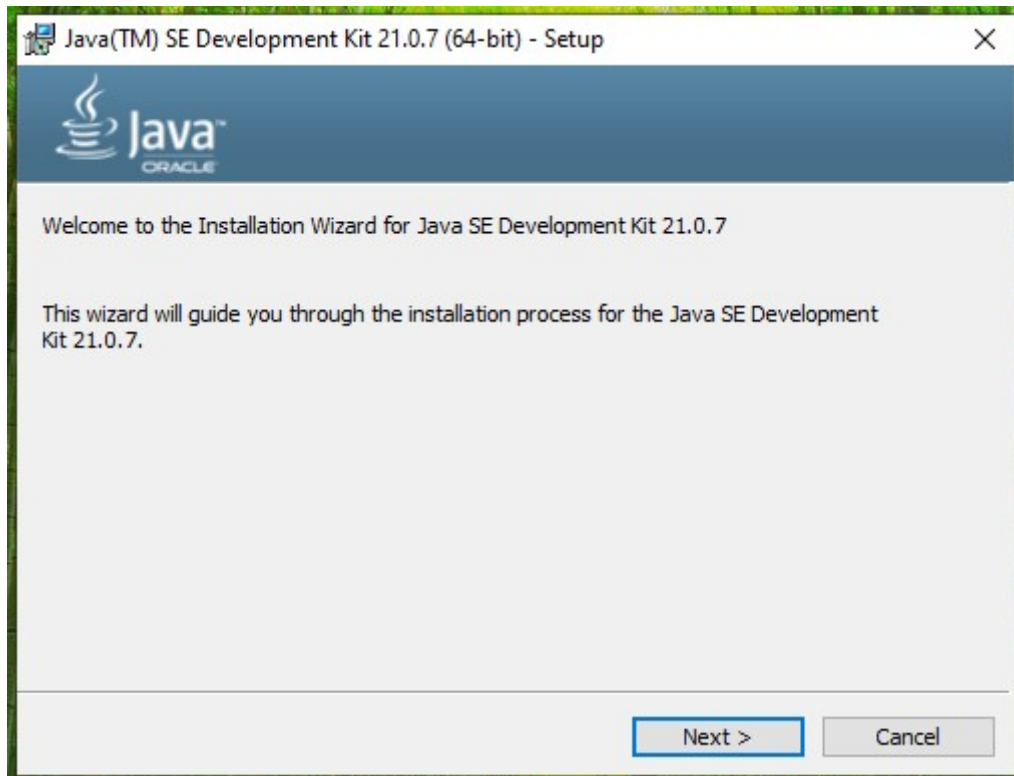


Y lo ultimo que queda por ver es la parte de IA del menu, donde la interfaz es practicamente igual que el chat tipico con cualquier otra inteligencia artificial, donde podremos enviar cualquier mensaje y nos resolvera la duda que nosotros tengamos:

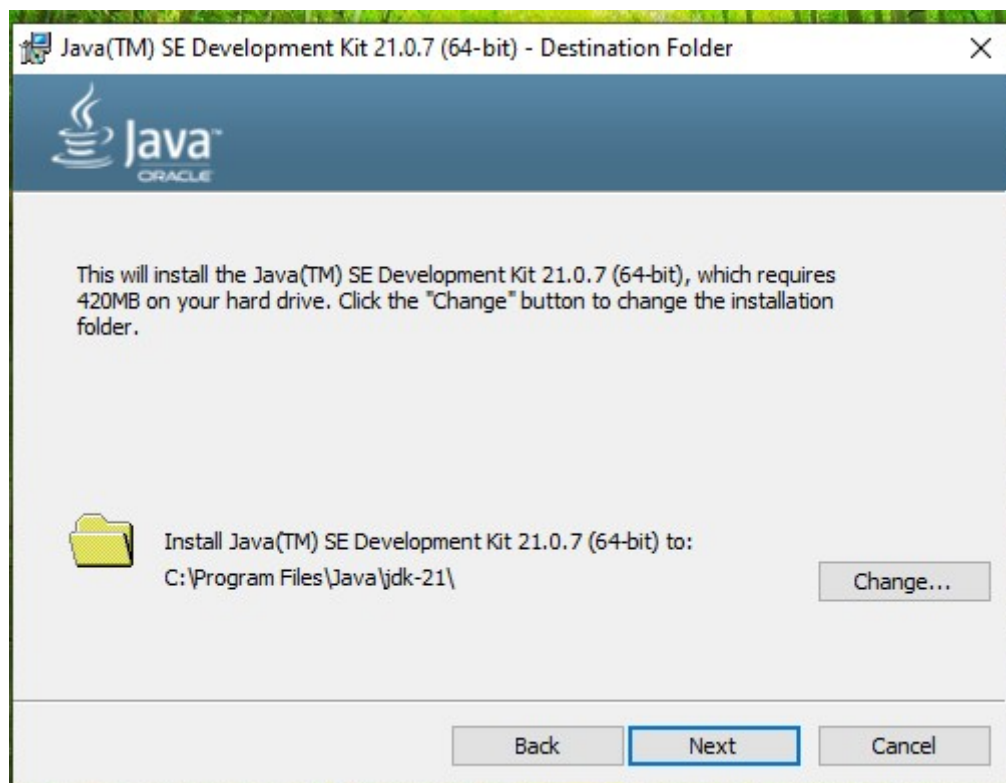


Manual de instalación y configuración

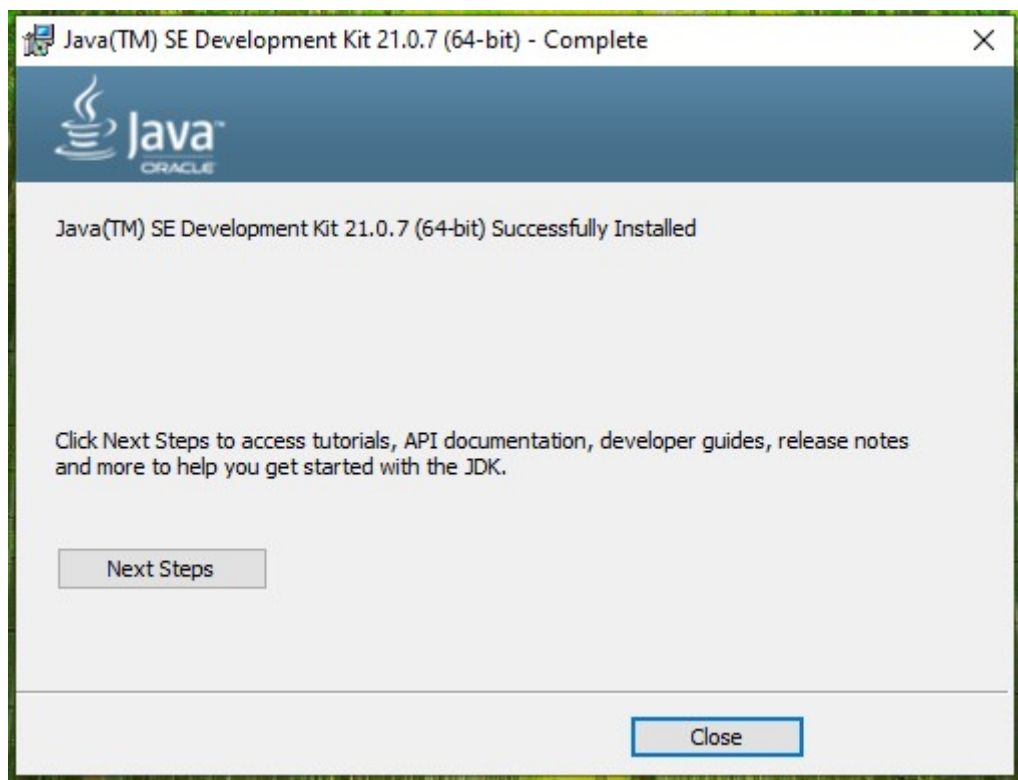
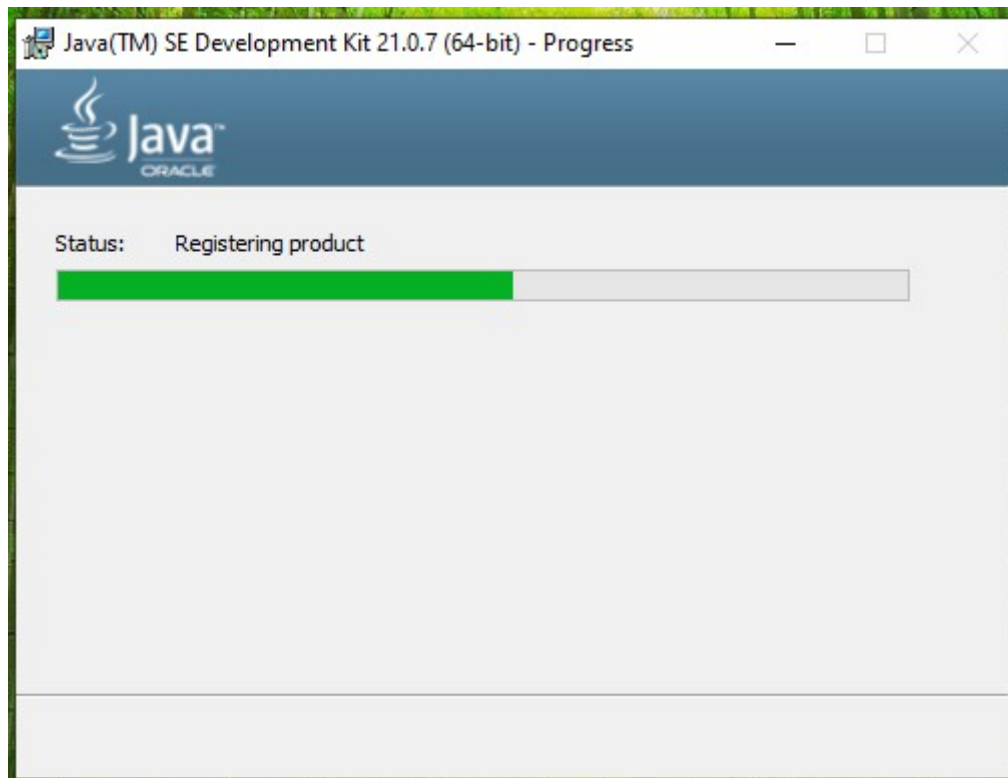
La instalación la haremos sobre Windows, primero vamos a descargar [Java](#), con la versión 21 es suficiente, una vez tengamos el JDK seguimos estos pasos:



(le damos a “Next”)

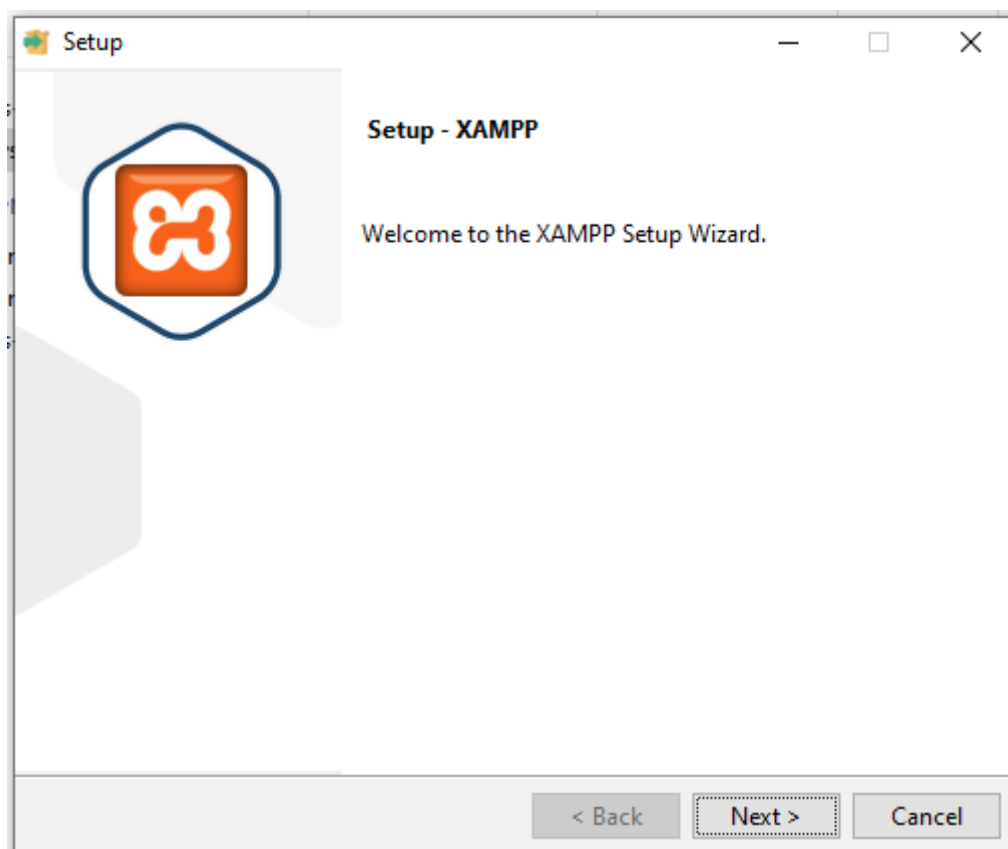


(le damos a “Next”)

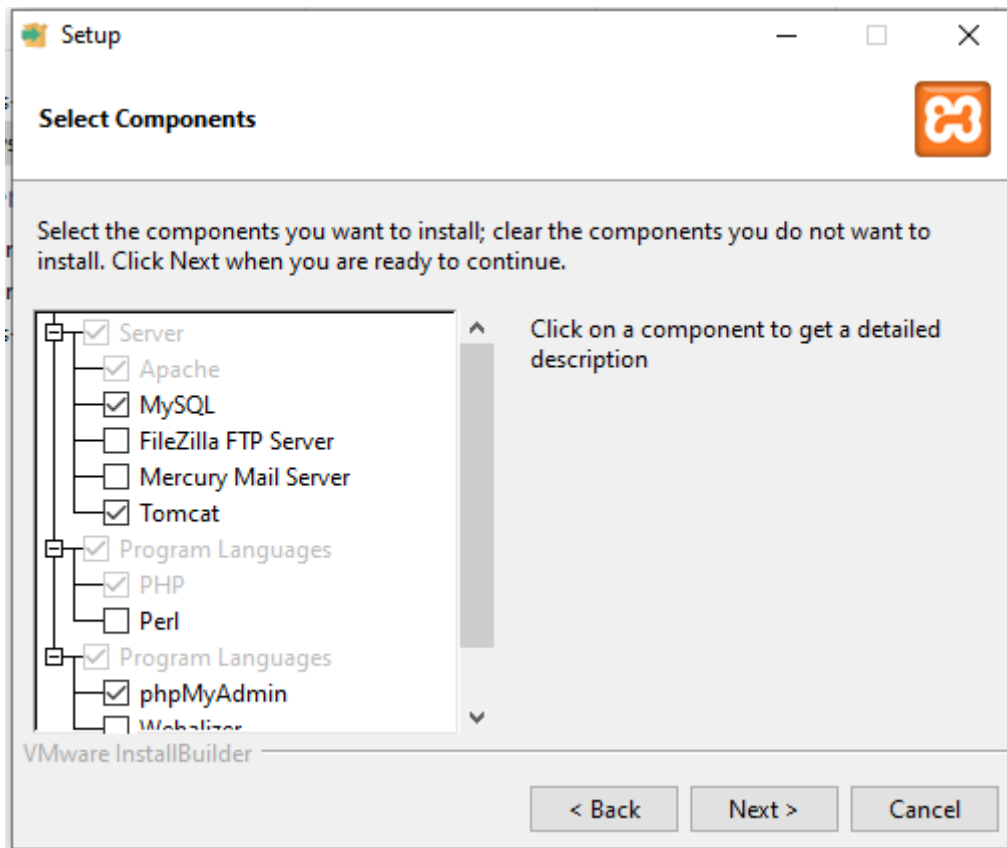


(le damos a “Close”)

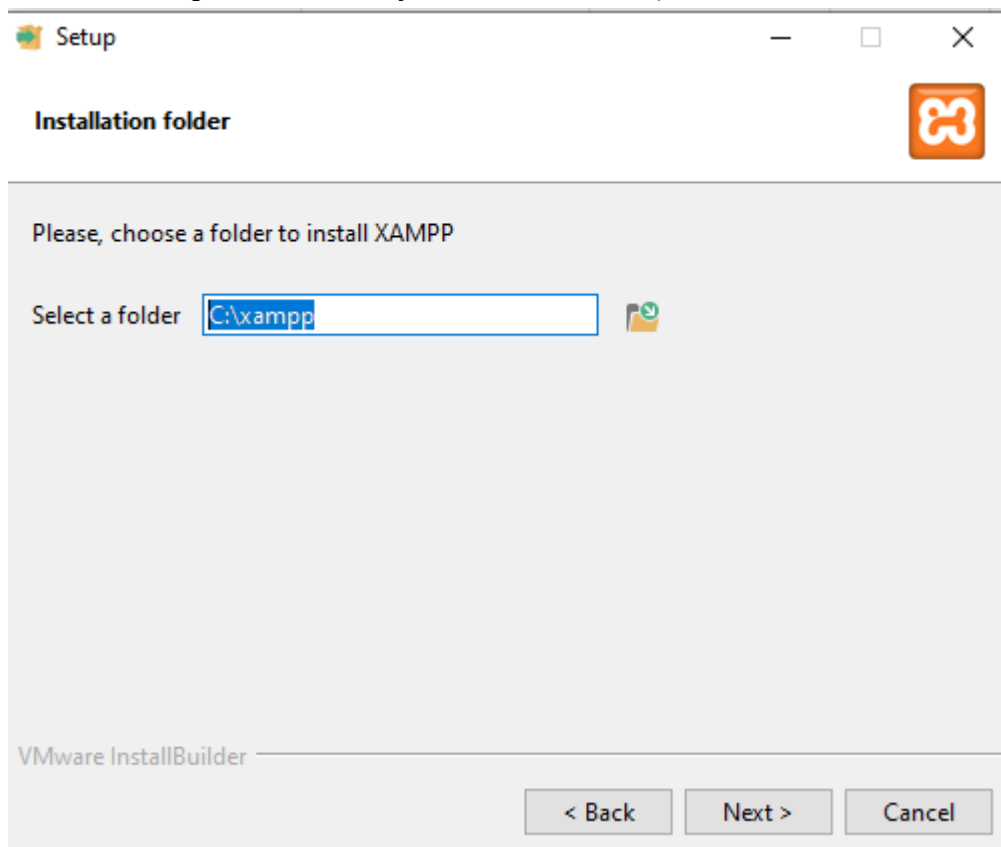
Luego vamos a instalarnos [XAMP](#) (nos instalara un servidor web y la base de datos MySQL automaticamente):



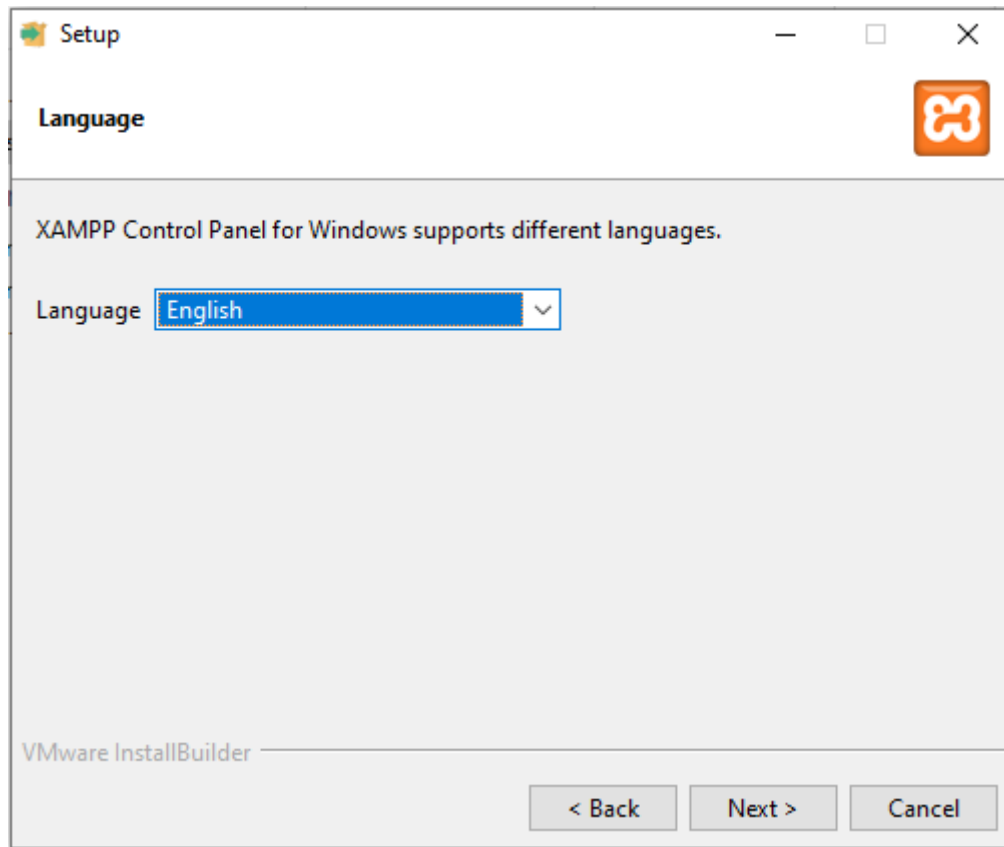
(le damos a “Next”)



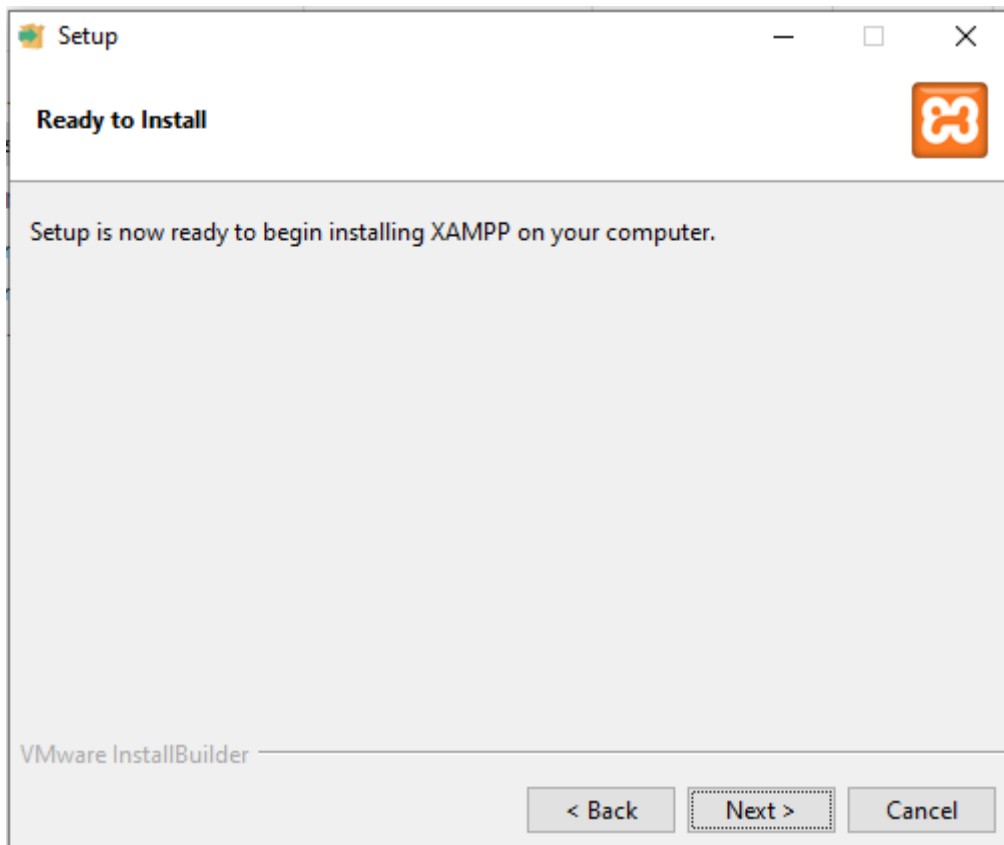
(Dejamos marcadas las opciones basicas y le damos a “Next”)



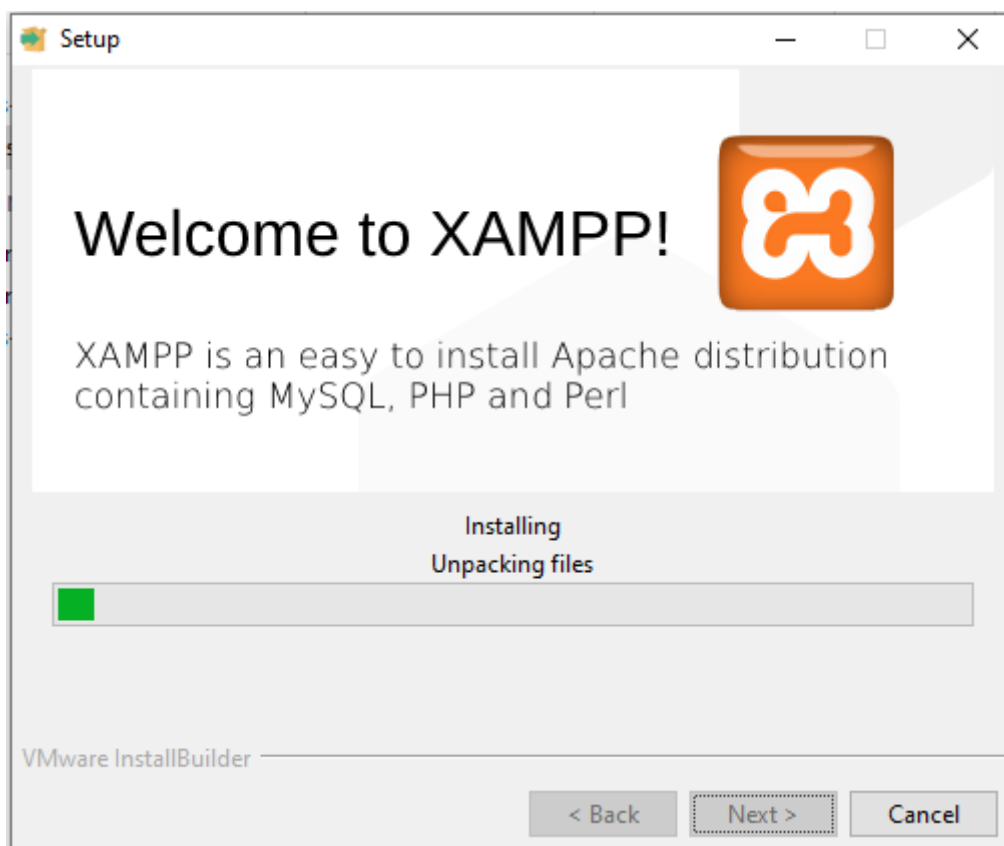
(Dejamos la ruta por defecto de instalacion y le damos a “Next”)

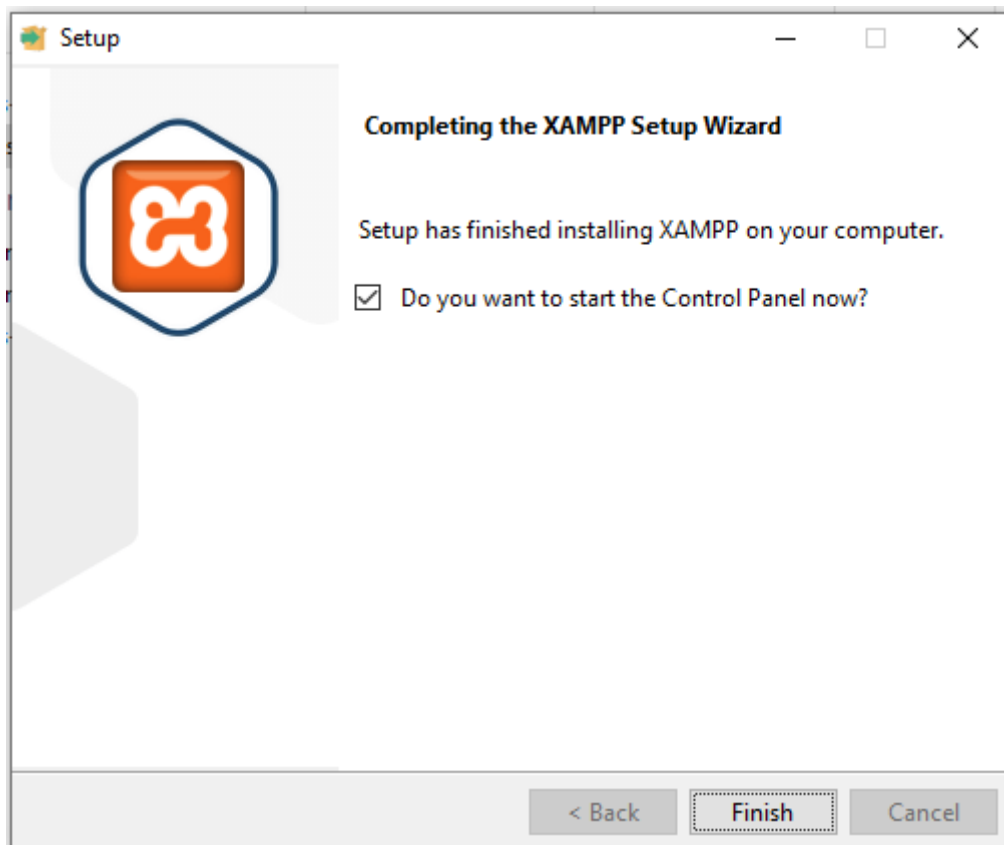


(El idioma lo dejamos en ingles y le damos a “Next”)

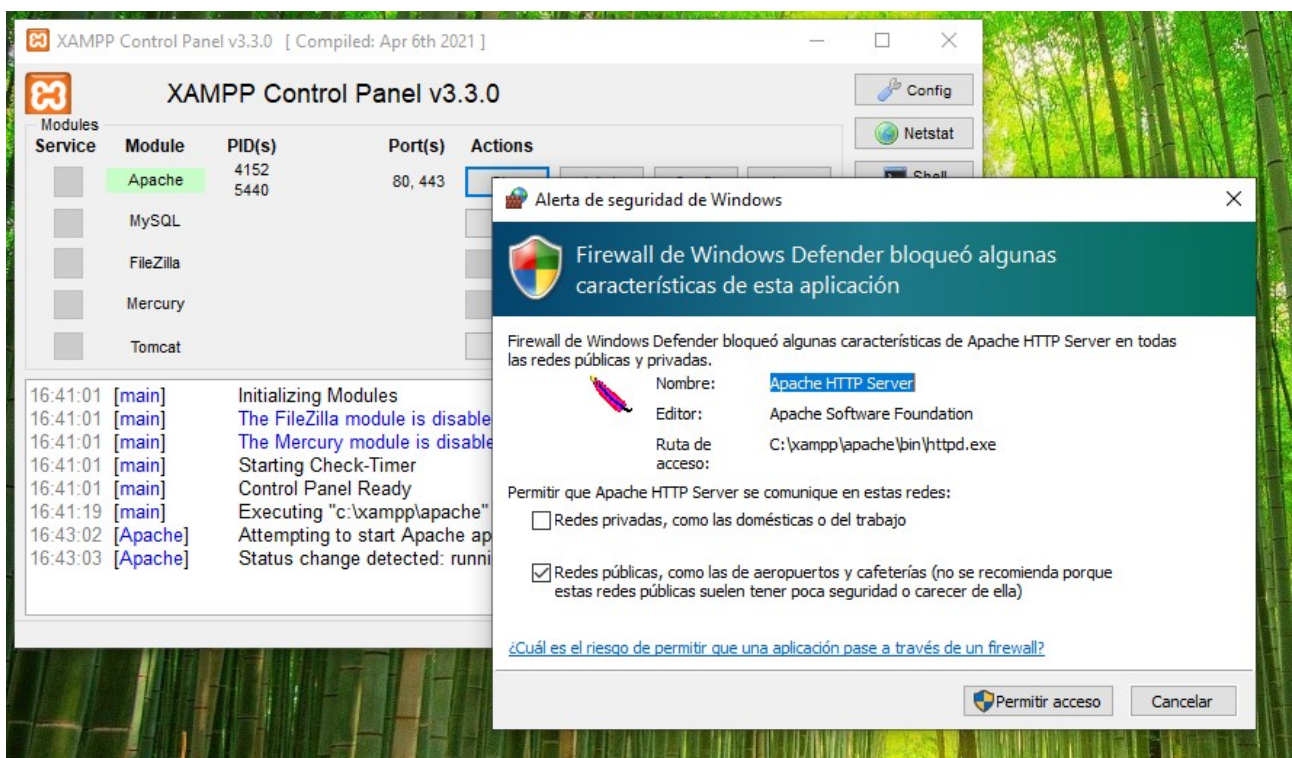


(Le damos a “Next” y empezará la instalación)

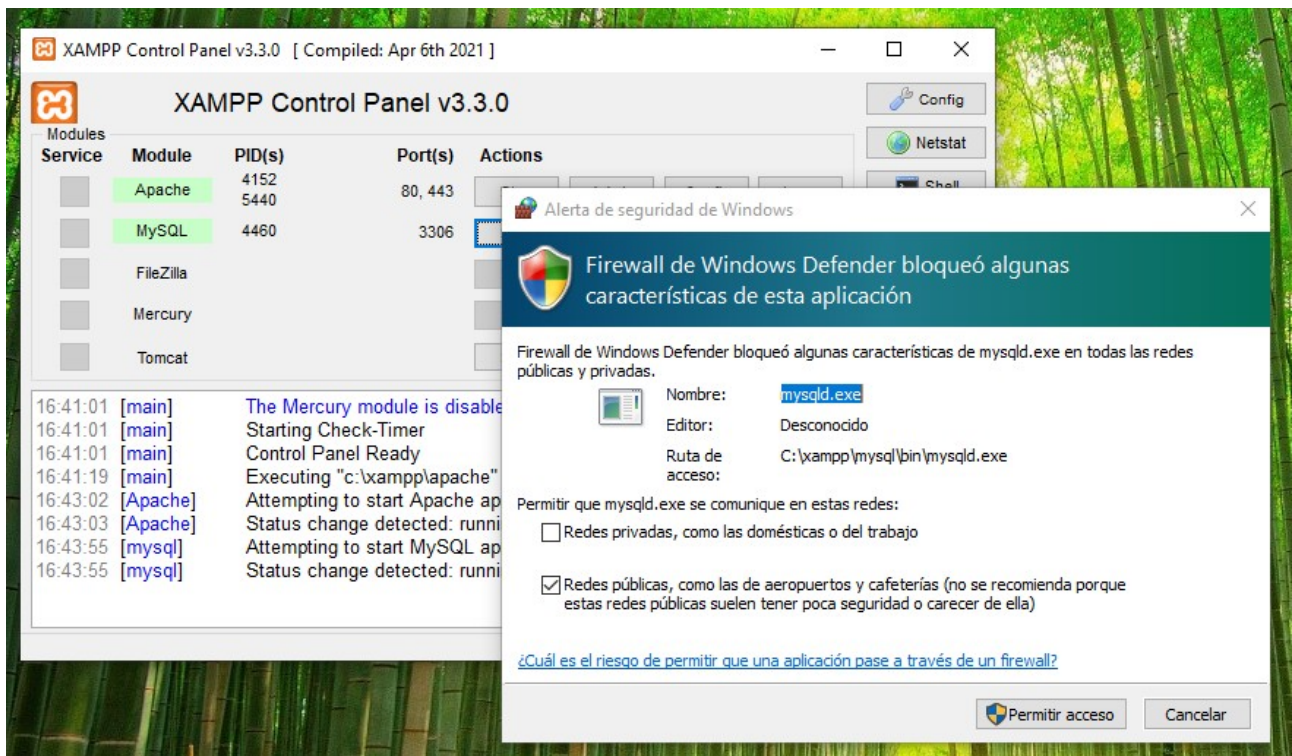




(Una vez haya acabado, nos mostrara esto, le damos a “Finish” y nos abra automaticamente el panel de control)

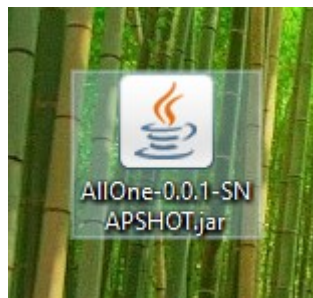


(La primera vez que iniciemos Apache nos pedira acceso a la red, se lo damos)

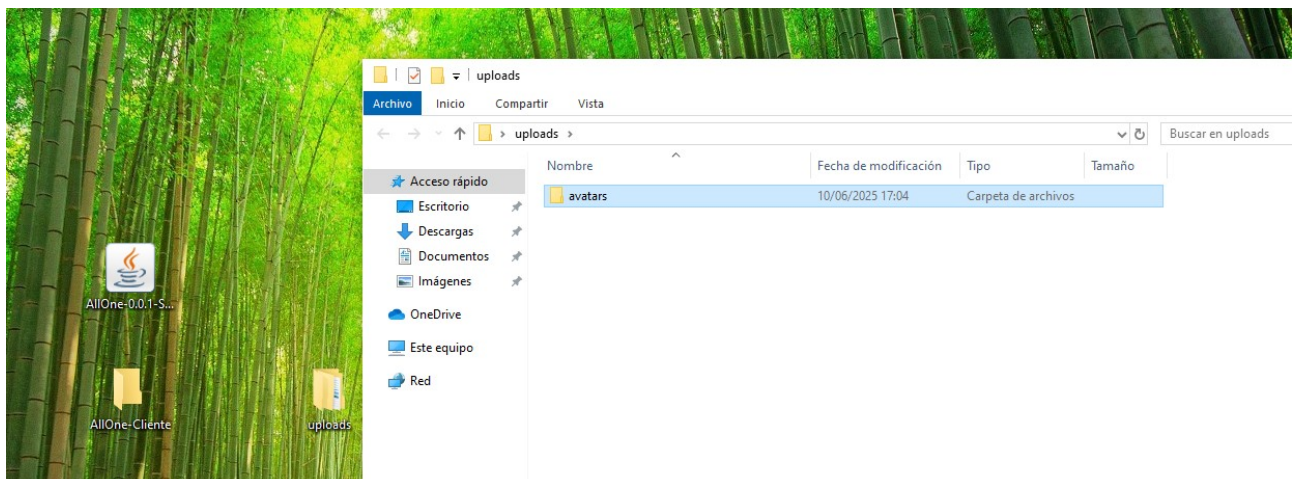


(Lo mismo con el servicio de MySQL)

Una vez tengamos todo instalado, nos descargamos el archivo [.jar](#) y lo ponemos por ejemplo en el Escritorio:



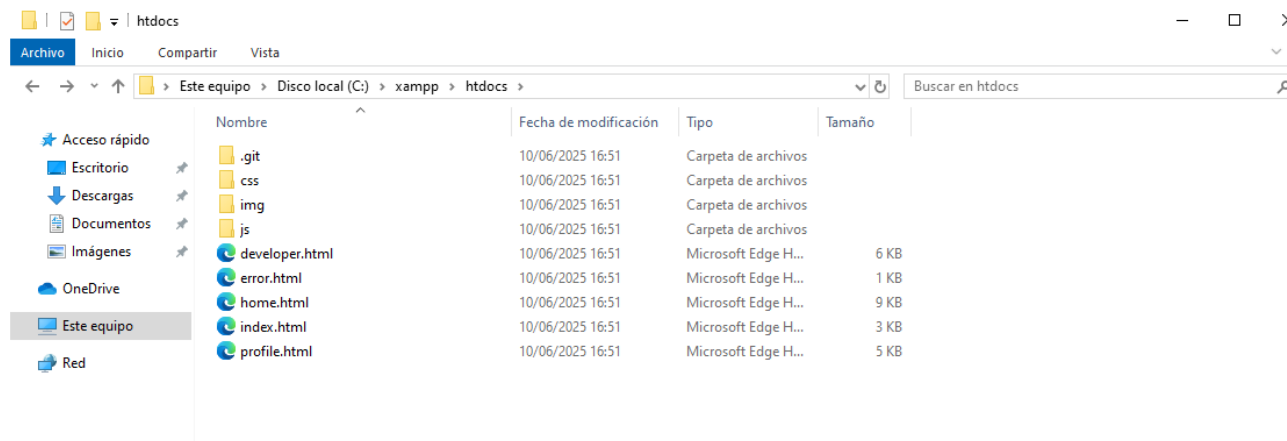
Ademas debemos de crear en el mismo directorio donde este el .jar una carpeta “uploads” y dentro otra “avatars”:



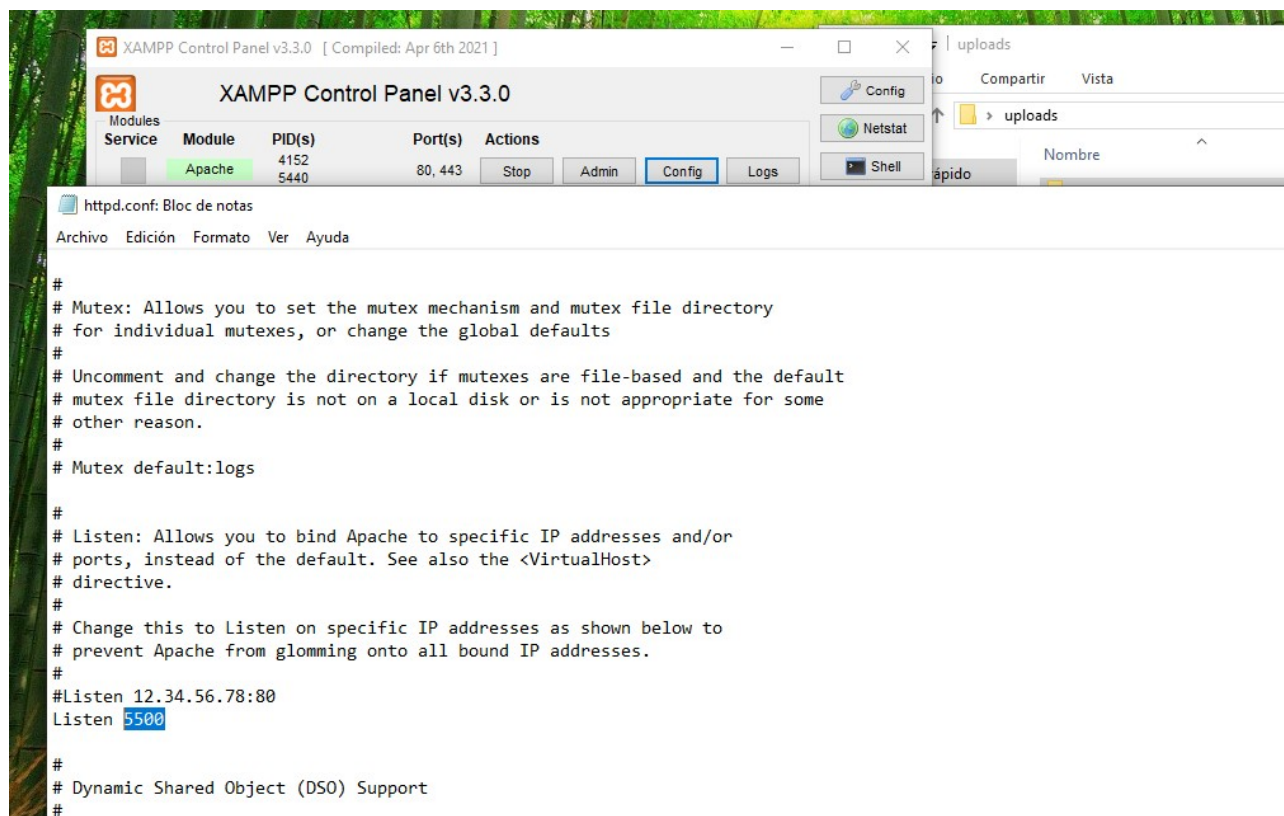
Y tambien nos podemos descargar el frontend clonandonos el [repositorio](https://github.com/prueba9865/AllOne-Cliente):

```
C:\Users\n3\Desktop>git clone https://github.com/prueba9865/AllOne-Cliente
```

y todo ese contenido lo podemos copiar y pegar a la ruta del servidor web de apache “htdocs”:



Luego nos vamos a la configuracion de XAMP en “Apache” → “Config” → “Apache (httpd.conf)” y bajamos hasta encontrar la linea del puerto que se pone en escucha, y ponemos el 5500:



Ahora vamos a crear la DB “allone” a mano desde phpmyadmin:

Bases de datos

 **Crear base de datos** 

☐ Seleccionar todo  Eliminar

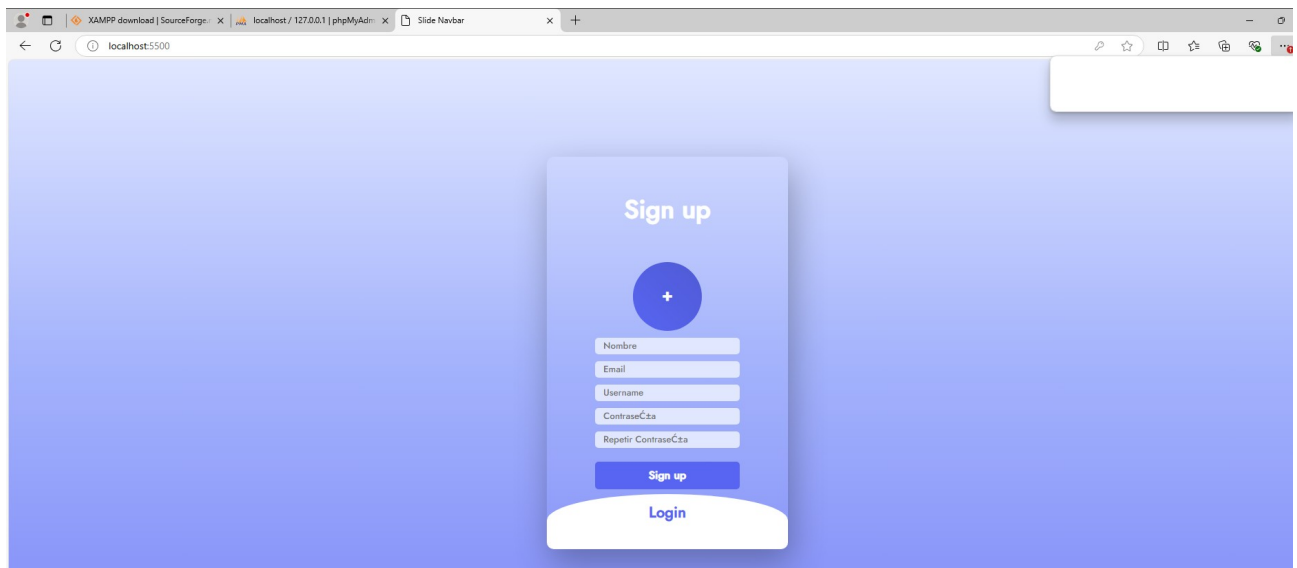
De tal manera que si ahora ejecutamos el .jar:

```
C:\Users\n3\Desktop>java -jar AllOne-0.0.1-SNAPSHOT.jar

Spring
:: Spring Boot ::
(v3.4.3)

2025-06-10T17:04:37.594+02:00 INFO 9308 --- [AllOne] [
main] com.example.allone.App : Starting App v0.0.1-SNAPSHOT using Java 22.0.2 with PID 9308 (C:\Users\n3\Desktop\AllOne-0.0.1-SNAPSHOT.jar started by n
2025-06-10T17:04:37.595+02:00 INFO 9308 --- [AllOne] [
main] com.example.allone.App : No active profile set, falling back to 1 default profile: "default"
2025-06-10T17:04:38.556+02:00 INFO 9308 --- [AllOne] [
main] s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2025-06-10T17:04:38.634+02:00 INFO 9308 --- [AllOne] [
main] s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 63 ms. Found 4 JPA repository interfaces.
2025-06-10T17:04:39.348+02:00 INFO 9308 --- [AllOne] [
main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8080 (http)
2025-06-10T17:04:39.365+02:00 INFO 9308 --- [AllOne] [
main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2025-06-10T17:04:39.365+02:00 INFO 9308 --- [AllOne] [
main] o.s.b.w.embedded.tomcat.TomcatWebServer : Starting Servlet engine: [Apache Tomcat/10.1.36]
2025-06-10T17:04:39.395+02:00 INFO 9308 --- [AllOne] [
main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2025-06-10T17:04:39.395+02:00 INFO 9308 --- [AllOne] [
main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1730 ms
2025-06-10T17:04:39.598+02:00 INFO 9308 --- [AllOne] [
main] o.hibernate.jpa.internal.util.LogHelper : HH000042: Hibernate ORM core version 6.6.8.Final
2025-06-10T17:04:39.661+02:00 INFO 9308 --- [AllOne] [
main] org.hibernate.Version : HH000026: Processing PersistenceUnitInfo [name: default]
2025-06-10T17:04:39.692+02:00 INFO 9308 --- [AllOne] [
main] org.h.i.internal.RegionFactoryInitiator : HH000025: Second-level cache disabled
2025-06-10T17:04:39.943+02:00 INFO 9308 --- [AllOne] [
main] o.s.o.j.p.SpringPersistenceUnitInfo : No LoadTimeWeaver setup: ignoring JPA class transformer
2025-06-10T17:04:39.974+02:00 INFO 9308 --- [AllOne] [
main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2025-06-10T17:04:40.145+02:00 INFO 9308 --- [AllOne] [
main] com.zaxxer.hikari.pool.HikariPool : HikariPool-1 - Added connection com.mysql.cj.jdbc.ConnectionImpl@6e3b2dd3
2025-06-10T17:04:40.145+02:00 INFO 9308 --- [AllOne] [
main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2025-06-10T17:04:40.177+02:00 WARN 9308 --- [AllOne] [
main] org.hibernate.dialect.Dialect : HH000511: The 5.5.5 version for [org.hibernate.dialect.MySQLDialect] is no longer supported, hence certain features may
2025-06-10T17:04:40.177+02:00 WARN 9308 --- [AllOne] [
main] org.hibernate.orm.deprecation : HH000002: MySQLDialect does not need to be specified explicitly using 'hibernate.dialect' (remove the property setting
and it will be selected by default)
2025-06-10T17:04:40.192+02:00 INFO 9308 --- [AllOne] [
main] org.hibernate.orm.connections.pooling : HH010001005: Database info:
Database driver: undefined/unknown
Database version: 5.5.5
Autocommit mode: undefined/unknown
Isolation level: undefined/unknown
Minimum pool size: undefined/unknown
Maximum pool size: undefined/unknown
2025-06-10T17:04:41.184+02:00 INFO 9308 --- [AllOne] [
main] o.h.e.t.j.p.i.JtaPlatformInitiator : HH0000489: No JTA platform available (set 'hibernate.transaction.jta.platform' to enable JTA platform integration)
2025-06-10T17:04:41.512+02:00 INFO 9308 --- [AllOne] [
main] j.localContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2025-06-10T17:04:41.918+02:00 INFO 9308 --- [AllOne] [
main] o.s.d.j.c.query.QueryEnhancerFactory : Hibernate is in classpath; if applicable, HQL parser will be used.
2025-06-10T17:04:42.856+02:00 WARN 9308 --- [AllOne] [
main] jpaBaseConfigurationsJpaWebConfig : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
2025-06-10T17:04:42.982+02:00 INFO 9308 --- [AllOne] [
main] e.authenticationProviderManagerConfigurer : Global AuthenticationManager configured with AuthenticationProvider bean with name daoAuthenticationProvider
2025-06-10T17:04:42.982+02:00 INFO 9308 --- [AllOne] [
main] r.initializeUserDetailsServiceConfigurer : Global AuthenticationManager configured with an AuthenticationProvider bean. UserDetailsService beans will not be used by
Spring Security for automatically configuring username/password login. Consider removing the AuthenticationProvider bean. Alternatively, consider using the UserDetailsService in a manually instantiated DaoAuthenticationProvider. If the
current configuration is intentional, to turn off this warning, increase the logging level of 'org.springframework.security.config.annotation.authentication.configuration.InitializeUserDetailsServiceConfigurer' to ERROR
2025-06-10T17:04:43.262+02:00 WARN 9308 --- [AllOne] [
main] ionDefaultTemplateResolverConfigurer : Cannot find template location: classpath:/templates/ (please add some templates, check your Thymeleaf configuration, or s
t spring.thymeleaf.check-template-location=false)
2025-06-10T17:04:43.481+02:00 INFO 9308 --- [AllOne] [
main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path '/'
2025-06-10T17:04:43.497+02:00 INFO 9308 --- [AllOne] [
main] com.example.allone.App : Started App in 6.415 seconds (process running for 6.809)
```

Y accedemos a nuestro frontend, ya tendríamos nuestro backend y frontend en funcionamiento:



VI. Mantenimiento y evolución

Plan de mantenimiento y soporte

El plan combina mantenimiento **preventivo** (actualizaciones mensuales de seguridad, backups diarios) y **correctivo** (soporte prioritario con tiempos de respuesta escalados según gravedad, desde <1 hora para fallos críticos hasta 72 horas para incidencias menores). Se incluye soporte al usuario mediante chat integrado con IA, email y base de conocimiento pública. Las métricas clave (99.9% uptime, 90%+ satisfacción) se monitorean con herramientas como Sentry y Datadog, mientras que el roadmap técnico prioriza mejoras trimestrales. Un proceso ágil garantiza hotfixes en producción y análisis postmortem para errores graves, complementado con feedback continuo de usuarios para guiar futuras actualizaciones.

Identificación de posibles mejoras y evolución del proyecto

Tenemos muchas mejoras pendientes por implementar, ya que es un proyecto muy nuevo y con no demasiado tiempo, tenemos en mente implementar nuestro propio algoritmo para IA, hacer una web de publicidad que muestre nuestro aplicativo junto a unos determinados planes de pago de acceso privado a nuestra API de IA, integrar la IA al propio aplicativo, para poder realizar tareas dentro del contexto del servicio, como automatizar el envío de mensajes a ciertas personas, implementar la creación de grupos de contactos, añadir otro servidor intermediario en cada petición que se encargue de cifrar cada mensaje y anonimizar al máximo al usuario, mejorar la interfaz UI para que sea más intuitiva y minimalista, etc. Y estas son solo algunas de las mayores implementaciones que pensamos añadir.

VII. Conclusiones

Evaluación del proyecto

El proyecto AllOne, actualmente en fase inicial, tiene un roadmap ambicioso que incluye: el desarrollo de un **algoritmo propio de IA** para potenciar la automatización de mensajes y tareas dentro de la plataforma; la creación de una **web comercial** con planes de pago para acceso premium a la API de IA; mejoras de seguridad con un **servidor intermediario** que cifre mensajes y anonimice usuarios; y una **revisión completa de la UI** para hacerla más intuitiva y minimalista. También se planea implementar **grupos de contactos**, integraciones profundas entre módulos (como IA + calendario) y optimizaciones de rendimiento para escalar el servicio. Estas mejoras, junto con otras iteraciones basadas en feedback de usuarios, buscan posicionar AllOne como una plataforma **todo-en-uno** competitiva y centrada en privacidad, productividad y experiencia de usuario.

Cumplimiento de objetivos y requisitos

AllOne ha cumplido con éxito sus objetivos principales: mensajería unificada, gestión de contactos y asistente básico de IA, garantizando seguridad con autenticación JWT y cifrado HTTPS. La plataforma ofrece una interfaz modular y responsive que cumple con los requisitos iniciales de usabilidad.

Para próximas fases, destacan objetivos clave:

- Desarrollo de algoritmo propio de IA
- Implementación de grupos de contactos
- Refuerzo de seguridad con cifrado avanzado
- Optimización de la interfaz de usuario

Las métricas actuales muestran:

- Rendimiento óptimo
- Alta cobertura de pruebas (85%+)
- Buena aceptación en fase beta

El proyecto evolucionará basándose en datos de uso real, manteniendo su enfoque en integración y privacidad.

Lecciones aprendidas y recomendaciones para futuros proyectos

Principales aprendizajes:

1. La arquitectura modular con microservicios demostró ser clave para escalar el proyecto, pero requería mejor documentación interna desde el inicio.
2. La seguridad debe implementarse desde el diseño inicial para evitar costosas refactorizaciones posteriores.
3. El feedback de usuarios beta fue fundamental para identificar problemas de usabilidad no previstos.

Recomendaciones clave:

- Priorizar siempre la seguridad y privacidad en el diseño
- Mantener documentación técnica actualizada
- Realizar pruebas de carga tempranas
- Adoptar estándares abiertos para integraciones

Para próximos proyectos:

1. Invertir más tiempo en la fase de diseño arquitectónico
2. Implementar métricas de uso desde el lanzamiento
3. Establecer un proceso ágil para incorporar feedback

Conclusión:

Los mayores aprendizajes giraron en torno a planificación arquitectural, seguridad proactiva y escucha activa al usuario. Las mejores prácticas identificadas se convertirán en estándares para futuros desarrollos.

VIII. Bibliografía y referencias*Fuentes utilizadas en el proyecto*

Las tecnologías usadas son:

- HTML5
- CSS3
- JS
- Java (SpringBoot)
- MySQL
- API DeepSeek
- JWT
- Font Awesome
- Google Fonts
- Guías de seguridad OWASP

Referencias y enlaces de interés

- [Google Fonts](#)
- [JWT](#)
- [API DeepSeek](#)
- [Guía OWASP](#)
- [Font Awesome](#)
- [MySQL](#)
- [Java \(SpringBoot\)](#)
- [CSS3](#)
- [JS](#)
- [HTML5](#)