Entrega 4 Visicontrol: Accesibilidad, usabilidad y pruebas

Integrantes:

- Kevin Adrián Gil Soto (@kevingS0712)
- Boris Guiillermo Briones Dupla
- Fernando Moreno Mora
- Rafael Alejandro Ajila Gallegos

Carrera: Ingeniería de Software

Fecha: [03/12/2025]

1. Resumen general de tipos de prueba

Tipo de prueba	Objetivo principal	Herramientas principales
Accesibilidad (WCAG)	Verificar contraste, navegación por teclado y soporte básico a lectores de pantalla.	Chrome + Lighthouse, inspección de DOM.
Usabilidad con usuarios reales	Evaluar si usuarios reales entienden y completan tareas sin dificultad.	Navegador (desktop/móvil), grabación de pantalla, guía de tareas.
Unitarias (código)	Validar funciones de negocio aisladas (sin Express ni BD).	Jest + ts-jest.
Integración	Probar el flujo completo entre API Express, middlewares de autenticación (JWT) y PostgreSQL, por ejemplo: POST /api/auth/login seguido de GET /api/auth/me, verificando que el token emitido sea válido y que se recupere correctamente la información del usuario autenticado.	Jest + Supertest, postgresql.
Funcionales / caja negra	Verificar flujos clave de negocio vía API y UI sin mirar el código.	Postman, navegador.
Rendimiento (carga 10 usuarios)	Medir tiempos promedio, mínimos y máximos de respuesta, throughput y porcentaje de errores al simular ~10 usuarios concurrentes realizando operaciones de login y consulta de visitas, para comprobar que el backend soporta una carga básica sin degradación severa.	Apache JMeter.
Seguridad básica (OWASP)	Detectar fallos básicos de inyección y de autenticación/autorización.	Postman.

2. Plan de pruebas de accesibilidad

Pantalla / área	Objetivo / Qué se prueba	Cómo se prueba (pasos)	Criterio de aceptación
Login	Contraste de texto y fondo en inputs y botones.	Ejecutar Lighthouse (sección *Accessibility*) en la pantalla de login. Revisar "Background and foreground colors have a sufficient contrast ratio".	Score de accesibilidad ≥ 80 y sin errores críticos de contraste.
Dashboard	Labels y nombres accesibles en botones, cards y enlaces.	Revisar resultados de Lighthouse y verificar que todos los botones y links tengan texto o `aria-label` descriptivo.	Sin errores en "Buttons have an accessible name" y "Links have a discernible name".
Historial de visitas	Etiquetas en filtros (inputs/select) y estructura de tabla.	Ejecutar Lighthouse y revisar sección "Names and labels". Comprobar que fecha y filtros tengan label asociado y que la tabla tenga encabezados correctos.	Ningún campo de filtro importante sin label; tabla con encabezados claros.
Login + Dashboard (teclado)	Navegación solo con teclado (Tab, Shift+Tab, Enter).	Usar únicamente teclado para: escribir usuario y contraseña, iniciar sesión, navegar por dashboard y abrir secciones básicas.	Se puede completar el flujo de login y navegación básica sin usar mouse; foco siempre visible.

2.1 Resultados

Para la parte de accesibilidad utilicé Lighthouse (pestaña *Accessibility*) desde Google Chrome, siempre en modo móvil, y fui probando las principales pantallas de VisiControl: login**,** dashboard**,** nueva visita (/visits) **e** historial de visitas (/history)**.** Los resultados globales fueron bastante buenos: la vista de login y la de creación de visitas obtuvieron **100/100**, mientras que el dashboard llegó a **86/100** y el historial a **87/100**.

En la práctica, esto significa que la aplicación ya cumple varias de las recomendaciones básicas de accesibilidad. Lighthouse confirma que en todas las páginas el documento HTML declara correctamente el idioma con el atributo lang y que siempre hay un <title> válido, algo que ayuda a los lectores de pantalla a anunciar el contenido. En las pantallas de login y de agendamiento de visitas se ve claramente el trabajo con formularios: los campos de entrada y los selects tienen sus respectivas etiquetas (label), los botones tienen nombres accesibles y las imágenes que se usan en esas vistas cuentan con atributo alt. Además, la

meta viewport permite hacer zoom (no se bloquea con user-scalable="no"), el contraste entre texto y fondo es aceptable y los elementos táctiles tienen tamaño suficiente para ser cómodos en celular. En el historial de visitas, Lighthouse también detecta que la tabla está bien estructurada, con cabeceras asociadas a las celdas, lo que facilita mucho la lectura con tecnologías asistivas.

Donde sí aparecieron observaciones fue en las vistas más “cargadas”, que son el dashboard y el historial. En el dashboard el reporte señala dos problemas concretos. El primero es que el botón de cerrar sesión aparece solo con un ícono y no tiene texto ni aria-label, así que para un lector de pantalla ese elemento se anuncia simplemente como “button”, sin indicar que en realidad sirve para salir de la sesión. Lo razonable aquí es añadir un aria-label="Cerrar sesión" o un texto oculto para lectores de pantalla, de modo que la acción quede clara. El segundo problema es el contraste en algunos textos, por ejemplo el mensaje que indica “Sesión iniciada como Administrador” y ciertos elementos del contenedor div.app-light. Lighthouse detecta que el color del texto y el del fondo están demasiado cerca, lo que dificulta la lectura para usuarios con baja visión. La corrección sería ajustar la paleta, oscureciendo el texto o aclarando el fondo hasta llegar a una relación de contraste que cumpla con los valores recomendados.

En el caso del historial de visitas, el puntaje de 87/100 se mantiene en un rango aceptable, pero surge otro tipo de problema: los filtros. El input de fecha y el select de estado funcionan visualmente, pero Lighthouse indica que no tienen un label correctamente asociado, por lo que en el informe aparecen como campos sin nombre accesible. Para una persona que usa lector de pantalla esto es un problema, porque escucha que hay un campo editable, pero no queda claro para qué sirve (si filtra por fecha, por estado, etc.). La solución propuesta es sencilla: añadir etiquetas visibles, o al menos aria-label o aria-labelledby, con descripciones del tipo “Filtrar por fecha” y “Filtrar por estado”, de forma que esos controles sean comprensibles también sin ver la pantalla.

En resumen, las pruebas de Lighthouse muestran que VisiControl parte de una base sólida de accesibilidad, sobre todo en las vistas críticas como login y creación de visitas, que ya están en 100/100. Las pantallas de dashboard e historial también están bien encaminadas, pero requieren pequeños ajustes: dar un nombre accesible al botón de cerrar sesión, mejorar el contraste de ciertos textos y etiquetar correctamente los filtros. Una vez aplicados esos cambios, la aplicación debería acercarse al 100 en todas las vistas y ofrecer una experiencia mucho más inclusiva para personas que navegan con teclado, con lectores de pantalla o que tienen dificultades visuales.

3. Plan de pruebas de usabilidad

Tarea para el usuario	Objetivo / Qué se observa	Pasos generales para la prueba	Criterios de aceptación (cualitativos)
Registro y login	Ver si el usuario entiende cómo crear cuenta e iniciar sesión.	Entregar al usuario las credenciales o pedirle que se registre. No dar instrucciones detalladas, solo el objetivo. Registrar tiempo y errores.	Completa el registro y login sin ayuda o con mínima guía.
Ver y editar perfil	Evaluar comprensión de la sección "Mi perfil" (datos personales) y guardado de cambios.	Pedir al usuario que cambie teléfono y dirección y guarde; observar si entiende los botones y mensajes.	El usuario localiza "Mi perfil" y actualiza datos sin perderse.
Cambiar contraseña	Comprobar si el flujo de cambio de contraseña es claro.	Pedir que cambie su contraseña actual por una nueva. Observar errores de validación (contraseñas que no coinciden, mensajes confusos, etc.).	El usuario puede completar el cambio tras el ajuste de validación realizado durante la prueba real.
Crear una visita	Ver si el usuario comprende el proceso de agendar visita para un interno.	Pedir que cree una nueva visita, elija interno, fecha y hora, y confirme. Observar si entiende el formulario y mensajes de error.	El flujo se completa sin confusión grave; si hay errores, la app muestra mensajes claros.
Consultar próximas visitas e historial	Evaluar si el usuario entiende la diferencia entre "Próxima visita" e "Historial de visitas".	Pedir que encuentre la visita recién creada y luego consulte visitas pasadas en el historial.	El usuario identifica dónde ver visitas futuras y dónde ver visitas pasadas sin ayuda.
Opinión general de uso	Recoger feedback sobre diseño, claridad y utilidad de la app.	Al final, hacer preguntas abiertas ("¿Qué fue fácil?", "¿Qué mejorarías?") y registrar comentarios, como el caso real de foto de perfil y contraseña.	La mayoría de comentarios son positivos (app simple y clara); se documentan mejoras sugeridas.

4. Plan de pruebas de unitarias

Módulo / función probada	Objetivo / Qué se valida	Herramienta / archivo de test	Criterio de aceptación
formatDateForText	Formato de fechas a dd/MM/yyyy y manejo de valores nulos/indefinidos.	Jest (src/backend/src/tests/visits-helpers.test.ts).	Devuelve formato correcto para fechas válidas y "" para valores no válidos.
formatTimeForText	Normalización de horas a HH:mm a partir de string o Date.	Jest (visits-helpers.test.ts).	Devuelve hora bien formateada y maneja horas incompletas (sin minutos).
formatStatusLabel	Mapeo de estados (PENDING, APPROVED, REJECTED, etc.) a etiquetas en español.	Jest (visits-helpers.test.ts).	Estados conocidos devuelven la etiqueta correcta; estados desconocidos devuelven texto original o "Pendiente".
buildVisitMeta	Construcción del objeto meta utilizado en notificaciones de visitas.	Jest (visits-helpers.test.ts).	Objeto incluye campos correctos y pone inmate_id en null si no viene definido.

4.1 Resultados

Para validar la lógica de formato y construcción de metadatos de las visitas en el backend de VisiControl se implementaron pruebas unitarias con Jest sobre los helpers definidos en el servicio visits.service.ts. El código de pruebas se encuentra en el archivo src/backend/src/tests/visits-helpers.test.ts. Desde la carpeta src/backend se ejecutó el comando npm test, que corre automáticamente todas las suites configuradas.

En esta suite se probaron cuatro funciones auxiliares: formatDateForText, formatTimeForText, formatStatusLabel y buildVisitMeta. En el caso de formatDateForText se comprobó que una instancia de Date se convierta correctamente al formato dd/MM/yyyy (por ejemplo, 4 de diciembre de 2025 se muestra como 04/12/2025) y que, cuando el valor es null o undefined, la función devuelva una cadena vacía. Para formatTimeForText se verificó que una hora en texto como "9:5" se normalice a "09:05", que las horas construidas a partir de un objeto Date se formateen como HH:mm y que, de nuevo, los valores nulos o indefinidos se transformen en una cadena vacía.

En formatStatusLabel se validó el mapeo entre los estados internos del sistema (PENDING, APPROVED, REJECTED, CANCELLED) y las etiquetas que se muestran al usuario (Pendiente, Aprobada, Rechazada, Cancelada). Además se probó el comportamiento cuando llega un estado desconocido o cuando el valor es nulo: en esos casos la función devuelve el texto original o "Pendiente" por defecto, tal como se definió en la lógica de negocio. Finalmente, con buildVisitMeta se comprobó que, a partir de los datos crudos de una visita (id, nombre del visitante, interno, fecha, hora y estado), se construya el objeto de metadatos que utiliza el sistema de notificaciones, incluyendo el campo status_label generado internamente con formatStatusLabel. También se verificó explícitamente que cuando no se envía inmate_id este campo se complete con null.

Al ejecutar la suite, Jest reporta 1 test suite ejecutada y 9 pruebas pasadas en aproximadamente 2–3 segundos, sin errores. Esto da confianza en que las funciones auxiliares que formatean fechas, horas, estados y metadatos de visitas están cubiertas por pruebas automatizadas y se comportan conforme a los criterios definidos en el plan de pruebas unitarias.

5. Plan de pruebas de integración

Flujo / Endpoint principal	Objetivo / Qué se valida	Herramienta / archivo de test	Pasos resumidos	Criterio de aceptación
Autenticación básica (/auth/login + /auth/me)	Validar integración entre Express, middleware de autenticación JWT, servicio de usuarios y PostgreSQL.	Jest + Supertest	1) Hacer POST /api/auth/login con usuario válido de la BD. 2) Verificar que responde 200 y devuelve token. 3) Con ese token, hacer GET /api/auth/me con header Authorization: Bearer <token>.	1) login responde 200 OK con ok: true y un token JWT. 2) /me responde 200 OK con ok: true y datos del usuario (id, name, email). 3) No se lanza ningún error no controlado.
Creación y lectura de visitas de usuario	Comprobar que la creación de visitas conecta correctamente controladores, servicio de visitas y base de datos, y que luego se pueden leer a través del mismo API.	Jest + Supertest	1) Autenticar usuario y obtener token. 2) Hacer POST /api/visits con fecha/hora válidas. 3) Verificar que responde 201/200 y devuelve la visita creada. 4) Hacer GET /api/visits con el mismo token. 5) Buscar en la respuesta la visita recién creada.	1) La visita se inserta en la tabla visits. 2) GET /api/visits devuelve la visita recién creada en el listado del usuario. 3) Campos clave (fecha, hora, interno, estado) coinciden con lo enviado.
Flujo de aprobación de visitas por admin	Validar interacción completa entre permisos de rol admin, rutas de administrador y lógica de cambio de estado de visita.	Postman	1) Autenticar como admin y obtener token. 2) Hacer GET /api/visits/admin para obtener una visita PENDING. 3) Hacer PATCH /api/visits/admin/:id con status: "APPROVED". 4) Volver a llamar GET /api/visits/admin y confirmar el nuevo estado.	1) Sólo usuarios admin pueden acceder a /api/visits/admin. 2) El PATCH cambia el estado de la visita en la BD. 3) El nuevo estado (APPROVED o REJECTED) se refleja en llamadas posteriores.

5.1 Resultados

Para comprobar que los módulos principales del backend de VisiControl funcionan bien cuando se combinan (rutas de Express, controladores, middlewares de autenticación, servicios y base de datos PostgreSQL), se implementaron pruebas de integración automatizadas con Jest + Supertest. En vez de llamar funciones sueltas, estas pruebas levantan la aplicación (app) en modo de prueba y le envían peticiones HTTP simuladas, como si fuera el frontend. Todo el código está en los archivos src/tests/auth.integration.test.ts y src/tests/visits.integration.test.ts, y se ejecuta desde la carpeta backend con el comando npm test.

En el caso de autenticación, se probó el flujo completo de login. Primero se envía un POST /api/auth/login con las credenciales reales del administrador de pruebas. La API debe responder 200 OK, con ok: true, un token JWT de longitud suficiente y los datos básicos del usuario (id, nombre, correo, rol). Luego se hace otro POST /api/auth/login pero con una contraseña incorrecta; en este escenario la respuesta esperada es 401 Unauthorized, ok: false y un mensaje que indica que las credenciales son inválidas. Con estos dos casos se valida en conjunto la consulta a la base de datos, la verificación de contraseña y la generación (o no) del token de autenticación.

También se agregó una prueba de integración para el flujo de creación y lectura de visitas**.** El test primero inicia sesión para obtener un token válido, y con ese token hace un POST /api/visits enviando los datos de una visita de prueba (interno, fecha, hora y estado). La API debe devolver 201/200 y los datos de la visita creada. A continuación, el mismo test realiza un GET /api/visits con el mismo token y comprueba que dentro del listado aparezca la visita recién creada, verificando que campos clave como fecha, hora, interno y estado coincidan con lo enviado. De esta manera se confirma que el controlador de visitas, el servicio de negocio y las consultas a PostgreSQL están bien integrados y devuelven información coherente.

Al ejecutar el comando npm test, Jest reporta actualmente 3 test suites ejecutados y 13 pruebas pasadas en total**,** sin fallos. Esto indica que los flujos críticos de autenticación y manejo de visitas están cubiertos por pruebas de integración automatizadas. Si en el futuro se cambia algo en las rutas, en los middlewares o en la lógica de servicios, estos tests sirven como red de seguridad para detectar errores antes de desplegar una nueva versión de VisiControl.

6. Plan de pruebas funcionales / caja negra

Flujo funcional	Objetivo / Qué se prueba	Herramienta / ruta	Criterio de aceptación
Registro y login	Crear usuario y comprobar que puede iniciar sesión correctamente.	Postman: POST /api/auth/register y POST /api/auth/login.	Respuestas 200 OK; registro devuelve usuario creado y login devuelve token.
Gestión de visitas de usuario	Crear visita, listar próximas visitas y ver historial propio.	Postman: POST /api/visits, GET /api/visits, GET /api/visits/history con token de usuario.	Se crea la visita, aparece en listado y, tras la fecha, se refleja en historial.
Panel de administración de visitas	Listado general de visitas y cambio de estado por parte de un admin.	Postman: GET /api/visits/admin y PATCH /api/visits/admin/:id con token de admin.	Admin puede ver visitas y cambiar estado a APPROVED/REJECTED obteniendo las respuestas esperadas.
Perfil de usuario (datos personales y notificaciones)	que el usuario pueda actualizar teléfono, dirección y el check de "Recibir notificaciones por correo".	GET /api/auth/me (para ver datos actuales) y PATCH /api/auth/me (para actualizar).	después del PATCH, un nuevo GET devuelve los datos actualizados.
Cambio de contraseña (caso positivo y negativo)	Caso positivo: con contraseña actual correcta, se actualiza y luego solo permite login con la nueva. Caso negativo: con contraseña actual incorrecta, responde 400/401 y NO cambia nada.	POST /api/auth/change-password	-200 + mensaje "Contraseña actualizada" y login solo con la nueva. -400/401 con mensaje de error y login sigue funcionando con la antigua.
Cancelación de visita por parte del usuario	que un usuario pueda cancelar una visita propia		después del DELETE, la visita ya no aparece como

	y que eso se vea reflejado en el historial.	-POST /api/visits (crear) -DELETE /api/visits/:id (cancelar) - GET /api/visits/history (ver resultado)	futura y en el historial se muestra con estado CANCELLED/Cancelada.

6.1 Resultados

Para las pruebas funcionales de VisiControl se ejecutaron distintos escenarios de caja negra usando **Postman** contra la API en http://localhost:4000, siempre verificando solo las respuestas y los efectos visibles en los datos.

1. **Registro y login.**
   Se envió POST /api/auth/register con datos válidos y luego POST /api/auth/login con esas credenciales. En ambos casos la API respondió **200 OK**: el registro devolvió el usuario creado y el login entregó un **token JWT** válido que permitió acceder a endpoints protegidos. El flujo de alta e inicio de sesión se considera correcto.
2. **Gestión de visitas del usuario.**
   Con un usuario autenticado se creó una visita mediante POST /api/visits y luego se consultaron las próximas visitas con GET /api/visits y el historial con GET /api/visits/history. La visita nueva apareció en el listado futuro y, al pasar la fecha o ajustarla a una fecha pasada, se reflejó en el historial. La creación y consulta de visitas funcionan según lo esperado.
3. **Panel de administración de visitas.**
   Con un token de **ADMIN** se probó GET /api/visits/admin para ver el listado general y PATCH /api/visits/admin/:id para cambiar el estado a APPROVED/REJECTED. El administrador pudo ver todas las visitas y actualizar su estado, y los cambios se reflejaron correctamente en las respuestas. El acceso quedó restringido al rol administrador.
4. **Perfil de usuario (datos personales y notificaciones).**
   Se consultaron los datos actuales con GET /api/auth/me y luego se actualizaron campos como teléfono, dirección y notify_email mediante PATCH /api/auth/me. Un nuevo GET devolvió los datos ya modificados. La edición del perfil y de la preferencia “Recibir notificaciones por correo” funciona de forma consistente.
5. **Cambio de contraseña (caso positivo y negativo).**
   En el caso positivo, POST /api/auth/change-password con la contraseña actual correcta devolvió **200 OK** y un mensaje de éxito; después solo fue posible iniciar

sesión con la nueva contraseña. En el caso negativo, al enviar una contraseña actual incorrecta la API respondió **400/401** con mensaje de error y el login continuó funcionando únicamente con la contraseña original. Se confirma que el cambio de contraseña está protegido.

6. **Cancelación de visita por parte del usuario.**
   Se creó una visita y luego se canceló desde el propio usuario actualizando su estado a CANCELLED (mediante la ruta de actualización correspondiente). Tras la operación, la visita dejó de aparecer como futura y pasó a mostrarse en el historial con estado **CANCELLED/Cancelada**, sin permitir que otros usuarios la modifiquen. El flujo de cancelación se comporta correctamente.

7. Plan de pruebas de rendimiento

Escenario de rendimiento	Objetivo / Qué se mide	Herramienta / configuración detallada	Métricas a observar	Criterio de aceptación
Carga concurrente de login (10 usuarios)	Evaluar tiempo de respuesta y estabilidad del endpoint de login bajo una pequeña carga simultánea.	Herramienta: Apache JMeter. Configuración: Thread Group con 10 hilos, 1 iteración, ramp-up de 0–5 segundos; HTTP Request POST /api/auth/login → URL http://localhost:4000/api/auth/login, método POST, tipo x-www-form-urlencoded con credenciales válidas. Agregar Summary Report y View Results in Table.	Promedio, mínimo, máximo, desviación estándar, porcentaje de errores, throughput (peticiones/segundo).	1) Error % = 0%. 2) Tiempo promedio de respuesta en entorno local ≈ 75 ms (según ejecución real) o, en general, < 500 ms. 3) Ningún error 5xx producido por sobrecarga.
Listado de visitas con filtros bajo carga ligera	Ver cómo responde el API de consultas (lecturas desde BD) cuando varios	Herramienta: Apache JMeter. Configuración: Thread Group con 10 hilos, 2 iteraciones. HTTP Request GET /api/visits?date=<fecha> y/o GET /api/visits/history. Añadir header Authorization: Bearer	Promedio y máximo del tiempo de respuesta, throughput, porcentaje de errores (tiempos de BD).	1) Error % = 0%. 2) Los tiempos medios siguen siendo aceptables (< 500–800 ms en local). 3) No se

	usuarios consultan sus visitas de forma casi simultánea.	<token> (token copiado de un login real). Incluir Summary Report para comparar los tiempos frente al login.		observan bloqueos ni respuestas 500 en el servidor.
Análisis cualitativo de resultados de JMeter	Documentar los hallazgos de las pruebas de rendimiento y posibles mejoras futuras.	A partir de los resultados de PERF-01 y PERF-02, revisar los reportes (pantalla de Summary Report y tabla de resultados) y capturar evidencias (capturas de pantalla) para incluirlas en el informe.	No aplica (es análisis), pero se revisan patrones: estabilidad de tiempos, ausencia de picos extremos.	El informe describe: tiempos promedio, ausencia de errores, y, si correspondiera, recomendaciones (por ejemplo: cachear consultas o aumentar recursos si el sistema creciera).

7.1 Resultados

Para evaluar el comportamiento del backend de VisiControl bajo una carga ligera, se montó un plan de pruebas en Apache JMeter con dos grupos de hilos. En ambos casos se usaron 10 usuarios simulados, con un ramp-up de 10 segundos y una sola iteración por usuario, de forma que las peticiones se generen casi al mismo tiempo pero sin saturar por completo la máquina. Todos los requests apuntan al servidor local de desarrollo y reutilizan cabeceras comunes mediante un HTTP Request Defaults y un HTTP Header Manager. En el segundo grupo, además, se añadió el header Authorization con un token JWT real obtenido de un login previo para poder acceder a los endpoints protegidos.

En el primer escenario se midió el endpoint de autenticación `POST /api/auth/login`. Con 10 usuarios iniciales y dos repeticiones de la prueba se registraron 20 solicitudes válidas. El Summary Report muestra un tiempo promedio de respuesta cercano a 66 ms, con mínimos alrededor de 60 ms y un máximo de 85 ms, desviación estándar moderada y porcentaje de error 0 %. El throughput se quedó aproximadamente en 2.2 peticiones por segundo y no se observaron respuestas 5xx ni timeouts. Estos valores están muy por debajo del umbral de 500 ms definido en el plan de pruebas y confirman que el flujo de login se mantiene estable con esta carga concurrente.

El segundo escenario se centró en las lecturas de visitas, utilizando el mismo grupo de 10 usuarios concurrentes pero llamando a los endpoints `GET /api/visits?date=...` y `GET /api/visits/history`, siempre con el token JWT en la cabecera. En total se generaron 40 lecturas (20 para cada ruta). Los tiempos de respuesta fueron bastante bajos: en ambos casos el promedio se ubicó alrededor de 2 ms, con máximos de 7 ms para el filtro por fecha y 4 ms para el historial, también con 0 % de errores. El throughput en estas rutas se mantuvo en unas 2.2 peticiones por segundo, similar al del login, pero con tiempos muchísimo menores por tratarse de consultas rápidas a la base de datos. De nuevo, todos los resultados se encuentran muy por debajo del rango de 500–800 ms planteado como aceptable para el entorno local.

A partir de estos dos escenarios se puede hacer un análisis cualitativo sencillo: bajo la carga simulada de 10 usuarios, el sistema responde de forma estable, sin errores y sin picos extremos de tiempo de respuesta. El login, que es la operación más "pesada" de las probadas, sigue resolviendo en torno a 60–80 ms, y las consultas de visitas prácticamente responden al instante. Para el contexto del proyecto (una aplicación académica con volumen moderado de usuarios), estos números son más que suficientes y dejan margen para crecer. Como mejora futura, si el sistema se llegara a usar en producción con más tráfico, se podría pensar en aumentar los recursos del servidor o en introducir cacheo para las consultas de listado de visitas, pero con la configuración actual se cumple sin problema el criterio de aceptación definido en el plan de pruebas de rendimiento.

1. Plan de pruebas de seguridad básica

Escenario de seguridad	**Objetivo / Riesgo probado**	**Herramienta / request**	**Resultado esperado / criterio de aceptación**
Login normal (control)	Confirmar el comportamiento esperado del login con credenciales válidas (caso base para comparar).	Postman: POST http://localhost:4000/api/auth/login con body JSON: { "email": "admin@visicontrol.dev", "password": "Admin123!" }.	Respuesta 200 OK, campo ok: true, se devuelve un token JWT y los datos básicos del usuario.
Inyección SQL en login	Verificar que no se pueda saltar la autenticación usando un password malicioso tipo SQL injection.	Postman: POST http://localhost:4000/api/auth/login con body JSON: { "email": "admin@visicontrol.dev", "password": "' OR '1'='1" }.	Respuesta 401 Unauthorized con ok: false y mensaje tipo "Credenciales inválidas"; no se devuelve token.

Inyección SQL en filtro de visitas	Comprobar que el backend no rompe la seguridad ni devuelve todas las visitas cuando se intenta alterar el filtro con SQL.	Postman: GET http://localhost:4000/api/visits?date=2025-12-04' OR '1'='1 con Authorization: Bearer <token válido>.	La API no expone información sensible ni detalles de SQL. Se obtiene un error controlado (por ejemplo 500 "Error interno") sin mostrar la consulta ni el stacktrace.
Acceso a endpoint protegido sin token	Verificar que las rutas protegidas exigen autenticación.	Postman: GET http://localhost:4000/api/auth/me sin header Authorization.	Respuesta 401 Unauthorized con ok: false y mensaje tipo "no token".
Acceso con token inválido	Validar que un token manipulado o inválido no sea aceptado.	Postman: GET http://localhost:4000/api/auth/me con header Authorization: Bearer <token_modificado> (token alterado a mano).	Respuesta 401 Unauthorized con ok: false y mensaje tipo "Token inválido".
Usuario normal accediendo a endpoint de administrador	Comprobar que solo usuarios con rol ADMIN puedan acceder a rutas administrativas.	Postman: obtener token de usuario normal (login); luego GET http://localhost:4000/api/visits/admin con Authorization: Bearer <token_usuario>	Respuesta 403 Forbidden con ok: false y mensaje tipo "Solo administradores".

8.1 Resultado

Con el objetivo de validar los controles de seguridad mínimos de la API de VisiControl, se ejecutó un conjunto de seis casos de prueba sobre los endpoints de autenticación, protección por token JWT y acceso a rutas administrativas. Todas las pruebas se realizaron con Postman, en el entorno local (`http://localhost:4000`).

Caso 1 – Login normal (control)
Primero se probó el inicio de sesión con credenciales válidas del usuario administrador. La petición POST /api/auth/login devolvió respuesta **200 OK**, con ok: true, un token JWT válido y los datos básicos del usuario. Este caso se usa como línea base para comparar el comportamiento cuando se envían credenciales maliciosas.

Caso 2 – Intento de inyección SQL en el password
Se envió de nuevo la petición POST /api/auth/login, pero esta vez el campo password se reemplazó por el payload "' OR '1'='1", típico de inyección SQL. La API respondió con **401 Unauthorized**, ok: false y mensaje **"credenciales inválidas"**, sin devolver token ni detalle

técnico. Esto confirma que la autenticación no se puede saltar mediante este tipo de cadenas y que el sistema trata el input como datos, no como parte de la consulta SQL.

Caso 3 – Inyección SQL en el filtro de visitas
Luego se probó el endpoint de listados con filtro de fecha: GET /api/visits?date=2025-12-04' OR '1'='1, usando un token válido en el header Authorization: Bearer. La respuesta del backend fue un **500 Internal Server Error** con mensaje genérico **"Error interno"**, sin mostrar la consulta SQL ni información de la base de datos. Aunque el código 500 no es ideal desde el punto de vista de usabilidad, desde el punto de vista de seguridad se considera aceptable, porque el sistema **no devuelve datos de más ni expone el detalle del error**. Como mejora futura se podría validar mejor el parámetro y devolver un 400 Bad Request más elegante.

Caso 4 – Acceso a endpoint protegido sin token
Se llamó al endpoint protegido GET /api/auth/me sin incluir el header Authorization. La API respondió con **401 Unauthorized** y mensaje **"no token"**, indicando que el acceso a esa ruta exige obligatoriamente un token JWT. Este comportamiento evita que usuarios no autenticados puedan obtener información del perfil.

Caso 5 – Acceso con token inválido o manipulado
En este caso se reutilizó el mismo endpoint GET /api/auth/me, pero enviando un token alterado manualmente en el header Authorization: Bearer <token_modificado>. La respuesta fue nuevamente **401 Unauthorized**, con mensaje **"Token inválido"**. Esto demuestra que el backend valida correctamente la firma del JWT y no acepta tokens corruptos o modificados.

Caso 6 – Usuario normal accediendo a endpoint de administrador
Por último, se probó el control de roles. Se obtuvo un token de un usuario normal (no admin) y se invocó GET /api/visits/admin con Authorization: Bearer <token_usuario>. La API respondió con **403 Forbidden** y mensaje **"Solo administradores"**, confirmando que este endpoint está limitado a usuarios con rol ADMIN y que no basta con estar autenticado.

Conclusiones de seguridad

Las pruebas realizadas muestran que la API implementa correctamente los controles básicos de seguridad:

- Los intentos de **inyección SQL** en login y en filtros de consulta no permiten saltarse autenticación ni exponen la consulta SQL.
- Las rutas protegidas exigen obligatoriamente un **token JWT válido**; los casos sin token o con token manipulado son rechazados con 401.
- El control de **roles** se cumple: un usuario normal no puede acceder a endpoints de administración, recibiendo 403 Forbidden.

Como mejora, se recomienda refinar el manejo de errores en el filtro de visitas para devolver códigos 4xx (por ejemplo 400 Bad Request) en lugar de 500, manteniendo

siempre mensajes genéricos sin detalles internos. En general, para el alcance del proyecto, los resultados son satisfactorios y evidencian un nivel de seguridad básico adecuado.