Evaluación de Arquitectura

Concerns

Con respecto a lo conversado con el cliente:

1. **Seguridad y privacidad:** Se solicita la autentificación de usuarios para el acceso a la aplicación. Para esto, a través de un login, los campos serán utilizados para comprobar si el usuario había sido o no registrado dentro de la base de datos utilizada en la aplicación. Simplemente se refiere que es necesario ser un paciente o parte del personal para poder utilizar la aplicación: De esta manera, en la pantalla inicial se presentará un pequeño formulario para corroborar la existencia del usuario, donde existe tipificación para que uno no pueda llenarlos de información aleatoria y sin sentido, además que el tipo de inicio de sesión se deriva según el usuario registrado sea un paciente o parte del personal. Como trade-off tenemos una pérdida de eficiencia en la experiencia del usuario ante el tiempo obtenido de necesidad de ingresar y comprobar credenciales para finalmente poder acceder a las funcionalidades principales de la aplicación.

2. **Usabilidad:** Se requiere una interfaz que sea cómoda para los usuarios de la aplicación. Para esto, se ha buscado crear una interfaz limpia junto con funcionalidades reducidas, pero intuitivas, facilitando la experiencia para el usuario. Esta necesidad expresa que los usuarios del sistema deben poder tener la capacidad de realizar sus objetivos con eficiencia, considerando que se sientan a gusto durante la utilización de la aplicación. De esta manera, como desarrolladores, constantemente tenemos que evaluar de manera autocrítica si el objetivo se ha cumplido realmente. Si bien a partir de nuestro propio criterio hemos intentado identificar que en efecto se cumple esta cobertura, es necesaria la intervención del cliente para comprobar su comodidad. El trade-off identificado consta en que nuevas funcionalidades podrían “ensuciar” la armonía de la interfaz usuaria, de manera que podría entorpecer la experiencia promedio del usuario.

3. **Interoperabilidad:** Debe ser capaz de recibir nuevas funcionalidades eficientemente. La aplicación es modularizada en forma de componentes para que se facilite la incorporación de requerimientos entrantes o se pueda dar con mayor facilidad el arreglo o modificación de funcionalidades ya existentes. Al constantemente recibir actualizaciones de requerimientos en forma de sprints, debe existir una adaptación para que el añadir estos no sea una complicación mayor para el equipo. Por detrás, se utilizan los componentes utilizan interfaces bien definidas para comunicarse, y además se implementan pruebas para comprobar las adiciones. Por último, este concern conlleva que el trade-off identificado consista en el aumento de la complejidad del diseño del sistema junto con la necesidad de pruebas exhaustivas, de manera que los componentes requieren mantención de calidad individual para el funcionamiento correcto del sistema.

Iteración de HUs

En resumen, las pruebas desarrolladas en el hito 3 tratan de pruebas unitarias donde se prueban distintas unidades del software en desarrollo para asegurar que estas unidades funcionen correctamente. Las pruebas fueron las siguientes:

Inserción de Horas: Se realizó una prueba en el componente de software llamado addEvent, que inserta horas en la base de datos para poder ser mostradas en el calendario de horas del centro médico. Aquí la prueba se basa en insertar de forma consecutiva 5 horas con campos con la misma información, es decir, una clonación de una hora, entonces se esperaba que solo 1 hora de las 5 insertadas estuviera en la base de datos y en el calendario del centro médico.

- **Contexto:** Una secretaria o parte del personal administrativo por falta de comunicación o al no ponerse de acuerdo con el colocar una hora dentro de sistema para un paciente (Satoru Gojo en este ejemplo) puede insertar la misma hora para la misma persona varias veces, por lo que el sistema debería responder con un aviso de error que el paciente que se está tratando de agendar ya tiene hora agendada dentro del sistema en ese instante de tiempo (Esa misma fecha y hora)

- **Resultados:** El resultado que entregó el software fue la inserción exitosa de las 5 horas duplicadas, lo que es incorrecto. La causa de este posible error es que antes de insertar una hora en específico dentro del sistema, no se verifica si la información a insertar de la hora ya está presente en la base de datos, por lo que solo inserta sin antes revisar.

Una posible solución a este problema es la búsqueda de la hora dentro de la base de datos antes de la inserción, esta búsqueda de la hora se realizaria buscando ciertos campos clave para identificar la hora que se está tratando de insertar, como pueden ser los atributos de rut del paciente, la hora y la fecha, entonces cuando se trate de insertar una hora en el calendario, antes se buscará si hay alguna otra hora dentro de la base de datos con los mismos atributos, si no encuentra nada, se insertará y si encuentra algo, no insertará nada. Esto servirá para evitar los posibles duplicados de hora dentro del sistema para asegurar la y la integridad dentro del software.

Inserción de horas con inputs aleatorios: Se realizó una prueba en el componente de software llamado addEvent, que inserta horas en la base de datos para poder ser mostradas en el calendario de horas del centro médico. Aquí la prueba se basa en insertar una hora con varios tipos de rut diferentes (7 ruts diferentes), donde hay ruts con inputs erróneos y correctos, por lo que se espera que solo la hora con el rut de input correcto pueda ser insertada en la base de datos y en el calendario.

- **Contexto:** Una secretaria o parte del personal administrativo puede equivocarse al anotar el rut del paciente o introducir el rut en un formato incorrecto en varias ocasiones, por lo que el software debe de estar preparado para estas situaciones, avisando de alguna manera que se ha cometido un error al introducir los datos dentro del formulario para agregar horas.

- **Resultados:** El resultado que entregó el software fue la inserción exitosa de las 7 pruebas de inserción de hora, lo que es incorrecto. La causa de este posible error, es que antes de insertar una hora en específico dentro del sistema, no se verifica si los campos o atributos de la hora a insertar han sido escritos correctamente, por lo que inserta sin preocuparse si los typo de los atributos de la hora están correctos.

Una posible solución a este problema es delimitar al usuario de lo que puede escribir, es decir, si el usuario debe escribir un número telefónico solo permitirá que escriba números y no caracteres o letras. En el caso de esta prueba, lo que se realizará de solución para identificar que un rut se escriba correctamente es que el rut tendrá dos secciones para escribir, una donde escriba los dígitos del rut sin guión ni dígito verificador y la segunda sección escriba sólo el dígito verificador. Esto servirá para que se puedan insertar horas sin errores de tipeo asegurando la consistencia y la integridad dentro del software.

Registro Médico: Se realizó una prueba en el componente de software llamado register, que crea un nuevo usuario de tipo médico dentro de la base de datos para que puedan ejercer su profesión de manera correcta en la aplicación. Aquí la prueba se basa en registrar 2 médicos, uno con datos correctos y otro con datos incorrectos (esto en cuanto a tipo de los datos de los médicos), por lo que se espera que solo el médico con los datos correctos sea registrado.

- **Contexto:** Un usuario puede registrarse en el software utilizando el formulario de registro, entonces el usuario puede equivocarse o poner cualquier cosa en los campos que no corresponden, entonces el software debe estar preparado para estas situaciones tanto cuando se ingresan los datos correctamente como cuando se ingresa incorrectamente

- **Resultados:** El resultado que entregó el software fue la inserción exitosa de ambos médicos, lo cual es incorrecto. La causa de este posible error, es que antes de registrar al médico no se revisa si los campos o atributos del usuario a crear han sido escritos correctamente (como puede ser el rut del médico), por lo que si se inserta sin preocuparse si los typo de los atributos del médico están correctos.

Una posible solución a este problema es delimitar al usuario de lo que puede escribir, es decir, si el usuario debe escribir. En el caso de esta prueba, lo que se realizará de solución para identificar que un rut se escriba correctamente es que el rut tendrá dos secciones para escribir, una donde escriba los dígitos del rut sin guión ni dígito verificador y la segunda sección escriba sólo el dígito verificador. Esto servirá para que se puedan registrar médicos sin errores de tipeo asegurando la consistencia y la integridad dentro del software.

Registro múltiple de un médico: Se realizó una prueba en el componente de software llamado register, que crea un nuevo usuario de tipo médico dentro de la base de datos para que puedan ejercer su profesión de manera correcta en la aplicación. Aquí la prueba se basa en registrar 5 médicos con los mismos datos o información, es decir, se están tratando de hacer registro de 5 médicos duplicados, por lo que se espera que se registre correctamente solo 1 médico.

- **Contexto:** Un usuario puede registrarse en el software para utilizarlo, sin embargo, el usuario puede olvidar si se ha registrado en el software con anterioridad, por lo que

es probable que intente registrarse más de una vez . Por lo tanto, el software debe estar preparado para esta situación, y no dejar al usuario tener múltiples perfiles con la misma información.

- **Resultados:** El resultado que entregó el software fue la inserción exitosa de solo un médico con información correcta, lo cual esta perfecto, no existe una falla de este tipo dentro del componente de software, por cual, el componente antes de insertar un nuevo usuario revisa dentro de la base de datos si es que existe un usuario con los mismos atributos.

Para la evaluación de la arquitectura del software desarrollado, primero hay que identificar cómo está estructurado hasta la última versión trabajada, la cual contiene lo siguiente.

Arquitectura General del Software:

- **Frontend:** Utiliza React para la interfaz de usuario con estilos en base a lo desarrollado en css, proporcionando una experiencia dinámica y responsiva.
- **Backend:** Desarrollado en Node.js, maneja el comportamiento del apartado del cliente y la lógica del servidor, con su correspondiente autenticación y las operaciones CRUD para la base de datos.
- **Base de Datos:** MongoDB almacena los datos del personal médico (médico y jefe de unidad), datos de los pacientes, horas agendadas y las solicitudes de horas médicas generadas por los pacientes.

Componentes Principales y su Interacción:

1. **Login y Registro:**
    - **Flujo de Login:** Al iniciar sesión, se verifica si la cuenta existe, si el resultado es positivo, en base a lo estructurado en código determinará el tipo de cuenta para llevarlo a su menú correspondiente, en el caso contrario simplemente lanzará un error de que los datos no están escritos correctamente y se tendrá que intentar de nuevo el login, impidiendo el paso.
    - **Flujo de Registro:** El usuario selecciona si quiere registrarse como paciente o personal médico. Dependiendo de la elección, se guardan diferentes secciones de la base de datos en MongoDB.
2. **Roles y Menús:**
    - **Pacientes:** Tienen acceso a un menú donde pueden ver sus resultados a exámenes médicos y una función de solicitar horas médicas.
    - **Secretarios:** Pueden visualizar las horas agendadas a través de un calendario, además de poder agendar, editar y borrar horas médicas.
    - **Jefes de Unidad:** Tienen todas las funciones de los secretarios y pueden aprobar o rechazar solicitudes de horas médicas de los pacientes.

Ahora que tenemos un resumen general de cómo es la arquitectura del software, podremos evaluar en cómo la arquitectura favorece o no a las pruebas diseñadas e implementadas.

Impactos que favorecen en las pruebas realizadas

- En base a cómo está estructurado el código, da la opción que las pruebas unitarias realizadas no afectan a las demás áreas y funciones del software, así cada componente pudo ser probado de manera independiente.

- Mediante el uso de la base de datos de MongoDB, este último da mayor libertad y flexibilidad a los datos a insertar, facilitando la opción de integrar datos independiente si contiene más o menos atributos de los ya establecidos.

Impactos que perjudican en las pruebas realizadas

- Al utilizar MongoDB que es una base de datos NoSQL, no impone restricciones sobre el tipo de datos a insertar en las pruebas realizadas, por lo tanto permite que los atributos puedan ser diferentes unos a los otros a los ya establecidos dentro de la arquitectura del software, provocando inconsistencias en los datos almacenados, perjudicando la integridad y fiabilidad del software.

Hacer las modificaciones de artefactos y código correspondiente (explique cuáles son esas modificaciones y entregar el rationale que justifica esas decisiones), e incorporarlas a la plataforma como nuevas versiones para dichas HUs.

Modificaciones de los Alphas:

StakeHolders: Hemos avanzado en ciertos puntos del StakeHolders en base a los casos bordes vistos por las pruebas realizadas, además de brindar posibles soluciones con una gestión de que tanto esfuerzo conlleva diseñarlas.

Opportunity: Lo mismo mencionado en la alpha anterior, una vez visto casos bordes, fueron llevadas sus correspondientes soluciones identificando los trade-off necesarios. Todo lo mencionado anteriormente ha demostrado un buen avance en el apartado del cliente para nuestro software.

Way of Working: En este punto hemos avanzado bastante en los términos de cómo trabajamos con el software cada uno, en otras palabras, actualmente el software puede ser accedido por cualquier miembro del equipo, además de que cada integrante sabe como utilizarlo adecuadamente para los casos de prueba y correcta ejecución de la arquitectura implementada, generando nuevas versiones con cambios y documentación de ser necesario.

Modificaciones del Código:

En base a las pruebas realizadas y casos bordes identificados del software, se efectuaron las correspondientes soluciones para solucionar aquellos fallos, provocando cambios dentro del código fuente para una nueva versión del software mucho más completa. Cabe recalcar que esto último no quiere decir que la versión actual del software es aprueba de fallos, sino más

bien nos brinda un abanico de oportunidades y un amplio margen de mejora para que el software pueda escalar a mejores niveles.

- Tenemos el caso del RUT para los usuarios que quieran registrarse como Personal médico o como paciente, independiente del caso el campo RUT admite el ingreso a cualquier carácter que quiera la persona, siendo este último cualquier campo menos el campo RUT como debe de corresponder. Quedando de la siguiente manera a nivel visual:

  La imagen mencionada anteriormente fue llevada a cabo en el código fuente de tal manera que limita al usuario los caracteres que puede usar para rellenar los campos del RUT, limitando el primer bloque a solo caracteres numéricos y el siguiente bloque limitado de igual manera que el anterior y añadiendo el caracter “K”. Acotando al usuario para minimizar posibles fallos al rellenar los campos, asegurando que lo que haya escrito sea un formato de tipo RUT.

- Para los casos donde el software permitía agendar múltiples horas para la misma persona en el mismo dia y hora, se realizó una lógica interna dentro del código fuente para una verificación antes de poder agendar una hora, en otras palabras, al momento de rellenar el formulario para agendar una hora y apretar el botón guardar, aquel gatillador da paso a un verificador por dentro que busca si para la misma persona (con el RUT) existe una hora ya agendada anteriormente para el bloque horario designado en el formulario, en el caso de hallar una coincidencia el software va a denegar el registro de la hora médica. Con la lógica de verificador de la siguiente manera:

```
const checkAvailability = (newEvent) => {
  const newStart = new Date(newEvent.inicio_fecha);
  const newEnd = new Date(newEvent.final_fecha);

  return !events.some(event => {
    const eventStart = new Date(event.inicio_fecha);
    const eventEnd = new Date(event.final_fecha);
    return event.description.rut_paciente === newEvent.description.rut_paciente &&
      eventStart < newEnd && newStart < eventEnd;
  });
};
```

Aquella nueva función extrae y comprueba si es que existe un RUT igual al RUT que se está registrando, y además verifica si que la nueva hora para agendar está dentro de los límites permitidos de la franja horaria que se quiere designar, una vez completada la verificación devolverá un Flag determinando si se puede o no agendar dicha hora médica.

Rationale del registro médico y paciente:

La limitación del formulario en el campo del RUT ayuda a garantizar que los usuarios ingresen la información de manera correcta y de acuerdo con el formato deseado. Al restringir los caracteres que se pueden ingresar, se reduce la posibilidad de errores por parte de los usuarios, lo que puede ayudar a mejorar la precisión y la calidad de los datos recopilados.

Rationale de los múltiples registros para el mismo paciente:

Al crear una función para verificar la disponibilidad del bloque horario, ayuda a prevenir la sobreocupación de horarios para un mismo paciente en un mismo día y bloque horario, lo cual garantiza que cada paciente tenga asignada una única hora médica por bloque horario y evita conflictos en la agenda del centro médico. Al verificar la disponibilidad del bloque horario antes de agendar una cita, se asegura que se cumpla con la lógica y la coherencia en la programación de las horas médicas, evitando posibles errores.