|  |  |
| --- | --- |
| Resultado de imagen de centralita  **Sistema complejo de emergencias (SCE)**  **Captura y Representación de Decisiones de Diseño** | Descripción breve  [Capte la atención del lector con un resumen atractivo. Este resumen es una breve descripción del documento. Cuando esté listo para agregar contenido, haga clic aquí y empiece a escribir.]  Javier Barrio, María Gutiérrez, David Robles, Álvaro Noguerales, Adrián Gómez de Juan y Alex Aguilar.  Diseño y Arquitectura de Software |

1. Roles:

* Arquitectos de Software Seniors: Javier Barrio y María Gutiérrez.
* Arquitectos Software Juniors: Adrián Gómez De Juan y Alejandro Aguilar.
* Arquitectos Software Cognitivos: David Robles y Álvaro Noguerales.

1. Capturas de Pantalla de ADMentor:

|  |  |  |
| --- | --- | --- |
| Hardware | | |
|  | Sensores | El sistema tendrá unos sensores para detectar las emergencias. |
|  | Sensores Información | El sistema tendrá unos sensores que envían la información a un centro de control remoto (BD SCE). |
|  | MTT | El sistema tendrá un modulo de traducción textual simultanea para conectarse internacionalmente en tiempo real y compartir la información. |
|  | Extensión | El sistema debe cubrir una región concreta, incluyendo ciudades, pueblos y costas. |

|  |  |  |
| --- | --- | --- |
| Base de Datos | | |
|  | BD SCE | El sistema tendrá una base de datos llamada SCE que almacena y permite el acceso a los datos. |

|  |  |  |
| --- | --- | --- |
| Interfaz | | |
|  | Interfaz Madre | El sistema tendrá una interfaz genérica que regulará al resto. |
|  | Interfaz Centralizada | El sistema tendrá una Interfaz, la cual, tendrá conexión activa con el resto del sistema |

|  |  |  |
| --- | --- | --- |
| Gestor | | |
|  | G. Emergencias | El sistema tendrá un gestor que regulará las emergencias. |
|  | G. Identificador Llamadas | El sistema tendrá un gestor que identificará las llamadas mediante un número y un audio. |
|  | G. Llamadas | El sistema tendrá un gestor que regulará las llamadas tanto internas como externas. |
|  | G. Rango Llamadas | El sistema tendrá un gestor que organiza las llamadas en una cola de máximo 20. |
|  | G. Distribución de llamadas. | El sistema tendrá un gestor que organiza las llamadas entre los operadores libres. |
|  | G. Eventos | El sistema tendrá un gestor que regulará eventos de manera simultánea. |
|  | G. Notificación Eventos | El sistema tendrá un gestor de eventos que enviará un SMS y una alerta al sistema |
|  | G. Usuarios | El sistema tendrá un gestor de usuarios que regulará un sistema de suscripciones para usuarios. |
|  | G. Notificación Usuarios | El sistema tendrá un gestor de usuarios que les notificará eventos en tiempo real a Smartphones y tablets. |
|  | G. Recursos | El sistema tendrá un gestor de recursos que lepermitirá organizarlos. |
|  | G. Recursos de Usuarios | El sistema tendrá un gestor de usuarios que les permitirá administrar sistemas, operarios de emergencia y de unidades. |
|  | G. Recursos para Emergencias | El sistema tendrá un gestor de recursos que les permitirá asignar recursos a emergencias concretas |
|  | G. Recursos Unidades | El sistema tendrá un gestor de recursos que permitirá asignar unidades activas para una emergencia. |
|  | G. Recursos Algoritmo | El sistema tendrá un gestor de recursos que utilizará un algoritmo para optimizar las rutas y a su vez los recursos. |
|  | G. Orden Eventos | El sistema tendrá un gestor de eventos que los organizará por prioridad. |
|  | G. Asignación Eventos | El sistema tendrá un gestor de eventos que los organizará entre los operarios libres. |
|  | G. Comunicaciones | El sistema tendrá un gestor de comunicaciones para regularlas. |
|  | G. Comunicaciones Policía | El sistema tendrá un gestor de comunicaciones que regulará estas con la policía y la UME mediante cifrado de radio. |

1. Descripción de los resultados para cada tarea:
2. Tabla de Tiempos:
3. Conclusiones en base a lecciones aprendidas:
4. Bibliografía:
5. Anexo con todos los tiempos Estimados:
6. Documentos en GitHub:
7. Arquitecturas producidas en cada Iteración:

**Informe de riesgos**

En este informe se reflejan los posibles riesgos o inconvenientes a tener en cuenta al utilizar los patrones y estilos de diseño elegidos para la arquitectura del sistema.

**-Estilo modelo visa controlador:**

Para desarrollar este modelo es necesario mayores tiempos de desarrollo en los inicios de desarrollo de la aplicación, ya que exige la programación de más clases.

Es necesaria una estructura previa para poder comunicar los diferentes módulos de la aplicación.

Es un estilo de diseño orientado a objetos por lo que implementarlo en un lenguaje que no siga este paradigma podría ser bastante costoso.

La estructura que sigue la de un sistema por capas por lo que podría aumentar la complejidad del sistema.

Si se desea hacer una aplicación web en la que toda la lógica esté en el front-end controlada por un framework de JavaScript como podría ser AngularJS, BackBone o Ember, dejaría de tener sentido utilizar MVC en la parte de back-end.

**-Estilo Rest:**

Usa una URI para leer o escribir un único recurso

Al realizar una solicitud no recibimos solo la información que hemos solicitado, sino todo el conjunto de datos relativos al resource alojado en esa URI.

En un esquema REST puedes tener varios servidores donde unos no saben que los otros existen. No sabes si un usuario ha iniciado sesión en un servidor y si le has enviado ciertos datos. Tampoco sabes realmente en qué servidor puede caer una solicitud.

No se mantiene un estado y eso hace que tengas que montar una infraestructura propia para poder conservar el conjunto de la aplicación. Generalmente mandarás un token que indique quien eres al servidor y qué has realizado en tu aplicación.

Puede producirse en determinadas circunstancias mayor rigidez en el desarrollo, sobre todo al ser dos proyectos independientes, tu back basada en REST y el/los frontales, pueden surgir situaciones de des-sincronización.

**-Estilo Publish Suscribe:**

El envío de mensajes no está programado para enviarlos a usuarios específicos, se envían a un servidor. Los mensajes se clasifican por clases y solo son enviados a los usuarios que se subscriben a esa clase.

**-Estilo por eventos:**

Entrega garantizada. En algunos sistemas es fundamental garantizar la entrega de los eventos.

Procesamiento de eventos en orden o exactamente una vez. Cada tipo de consumidor normalmente se ejecuta en varias instancias, a fin de conseguir resistencia y escalabilidad. Esto puede suponer un desafío si se deben procesar los eventos en orden (dentro de un tipo de consumidor), o si la lógica de procesamiento no es impotente.

**-Patrón Facade:**

Se tiene que considerar el caso en que varios clientes necesiten acceder a subconjuntos diferentes de la funcionalidad que provee el sistema, podrían acabar usando sólo una pequeña parte de la fachada, por lo que sería conveniente utilizar varias fachadas más específicas en lugar de una única global.

**-Patrón Chain of Responsibility:**

No se garantiza la recepción: Dado que las peticiones no tienen un receptor explícito, no hay garantía de que sean manejadas (la petición puede alcanzar el final de la cadena sin haber sido procesada). Una petición también puede quedar sin tratar cuando la cadena no está configurada correctamente (Desventaja).

**-Patrón Comand:**

Aumenta el volumen de nuestro código.

**-Patrón Interpreter:**

Limitación en el tipo de gramáticas (y, por extensión, de problemas) para los que sirve esta aproximación: simples y sin grandes necesidades de eficiencia. Aunque el uso de otros patrones puede mitigar este problema, hay circunstancias en que un Intérprete no es lo más apropiado.

No es conveniente implementarlo si la eficiencia es un punto clave.