



**UNIVERSIDAD DE MENDOZA
FACULTAD DE INGENIERÍA
INGENIERÍA EN INFORMÁTICA**

Trabajo Final de Grado

**QRSec: Sistema de Control de Acceso
de Invitados a Barrios Privados**

Lucas Damián Soria Gava

Asesor Especialista: Mag. Ing. Córdoba, Diego
Profesor a cargo: Dra. Prof. Ruth Leiton

2022

Dedicado a quien quiera utilizar este texto
para aprender, inspirarse o tomarlo
como base para su propio trabajo.

Resumen:

Español: Ante las problemáticas actuales de falta de agilidad y practicidad presentes en los sistemas de control de acceso de personas invitadas a barrios privados o cerrados, el siguiente trabajo integrador propone un nuevo sistema que elimine o mitigue dichas problemáticas.

En este trabajo se describe el proceso completo de creación de un sistema que permita a los residentes de un barrio, poder invitar a personas a sus viviendas de forma inteligente, a través de una aplicación web y con alta disponibilidad.

Palabras clave: *Control de acceso, Aplicación web, API, código QR, Nube, AWS, DNS, Dominio, Git, React, Spring, MongoDB, Docker, Scrum.*

English: In light of the current agility and practicality problems present on guests' access control systems of private or closed neighborhoods, the following paper proposes a new system that eliminates or mitigates those problems.

This paper describes the whole process that involves the creation of a system that allows residents of a neighborhood to invite people to their houses smartly, through a web application and with high availability.

Key words: *Access control, Web application, API, QR code, Cloud, AWS, DNS, Domain, Git, React, Spring, MongoDB, Docker, Scrum.*

Índice general:

Resumen:	III
Índice general:	IV
Índice de figuras:	VII
Índice de tablas:	IX
Introducción:	1
Parte 1: Marco Teórico	2
Capítulo 1: Marco referencial	2
Sección 1: Identificación del problema	2
Sección 2: Justificación del problema	3
Sección 3: Estado del arte	4
Sección 4: Objetivos	6
1.4.1 Objetivo generales:	6
1.4.2 Objetivos específicos:	6
Capítulo 2: Marco conceptual	7
Sección 1: Marco interdisciplinario	7
2.1.1 Control de acceso:	7
Sección 2: Marco tecnológico ingenieril	7
2.2.1 Aplicación web progresiva:	8
2.2.2 API REST:	8
2.2.3 Código QR:	10
2.2.4 Nube:	11
2.2.4.1 AWS:	12
2.2.5 Virtualización:	12
2.2.5.1 Docker:	13
2.2.6 Dominio:	14
2.2.7 DNS:	15
2.2.8 Framework:	16
2.2.8.1 Spring:	16

Parte 2: Marco metodológico	18
Capítulo 1: Desarrollo de ingeniería	18
1.1 Configuración de las tecnologías usadas:	18
1.1.1 Estructura base del proyecto:	18
1.1.2 Diseño gráfico de la página:	18
1.1.3 Frontend:	19
1.1.4 Backend:	20
1.1.5 Base de datos:	21
1.1.6 Herramienta de control de versiones:	23
1.1.7 Despliegue:	24
1.2 Desarrollo:	25
1.2.1 Sprint 1:	27
1.2.1.1 Sprint Planning:	27
1.2.1.2 Diseño Gráfico:	29
1.2.1.2.1 Definición de una paleta de colores:	29
1.2.1.2.2 Pantalla de creación de una Invitación:	29
1.2.1.2.3 Pantalla para compartir la Invitación:	31
1.2.1.2.4 Pantalla de creación de un nuevo Invitado:	32
1.2.1.3 Desarrollo del Frontend:	33
1.2.1.3.1 Desarrollo de las pantallas de creación de Invitación y compartir Invitación:	33
1.2.1.3.2 Desarrollo de la barra superior:	35
1.2.1.4 Desarrollo del Backend:	36
1.2.1.4.1 Asociación de la base de datos con una aplicación de Spring:	36
1.2.1.4.2 Creación de los modelos:	37
1.2.1.4.3 Creación de los primeros endpoint:	41
1.2.1.4.4 Configuración de la política de CORS:	42
1.2.1.5 Sprint Review:	43
1.2.2 Sprint 2:	44
1.2.2.1 Sprint Planning:	44
1.2.2.2 Diseño Gráfico:	45
1.2.2.2.1 Pantalla de visualización de la Invitación:	45

1.2.2.2.2 Pantalla de escaneo de Invitaciones:	46
1.2.2.2.3 Pantalla de inicio de sesión:	47
1.2.2.3 Desarrollo del Frontend:	47
1.2.2.3.1 Desarrollo de la pantalla de visualización de la Invitación:	47
1.2.2.3.2 Desarrollo de la pantalla de escaneo de Invitaciones:	49
1.2.2.3.3 Desarrollo de la pantalla de inicio de sesión:	50
1.2.2.3.4 Modificación de la barra superior:	51
1.2.2.3.5 Desarrollo de la pantalla “Agregar invitado”:	52
1.2.2.4 Desarrollo del Backend:	53
1.2.2.4.1 Implementación de nuevos métodos HTTP:	53
1.2.2.4.2 Creación del endpoint de login:	54
1.2.2.4.3 Modificación de los métodos HTTP existentes:	56
1.2.2.5 Despliegue en la nube:	57
1.2.2.6 Sprint Review:	63
Capítulo 2: Análisis de resultados	65
Conclusiones:	66
Anexos:	68
Anexo 1: Repositorio del proyecto	68
Anexo 2: Dominio de la página web	68
Fuentes bibliográficas:	69
Referencias bibliográficas:	69
Bibliografía:	70

Índice de figuras:

Figura 1: Comunicación entre distintos dispositivos y una API genérica.	9
Figura 2: Estructura del código QR.	10
Figura 3: Comunicación entre un dispositivo y la Nube.	12
Figura 4: Máquinas Virtuales.	13
Figura 5: Aplicaciones contenerizadas.	14
Figura 6: Proceso de búsqueda DNS y petición web.	16
Figura 7: Estructura base de ficheros del proyecto.	18
Figura 8: Estructura de ficheros del Frontend.	20
Figura 9: Estructura de ficheros del backend.	21
Figura 10: Estructura de ficheros de la base de datos.	23
Figura 11: Github, una herramienta de versionado basado en git.	23
Figura 12: Configuración de DNS en GoDaddy.	25
Figura 13: Configuración de DNS en Cloudflare.	25
Figura 14: Framework Scrum.	26
Figura 15: HU establecer el entorno de desarrollo.	28
Figura 16: HU asociación backend - base de datos.	28
Figura 17: HU aprender las bases del framework de React.	28
Figura 18: HU creación de <i>Invitaciones</i> a través de la página web.	28
Figura 19: HU documentación del proceso de creación.	28
Figura 20: Paleta de colores.	29
Figura 21: Diseño de la pantalla base de creación de <i>Invitaciones</i> .	30

Figura 22: Diseño del botón desplegable de selección de <i>Invitados</i> .	30
Figura 23: Diseño de la pantalla para compartir <i>Invitaciones</i> .	32
Figura 24: Diseño de la pantalla de creación de un nuevo <i>Invitado</i> .	32
Figura 25: Implementación de la pantalla de creación de <i>Invitaciones</i> .	34
Figura 26: Implementación de la pantalla para compartir la <i>Invitación</i> .	35
Figura 27: Modelo de la base de datos.	41
Figura 28: HU inicio de sesión.	44
Figura 29: HU aprender a leer códigos QR.	44
Figura 30: HU validar <i>Invitaciones</i> .	45
Figura 31: HU entrar al barrio.	45
Figura 32: Diseño de la pantalla de visualización de la <i>Invitación</i> .	46
Figura 33: Diseño de la pantalla de escaneo de <i>Invitaciones</i> .	47
Figura 34: Diseño de la pantalla de inicio de sesión.	47
Figura 35: Implementación de la pantalla de visualización de la <i>Invitación</i> .	49
Figura 36: Implementación de la pantalla de escaneo de <i>Invitaciones</i> .	50
Figura 37: Implementación de la pantalla de inicio de sesión.	51
Figura 38: HU agregar <i>Invitados</i> .	52
Figura 39: Implementación de la pantalla para agregar nuevos <i>Invitados</i> .	53
Figura 40: HU despliegue a producción.	58
Figura 41: Imprevisto, generar un entorno seguro.	58

Índice de tablas:

Tabla 1: Estructura del documento <i>"User"</i> .	37
Tabla 2: Valores posibles para el campo <i>"Authority"</i> .	38
Tabla 3: Estructura del documento <i>"Guest"</i> .	38
Tabla 4: Estructura del documento <i>"Invite"</i> .	39
Tabla 5: Estructura del documento <i>"Address"</i> .	39
Tabla 6: Atributos de la clase <i>"House"</i> .	40
Tabla 7: Atributos de la clase <i>"Location"</i> .	40

En agradecimiento a la comunidad de desarrolladores mundial,
que me ayudó a lo largo de mi carrera y lo seguirá haciendo.

Gracias a mi familia, por todo el soporte que me dieron
y por haber leído todas las versiones de este documento.

Y gracias a mis amigos y profesores, por
haberme acompañado todos estos años.

Introducción:

En el siguiente escrito, se encuentra la descripción del proceso de creación de un sistema de control de acceso de invitados a barrios privados o cerrados. El desarrollo del mismo se divide en dos grandes secciones, que pasan por diversas etapas que permiten la creación de un MVP (Producto Mínimo Viable, según sus siglas en inglés) de una aplicación web en forma iterativa.

La forma en que el sistema lleva a cabo el control de acceso es mediante la emisión y validación de invitaciones, las cuales son generadas por los habitantes de los barrios, para que sus conocidos, amigos o familiares puedan visitarlos.

Este sistema está compuesto por dos grandes partes o componentes: el componente de creación de invitaciones, disponible para los residentes del barrio, y el componente de verificación de las invitaciones, disponible para los guardias.

Para el desarrollo del mismo se utilizará la metodología ágil Scrum, dividiendo el proceso en dos Sprints. Dentro de cada Sprint, se llevan a cabo actividades que están relacionadas al diseño e implementación del sistema.

Para el final del segundo Sprint y de este documento, el sistema producido será capaz de crear y verificar las invitaciones generadas por los residentes, y estará disponible para su uso accediendo desde un navegador, ya que la última actividad ejecutada será su despliegue en la nube.

Parte 1: Marco Teórico

Capítulo 1: Marco referencial

Sección 1: Identificación del problema

Una situación común para los habitantes de barrios cerrados es querer invitar personas. Para ello deben registrar a sus visitantes para que los guardias de seguridad les permitan el acceso.

En el momento en que los visitantes llegan al complejo, deben transitar un proceso de verificación de identidad y registro del ingreso. Ambas actividades resultan ser lentas e intrincadas; el resultado, una mala experiencia. Entonces, el proceso de visita a un barrio cerrado, está dividido en dos fases: la fase de registro del futuro visitante y la fase de entrada del mismo.

La primera fase requiere que el anfitrión conozca datos personales del visitante como: nombre completo y DNI, a fin de comunicárselo al guardia. En caso de no poseer estos datos, debe consultar con el invitado antes de registrarlo.

Por otro lado, la fase de entrada es más problemática, ya que el personal de seguridad debe corroborar la lista de invitados y debe asegurar la identidad de la persona. Situación que, a menudo, desemboca en llamados telefónicos al residente, consultas sobre los datos del visitante y demoras.

Sección 2: Justificación del problema

A fin de agilizar ambas fases, se propone automatizar el control de acceso de invitados a un barrio cerrado mediante la emisión de un código QR. La solución que se plantea en este proyecto implementa un esquema basado en la nube, donde los residentes del barrio, los guardias de seguridad, y los invitados pueden acceder para poder acelerar el proceso de ingreso.

El servidor remoto responderá las consultas de los usuarios, los cuales tienen acceso a distintos niveles de información sobre el estado de las invitaciones del barrio, dependiendo de su tipo (guardia, residente o invitado). Los residentes pueden crearlas, los invitados pueden ver sus invitaciones e indicaciones para ir a la casa del residente, y los guardias pueden validar la invitación y ver la información de los otros dos roles.

El componente al que tiene acceso el guardia tendrá la tarea de leer el código QR que representa la invitación desde la cámara ubicada en la entrada del barrio. Se utilizará este código porque permite condensar la información necesaria para validar el permiso de acceso, la identidad del invitado y evita que el mismo intercambie información de ingreso al barrio con un tercero malintencionado.

Por otro lado, las invitaciones deben ser altamente configurables, ya que el anfitrión tendrá total control sobre las mismas. Entre la información que administra a través del permiso, se encuentran los días y rangos horarios en que se permite el acceso, cantidad de acompañantes, duración permitida de estadía y cantidad de personas a invitar, a las que se le enviará automáticamente un link para generar el código con su acceso.

Sección 3: Estado del arte

Históricamente, los guardias de seguridad han controlado el acceso de invitados a barrios cerrados consultando una lista física con la información de dichas personas, a las cuales se le debe consultar sus datos personales para contrastarlos con la información contenida en esta.

Posteriormente, por medidas de seguridad para el guardia, se agregó en las barreras de entrada un dispositivo de comunicación por voz, para que estos no tengan la necesidad de aproximarse al vehículo del visitante, ya que se registraron casos de asaltos y violencia hacia los mismos por parte de delincuentes que logran impersonar a visitantes con buenas intenciones.

Sin embargo, los guardias suelen preferir hablar personalmente con los invitados debido a la baja calidad de audio que poseen dichos dispositivos. Además, a menudo los guardias deben llamar telefónicamente al anfitrión para consultar por personas que intentan ingresar y no están registradas, porque las listas suelen estar incompletas debido a errores humanos.

Actualmente existen pocos sistemas competentes de control y registro del acceso de personas a espacios cerrados, la mayor parte enfocados al control del acceso a edificios, y al acceso de clientes por turnos, generalmente en estudios médicos o ámbitos similares. Asimismo, estos sistemas utilizan contraseñas (PINs numéricos o contraseñas alfanuméricas), tokens (tarjetas RFID, JWTs o códigos QR) o tecnologías biométricas (lectores de huellas digitales, lectores de iris o reconocimiento facial).

De los sistemas que controlan el acceso de invitados a barrios cerrados se destacan Control Global, Sevnt, Accessin y Bodet Software. El sistema más maduro lo posee Accessin, sin embargo, al momento de la redacción de este trabajo, sus usuarios no están contentos o satisfechos con el producto, ya que la experiencia de usuario no es ideal. Otro evento que tuvo lugar durante la redacción fue la popularización de estos sistemas, surgiendo nuevas soluciones en diversas localizaciones.

Estas empresas no especifican características sobre la disponibilidad de la información, y para usar sus servicios, es necesario instalar sus aplicaciones nativas.

La solución propuesta en este trabajo plantea una aplicación web progresiva de consultas de invitaciones, montada sobre una infraestructura basada en la nube. Esto permite a los usuarios utilizar el sistema desde cualquier dispositivo con conexión a internet, sin importar el sistema operativo y sin la necesidad de instalar ninguna aplicación. Adicionalmente, estas características facilitan su desarrollo y distribución. En palabras de Tal Ater (2017):

“What started as a simple website has slowly acquired new powers until it was just as capable as any native app on your phone. [...] This new progressive model replaces the binary installed/not installed nature of native apps.”

En cuanto a la base de datos, se utilizará un motor de base de datos NoSQL debido a sus ventajosas características, descritas por Padhy et al. (2015).

Por último, la utilización de códigos QR para la identificación de personas y objetos es una práctica común a través de distintos ámbitos, por ejemplo sistemas de salud (Uzun, 2016; Kieseberg et al., 2010).

Sección 4: Objetivos

1.4.1 Objetivo generales:

Elaborar un sistema informático que controle y registre el acceso de invitados a barrios cerrados.

1.4.2 Objetivos específicos:

- Elaborar una plataforma que permite crear y consultar invitaciones en el sistema con distintos niveles de autoridad.
- Crear un sistema que permita un rápido ingreso del invitado al barrio, sin la necesidad de intercambiar información con el guardia de seguridad.
- Producir una plataforma que interactúe con una cámara y lea los códigos pertenecientes a las invitaciones.

Capítulo 2: Marco conceptual

Sección 1: Marco interdisciplinario

2.1.1 Control de acceso:

“El control de acceso es el proceso de mediar cada petición de recursos o datos mantenidos en un sistema y determinar si dicha solicitud debe ser otorgada o denegada. La decisión de control de acceso es reforzada por un mecanismo que implementa regulaciones establecidas por una política de seguridad”. Samarati, P., de Virmercati, S. C. (2001). Representa una limitación a las acciones que puede realizar un usuario sobre un recurso específico para evitar violaciones de seguridad.

El modelo de seguridad es una representación de la política de seguridad y su funcionamiento, que permite probar las propiedades de seguridad previstas por el sistema de control de acceso. El mecanismo de seguridad define el funcionamiento, la implementación de los controles establecidos por la política y descritas por el modelo. Entre los modelos más comunes de control de acceso se encuentra la Matriz de Acceso, cuya implementación más común es la Lista de Control de Acceso (ACL). Esta lista asocia a cada *sujeto* con los *recursos* a los que posee acceso y es la implementación más usada actualmente en los sistemas de control de acceso a barrios cerrados, donde los sujetos son los dueños de casa y los *recursos* representan a los invitados.

Sección 2: Marco tecnológico ingenieril

Los conceptos, terminologías y paradigmas explicados a continuación tienen influencia directa sobre la realización del sistema de control de acceso planteado en las secciones anteriores y serán explicados uno a uno, para luego justificar su importancia para el desarrollo de este sistema.

2.2.1 Aplicación web progresiva:

Una aplicación web progresiva es un tipo de aplicación que no necesita ser instalada ya que está diseñada y construida para su uso en un navegador web. Funciona como una página web que se adapta al dispositivo que la está utilizando en ese momento. Están pensadas para reducir la cantidad de aplicaciones instaladas en los teléfonos móviles, dando a sus usuarios una experiencia entre ambos mundos, sin perder los beneficios que aportan individualmente.

A su vez, las aplicaciones web progresivas ofrecen disponibilidad de información sin importar la conexión a internet, rápidos tiempos de carga, envío de notificaciones y accesos directos en la pantalla de inicio, características que antes solo poseían las aplicaciones nativas tradicionales.

2.2.2 API REST:

API o Application Programming Interface, es un conjunto de definiciones y protocolos para construir e integrar aplicaciones de software. Le permite a cada programa o servicio poder comunicarse con otros programas o servicios, permitiendo así la reutilización de código, que desemboca en la simplificación del desarrollo de aplicaciones, ahorrando tiempo y dinero, a la vez que flexibiliza y simplifica el uso de dicha aplicación. Esta flexibilización permite una rápida adaptación al entorno de necesidades cambiantes de la industria, lo que a su vez le permite mantenerse competitiva.

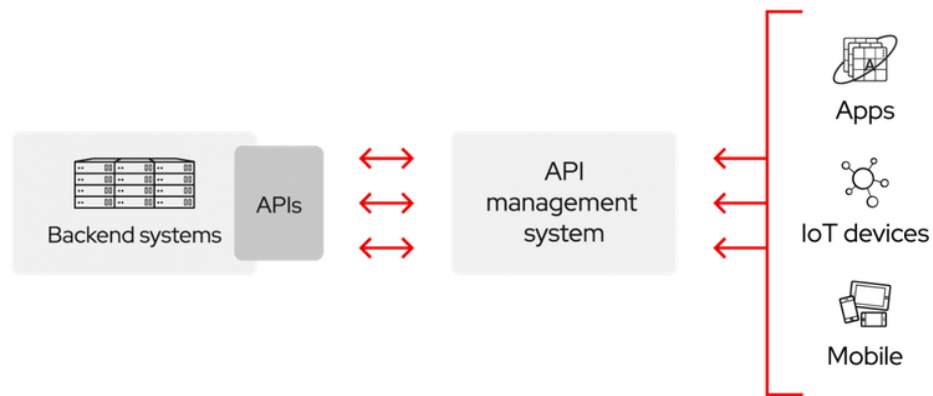


Figura 1: Comunicación entre distintos dispositivos y una API genérica.

Fuente:

<https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces>

Cuando las APIs se encuentran en dispositivos distintos al que realizó la solicitud de información, se obtiene una API remota, pensada para operar a través de redes de comunicación. Internet es la red de comunicación más usada en la actualidad, por lo que es natural que la mayoría de APIs se diseñen pensando en su uso en la web. Las APIs web utilizan el protocolo HTTP (Hypertext Transfer Protocol) para definir una estructura de envío de mensajes en la forma de archivos XML (Extensible Markup Language) o JSON (JavaScript Object Notation).

Las RESTful APIs son APIs que se adhieren a las restricciones arquitecturales REST (Representational State Transfer), lo que quiere decir que cumple con: una arquitectura cliente-servidor, es *stateless* (el servidor no guarda información sobre el estado de la sesión), cacheable, un sistema en capas (cache, seguridad, interacciones cliente-servidor) y una interfaz uniforme.

En el sistema planteado se utiliza una API REST para comunicar el Backend con el Frontend a través de archivos JSON, con el objetivo de que el último muestre información al usuario o envíe al servidor la información generada por el cliente.

2.2.3 Código QR:

Un código QR o código de respuesta rápida (Quick Response) es un código de barras de dos dimensiones cuya información está codificada tanto en la dirección horizontal como en la vertical, lo que le permite almacenar mucha más información que un código de barras tradicional. La información que estos almacenan se puede acceder a través de una imagen del mismo y un programa lector de códigos QR.

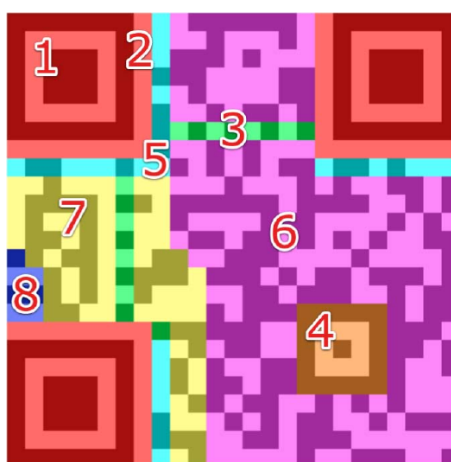


Figura 2: Estructura del código QR. Fuente: Kieseberg, P., et al. QR Code Security (2010)

La información almacenada en su imagen está dividida en siete partes o áreas. Estas están dedicadas a:

1. Poder encontrar el patrón o los límites del código
2. Separación entre el código y la primera área.
3. Patrón de sincronización que indica el largo de cada recuadro.
4. Patrones de alineamiento, para corregir distorsiones en la imagen.
5. Información del formato de la carga.
6. La carga o información almacenada.
7. Módulos de corrección de errores.
8. Bits vacíos de relleno en caso de que la información y los bits de corrección de errores no haya sido posible dividirlos en bloques de 8 bits.

La corrección de errores se basa en códigos de Reed-Solomon y permite una corrección de hasta el 30% del código. A medida que se aumenta el nivel de corrección de errores, disminuye la cantidad de módulos usables para codificar información.

La solución propuesta utiliza estos códigos para representar las invitaciones y permitir un escaneo rápido de la misma. La corrección de errores utilizada en este trabajo es del 30%, es decir, la máxima capacidad de corrección de errores, esto es posible debido que se guarda muy poca información en cada una de las imágenes y no se alcanza el límite de módulos utilizables. Por último, utilizar este tipo de códigos facilita el intercambio de información sin la necesidad de exponer información sensible del invitado a cualquiera que pueda tener acceso a la información.

2.2.4 Nube:

“La definición de la nube puede parecer poco clara, pero, básicamente, es un término que se utiliza para describir una red mundial de servidores, cada uno con una función única. [...] Estos servidores están diseñados para almacenar y administrar datos, ejecutar aplicaciones o entregar contenido o servicios, como streaming de vídeos, correo web, software de ofimática o medios sociales.” Azure (recuperado 2022).

Son un conjunto de servidores, software y bases de datos a los que se puede acceder a través de internet, evitando la necesidad de que sus clientes tengan que correr los programas localmente y permitiendo que puedan acceder a su información en cualquier parte del planeta, desde casi cualquier dispositivo. Existen nubes públicas (accesibles por todo el mundo), comunitarias (comparten recursos sólo entre organizaciones), privadas (accesible a través de una red privada) e híbridas.

La nube híbrida combina nubes públicas y privadas. Una organización puede utilizar su nube privada para algunos servicios y la

nube pública para otros, o puede emplear una como copia de seguridad de la otra.

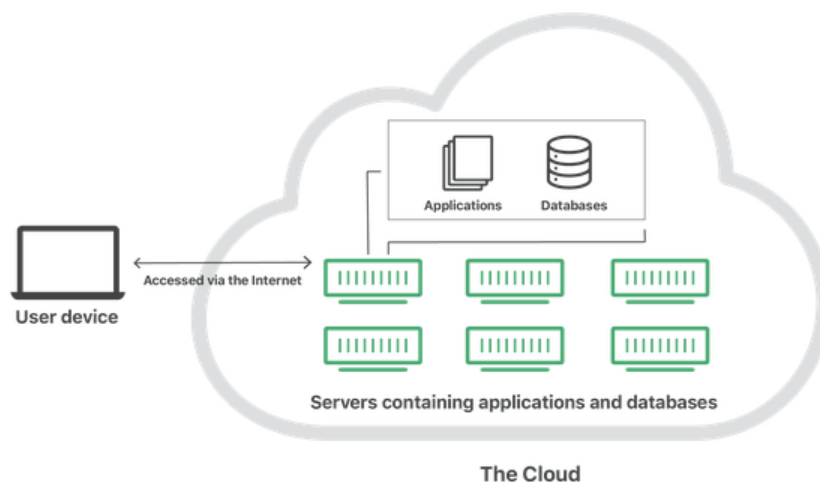


Figura 3: Comunicación entre un dispositivo y la Nube. Fuente: <https://www.cloudflare.com/es-es/learning/cloud/what-is-the-cloud/>

2.2.4.1 AWS:

AWS (Amazon Web Services) es una de las plataformas Cloud más exhaustivas y adoptadas a nivel mundial. Entre sus servicios ofrece soluciones de infraestructura, almacenamiento, tecnología de inteligencia artificial, IoT (Internet of Things), seguridad, etc. que le permiten tanto a desarrolladores individuales como empresas poder ejecutar casi todas sus aplicaciones sobre este servicio y tener disponibilidad del mismo a nivel mundial, con un downtime mínimo.

Para el despliegue de la aplicación web final se utilizarán los servidores de nube pública de AWS, haciendo uso de sus instancias gratuitas de máquinas virtuales.

2.2.5 Virtualización:

“La informática en la nube es posible gracias a una tecnología conocida como virtualización. La virtualización permite la creación de un ordenador virtual, simulado y digital que se comporta como si fuera un

ordenador físico con su propio hardware. El término técnico para este ordenador es máquina virtual.” Cloudflare (recuperado 2022).

La virtualización gracias al hipervisor permite la ejecución de múltiples máquinas virtuales en el mismo hardware, permitiendo así un uso más eficiente de los recursos, siendo la razón por la que los proveedores de Cloud pueden ofrecer el uso de sus servidores a muchos clientes simultáneamente a un bajo costo.

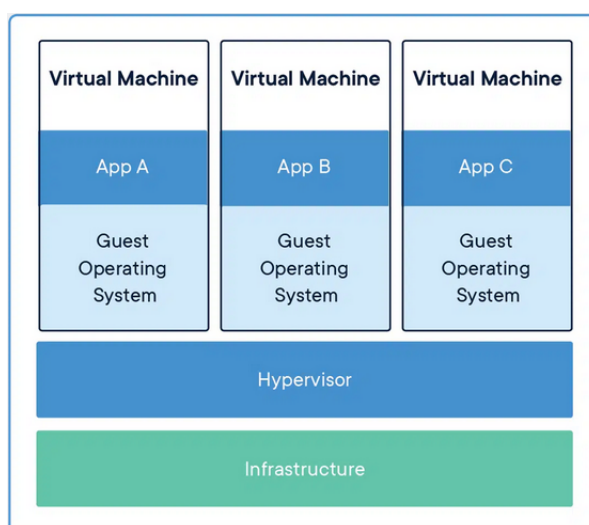


Figura 4: Máquinas Virtuales. Fuente: <https://www.docker.com/resources/what-container/>

2.2.5.1 Docker:

Los contenedores son una unidad estándar de software que empaquetan código y sus dependencias para que la aplicación corra de forma rápida y confiable. Aunque están aislados unos de otros, entre ellos comparten el kernel de sistema operativo y permiten migrar, abrir y utilizar en las configuraciones de desarrollo, pruebas y producción de una aplicación. Los contenedores son livianos, seguros y estandarizados. Docker es una plataforma open source para desarrollar, montar y correr aplicaciones en contenedores.

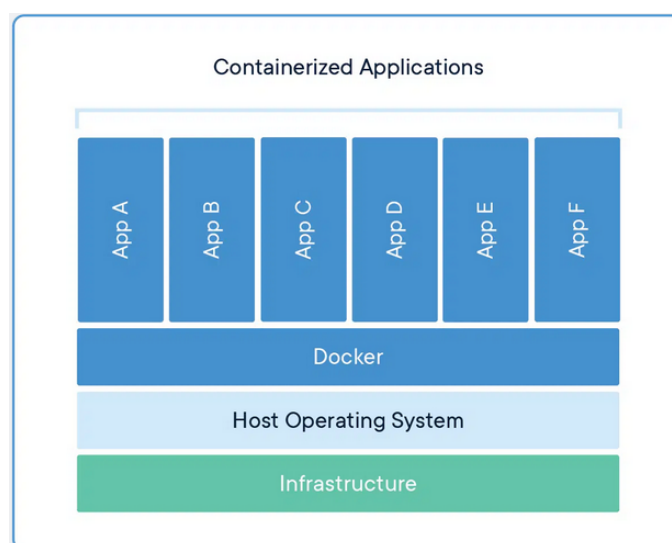


Figura 5: Aplicaciones contenerizadas. Fuente: <https://www.docker.com/resources/what-container/>

Docker es la herramienta de virtualización utilizada en este trabajo para desplegar los servicios estandarizados como la base de datos de forma rápida y sencilla.

2.2.6 Dominio:

Un dominio es una dirección fácil de recordar usada para acceder a sitios web, como por ejemplo *"Isoria.com"*. Está enlazada a una dirección IP (Internet Protocol), donde se encuentra el servicio solicitado. Los usuarios pueden usar estos textos fáciles de comprender gracias a los DNS (Domain Name Server) que rutean los dominios a dichas direcciones IP.

Los dominios son manejados por registros de dominio, que delegan la responsabilidad de reserva de dichos dominios a los registradores de dominios.

La diferencia entre un dominio y una URL (Uniform Resource Locator) radica en que esta última es la combinación de un protocolo, un dominio y una dirección de un recurso. Por ejemplo, en la URL ["https://Isoria.com/invite/view/62b435465c3c484689b487ef"](https://Isoria.com/invite/view/62b435465c3c484689b487ef) hace referencia a una invitación específica (recurso), *"https"* es el protocolo,

“Isoria.com” el dominio y *“/invite/view/62b435465c3c484689b487ef”* es la ruta a dicho recurso.

Los dominios están compuestos generalmente por dos o tres partes separadas por un punto. Leyendo de derecha a izquierda, nos encontramos con el Top Level Domain (TLD), cuya funcionalidad es clasificar los dominios. A medida que se siguen leyendo las distintas porciones del dominio, se pueden encontrar dominios de segundo nivel (2LD), de tercero (3LD), etc.

2.2.7 DNS:

DNS o Domain Name System es un registro de nombres de dominio y direcciones IP. Las personas navegan por internet a través de los dominios, pero los navegadores se comunican utilizando el protocolo IP. Los DNS traducen de dominios a direcciones para que la navegación por internet sea posible, sin la necesidad de recordar todas las direcciones IP de los sitios que se utilizan en el día a día.

Los navegadores cada vez que se comunican con un dominio no cacheado (cuya dirección IP desconocen) deben realizar un proceso conocido como búsqueda DNS, y una vez conocen la dirección que corresponde a dicho dominio, se comunican con él directamente, a través de su IP.

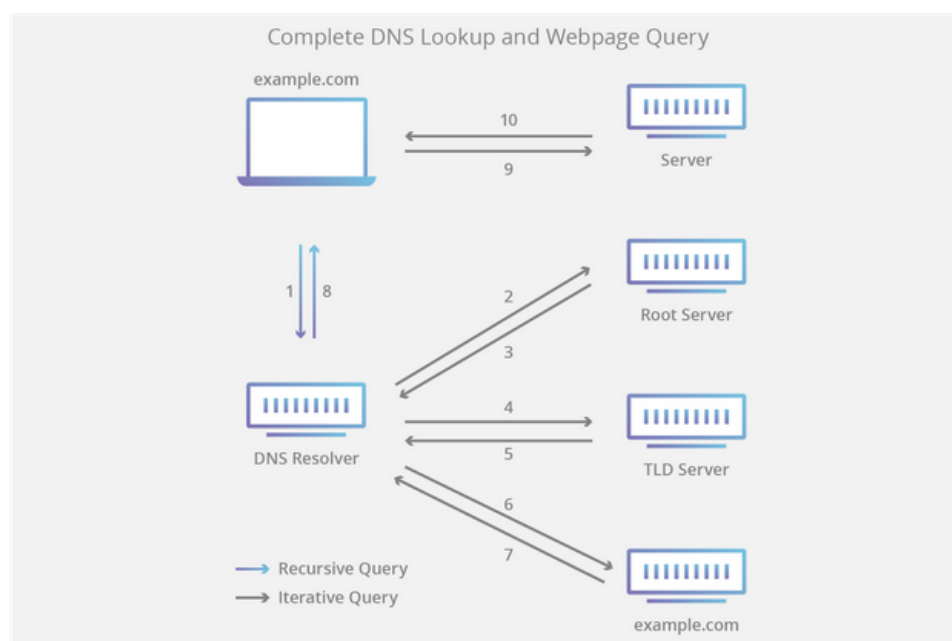


Figura 6: Proceso de búsqueda DNS y petición web. Fuente: <https://www.cloudflare.com/en-gb/learning/dns/what-is-dns/>

2.2.8 Framework:

Los marcos de trabajo o frameworks son soluciones de software desarrolladas y usadas por desarrolladores para construir una estructura estandarizada en otras aplicaciones. Resuelven problemas comunes, establecen una base sólida que sigue las *buenas prácticas* de la industria, son robustos, eficientes, seguros y versátiles.

2.2.8.1 Spring:

Spring es un framework open source que provee múltiples modelos de programación y configuración exhaustivos para aplicaciones web basadas en el lenguaje Java y desplegadas en cualquier tipo de plataformas.

Al resolver las características de más bajo nivel, Spring permite a las empresas y desarrolladores enfocarse en la lógica de negocio a nivel de aplicación. Spring facilita y se utiliza especialmente para el desarrollo del backend, manejando su comunicación con la base de datos y el desarrollo de infraestructuras tanto monolitas como microservicios.

2.2.8.2 React:

React es un framework que en realidad no es un framework, sino una librería open source de JavaScript usada para el desarrollo de frontend, interfaces de usuario (UI). La misma permite renderizar componentes que se actualizan en el cliente, por lo que la página no necesita refrescarse.

La razón por la que es una librería y no un framework es que los frameworks definen la forma en la que los desarrolladores deben diseñar sus aplicaciones, mientras que una librería es un conjunto de funciones que una aplicación puede llamar para realizar distintas tareas. Esto le da su flexibilidad característica, razón por la que se popularizó, junto a su velocidad y su amena curva de aprendizaje. Sin embargo, se lo suele considerar un framework ya que existen buenas prácticas que llevan a los programadores a desarrollar aplicaciones de una manera común.

Parte 2: Marco metodológico

Capítulo 1: Desarrollo de ingeniería

1.1 Configuración de las tecnologías usadas:

1.1.1 Estructura base del proyecto:

El código se encuentra dividido en 3 directorios principales: frontend, backend y base de datos. De esta forma todo lo desarrollado de la aplicación web se encuentra contenida dentro de la capa a la que pertenece en el stack.

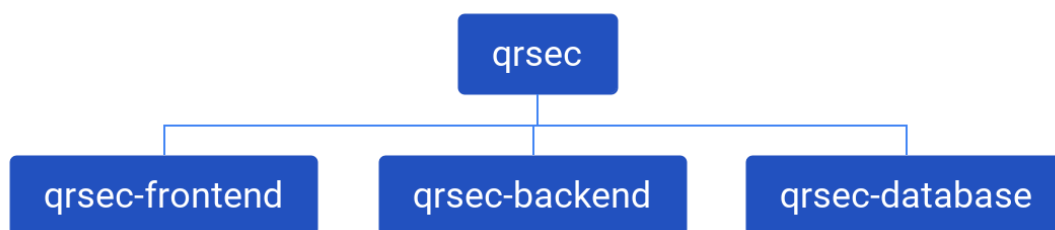


Figura 7: Estructura base de ficheros del proyecto. Fuente: Imágen de elaboración propia.

Luego la relación entre estas tres capas se da también en ese orden, el frontend se relaciona con el backend a través de peticiones a la API, y este a su vez se relaciona con la base de datos.

1.1.2 Diseño gráfico de la página:

Para tener una noción del diseño y la distribución del contenido de las páginas web que se desarrollaron, se usó la herramienta de diseño UX/UI Marvel. El diseño se pensó desde un principio como una guía para el desarrollo y no como un estilo fijo, por lo tanto, las imágenes que se produjeron son aproximaciones del desarrollo final. Además, para que el desarrollo de las mismas siga un diseño de colores uniforme a lo largo de todas sus pantallas, se definió una paleta de colores, la cual se hizo con la ayuda de la herramienta web colors.

1.1.3 Frontend:

El frontend está principalmente desarrollado en el lenguaje de programación JavaScript, con npm como su manejador de paquetes, para instalar las librerías necesarias. Las librerías más usadas fueron las de React para crear las distintas funcionalidades de la aplicación web, Material UI (MUI) para darle el estilo a las mismas y react-qr-code para poder crear el código QR de la *Invitación*.

El código principal se encuentra dentro del directorio *"src"*, dividido en los subdirectorios *"components"*, *"data"* y *"pages"*, cada uno contiene diferentes tipos de ficheros, dependiendo de las funciones que desempeñen.

El directorio *"pages"* contiene los componentes de React de más alto nivel, que simbolizan las páginas que los diferentes tipos de *Usuarios* podrán ver. El directorio *"components"* contiene los elementos de código que *componen* a cada página. El directorio *"data"* contiene todo el código que manipule información, principalmente llamadas al backend.

Por otro lado, ficheros como Dockerfile y Jenkinsfile tienen el propósito de en un futuro poder configurar, crear y publicar una imagen de Docker automáticamente cuando se haga un *commit* a GitHub.

Las dependencias del proyecto se encuentran dentro del fichero *"package.json"*, el cual es generado automáticamente por *npm*.

Dentro de los archivos *.env.development* y *.env.production*, conocidos como archivos *"dotenv"*, se encuentran las constantes de la aplicación que son dependientes de los entornos de desarrollo y producción correspondientemente.

Por último, la versión de producción de la aplicación se encuentra en el directorio *"build"* y dentro de *"public"* se encuentran disponibles los archivos estáticos de la misma, junto con los archivos de estilos, entre los cuales se encuentran las modificaciones a los componentes de MUI.

Arbitrariamente para la programación del proyecto se decidió utilizar Visual Studio Code, uno de los IDEs más populares actualmente.

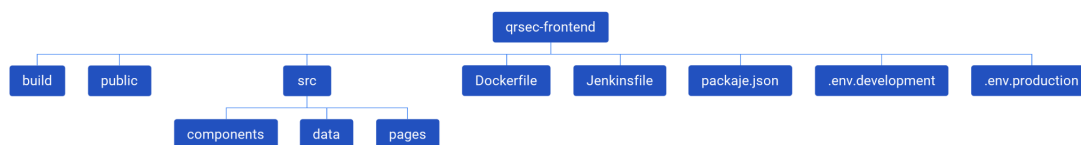


Figura 8: Estructura de ficheros del Frontend. Fuente: Imágen de elaboración propia.

1.1.4 Backend:

El desarrollo del backend se basó sobre el lenguaje Java (versión 11), utilizando el framework de Spring. Junto con Maven, una herramienta de software para la gestión y construcción de proyectos Java. La dependencia usada más importante fue `spring-boot-starter-data-mongodb`, que permitió la comunicación del backend con la base de datos.

Dentro del directorio `qrsec-backend` nos encontramos con una estructura de archivos común para un proyecto de Spring. Los archivos principales en este directorio son: `pom.xml`, `Jenkinsfile`, y `application.properties`. El primer archivo contiene todas las dependencias del proyecto, el segundo da la posibilidad de que en un futuro se defina el procedimiento que la herramienta Jenkins debe seguir para poder hacer una imagen de Docker a partir del proyecto y poder publicarla a DockerHub automáticamente luego de un commit en GitHub, y el archivo restante contiene las constantes de la aplicación que son dependientes de los entornos, como en el frontend lo son los archivos `dotenv`.

La estructura de directorios elegida fue una compuesta por seis carpetas principales ubicadas dentro de `src`: `config`, `controller`, `domain`, `repository`, `security` y `service`. Dentro del directorio `config` se encuentran las clases encargadas de las configuraciones de la aplicación, como por ejemplo, las políticas de CORS, `domain` contiene las clases que representan los documentos de la base de datos (*models*) y la información de transferencia entre la api y el unco externo (*DTOs*), `repository` contiene solicitudes más complejas a la base de datos o que

no estén incluidas en la librería usada, *service* contiene la lógica detrás de cada endpoint de la API, y por último, *controller* expone dichos endpoints, los métodos HTTP que permite y los permisos necesarios para acceder a estos, los cuales son controlados por la capa de *security*.

El directorio *target* contiene entre otras cosas la versión compilada para producción de la aplicación.

Para el testeo de la API desarrollada se utilizó Insomnia, un cliente para APIs diseñado para el testing de las mismas.

Por último, el IDE seleccionado para el desarrollo fue IntelliJ IDEA, desarrollado por JetBrains y diseñado para el desarrollo en Java.

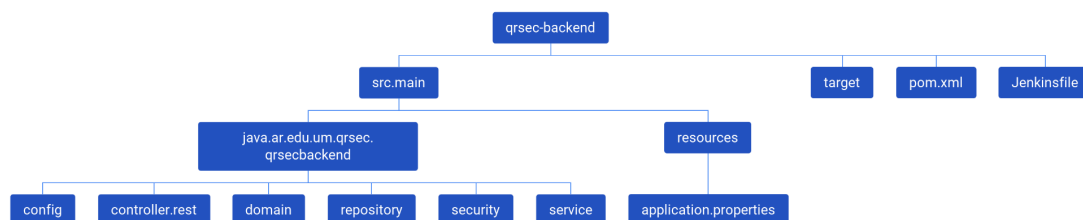


Figura 9: Estructura de ficheros del backend. Fuente: Imágen de elaboración propia.

Las compilaciones de la aplicación tanto para el backend como para el frontend siguen el standard de versionado “X.Y.Z”, donde:

- X: Cambios mayores.
- Y: Cambios menores.
- Z: Patches o fixes.

1.1.5 Base de datos:

El motor de base de datos utilizado para el proyecto es MongoDB. Debido a que el sistema operativo en el que se desarrolló la aplicación está basado en una distribución de Arch Linux, existen problemas de compatibilidad con el motor de la base de datos, y no es posible utilizar establecer uno localmente para el desarrollo. Por esta razón, se decidió virtualizar la misma con Docker, a continuación se presenta el código para la configuración de la misma.

```

version: '3'
services:
  mongodb:
    image: mongo
    container_name: mongodb
    ports:
      - 27017:27017
    volumes:
      -
        /home/lucas/Documents/qrsec/qrsec-database/data:/data
    environment:
      - MONGO_INITDB_ROOT_USERNAME=root
      - MONGO_INITDB_ROOT_PASSWORD=password
    restart: unless-stopped
  mongo-express:
    image: mongo-express
    container_name: mongo-express
    ports:
      - 8081:8081
    environment:
      - ME_CONFIG_MONGODB_ADMINUSERNAME=root
      - ME_CONFIG_MONGODB_ADMINPASSWORD=password
      - ME_CONFIG_MONGODB_SERVER=mongodb
    restart: unless-stopped

```

En este archivo `docker-compose.yml` se puede apreciar la configuración de dos servicios, el primero es el motor de base de datos junto con la configuración de las credenciales necesarias para ingresar al mismo. El segundo servicio es un contenedor de *mongo-express*, el cual posibilita explorar las bases de datos presentes en el contenedor anterior a través de una interfaz gráfica en la web, especialmente útil para la etapa de desarrollo.

El primer servicio (mongodb) presenta un volumen para poder persistir tanto las bases de datos como sus configuraciones, siguiendo el sistema de ficheros presentes en la Figura 10. Como se puede observar, tanto la base de datos persistida como el archivo de configuración de los servicios están almacenados en el directorio qrsec-database.

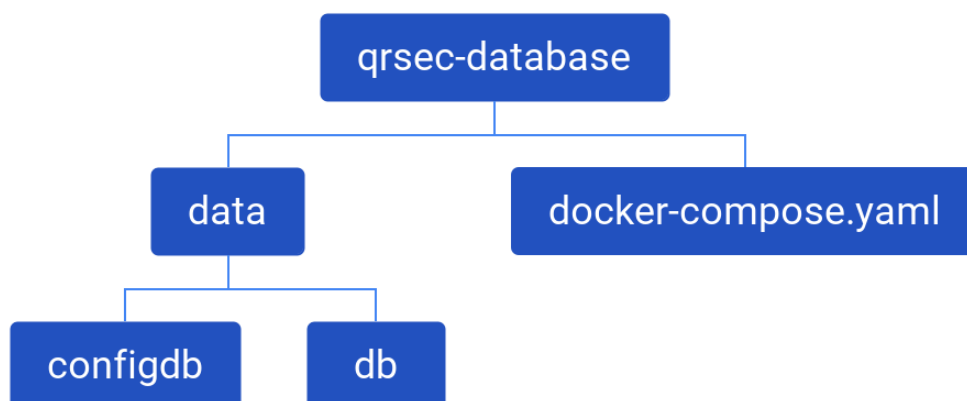


Figura 10: Estructura de ficheros de la base de datos. Fuente: Imágen de elaboración propia.

1.1.6 Herramienta de control de versiones:

Para el control de versiones se utilizó la herramienta Git, junto con la plataforma de desarrollo colaborativo GitHub.



Figura 11: Github, una herramienta de versionado basado en git. Fuente: <https://github.com/bryan2811/Curso-de-Git-y-Github-Platzi>

Para cada mensaje de *commit* se utilizó un estándar propio que es una simplificación de buenas prácticas propuestas por Conventional Commits o utilizadas en la industria para la generación de los mismos. El mismo tiene la siguiente estructura:

- <tipo>: <mensaje>

Donde los **tipos** pueden ser:

- FEAT: Nuevas funcionalidades.
- FIX: Arreglos de fallas o bugs.
- REFACTOR: Cambios en la estructura de archivos del código o renombre de archivos.

El **mensaje** es simplemente un texto que sintetiza y describe los cambios realizados en ese *commit*.

1.1.7 Despliegue:

Una vez que se crearon las versiones de producción del frontend y backend, se las debe alojar en un servidor desde el cual exponerlas al público, es por ello que se decidió utilizar los servicios de Amazon, AWS. Se eligió AWS porque ofrecen un servicio de creación de instancias pequeñas gratuitas (mientras el consumo de recursos sea menor a un límite establecido por ellos) durante el primer año después de crear una cuenta.

Además de un servidor desde el cual ofrecer el servicio de la aplicación, se necesita un dominio y un DNS para que este sea fácilmente accesible. Como registrador de dominios se eligió contratar los servicios de GoDaddy debido a que el precio de compra del dominio por un año era el menor, el dominio comprado fue <http://Isoria.com>. Por último, el DNS seleccionado fue Cloudflare, ya que este ofrece sus servicios gratuitamente, como por ejemplo los servicios de prevención de ataques DoS y DDoS. La configuración del mismo para el dominio mencionado anteriormente se muestra en las figuras N°12 y N°13.

Figura 12: Configuración de DNS en GoDaddy. Fuente: Imagen de elaboración propia.

Type	Name	Content	Proxy status	TTL	Actions
A	Isoria.com	3.208.25.203	Proxied	Auto	Edit ▶
CNAME	_domainconnect	_domainconnect.gd.domainc...	Proxied	Auto	Edit ▶
CNAME	www	Isoria.com	Proxied	Auto	Edit ▶

Figura 13: Configuración de DNS en Cloudflare. Fuente: Imagen de elaboración propia.

1.2 Desarrollo:

Esta sección del trabajo se encuentra dividida en dos grandes etapas: Sprint 1 y Sprint 2. Ambas corresponden con dos iteraciones del marco de trabajo Scrum, el cual fué elegido por las autoridades de la facultad de ingeniería como la metodología sobre la cual desenvolver la fase de desarrollo de las aplicaciones o soluciones propuestas por los estudiantes. Esta fue seleccionada ya que permite a las desarrollar productos de forma ágil e iterativa. En palabras de Schwaber, K., Sutherland, J. (2020):

“Scrum es un marco de trabajo liviano que ayuda a las personas, equipos y organizaciones a generar valor a través de soluciones adaptativas para problemas complejos.”

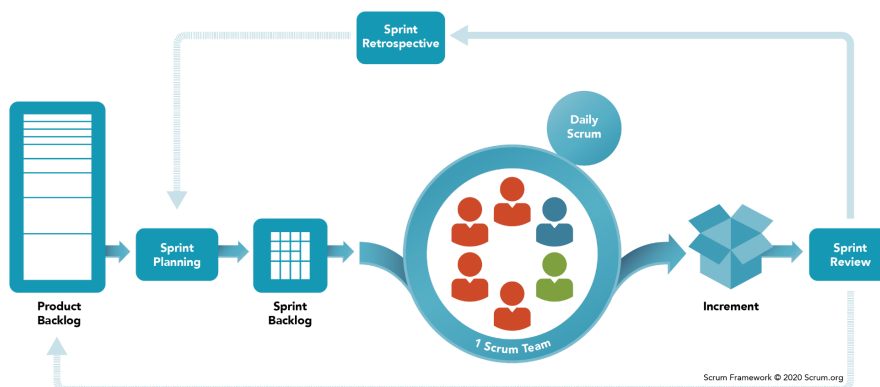


Figura 14: Framework Scrum. Fuente:
<https://www.scrum.org/resources/what-is-scrum>

El marco de trabajo se modificó para acomodarse a las necesidades de la materia de Trabajo Integrador Final III y para comodidad de todas las partes involucradas. Por lo que se dispuso el mismo de forma tal que cada Sprint tuviera una duración de tres (3) semanas y en oposición a la metodología pura, se estableció un periodo de una (1) semana entre cada Sprint.

Los roles de Scrum asimismo se asignaron de la siguiente forma:

- Equipo de desarrollo: Es el mismo alumno, Lucas Damián Soria Gava.
- Scrum Master: El profesor tutor, Mag. Ing. Córdoba, Diego.
- Dueño del producto: También compuesto por el alumno, Lucas Damián Soria Gava, el cual es aconsejado por el tutor a cargo, Mag. Ing. Córdoba, Diego.

A continuación se detalla la implementación de cada uno de los eventos de Scrum:

- Sprint planning: Este evento partía de un acuerdo entre alumno y tutor, donde se definía el alcance máximo del MVP a desarrollar a lo largo de ambos Sprint. Ambos actores establecen el objetivo del Sprint que está empezando, eligiendo y estimando las historias de *Usuario* del backlog del producto que ofrezcan el mayor valor al *Usuario* final lo más rápido posible y generen una

aplicación usable. Así mismo, el tutor bajo el rol de Scrum Master, consulta al alumno sobre dudas o situaciones en las que pueda ayudar, tanto de la metodología como del desarrollo.

- Daily Scrum: Las Daily Meetings fueron modificadas, para que resulten adecuadas a la disponibilidad horaria de los alumnos, ya que a diferencia de los equipos de desarrollo comunes, los alumnos no dedican todo su día solo al desarrollo del Sprint. Las Dailies se reemplazaron por Weeklies, donde se responden a las típicas preguntas de las Dailies y el profesor tutor ofrece su ayuda y consejos basado en su conocimiento y experiencia. Esta ayuda responde tanto a la metodología como al desarrollo del MVP. Del mismo modo, en este evento se re-estiman las historias de *Usuario*, en base a los avances o inconvenientes que se presentaron durante la semana.
- Sprint Review: Se realiza una Demo o demostración de los resultados de cada Sprint. Se analiza lo que se logró en cada Sprint, que imprevistos surgieron, que se objetivos no se cumplieron, y cuál era la expectativa para esta iteración. Por último, en base a estos factores, se vuelve a definir el alcance del MVP.
- Sprint retrospective: Este evento tiene lugar a la vez que el Sprint Review, esa es otra de las modificaciones que se le hicieron a la metodología.

1.2.1 Sprint 1:

1.2.1.1 Sprint Planning:

Como se definió anteriormente, con el objetivo de generar valor para el *Usuario* final lo más pronto posible, se seleccionaron las Historias de *Usuario* planteadas en las siguientes figuras, las cuales componen el Sprint Backlog:

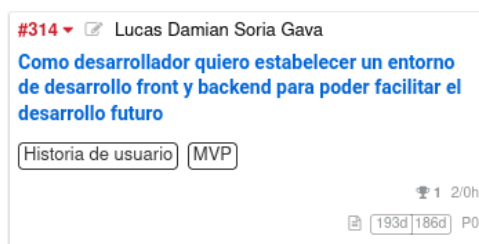


Figura 15: HU establecer el entorno de desarrollo.

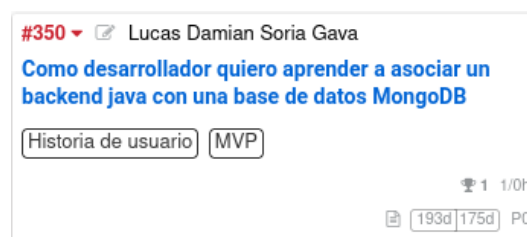


Figura 16: HU asociación backend - base de datos.

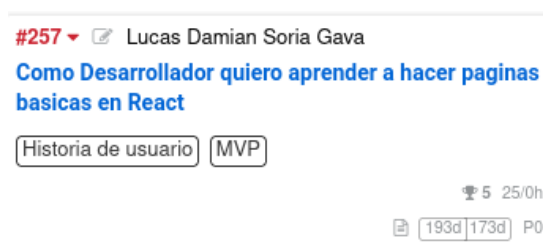


Figura 17: HU aprender las bases del framework de React.

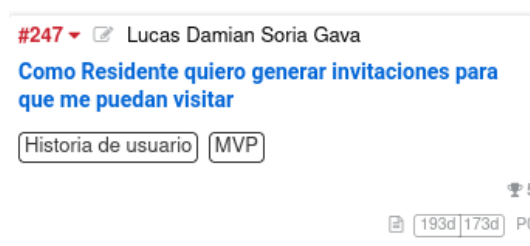


Figura 18: HU creación de *Invitaciones* a través de la página web.

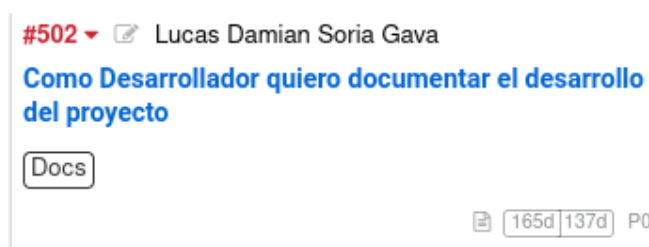


Figura 19: HU documentación del proceso de creación.
Fuente: Imágenes de elaboración propia.

En este primer Sprint, las Historias de *Usuario* (HU) están mayormente orientadas a establecer la base del proyecto y entender el funcionamiento de las tecnologías utilizadas. Sin embargo, este entendimiento se hace a través del mismo desarrollo, por estas HU están relacionadas a la funcionalidad más importante de la aplicación, la generación de *Invitaciones*. Estas están pensadas para que se generen con algunos datos estáticos, como por ejemplo, el *Residente* que la genera. La finalidad de este Sprint es poder generar sostén del MVP.

Es importante destacar la UH dedicada a la documentación del proceso de desarrollo, la cual tiene como fin facilitar la futura redacción de este escrito y el almacenamiento de los recursos que se utilizaron para la programación del sistema.

1.2.1.2 Diseño Gráfico:

1.2.1.2.1 Definición de una paleta de colores:

A partir de la herramienta de diseño mencionada en la sección 1.1.2, y siguiendo los significados de la *teoría del color* y los estándares de UX/UI de la industria, se definieron cinco colores principales presentes en el diseño del producto:

- #EBEBEB
- #6FEC82
- #FF101F
- #2251BF
- #103C89



Figura 20: Paleta de colores. Fuente:

<https://colors.co/ebEBEB-6fec82-ff101f-2251bf-103c89>

1.2.1.2.2 Pantalla de creación de una Invitación:

Las figuras N°21 y N°22 corresponden a la secuencia de acciones requeridas para la generación de nuevas *Invitaciones*. En ellas se incluye el comportamiento del botón desplegable para la selección de *Invitados*,

la selección múltiple de días de la semana y de rangos horarios y por último, los posibles colores que puede tomar el botón de tipo “switch” propio de la opción de “Dejar persona”.

Esta es la pantalla más importante del desarrollo, en la que se concentra la mayor parte del desarrollo y es la que le permitirá al *Usuario Residente* poder crear y configurar nuevas *Invitaciones*.

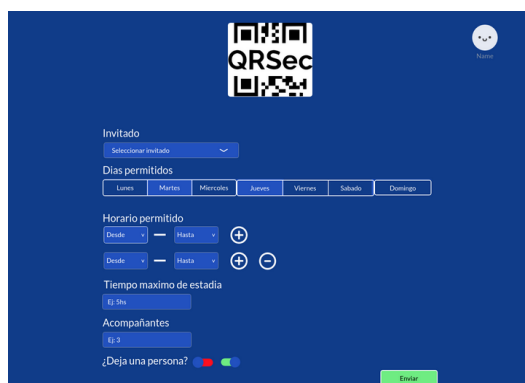


Figura 21: Diseño de la pantalla base de creación de *Invitaciones*.

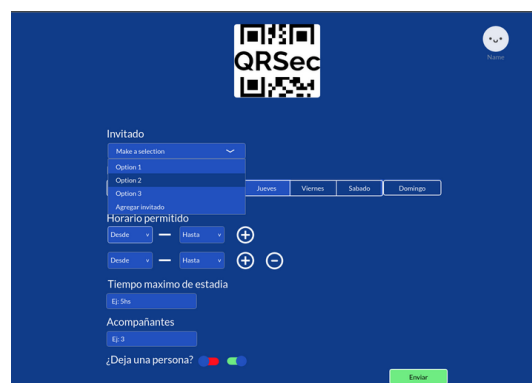


Figura 22: Diseño del botón desplegable de selección de *Invitados*.

Fuente: Imágenes de elaboración propia.

Los campos más importantes son los de “*Invitado*”, “*Días permitidos*” y “*Horario permitido*”:

- *Invitado* es un campo de selección múltiple desplegable que permite elegir precisamente el *Invitado* al que está dirigida la *Invitación*. La lista de *Invitados* coincide con la lista que posee registrada el *Residente* en la base de datos y tiene la opción de crear uno nuevo en caso de que el *Invitado* deseado no se encuentre inscripto en la misma.
- *Días permitidos* también es un campo de selección múltiple, encargado de seleccionar los días de la semana en que el *Invitado* escogido anteriormente puede ingresar al barrio para visitar al *Residente* que emitió esa *Invitación*.
- *Horario permitido* es un campo variable, que tiene la opción de asignar uno o más horarios en los que la persona tiene permitido el ingreso y estadía en el barrio, para todos

los días seleccionados en el campo anterior. Pasado el rango asignado, es responsabilidad del barrio decidir qué hacer con la persona.

- Tiempo máximo de estadía funciona de forma similar a “Horario permitido”, establece la cantidad de horas que el *Invitado* tiene permitida la estadía en el barrio después de su ingreso al mismo.
- Acompañantes es un campo que permite establecer cuántas personas acompañan al *Invitado* que figura en el campo “*Invitado*”. Este es un espacio informativo, principalmente para el *Guardia de Seguridad*, el cual puede tener una noción de la cantidad de personas que debe esperar que ingresen juntas, además de que queda un registro en caso de que pase algún siniestro o percance.
- ¿Deja una persona? es otro campo informativo, en este caso en el formato de un botón “switch”, cuyo rol es dejar por sentado que la persona invitada (generalmente el conductor), tiene como objetivo dejar a otra persona en el barrio y retirarse. Su utilidad se evidencia en eventos como cumpleaños o fiestas, donde padres pueden conducir a sus hijos hasta las *Casas* de sus amigos.

1.2.1.2.3 Pantalla para compartir la Invitación:

Es posible visualizar esta pantalla una vez se generó con éxito una nueva *Invitación*. Es una pantalla simple, ya que solo contiene el link que luego el *Invitado* debe visitar para poder visualizar su *Invitación* y un botón para copiar el mismo al portapapeles.



Figura 23: Diseño de la pantalla para compartir *Invitaciones*. Fuente: Imágen de elaboración propia.

1.2.1.2.4 Pantalla de creación de un nuevo Invitado:

Esta pantalla se puede ver cuando uno hace click en “Agregar Invitado” dentro del menú de selección de *Invitados*. Permite generar un nuevo *Invitado*, el cual está relacionado al *Usuario* que lo crea, es decir, al *Residente* cuya sesión está abierta en ese momento.



Figura 24: Diseño de la pantalla de creación de un nuevo *Invitado*. Fuente: Imágen de elaboración propia.

Cómo se mencionó en la sección 1.2.1.2.2 (Pantalla de creación de una Invitación), esta pantalla permite agregar *Invitados* que no se encuentran registrados en la lista del *Residente*. Los campos que necesita para la creación del *Invitado* son: “Nombre”, “Apellido”, “DNI” y

“Telefono”. En caso de que el *Invitado* ya se encuentre listado en la base de datos del barrio, confirmado a partir del *“DNI”*, simplemente se asocia el *Invitado* existente al repertorio del *Residente*.

1.2.1.3 Desarrollo del Frontend:

1.2.1.3.1 Desarrollo de las pantallas de creación de Invitación y compartir Invitación:

Luego de realizar el diseño de las primeras pantallas se decidió utilizar para el desarrollo del frontend la librería de Material UI (como ya se especificó en la sección 1.1.3) ya que esta ofrece componentes de React muy útiles y estilizados que son fáciles de modificar para que se ajusten a las necesidades del proyecto. Para poder utilizar dicha librería se examinó la documentación de la misma y se consultaron guías y cursos online, los cuales facilitaron el proceso de aprendizaje de ambas tecnologías (tanto React como MUI).

Las modificaciones al estilo de los componentes provistos por la librería de Material UI se encuentran dentro del directorio *“public”*, como se detalla en la sección 1.1.3.

Finalmente se logró implementar una página capaz de:

- Tomar la lista de *Invitados* almacenada en la base de datos a través de la API desarrollada en el Backend y mostrar sus nombres completos para su posterior selección.
- Hacer una selección múltiple de los días de la semana que esa persona tiene permitido ingresar al barrio [opcional].
- Hacer una selección de los rangos horarios donde tiene permitido ingresar al barrio [opcional].
- Establecer un tiempo máximo de estadía una vez ingresó al barrio [opcional].
- Fijar la cantidad de personas que ingresan con el *Invitado* [opcional].

- Establecer si dicho *Invitado* en realidad ingresa al barrio para dejar a una persona, por ejemplo, si es un padre que viene a dejar a su hijo en la casa de un amigo.

Invitado:
Seleccionar invitado
Lucas Damián Soria Gava

Días permitidos:
LUNES MARTES MIÉRCOLES JUEVES VIERNES SABADO DOMINGO

Horario permitido:
15:30 18:30
20:00 22:45

Tiempo máximo de estadía:
Ej: 3hs
3

Acompañantes:
Ej: 3 acompañantes

¿Deja una o más personas?

GENERAR LINK

Figura 25: Implementación de la pantalla de creación de *Invitaciones*.

Fuente: Imágen de elaboración propia.

Una vez se completan los campos deseados, se oprime el botón de “Generar link” y a través de la API se genera una nueva *Invitación* y se muestra una ventana de diálogo que permite copiar el link con la *Invitación* al portapapeles para compartir con el *Invitado*, como lo demuestra la figura N°26.

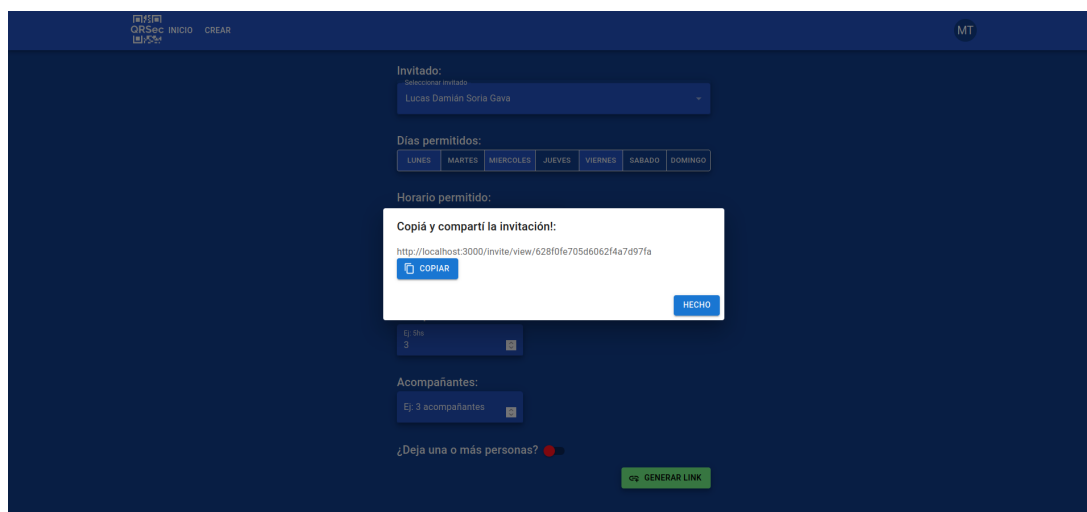


Figura 26: Implementación de la pantalla para compartir la *Invitación*.
Fuente: Imágen de elaboración propia.

Para el desarrollo de estas funcionalidades, se utilizaron los componentes “*Input*” de la librería MUI y se los modificó para que posean una aproximación al estilo diseñado en la sección 1.2.1.2.2. Para la comunicación con el Backend a través de la API, se utilizó el método `fetch()` de JavaScript y los endpoint “`/api/v1/invites`” (POST) y “`/api/v1/guests`” (GET).

1.2.1.3.2 Desarrollo de la barra superior:

Uno de los cambios más grandes que se pueden apreciar con respecto al diseño original es la introducción de una barra superior que permite navegar a las distintas páginas de la aplicación. Además de poseer dichos enlaces, la misma muestra en todo momento el ícono de QRSec y un ícono (por el momento sin funcionalidad) que muestra las primeras dos iniciales del *Usuario* que esté utilizando la página en ese momento.

El componente que sirve de base para este desarrollo es “*AppBar*”, también extraído de la librería MUI y modificado para adecuarlo al proyecto planteado.

1.2.1.4 Desarrollo del Backend:

1.2.1.4.1 Asociación de la base de datos con una aplicación de Spring:

Para poder comunicar la base de datos con la aplicación de Spring se necesitó principalmente realizar la configuración de la misma dentro de las variables de entorno que se encuentran en el archivo *application.properties*, descrito en la sección 1.1.4. Dicha configuración es la siguiente:

```
spring.data.mongodb.authentication-database=admin
spring.data.mongodb.username=root
spring.data.mongodb.password=password
spring.data.mongodb.database=qrsec
spring.data.mongodb.port=27017
spring.data.mongodb.host=localhost
spring.data.mongodb.auto-index-creation=true

spring.security.user.name=root
spring.security.user.password=password

api.path=/api/v1
api.path.invite=/invites
api.path.guest=/guests
api.path.user=/users
api.path.login=/login
api.path.refresh=/token/refresh

server.port=8080
```

Estas variables concuerdan con la configuración de la base de datos definida en la sección 1.1.5. Este tipo de configuración es común en los proyectos donde se integra el framework Spring con las bases de datos MongoDB. Las variables que comienzan con “*api.path*” fueron creadas para su uso dentro del desarrollo de la aplicación, con el

objetivo de facilitar la definición de los endpoints y los controles de los mismo en las políticas de CORS.

1.2.1.4.2 Creación de los modelos:

Para poder relacionar un *Documento* de la base de datos NoSQL y la aplicación de Spring, se debe crear una clase *modelo* que represente dicho *Documento*. Inicialmente se plantearon siete *Clases* que los representan, algunos de ellos son *Documentos* independientes y otros son *Objetos* embebidos. A continuación se listan mostrando su estereotipo, nombre, atributos y tipo de dato:

<Document> User	
email	String (Indexed)
password	String
authority	Authority
firstName	String
lastName	String
dni	String (Indexed)
address	Address (DocumentReference)
phone	String

Tabla 1: Estructura del documento “User”. Fuente: Tabla de elaboración propia.

<Enum> Role
ADMIN
OWNER
GUARD

Tabla 2: Valores posibles para el campo “*Authority*”. Fuente: Tabla de elaboración propia.

<Document> Guest	
firstName	String
lastName	String
dni	String (Indexed)
phone	String

Tabla 3: Estructura del documento “*Guest*”. Fuente: Tabla de elaboración propia.

<Document> Invite	
owner	User (DocumentReference)
guest	Guest (DocumentReferece)
days	List<String>
hours	List<List<String>>
maxTime	Integer
passengers	Integer
drop	Boolean
arrival	LocalDateTime
departure	LocalDateTime
created	LocalDateTime
modified	LocalDateTime

Tabla 4: Estructura del documento “*Invite*”. Fuente: Tabla de elaboración propia.

<Document> Address	
street	String
number	Integer
house	House (Indexed)
location	Location

Tabla 5: Estructura del documento “*Address*”. Fuente: Tabla de elaboración propia.

<Class> House	
block	String
house	Integer

Tabla 6: Atributos de la clase “*House*”. Fuente: Tabla de elaboración propia.

<Class> Location	
type	String
coordinates	List<Double>

Tabla 7: Atributos de la clase “*Location*”. Fuente: Tabla de elaboración propia.

Es importante destacar que cada clase con el estereotipo de *Documento* posee un atributo adicional *id*, el cual lo identifica de forma única dentro de la base de datos.

La última clase especificada (“*Location*”) sigue las especificaciones de la documentación del motor de base de datos MongoDB para una petición geoespacial. En el caso de la aplicación desarrollada, el tipo de localización siempre será “*Point*”, ya que es de interés guardar en ella las coordenadas donde se encuentra ubicada la casa del *Residente*.

La figura N°27 representa gráficamente el modelo de la base de datos que las tablas N°1 a N°7 representaban en forma de clases.

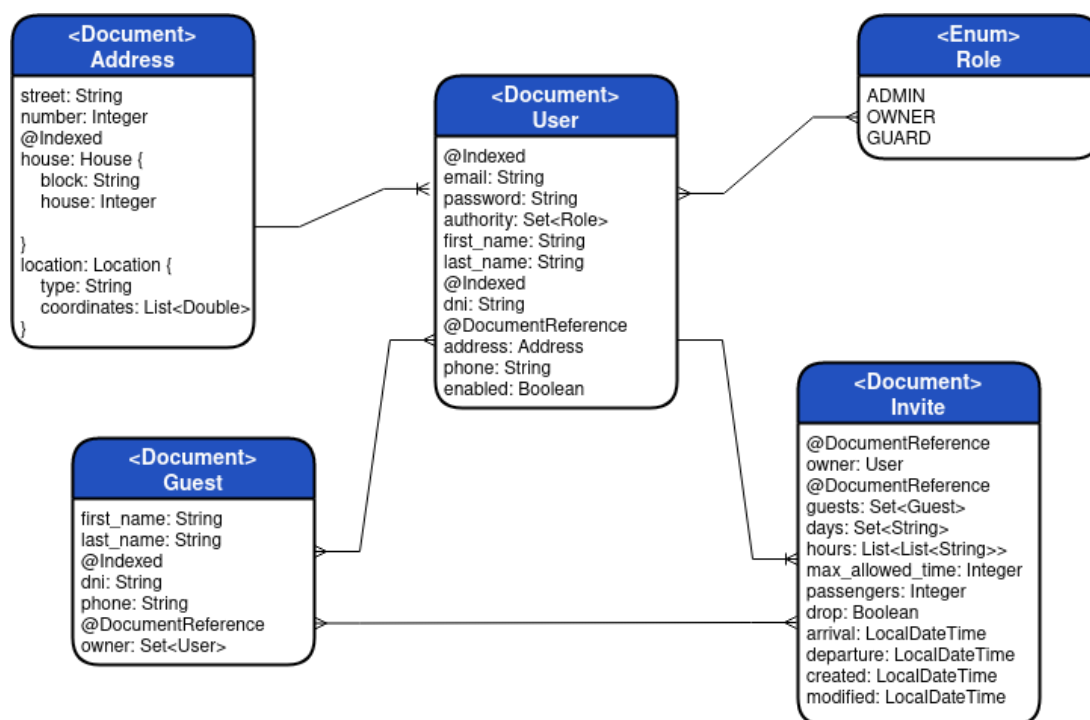


Figura 27: Modelo de la base de datos. Fuente: Imagen de elaboración propia.

1.2.1.4.3 Creación de los primeros endpoint:

Una vez creadas las clases que se relacionan con la base de datos, se debe definir una clase *Repository* por cada una de ellas para que controle los *Documentos* de la base de datos y facilite las peticiones a la misma abstrayéndose del lenguaje necesario para su uso. En el caso de QRSec se lo hizo para las clases: *Address*, *Guest*, *Invite* y *User*.

Posteriormente, se definieron las clases *Controller* para *Guest* e *Invite*. Específicamente se crearon los endpoint de `"/api/v1/guests"` y `"/api/v1/invites"`, el primero tiene definido el método GET, para que la pantalla "Creación de *Invitación*" pueda listar los *Invitados* asociados a ese *Residente* que estén presentes en la base de datos. Como se mencionó al principio del Sprint, este endpoint entrega todos los *Invitados*, sin importar el *Residente* que los consulte porque todavía no se implementa la funcionalidad de las sesiones, por lo que no se puede conocer quién es el que realiza la consulta o que nivel de autoridad posee.

Para el segundo endpoint se definió el método POST para que la pantalla de creación de *Invitaciones* pueda efectivamente generar una nueva *Invitación* y asociarla a su correspondiente *Residente* e *Invitado*.

Para cada método HTTP se creó también la lógica de negocio dentro de su correspondiente método en las clases *Service* pertinentes.

Durante el desarrollo de estos endpoint surgieron problemas de mapeo entre las clases y sus respectivos *DTOs* (Data Transfer Objects), debido a que se está utilizando una base de datos NoSQL, por lo que se decidió omitir el desarrollo de los mismos, ya que no son una parte indispensable del desarrollo de la aplicación.

1.2.1.4.4 Configuración de la política de CORS:

Otro problema encontrado durante el desarrollo del Frontend fue la configuración de las políticas de CORS, la cual impedía poder hacer peticiones a la API desde la página web. Se logró superar dicho problema utilizando las guías presentes en la documentación de Spring al respecto, culminando en la creación de la Clase "*WebConfig.java*" detallada a continuación, que se encuentra dentro del directorio "*config*", mencionado anteriormente.

```
@Configuration
@EnableWebMvc
public class WebConfig implements WebMvcConfigurer {

    @Value("${api.path}")
    private String path;

    @Override
    public void addCorsMappings(CorsRegistry
registry) {
        registry.addMapping(this.path + "**").
            allowedOrigins("http://localhost",
                "https://localhost",
```

```
"http://localhost:8080",  
"http://localhost:3000");  
    }  
  
}
```

1.2.1.5 Sprint Review:

Para este Sprint se planteó completar cinco (5) Historias de Usuario del backlog del producto. Una de estas HU consiste en el establecimiento de un entorno en el cual poder programar la aplicación, dos consisten en aprender a utilizar las herramientas necesarias para el desarrollo de la misma, otra en la documentación del proceso, y la última en la generación de la funcionalidad principal del sistema.

Durante este Sprint se esperaba el surgimiento de distintos errores o problemas asociados al desconocimiento de algunas de las tecnologías que se debían utilizar o a la forma de integrarlas. Las dificultades que se presentaron fueron: la asociación de la base de datos con la aplicación del Backend y las políticas de CORS que no permitían la comunicación entre Frontend y Backend. Ambos problemas pudieron ser solucionados eventualmente.

En este periodo de desarrollo se pudo implementar una primera aproximación a las funcionalidades de la página de creación de *Invitaciones* y a los endpoints del Backend necesarios para realizar dicha acción.

Por lo tanto, se pudo cumplir tanto con el Sprint Backlog planteado como con el cumplimiento de la metodología y sus eventos especificados al inicio de la sección.

1.2.2 Sprint 2:

1.2.2.1 Sprint Planning:

Para el segundo Sprint se seleccionaron cuatro (5) Historias de Usuario, la primera de ellas tenía como objetivo completar la funcionalidad de generación de *Invitaciones*, cuyo desarrollo se inició en el Sprint anterior. Para completar esta funcionalidad, se decidió implementar la gestión de sesiones, lo que permite asignar dinámicamente el *Residente* a cada una de las *Invitaciones* generadas.

Otras tres (3) HU tienen como objetivo darle sentido a las *Invitaciones*, es decir, permitir que éstas concedan al *Invitado* el acceso al barrio. Para cumplir este objetivo, se necesitó estudiar las formas en que React permite al desarrollador escanear los códigos QR. Con el conocimiento adquirido sobre la lectura de códigos QR, se pudo programar una página destinada al *Guardia* de seguridad, para que este pueda realizar la interpretación de dichos códigos y autorice el ingreso del *Invitado* al barrio.

La última HU es la documentación del desarrollo, la cual se continúa ejecutando desde el comienzo del proyecto. Las siguientes figuras representan dichas HU:

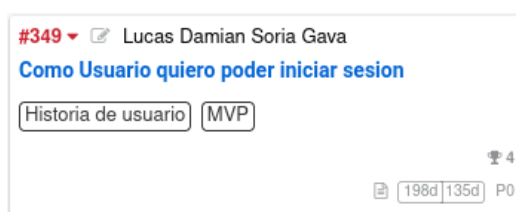


Figura 28: HU inicio de sesión.

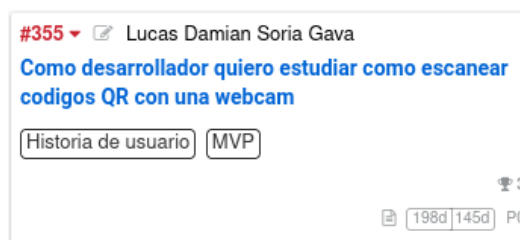


Figura 29: HU aprender a leer códigos QR.

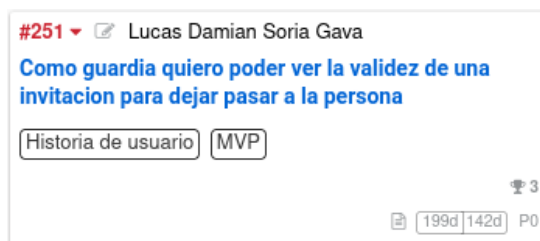


Figura 30: HU validar *Invitaciones*.

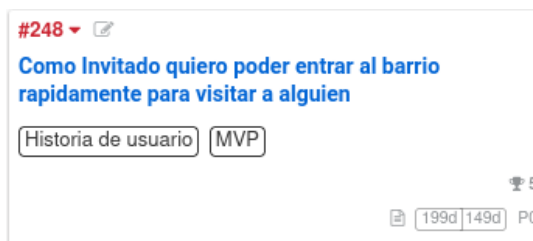


Figura 31: HU entrar al barrio.

Fuente: Imágenes de elaboración propia.

1.2.2.2 Diseño Gráfico:

1.2.2.2.1 Pantalla de visualización de la Invitación:

El segundo Sprint comenzó por la implementación de página de visualización de la *Invitación*, ya que es necesario poder tener acceso al código QR para su posterior lectura. Para esta pantalla, que es a la que tiene acceso el *Invitado*, se pensó en un diseño simple pero que permita ver la información relevante para su destinatario. En ella se pueden ver el código QR que representa la *Invitación* y un mapa que señala con un punto, la ubicación de la casa del *Residente* que generó dicha *Invitación*. Este mapa debe estar integrado con Google, para permitirle al *Usuario* hacer *click* en él y que este abra en la aplicación móvil (o en su defecto en el navegador) el mapa con el punto seleccionado, permitiendo recibir indicaciones de como llegar hasta la casa de su conocido.

El código QR ocupa la primera posición en la pantalla con el objetivo de presentar el mismo en la entrada del barrio con solo mostrar el display del dispositivo móvil en una cámara, sin la necesidad de que el *Invitado* deba realizar acciones adicionales.

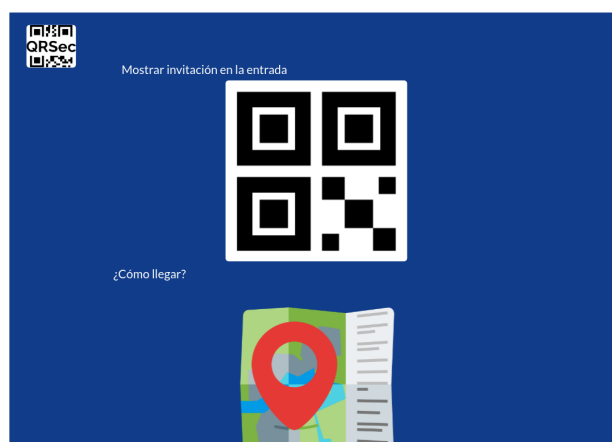


Figura 32: Diseño de la pantalla de visualización de la *Invitación*. Fuente: Imágen de elaboración propia.

1.2.2.2 Pantalla de escaneo de Invitaciones:

Una vez diseñada la página de visualización de *Invitaciones*, corresponde diseñar la interfaz por la que se validan las mismas. La pantalla de escaneo de *Invitaciones* es aquella a la que tendrá acceso el *Guardia* de la entrada del barrio, está compuesta de una sección que muestra las imágenes captadas por la cámara de la barrera y debajo de esta, un recuadro de texto que informa si la *Invitación* detectada es válida o no.

En caso de que la invitación sea válida, la sección inferior debe otorgar información al *Guardia* respecto de la misma. El cuadro de texto en estos casos estará dividido en dos partes, la primera sección está destinada a la información de la persona *Invitada*, presentando su nombre, apellido, dni y el tiempo máximo (en horas) de estadía. La segunda parte, estará compuesta por la información del *Residente*, también se mostrará su nombre y apellido, pero a diferencia del *Invitado*, se indicará el número de teléfono y la información de su casa.

Esta información le permitirá al Guardia tener una mayor noción de donde se encuentran las personas que ingresan al barrio y también, le permitirá ayudar a los mismos a encontrar el lugar al que se dirigen.

Por último, se debe destacar que para reconocer rápidamente la validez de la *Invitación*, se dispuso un código de colores simple: verde es válido, rojo es inválido.

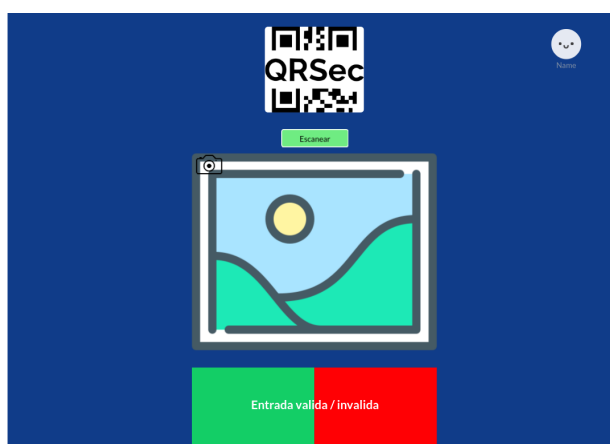


Figura 33: Diseño de la pantalla de escaneo de *Invitaciones*. Fuente: Imágen de elaboración propia.

1.2.2.2.3 Pantalla de inicio de sesión:

Por último, se presenta el diseño de la pantalla de inicio de sesión, esta es una página simple, conformada por un único formulario que solicita el nombre del *Usuario* y su contraseña. En QRSec el email funciona como nombre de usuario y permite la identificación única del mismo dentro del sistema.

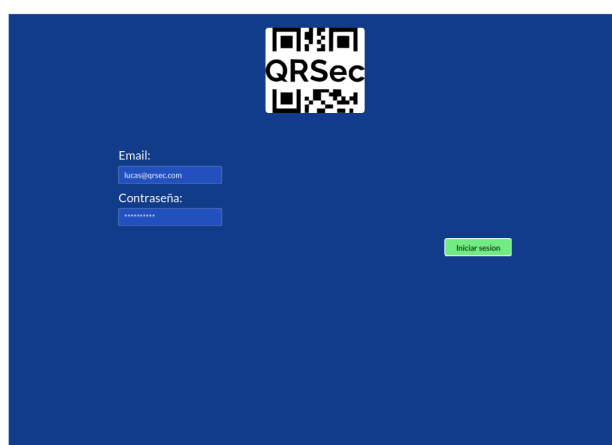


Figura 34: Diseño de la pantalla de inicio de sesión. Fuente: Imágen de elaboración propia.

1.2.2.3 Desarrollo del Frontend:

1.2.2.3.1 Desarrollo de la pantalla de visualización de la Invitación:

Para el desarrollo de la pantalla de visualización se hizo uso de la librería de react-qr-code que se describió en la sección 1.1.3. Esta permite que a través de un conjunto de bytes se genere el código QR, para esta aplicación se utilizó texto (el id de la *Invitación*) para generar la imagen. La librería react-qr-code permite configurar el código según la necesidad del caso, QRSec hace uso del máximo nivel de tolerancia a errores, tal como lo describe Peter Kieseberg, et al. (2010) en su paper “QR Code Security”. El máximo nivel es el “H” y permite una tolerancia al error de hasta el 30%, esto resulta especialmente útil para cuando se

requiere una lectura más confiable y no importa perder capacidad de codificación. En el caso de QRSec, el largo de la información es de 24 caracteres, tamaño inferior al límite de codificación.

Adicionalmente debajo del código QR se localiza un mapa con la *Dirección* de la casa del *Residente* marcada, para el caso en que el *Invitado* nunca haya asistido a la misma. La integración del mismo a la página ha sido un problema inesperado ya que para hacerlo se debe crear una cuenta en el servicio de Cloud de Google (GCP) y se debe generar un token de acceso a la API de Maps de Google, la cual se ofrece de manera gratuita a todos sus usuarios mientras éstos realicen menos de trescientas (300) peticiones por mes. Esta a su vez debe ser consumida a través de la librería de react-google-maps/api, implementada y ofrecida por los mismos desarrolladores de Google.

Al hacer click sobre la marca roja que señala la ubicación de la casa, es posible abrir el mapa en una nueva ventana o en la aplicación de google maps para teléfonos móviles.

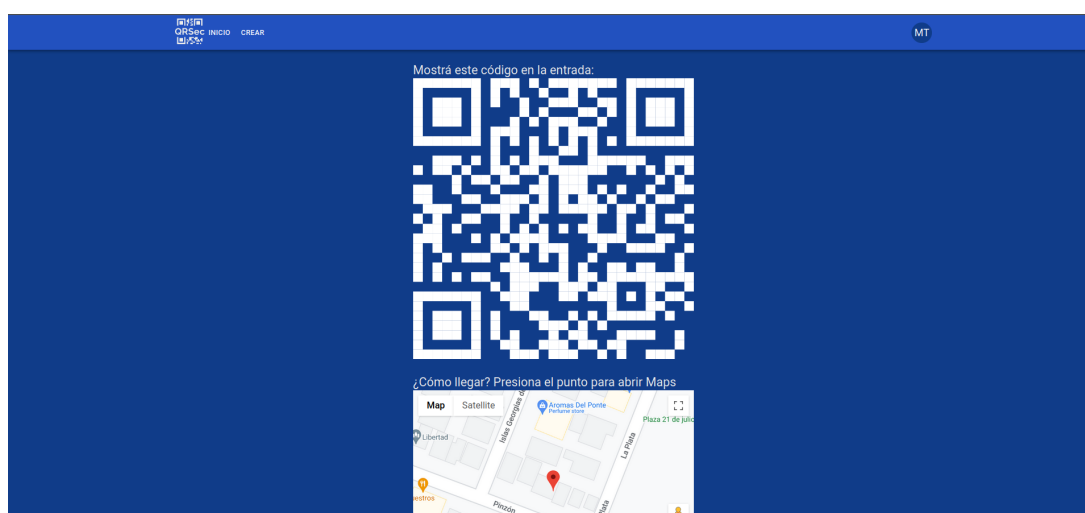


Figura 35: Implementación de la pantalla de visualización de la *Invitación*. Fuente: Imágen de elaboración propia.

1.2.2.3.2 Desarrollo de la pantalla de escaneo de Invitaciones:

Después de haber podido generar la página de visualización de los códigos QR, se desarrolló la página de escaneo de los mismos. A esta

página tienen acceso los *Guardias* de la entrada del barrio y es a través de ella que pueden validar las entradas.

El escaneo del código resultó especialmente desafiante ya que las librerías que se probaron para realizar este proceso contenían todos problemas de compatibilidad ya sea con la versión de React que se utilizó (React 18), con el navegador o el sistema operativo. Por esta razón, se optó buscar una solución implementada al nivel más bajo posible, haciendo mayor uso de Javascript y HTML5, evitando las dependencias de funcionalidad con la librería de React. Encontrar una librería que cumpla con este requerimiento fue lo que más demoró al desarrollo en el segundo Sprint, pero finalmente se pudo localizar a “qrreader” y posibilitó avanzar con el desarrollo de la verificación de las *Invitaciones*.

Un inconveniente que surge al utilizar cámaras en las aplicaciones web es que las mismas requieren de un ambiente seguro para poder funcionar. Esto quiere decir que el módulo de cámara (“qrreader”) sólo funcionará en ambientes de prueba locales y páginas que implementen el protocolo HTTPS.

Luego de realizar un escaneo exitoso del código, se hace una solicitud a la API con el id de la *Invitación* (“/api/v1/invites/:id”), en caso de no encontrar la entrada en la base de datos, se la muestra como inválida. En el caso que la *Invitación* exista, el Frontend analiza la validez de sus campos. Por escasez de tiempo para el desarrollo de esta funcionalidad, solo se verifica que la persona esté intentando ingresar dentro del horario permitido o en alguno de los días que tiene permitido ingresar.

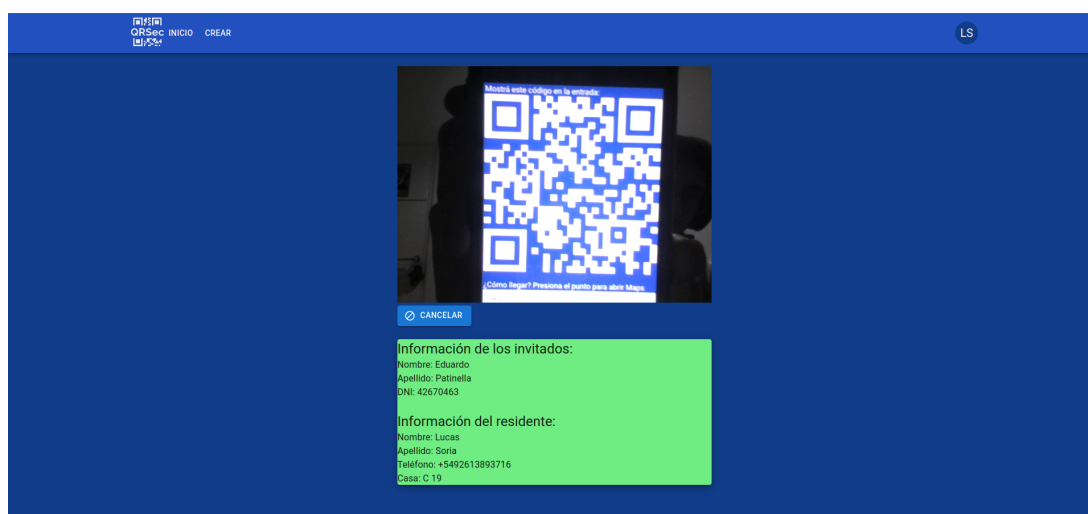


Figura 36: Implementación de la pantalla de escaneo de *Invitaciones*.
 Fuente: Imágen de elaboración propia.

1.2.2.3.3 Desarrollo de la pantalla de inicio de sesión:

Como se especificó en la sección 1.2.2.2.3, esta es una pantalla simple que contiene un pequeño formulario cuya funcionalidad es la de obtener el JWT (JSON Web Token) que el *Usuario* utilizará en la aplicación para identificarse. Este token se utilizará para dirigirlo a la pantalla correspondiente, creación de *Invitaciones* para el *Residente* y escaneo de códigos para el *Guardia*.

Con el objetivo de mantener la sesión abierta, incluso después de que el *Usuario* haya abandonado la página y regresado, el Frontend guarda en el almacenamiento local del dispositivo el JWT correspondiente a dicho *Usuario*.

El botón “*Iniciar sesión*” realiza una petición POST con la información de los campos al endpoint “*/api/v1/login*” de la API del Backend que se encarga de validar los datos y generar el JWT que posteriormente almacenará el Frontend.

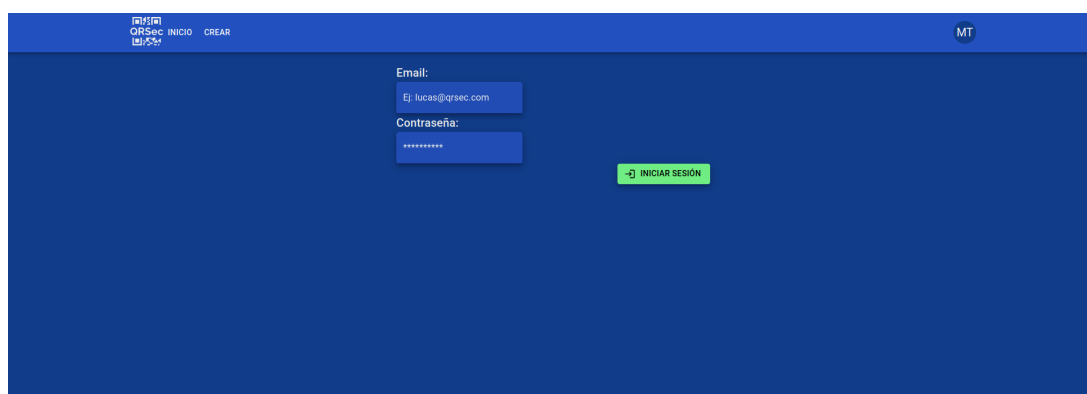


Figura 37: Implementación de la pantalla de inicio de sesión. Fuente: Imágen de elaboración propia.

1.2.2.3.4 Modificación de la barra superior:

Con la implementación del manejo de sesiones, se hizo posible darle funcionalidad a la barra superior de la interfaz gráfica de la página web. Anteriormente se mencionó que el icono presente en la parte superior derecha representaba las iniciales del *Usuario* activo, pero que en realidad este era fijo, ya que no había forma de distinguirlo.

El uso de JWTs permite extraer del mismo el nombre y apellido del *Usuario* y asignar sus iniciales al logo mencionado para poder darle identidad y que este se identifique durante el uso de la aplicación.

1.2.2.3.5 Desarrollo de la pantalla “Agregar invitado”:

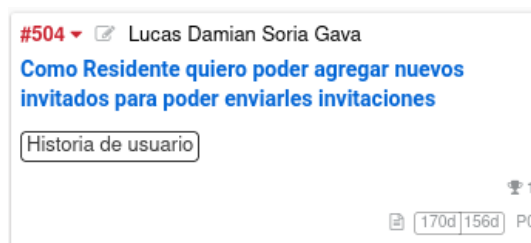


Figura 38: HU agregar *Invitados*. Fuente: Imágen de elaboración propia.

Durante el segundo Sprint se desarrolló la funcionalidad de “Agregar invitado” dentro del menú de selección de *Invitados*, a pesar de que no formaba parte del Sprint Backlog. La decisión de implementar esta funcionalidad se fundamenta en que la misma facilita la realización de pruebas sobre la pantalla de generación de invitaciones y agrega valor al Usuario.

Si el botón “Agregar invitado” es seleccionado, se despliega una ventana de diálogo que permite completar la información pertinente para generar un nuevo *Invitado*. Como se especificó en la sección 1.2.1.2.4 del primer Sprint, los datos solicitados son: nombre, apellido, DNI y teléfono.

Una vez se completan los campos y se presiona el botón de “CREAR”, la página se comunica con el Backend a través del endpoint “/api/v1/guests” de la API y este crea el nuevo *Invitado*. De este modo, la próxima vez que se abra el menú desplegable, el nuevo *Invitado* estará disponible junto con aquellos que se encuentren asociados a ese *Usuario*.

Figura 39: Implementación de la pantalla para agregar nuevos *Invitados*.
Fuente: Imágen de elaboración propia.

1.2.2.4 Desarrollo del Backend:

1.2.2.4.1 Implementación de nuevos métodos HTTP:

Para el correcto funcionamiento de las nuevas características desarrolladas en el Frontend, se necesitó programar nuevos métodos para endpoints existentes. Específicamente se desarrollaron los métodos GET `"/api/v1/invites/:id"` y POST `"/api/v1/guests"`.

El primer método tiene como objetivo conseguir la información de una *Invitación* específica y es usada por *Invitados* y *Guardias*, el primero para visualizar la página con el código QR y el mapa para llegar a la casa del *Residente*, y el segundo lo utiliza para validar la información de la mismas y permitir el acceso del *Invitado*.

El método POST en el endpoint de *Invitados* permite registrar los mismos en el sistema, y que a su vez estos se asocien al *Residente* que los creó. Esto se logra tomando el *username* (email) presente en el JWT que envía el agente que realiza la petición al servidor. Esta funcionalidad es consumida por el mismo *Residente* en la pantalla de creación de *Invitaciones*, como fue explicado en las secciones 1.2.1.2.4 y 1.2.2.3.5, correspondientes al diseño e implementación de dicha página.

Para poder registrar un nuevo *Invitado* asociado a un *Residente*, primero se debe desarrollar un sistema de gestión de sesiones. Este sistema es más complejo y por lo tanto se decidió describirlo en una sección aparte.

1.2.2.4.2 Creación del endpoint de login:

El último endpoint se programó fue el sistema de generación de JWT o endpoint de *login* (POST `"/api/v1/login"`), el cual resultó tener una implementación más compleja, que supuso una dificultad en el proceso de desarrollo de la aplicación.

Para la implementación de esta funcionalidad, se estableció un filtro de autenticación, para el cual Spring define que debe ser especificado en la configuración de seguridad de la API, dentro del archivo `"SecurityConfig.java"`, el cual debe extender la clase `"WebSecurityConfigurerAdapter"`. La sección de código pertinente se muestra a continuación, teniendo en cuenta que `"..."` simboliza una omisión de código no importante para la configuración del filtro.

```
@EnableWebSecurity
@Configuration
@RequiredArgsConstructor
@EnableGlobalMethodSecurity(prePostEnabled = true, securedEnabled = true)
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    ...

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.cors().and();
        CustomAuthenticationFilter customAuthenticationFilter = new
CustomAuthenticationFilter(authenticationManagerBean());
        customAuthenticationFilter.setFilterProcessesUrl(this.path +
this.login);

        ...

        http.authorizeRequests().antMatchers(this.path + this.login +
```



```

"/**", this.path + this.refreshToken + "**").permitAll();

...

http.addFilter(customAuthenticationFilter);
http.addFilterBefore(new CustomAuthorizationFilter(),
UsernamePasswordAuthenticationFilter.class);
}

...
}

```

Una vez establecido el filtro de autenticación y configurados los endpoints que requerirán del mismo, se debe definir un algoritmo para autenticar a los *Usuarios*. Dicho algoritmo se encuentra contenido dentro de la clase *CustomAuthenticationFilter.java*, la cual extiende la clase *UsernamePasswordAuthenticationFilter*. De esta clase es importante mencionar dos métodos: *attemptAuthentication* y *successfulAuthentication*, los cuales se encargarán de efectivamente verificar que *email* y *contraseña* coinciden con el *Usuario* de la base de datos, y expedirán el JWT correspondiente.

```

@Slf4j
public class CustomAuthenticationFilter extends
UsernamePasswordAuthenticationFilter {

...

@Override
public Authentication attemptAuthentication(HttpServletRequest
request, HttpServletResponse response) {

...

UsernamePasswordAuthenticationToken authenticationToken = new
UsernamePasswordAuthenticationToken(this.jsonUsername,
this.jsonPassword);
return authenticationManager.authenticate(authenticationToken);
}

...
}

```

```

@Override
public void successfulAuthentication(HttpServletRequest request,
    HttpServletResponse response, FilterChain chain, Authentication
    authentication) throws IOException {
    User user = (User)authentication.getPrincipal();
    Algorithm algorithm = Algorithm.HMAC256("secret".getBytes());
    String access_token = JWT.create()
        .withIssuer(request.getRequestURL().toString())
        .withSubject(user.getUsername())
        .withClaim("first_name", user.getFirstName())
        .withClaim("last_name", user.getLastName())
        .withClaim("roles",
            user.getAuthorities().stream().map(GrantedAuthority::getAuthority).collect(
                Collectors.toList()))
        .withExpiresAt(Date.from(Instant.now().plusSeconds(5 * 60 *
            60)))
        .sign(algorithm);

    ...

    Map<String, String> tokens = new HashMap<>();
    tokens.put("access_token", access_token);
    response.setContentType(APPLICATION_JSON_VALUE);
    new ObjectMapper().writeValue(response.getOutputStream(),
    tokens);
}
...
}

```

Por último, debe implementarse otro filtro que realice el camino inverso de *successfulAuthentication*, permitiendo a la API obtener el *email* y el nivel de *autoridad* del *Usuario* a partir de su JWT. Este filtro se ejecutará cada vez que alguien realice una petición a un endpoint que requiera autenticación.

1.2.2.4.3 Modificación de los métodos HTTP existentes:

Ya que los endpoints básicos y sus configuraciones fueron mayormente programadas durante el primer Sprint, en este Sprint se modificaron los mismos para admitir el manejo de sesiones. Concretamente se modificaron los métodos: POST *"/api/v1/invites"*, y GET *"/api/v1/guests"*.

Para el endpoint de *Invitaciones*, la modificación está relacionada con vincular cada nueva *Invitación* con el *Residente* que la creó, específicamente con el *Usuario* dueño del JWT que realizó dicha petición.

En el caso del endpoint de *Invitados*, la petición GET fue alterada para que responda únicamente con los *Invitados* cuyo “dueño” sea el *Usuario* que está consultando el sistema. De esta forma, se reduce la cantidad de personas que puede visualizar el *Residente* en el menú desplegable de la pantalla de creación de *Invitaciones*, y facilita el proceso de selección del *Invitado*.

1.2.2.5 Despliegue en la nube:

La última actividad llevada a cabo en el segundo Sprint fue una Historia de Usuario que no fue prevista en el Sprint Planning. La misma consiste en dar accesibilidad a los usuarios al sistema desde internet, para lo cual es necesario principalmente un Cloud Service Provider (CSP), el cual es el encargado de disponibilizar un servidor de propósito general en la nube, en este trabajo se utilizaran los servicios de instancias de máquinas virtuales.

En la sección 1.1.7 se explicaron las tecnologías utilizadas con el propósito de disponibilizar fácilmente el servicio, en esta sección se explica cómo se logró disponibilizar. En la sección 1.2.2.3.2 se mencionó que para el correcto funcionamiento de la cámara en la aplicación, es necesario utilizar un entorno seguro, ahora se explicará cómo se logró establecer dicho entorno.

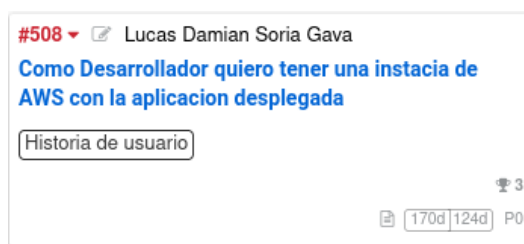


Figura 40: HU despliegue a producción.

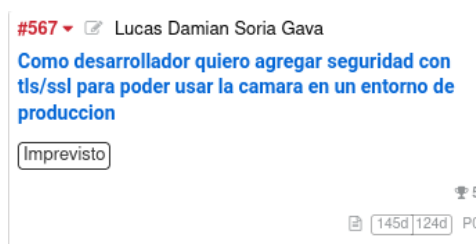


Figura 41: Imprevisto generar un entorno seguro.

Fuente: Imágenes de elaboración propia.

Esta sección puede dividirse en tres partes: configuración de la instancia del servidor, configuración del servidor para disponibilizar la aplicación web, y por último, configuración del entorno seguro con certificados *SSL/TLS*.

En primer lugar, utilizando los servicios de instancias *E2C* de *AWS*, se creó un servidor con una máquina virtual con las siguientes características:

- *Sistema operativo:* Ubuntu Server 22.04 x64 bits.
- *Procesador:* 1vCPU.
- *Memoria:* 1GiB.
- *Almacenamiento:* 8GiB.
- *Puertos habilitados:* 22 (SSH) y 443 (HTTPS).

Esta configuración permite mantener la instancia bajo la categoría de “*free-tier-eligible*”, con opción de expandir el almacenamiento hasta los 30GiB.

Es esta pequeña, pero suficiente máquina virtual (VM) la que da origen al servidor web que se aloja la página de QRSec. Para ello se necesita configurar el mismo, en este caso, a partir de la herramienta *NGINX*. A continuación se detallarán dichos ajustes:

```
$ sudo apt install nginx
$ sudo ufw allow 'Nginx Full'
$ sudo ufw allow 'OpenSSH'
$ systemctl status nginx
```

Para empezar a configurar el servidor de *NGINX*, primero debemos instalarlo y permitir al firewall de la VM el paso del tráfico de red por los puertos 22 y 443, independientemente de que se lo haya permitido en la creación de la misma. Esto se debe a que los grupos de seguridad de *AWS* son distintos del firewall de la instancia, ambos deben estar permitidos, tanto para no perder la comunicación a través del protocolo *SSH*, como para poder visitar el sitio expuesto por el servidor *NGINX*. Es una buena política de seguridad cerrar cualquier otro puerto que no se use expresamente o restringir el rango de IPs que tienen permitido el uso del puerto 22, por ejemplo.

El último comando permite ver el estado del servicio de *NGINX*, el cual debe estar activo y corriendo para poder seguir configurándolo. En caso de que no haya problemas, el servidor debería encontrarse exponiendo una página por defecto o de prueba, a la que se puede acceder a través del enlace "*http://<server-ip>*", donde *<server-ip>* corresponde con la IP de la instancia de *AWS*. Para configurar un dominio sobre esa ip se deberían seguir las configuraciones demostradas en las figuras N°12 y N°13, editando los registros A del DNS para que apunten del dominio ("*lsoria.com*") a la IP del servidor.

Para pasar de un servidor web de prueba al real de producción, se debe: registrar el dominio como uno del servidor y exponer los archivos de la aplicación. Para lo primero, se pueden conformar "*server blocks*", los cuales son especialmente útiles cuando se quiere registrar varios dominios en un mismo servidor, no es el caso pero se utilizarán de igual manera porque permiten una configuración más organizada.

```
$ sudo mkdir -p /var/www/lsoria.com/html
$ sudo chown -R $USER:$USER /var/www/lsoria.com/html
$ sudo chmod -R 755 /var/www/lsoria.com
```

Una vez creada la carpeta correspondiente al dominio "*lsoria.com*", se debe cambiar el dueño del archivo a *USER* y permitir a los demás usuario la ejecución de los archivos contenidos dentro del mismo.

El código fuente de la aplicación debería ubicarse dentro de la carpeta creada. Existen varias formas de hacerlo, aquí se expondrán dos: utilizar el protocolo SSH para enviar los archivos de una computadora local al servidor web remoto, o clonar el repositorio de git para luego “compilar” la aplicación en el servidor remoto.

```
$ scp -r ./build/* <username>@<server-ip>:/var/www/lsoria.com/html

OR

$ git pull https://github.com/soria-lucas/qrsec.git
$ cd qrsec-frontend/
$ npm run build
$ mv -r ./build/* /var/www/lsoria.com/html
```

En este caso *<username>* y *<server-ip>* corresponden con el nombre de usuario y la IP específicos del servidor, información provista por el mismo CSP.

En definitiva, no importa el método, pero el código fuente (y especialmente el archivo `index.html`) debe estar ubicado dentro del directorio `“/var/www/lsoria.com/html”`.

Como esta configuración del servidor web utiliza *“server blocks”* no se pueden modificar las configuraciones por defecto para disponibilizar el sitio, se debe detallar la misma en otro fichero creado para estos casos. El siguiente comando y su configuración resuelven este problema.

```
$ sudo nano /etc/nginx/sites-available/lsoria.com
```

```
server {
    root /var/www/lsoria.com/html;
    index index.html index.htm index.nginx-debian.html;

    server_name lsoria.com www.lsoria.com;

    location / {
        try_files $uri $uri/ =404;
    }
}
```

```

listen [::]:443 ssl ipv6only=on; # managed by Certbot
listen 443 ssl; # managed by Certbot
ssl_certificate /etc/letsencrypt/live/lsonia.com/fullchain.pem; #
Certbot
ssl_certificate_key /etc/letsencrypt/live/lsonia.com/privkey.pem; #
Certbot
include /etc/letsencrypt/options-ssl-nginx.conf; # managed by
Certbot
ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot
}

server {
    if ($host = www.lsonia.com) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

    if ($host = lsonia.com) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

    listen 80;
    listen [::]:80;

    server_name lsonia.com www.lsonia.com;
    return 404; # managed by Certbot
}

```

En este archivo se puede observar que la aplicación no solo tiene configurados los puertos 80 y 443 para atender solicitudes, sino que el puerto 80 redirige el tráfico al entorno seguro, y que el puerto 443 presenta configuraciones para el manejo de certificados *SSL/TLS* gestionados por *Certbot*, una herramienta que será configurada más adelante.

Una vez se configura el servidor para atender en el dominio y puertos especificados, se debe habilitar la página, esto se hace creando un enlace simbólico desde los sitios disponibles a los sitios habilitados.

```

$ sudo ln -s /etc/nginx/sites-available/your_domain
/etc/nginx/sites-enabled/
$ sudo nano /etc/nginx/nginx.conf

```

Esta técnica permite habilitar y deshabilitar fácilmente distintos sitios porque se basta con crear o eliminar symlinks, no hay necesidad de mover información o modificar archivos de configuración, y toda la información y código fuente de las mismas se mantiene en un sitio.

Opcionalmente se puede ejecutar el segundo comando para editar el número de “*server names*” que permite generar NGINX, para evitar problemas de memoria o creación de servidores por permitir la creación de muy pocos “*server names*”. Esta configuración sólo es útil si se configuran más sitios.

```
...  
http {  
    ...  
    server_names_hash_bucket_size 64;  
    ...  
}  
...
```

Para terminar la configuración de NGINX, se pueden buscar errores de sintaxis en las configuraciones y se debe reiniciar el servicio.

```
$ sudo nginx -t  
$ sudo systemctl restart nginx
```

Por último, para aplicar los certificados *SSL/TLS* con *Let's encrypt*, se debe ejecutar *Certbot*:

```
$ sudo apt install certbot python3-certbot-nginx  
$ sudo certbot --nginx -d example.com -d www.example.com
```


Certbot se encarga automáticamente de integrar el plugin de *NGINX* con *Let's encrypt* y especifica el dominio -d donde debe tomar efecto el certificado.

Los certificados emitidos por *Let's encrypt* tienen validez por 90 días, con el propósito de que los usuarios automaticen la seguridad de sus sitios. *Certbot* se encarga de renovar dichos certificados automáticamente aplicando un timer de *systemd* que se ejecuta dos veces al día. Se puede consultar el estado de ese servicio con el primer comando, mientras que el segundo comando fuerza una renovación de certificados, especialmente útil para comprobar que todas las configuraciones se realizaron correctamente.

```
$ sudo systemctl status certbot.timer  
$ sudo certbot renew --dry-run
```

1.2.2.6 Sprint Review:

En este segundo y último Sprint se planteó realizar cinco (5) Historias de Usuario que a grandes rasgos agregarían las funcionalidades de: inicio de sesión, lectura y validación de *Invitaciones*, y entrada al barrio. Además de que se sigue cumpliendo con el objetivo de documentar el proceso de desarrollo, que también tuvo lugar durante el periodo anterior.

Durante este Sprint no solo se logró cumplir con los objetivos planteados, sino que se pudo dar un paso extra y se implementaron los sistemas de generación de *Invitados* y el despliegue del MVP en la cloud.

Para lograr estos objetivos, fue necesario diseñar e implementar las pantallas de inicio de sesión, generación de *Invitados*, visualización de la *Invitación*, y escaneo de las mismas. Para el correcto funcionamiento de estas pantallas, fue necesaria la creación o modificación de los métodos HTTP de los endpoint `"/api/v1/login"`, `"/api/v1/invites/"`, `"/api/v1/invites/:id"` y `"/api/v1/guests"`, los cuales deben hacer uso de un JWT para enviar respuestas al *Usuario* en base a

su identidad y nivel de *Autoridad*. Adicionalmente, se dió funcionalidad a la barra superior de la página web, presente en todas las pantallas.

Se esperaba tener menor cantidad de problemas o dificultades durante este Sprint que durante el anterior, sin embargo, eso no fue así. Surgieron diversos problemas durante el desarrollo de la funcionalidad de escaneo de códigos QR y la exhibición del mapa de google maps. Con la primera funcionalidad hubo un problema de compatibilidad entre la nueva versión del framework React (v18) y sus librerías, programadas para versiones anteriores. El segundo problema que se originó con la pantalla de lectura de códigos fue la necesidad de tener un entorno seguro para utilizar la cámara del *Guardia* de seguridad, sin este el framework no permite el uso de la misma.

A su vez, el despliegue del MVP supuso una dificultad extra, ya que al estar en un entorno de producción, y utilizar la cámara del *Guardia*, el canal de comunicación con el servidor debe ser seguro. Para tener un entorno seguro se aplicaron certificados SSL/TLS,

El último inconveniente fué la presentación del mapa de Google Maps con un punto que indique la dirección de la casa del *Residente*, dicho problema fue solucionado con el uso de la librería desarrollada por Google para tal aplicación.

Se resalta de este Sprint no solo el hecho de haber cumplido con los objetivos planteados al inicio, sino el haber podido ir más lejos para ampliar las funcionalidades del MVP y lograr su despliegue en un entorno de producción funcional. Este último hito sirve como confirmación de que el producto, a pesar de ser mínimo, es viable.

Capítulo 2: Análisis de resultados

Al finalizar ambos Sprints y el desarrollo del proyecto, lo que se obtuvo fue el cumplimiento de los objetivos planteados al inicio de cada Sprint, en su Planning respectivo, y la obtención de resultados más allá de lo estipulado o estimado.

Como fruto de dos meses de trabajo, se produjo un sistema disponibilizado al Internet a través de servicios cloud capaz de permitir a sus *Usuarios* agendar *Invitados* y expedir *Invitaciones* para que los mismos puedan visitarlos ingresando a un barrio cerrado de forma rápida y semi-automatizada.

Los *Invitados* son capaces de recibir indicaciones para llegar a su destino, y una vez que lo alcanzan, pueden ingresar al barrio ágilmente, utilizando la herramienta de validación automática de sus *Invitaciones*, sin la necesidad de intercambiar datos múltiples veces con los *Guardias* de seguridad y los *Residentes*. Los *Guardias* tienen acceso a la información de la *Invitación*, de su dueño y del *Invitado* a la que corresponde, lo que les permite tener un mayor control sobre las personas que ingresan al barrio.

El producto final logra también interactuar y comunicarse con otros sistemas fuera de sí mismo, como lo son las APIs de Google Maps y los certificados de *Let's encrypt*. Esto genera un sistema más completo y complejo que agrega valor al Usuario y mejora la experiencia de uso, a pesar de que presentaron complicaciones imprevistas en el desarrollo de la solución.

Conclusiones:

Luego de analizar los resultados obtenidos en cada etapa del desarrollo, se puede afirmar que no solo se logró cumplir con los objetivos planteados en la sección de objetivos, sino que se los superó, es decir, se elaboró el MVP de un sistema informático capaz de controlar y registrar el acceso de invitados a barrios cerrados, que además está disponible en la nube.

El sistema elaborado es capaz de crear y consultar invitaciones para el ingreso a un barrio cerrado, permitiendo a los invitados una entrada ágil, y lo consigue a través de un sistema que lee e interpreta códigos QR asociados a dichas invitaciones y muestra su información a los guardias del barrio, quienes deciden sobre el ingreso de esa persona al barrio.

Sin embargo, éste al ser un producto mínimo viable, es un sistema incompleto y se pueden destacar los siguientes aspectos a tener en cuenta para un futuro desarrollo más completo, confiable y seguro:

Lo más importante es poder completar la verificación de la información de las *Invitaciones*, que actualmente solo confirman que el día sea el correcto. Así mismo, es necesario agregar las funcionalidades de edición y eliminación de entidades, entendiéndose por entidades a *Usuarios (Residentes, Guardias y Encargados)*, *Invitados*, *Invitaciones* y *Direcciones* de casas.

Del mismo modo, se pueden agregar las pantallas correspondientes al *Encargado* del barrio, el cual debe ser capaz de visualizar información general sobre las estadísticas de *Invitaciones* del barrio, así como el detalle de cada una. A los Guardias puede se les puede agregar una pantalla que les informe la cantidad de *Invitaciones* que hay para el día, o las personas que ya han ingresado al mismo, con la información de cada una.

Para que el sistema sea más confiable o resiliente, es importante que el mismo aplique un esquema de nube híbrida, donde se coloque un servidor en cada barrio, que genere una copia de la base de datos en

la nube y que, de esta forma, se pueda permitir el acceso al barrio incluso si se pierde conexión a internet.

Ahora el sistema solo toma 3 datos del *Invitado*, en un futuro, con la intención de hacer el sistema más confiable, el ingreso más rápido y automatizado, se podría registrar más datos del *Invitado*, como una fotografía, para poder implementar un sistema de reconocimiento facial, el cual junto con la *Invitación* puede automatizar el izado de la barrera y el registro del *Residente* al cual visita. La *Invitación*, también puede mejorarse para que el campo de DNI pueda admitir más tipos de identificaciones, por ejemplo, en el caso de que se invite a una persona extranjera. Otra forma de mejorar la *Invitación* puede ser permitiendo cambiar la dirección de destino a lugares comunes del barrio, como los llamados “Club house”.

Por último, en el espíritu de aumentar la seguridad, debería hacerse un manejo de las variables de entorno a través de *secrets*, para por ejemplo, ocultar los usuarios y contraseñas que configuren los servicios. También, es conveniente o imperativo, almacenar los datos de las personas en almacenamientos seguros, para que no puedan ser accedidos por atacantes malintencionados. En última instancia, el acceso a los endpoints y el nivel de información que estos entregan debe estar asegurado por los roles de cada usuario, por ejemplo, el invitado solo puede ver la dirección de la casa del Residente, mientras que el guardia ve toda la info de la invitación.

Anexos:

Anexo 1: Repositorio del proyecto

El repositorio de git utilizado para almacenar el código del proyecto se encuentra ubicado en GitHub en el siguiente enlace: "<https://github.com/soria-lucas/qrsec>".

Anexo 2: Dominio de la página web



El dominio de la página web donde se puede encontrar el desarrollo funcional de la aplicación web es: "<https://lsoria.com>".

Fuentes bibliográficas:

Referencias bibliográficas:

1. Ater, T. (s/f). Building Progressive Web Apps: Bringing the Power of Native to the Browser. 429.
2. Kieseberg, P., Leithner, M., Mulazzani, M., Munroe, L., Schrittwieser, S., Sinha, M., & Weippl, E. (2010). QR code security. Proceedings of the 8th International Conference on Advances in Mobile Computing and Multimedia, 430–435. <https://doi.org/10.1145/1971519.1971593>
3. Padhy, R. P., & Panigrahy, D. (2015). NoSQL Databases: State-of-the-art and Security Challenges. 2(5), 8.
4. ¿Qué es la nube? | Conceptos esenciales. (s/f). Cloudflare. Recuperado el 20 de octubre de 2022, de <https://www.cloudflare.com/es-es/learning/cloud/what-is-the-cloud/>
5. Qué es la nube: Definición | Microsoft Azure. (s/f). Recuperado el 30 de agosto de 2022, de <https://azure.microsoft.com/es-es/resources/cloud-computing-dictionary/what-is-the-cloud/>
6. Samarati, P., & de Vimercati, S. C. (2001). Access Control: Policies, Models, and Mechanisms. En R. Focardi & R. Gorrieri (Eds.), Foundations of Security Analysis and Design (Vol. 2171, pp. 137–196). Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-45608-2_3
7. Uzun, V. (2016). QR-Code Based Hospital Systems for Healthcare in Turkey. 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), 71–76. <https://doi.org/10.1109/COMPSAC.2016.173>

Bibliografía:

1. Adam, J. (2020, octubre 26). The React Framework (library) – A Brief Overview.
<https://kruschecompany.com/react-framework-library/>
2. Adding Custom Environment Variables | Create React App. (s/f). Recuperado el 21 de junio de 2022, de
<https://create-react-app.dev/docs/adding-custom-environment-variables/>
3. Advanced Load Balancer, Web Server, & Reverse Proxy. (s/f). NGINX. Recuperado el 22 de junio de 2022, de
<https://www.nginx.com/>
4. Amigoscode (Director). (2021a, junio 1). Spring Boot Tutorial—Build a Rest Api with MongoDB.
<https://www.youtube.com/watch?v=ssj0CGxv60k>
5. Amigoscode (Director). (2021b, julio 25). Spring Boot and Spring Security with JWT including Access and Refresh Tokens .
<https://www.youtube.com/watch?v=VVn9OG9nfH0>
6. Arch Linux. (s/f). Recuperado el 20 de junio de 2022, de
<https://archlinux.org/>
7. baeldung. (2017, enero 30). CORS with Spring | Baeldung.
<https://www.baeldung.com/spring-cors>
8. baeldung. (2018, enero 20). Introduction to Spring Method Security | Baeldung.
<https://www.baeldung.com/spring-security-method-security>
9. Build software better, together. (s/f). GitHub. Recuperado el 20 de junio de 2022, de <https://github.com>
10. Carlos Azaustre - Aprende JavaScript (Director). (2021, abril 23). APRENDE REACT BÁSICO en 30 MINUTOS  —Tutorial de React.js Desde Cero.
<https://www.youtube.com/watch?v=EMk6nom1aS4>

11. Cloud Computing Services—Amazon Web Services (AWS). (s/f). Amazon Web Services, Inc. Recuperado el 20 de junio de 2022, de <https://aws.amazon.com/>
12. Cloudflare—The Web Performance & Security Company. (s/f). Cloudflare. Recuperado el 20 de junio de 2022, de <https://www.cloudflare.com/en-gb/>
13. ¡Compra dominio y hosting, y haz tu página web con el proveedor #1! GoDaddy AR. (s/f). GoDaddy. Recuperado el 20 de junio de 2022, de <https://www.godaddy.com/es>
14. Conventional Commits. (s/f). Conventional Commits. Recuperado el 20 de octubre de 2022, de <https://www.conventionalcommits.org/en/v1.0.0/>
15. Coolors.co. (s/f). Coolors.Co. Recuperado el 20 de octubre de 2022, de <https://coolors.co/ebebeb-6fec82-ff101f-2251bf-103c89>
16. Create a Palette—Coolors. (s/f). Coolors.Co. Recuperado el 20 de junio de 2022, de <https://coolors.co/ebebeb-6fec82-ff101f-2251bf-103c89>
17. Dewi, L. P., Noertjahyana, A., Palit, H. N., & Yedutun, K. (2019). Server Scalability Using Kubernetes. 2019 4th Technology Innovation Management and Engineering Science International Conference (TIMES-ICON), 1–4. <https://doi.org/10.1109/TIMES-ICON47539.2019.9024501>
18. Docker Hub Container Image Library | App Containerization. (s/f). Recuperado el 20 de octubre de 2022, de <https://hub.docker.com/>
19. Docker overview. (2022, agosto 30). Docker Documentation. <https://docs.docker.com/get-started/overview/>
20. Enabling Cross Origin Requests for a RESTful Web Service. (s/f). Recuperado el 20 de junio de 2022, de <https://spring.io/guides/gs/rest-service-cors/>
21. Geospatial Queries—MongoDB Manual. (s/f). Recuperado el 20 de junio de 2022, de

- <https://www.mongodb.com/docs/manual/geospatial-queries/>
22. Git. (s/f). Recuperado el 20 de junio de 2022, de <https://git-scm.com/>
 23. GitHub: Where the world builds software · GitHub. (s/f). Recuperado el 20 de octubre de 2022, de <https://github.com/>
 24. Home—Docker. (2022, mayo 10). <https://www.docker.com/>
 25. How To Deploy a React Application with Nginx on Ubuntu 20.04 | DigitalOcean. (s/f). Recuperado el 22 de junio de 2022, de <https://www.digitalocean.com/community/tutorials/how-to-deploy-a-react-application-with-nginx-on-ubuntu-20-04>
 26. How To Install Nginx on Ubuntu 20.04 | DigitalOcean. (s/f). Recuperado el 22 de junio de 2022, de <https://www.digitalocean.com/community/tutorials/how-to-install-nginx-on-ubuntu-20-04>
 27. How To Secure Nginx with Let's Encrypt on Ubuntu 20.04 | DigitalOcean. (s/f). Recuperado el 22 de junio de 2022, de <https://www.digitalocean.com/community/tutorials/how-to-secure-nginx-with-let-s-encrypt-on-ubuntu-20-04>
 28. Jenkins. (s/f). Jenkins. Recuperado el 20 de octubre de 2022, de <https://www.jenkins.io/>
 29. Let's Encrypt. (s/f). Recuperado el 22 de junio de 2022, de <https://letsencrypt.org/>
 30. Marvel—The design platform for digital products. Get started for free. (s/f). Recuperado el 20 de junio de 2022, de <https://marvelapp.com/>
 31. Maven – Welcome to Apache Maven. (s/f). Recuperado el 20 de junio de 2022, de <https://maven.apache.org/>
 32. Maven Repository: Org.springframework.boot » spring-boot-starter-data-mongodb. (s/f). Recuperado el 20 de junio de 2022, de

- <https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-data-mongodb>
33. MongoDB | Build Faster. Build Smarter. (s/f). MongoDB. Recuperado el 20 de junio de 2022, de <https://www.mongodb.com>
 34. MUI: The React component library you always wanted. (s/f). Recuperado el 20 de junio de 2022, de <https://mui.com/>
 35. Network ports · Cloudflare Fundamentals docs. (s/f). Recuperado el 22 de junio de 2022, de <https://developers.cloudflare.com/fundamentals/get-started/reference/network-ports/>
 36. Npm. (s/f). Recuperado el 20 de junio de 2022, de <https://www.npmjs.com/>
 37. oznu/cloudflare-ddns—Docker Image | Docker Hub. (s/f). Recuperado el 22 de junio de 2022, de <https://hub.docker.com/r/oznu/cloudflare-ddns>
 38. Qrreader. (s/f). Npm. Recuperado el 22 de junio de 2022, de <https://www.npmjs.com/package/qrreader>
 39. Rasheed, Y., Qutqut, M. H., & Almasalha, F. (2019). Overview of the Current Status of NoSQL Database. 8.
 40. React – A JavaScript library for building user interfaces. (s/f). Recuperado el 20 de junio de 2022, de <https://reactjs.org/>
 41. @react-google-maps/api. (s/f). Npm. Recuperado el 22 de junio de 2022, de <https://www.npmjs.com/package/@react-google-maps/api>
 42. React-qr-code. (s/f). Npm. Recuperado el 20 de junio de 2022, de <https://www.npmjs.com/package/react-qr-code>
 43. Replication—MongoDB Manual. (s/f). Recuperado el 30 de agosto de 2022, de <https://www.mongodb.com/docs/manual/replication/>

44. Spring Boot Integration With MongoDB Tutorial. (s/f). MongoDB. Recuperado el 20 de junio de 2022, de <https://www.mongodb.com/compatibility/spring-boot>
45. Spring Boot—Application Properties. (s/f). Recuperado el 21 de junio de 2022, de https://www.tutorialspoint.com/spring_boot/spring_boot_application_properties.htm
46. Spring Framework. (s/f). Recuperado el 30 de agosto de 2022, de <https://spring.io/projects/spring-framework>
47. Spring makes Java simple. (s/f). Spring. Recuperado el 20 de junio de 2022, de <https://spring.io/>
48. Srinivasan, A., Quadir, M. A., & Vijayakumar, V. (2015). Era of Cloud Computing: A New Insight to Hybrid Cloud. *Procedia Computer Science*, 50, 42–51. <https://doi.org/10.1016/j.procs.2015.04.059>
49. The API Design Platform and API Client. (s/f-a). Recuperado el 22 de junio de 2022, de <https://insomnia.rest/>
50. The API Design Platform and API Client. (s/f-b). Recuperado el 20 de octubre de 2022, de <https://insomnia.rest/>
51. Traversy Media (Director). (2020, octubre 9). Material UI React Tutorial. <https://www.youtube.com/watch?v=vyJU9efvUtQ>
52. Virtualización. (s/f). Recuperado el 30 de agosto de 2022, de <https://www.redhat.com/es/topics/virtualization>
53. Visual Studio Code—Code Editing. Redefined. (s/f). Recuperado el 20 de octubre de 2022, de <https://code.visualstudio.com/>
54. What is a Container? - Docker. (2021, noviembre 11). <https://www.docker.com/resources/what-container/>
55. What is an API? (s/f). Recuperado el 30 de agosto de 2022, de <https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces>

56. What is DNS? | How DNS works. (s/f). Cloudflare.
Recuperado el 20 de octubre de 2022, de
<https://www.cloudflare.com/learning/dns/what-is-dns/>
57. What is Scrum? (s/f). Scrum.Org. Recuperado el 23 de octubre de 2022, de
<https://www.scrum.org/resources/what-is-scrum>