“Sistema de control de acceso de invitados a barrios privados”

Autor: Soria Gava, Lucas Damián

Documentación de tesis

Facultad de Ingeniería

Carrera: Ingeniería en informática

Universidad de Mendoza

Mes* 2022

Tutor especialista: Mag. Ing. Córdoba, Diego

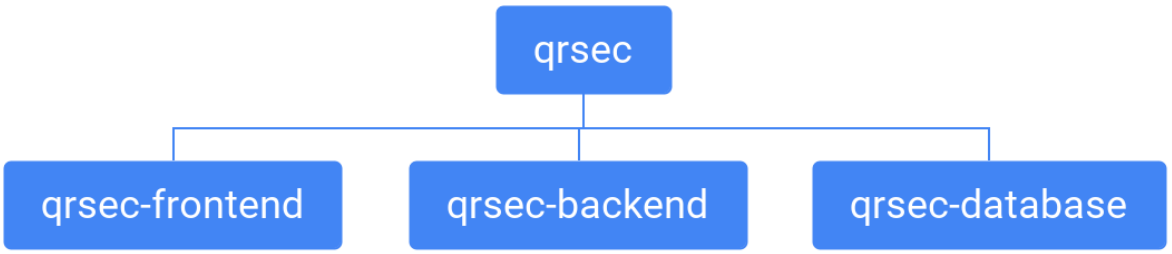
Índice:

4 Metodología:

4.1 Tecnologías usadas:

4.1.1 Estructura base del proyecto:

El código se encuentra divido 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.



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 desarrollada en este, y este a su vez se relaciona con la base de datos.

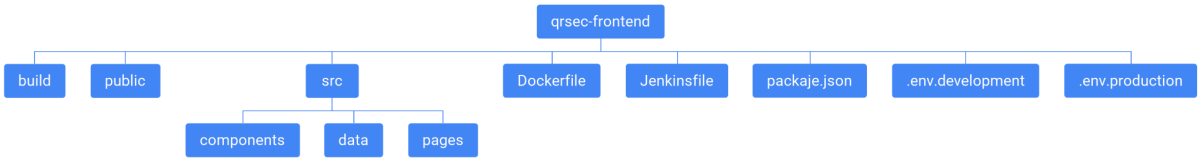
4.1.2 Diseño gráfico de la página:

Para tener una noción del diseño y la distribución del contenido de las páginas web que se desarrolló, 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 siguen 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 coolors.

4.1.3 Frontend:

El frontend está principalmente desarrollado en el lenguaje de programación JavaScript. Junto 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 las páginas. 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 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”.
Arbitrariamente para la programación del proyecto se decidió utilizar Visual Studio Code, uno de los IDEs más populares actualmente.

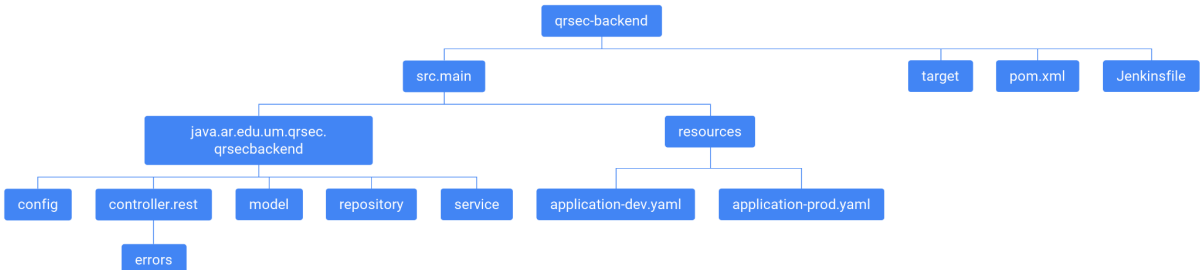


Imágen de elaboración propia

4.1.4 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 posibilitó 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, application-dev.yaml y application-prod.yaml. El primer archivo contiene todas las dependencias del proyecto, el segundo contiene 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 los dos archivos restantes contienen 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 cinco carpetas principales ubicadas dentro de src: config, controller, model, repository y service. Dentro del directorio *config* se encuentran las clases encargadas de las

configuraciones de la aplicación, como por ejemplo, las políticas de CORS, *model* contiene las clases que representan los documentos de la base de datos y cómo se relacionan entre sí, *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.
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 muy popular actualmente para el desarrollo en Java.



Imágen de elaboración propia

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

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

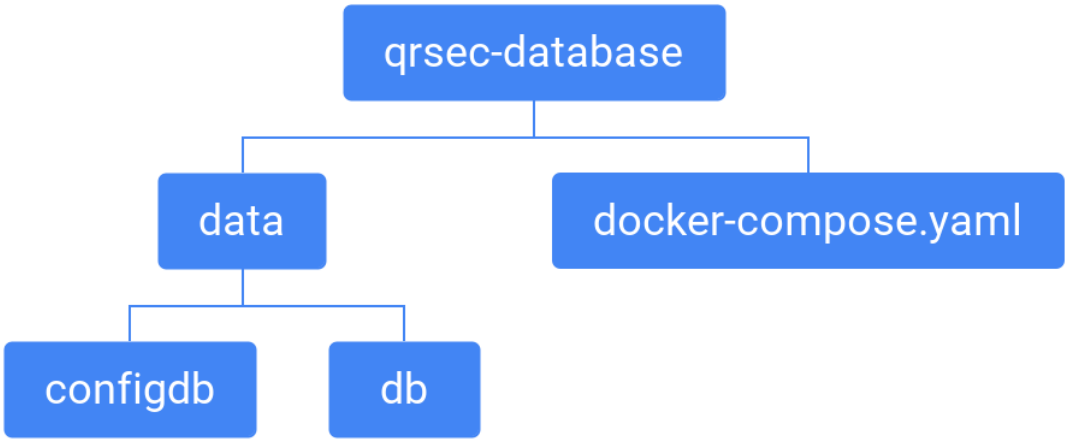
4.1.5 Base de datos:

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, no es posible utilizar una base de datos local para el desarrollo. Por eso 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.yaml 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 siguiente imagen.



Imágen de elaboración propia

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 del que se habló en la sección 4.1.1.

4.1.6 Herramienta de control de versiones:

Para el control de versiones se utilizó la herramienta Git, junto con la plataforma de desarrollo colaborativo GitHub.
El repositorio utilizado para almacenar el código del proyecto es:

https://github.com/LucaSor1a/qrsec

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 comúnmente utilizadas en la industria para la generación de los mismos. El mismo tiene la siguiente estructura:

- <tipo>: <resumen>

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

4.1.7 Virtualización:

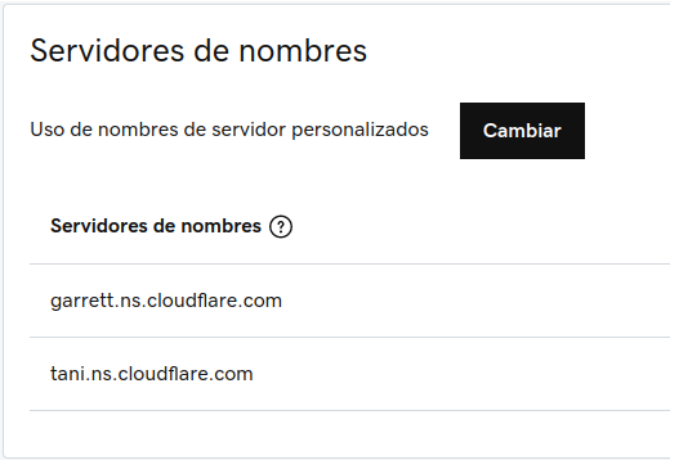
Pensando en un despliegue rápido y uniforme, sin importar la plataforma en la que se lo haga, se decidió utilizar Docker y generar contenedores a partir de las versiones de producción tanto del frontend como del backend. Para ello se necesita

generar los mismos a partir de las imágenes de Docker de openjdk (backend), node (frontend) y mongo (base de datos).
La configuración de la base de datos en Docker ya fue detallada en la sección 4.1.5, la creación de la imagen del backend puede hacerse a través de su mismo framework, y la creación de la imagen del frontend debe hacerse a través de una imagen de node con el siguiente archivo de configuración Dockerfile:

```
FROM node:16.13-alpine

# set working directory
WORKDIR /front

ENV PATH /qrsec-frontend/node_modules/.bin:$PATH

# add app
COPY . ./

# install app dependencies
RUN npm install --silent
RUN npm run build
RUN npm install -g serve

# start app
CMD ["serve", "-s", "build", "-p", "80"]
```

4.1.8 Despliegue:

Una vez se crearon las versiones de producción del frontend y backend, se necesita un servidor desde el cual exponer la aplicación 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://lsoria.com. Por último, el DNS seleccionado fue Cloudflare, ya que este ofrece sus servicios gratuitamente y su configuración para el dominio mencionado anteriormente se muestra en las siguientes imágenes. Adicionalmente, Cloudflare ofrece servicios de prevención contra ataques DoS y DDoS.



Configuración de DNS en GoDaddy (Imagen de elaboración propia)

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

Configuración de DNS en Cloudflare(Imagen de elaboración propia)

4.2 Desarrollo:

4.2.1 Diseño gráfico:

4.2.1.1 Sprint 1:

4.2.1.1.1 Definición de una paleta de colores:

A partir de la herramienta de diseño mencionada en la sección 4.1.2, y siguiendo los significados de la *teoría de colores*, se definieron cinco principales:

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

Fuente: https://coolors.co/ebebeb-6fec82-ff101f-2251bf-103c89

4.2.1.1.2 Pantalla de creación de una invitación:

Estas imágenes corresponden a la secuencia de acciones 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”.

Pantalla base de creación de invitación

Botón desplegable de selección de invitados

Imágenes de elaboración propia

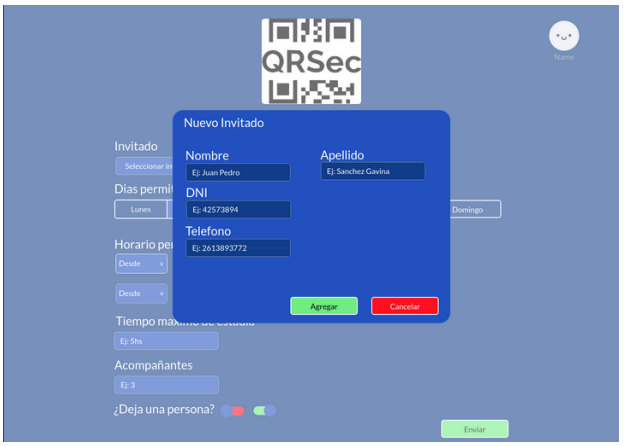
4.2.1.1.3 Pantalla para compartir la invitación:

Es posible visualizar esta pantalla una vez se genera una nueva invitación. Contiene el link que luego el invitado debe visitar para poder visualizar su invitación y un botón para copiar el mismo al portapapeles.

Imágen de elaboración propia

4.2.1.1.4 Pantalla de creación de un Invitado nuevo:

Esta pantalla se puede ver cuando uno hace click en ‘Agregar Invitado’ dentro del menú de selección de invitados. Permite agregar un nuevo invitado que esté relacionado al usuario que lo crea.



Imágen de elaboración propia

4.2.1.2 Sprint 2:

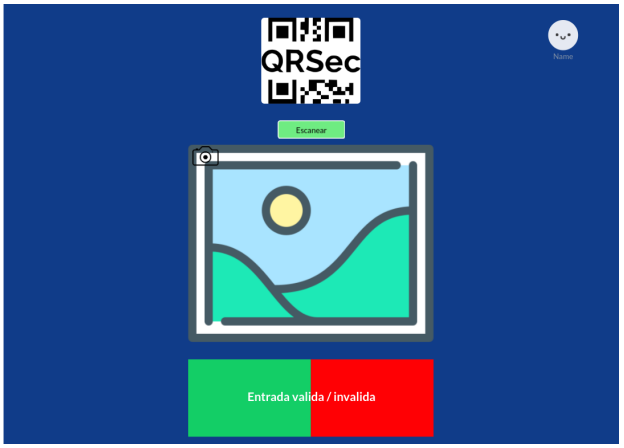
4.2.1.2.1 Pantalla de visualización de la invitación:

Esta es la pantalla a la que tiene acceso el invitado, en ella se puede ver el código QR que representa su invitación y un mapa que señala como llegar hasta la entrada al barrio al que fue invitado.

Imágen de elaboración propia

4.2.1.2.2 Pantalla de escaneo de invitaciones:

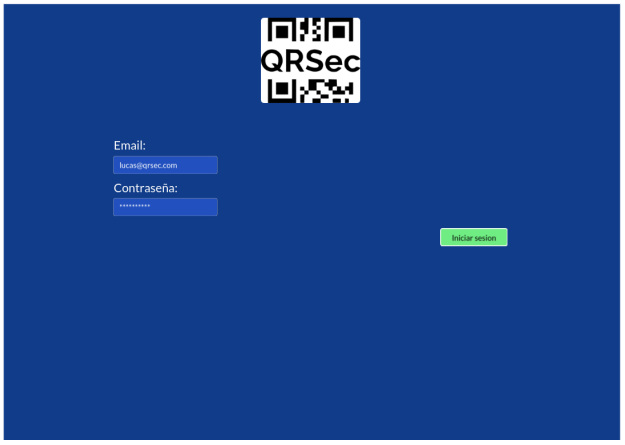
Esta es la pantalla a la que tendrá acceso el guardia de la entrada del barrio. Está compuesta de una sección que muestra la cámara de la barrera y debajo un recuadro que informa si la invitación que detecta es valida o invalida.



Imágen de elaboración propia

4.2.1.2.3 Pantalla de inicio de sesión:

En esta última pantalla se encuentra un simple formulario para ingresar los datos de inicio de sesión.



Imágen de elaboración propia

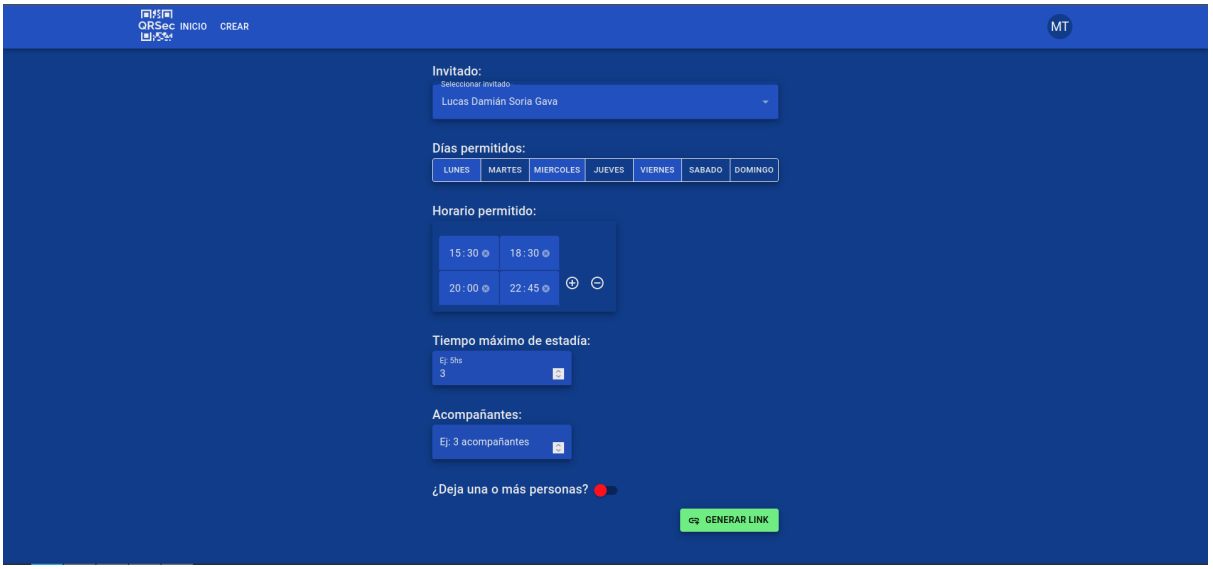
4.2.2 Desarrollo del Frontend:

4.2.2.1 Sprint 1:

4.2.2.1.1 Desarrollo de las pantallas de creación de invitación y compartir invitación:

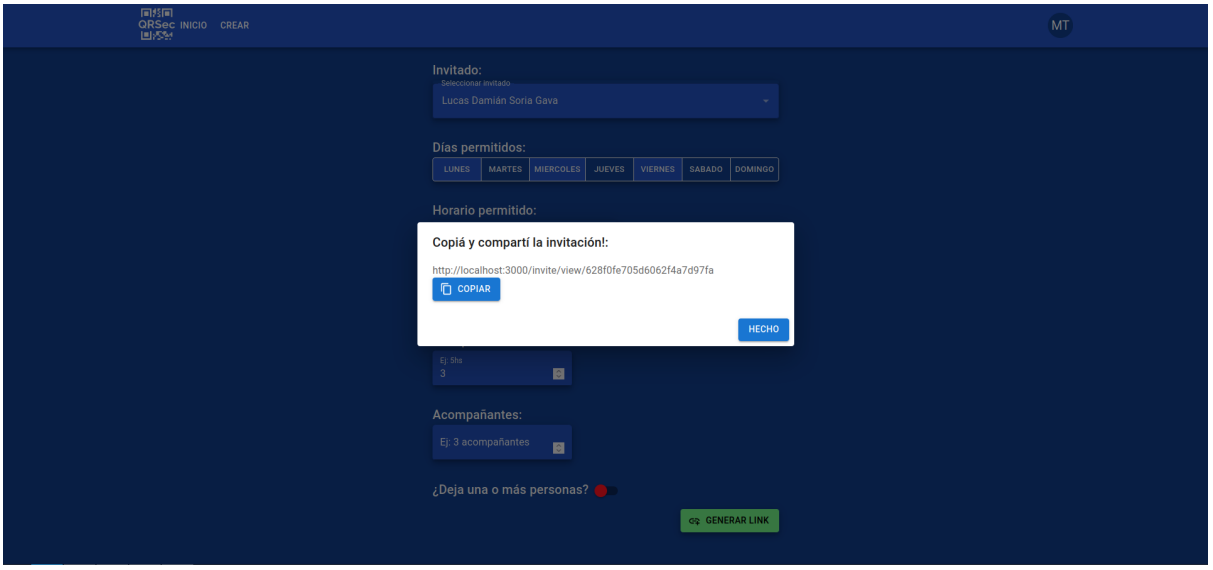
Luego de realizar el diseño de las primeras pantallas se decidió utilizar para el desarrollo del frontend la librería de Material UI (como ya se especificó en la sección 4.1.3) ya que esta ofrece componentes de React muy útiles y estilizados que son fáciles de modificar para que se ajusten a las necesidades del proyecto. Para poder utilizar dicha librería se consultó la documentación de la misma y se complementó la misma con tutoriales básicos de YouTube respecto a dichas tecnologías para dar un comienzo más ameno al proceso de aprendizaje. Dichos videos fueron: un tutorial de React y un tutorial de 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 4.1.3.
Finalmente se logró implementar una página capaz de:

- Tomar la lista de Invitados almacenada en la base de datos a través de la API 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.



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 un diálogo que permite copiar el link con la Invitación para compartir con el Invitado, como se muestra a continuación.



Imágen de elaboración propia

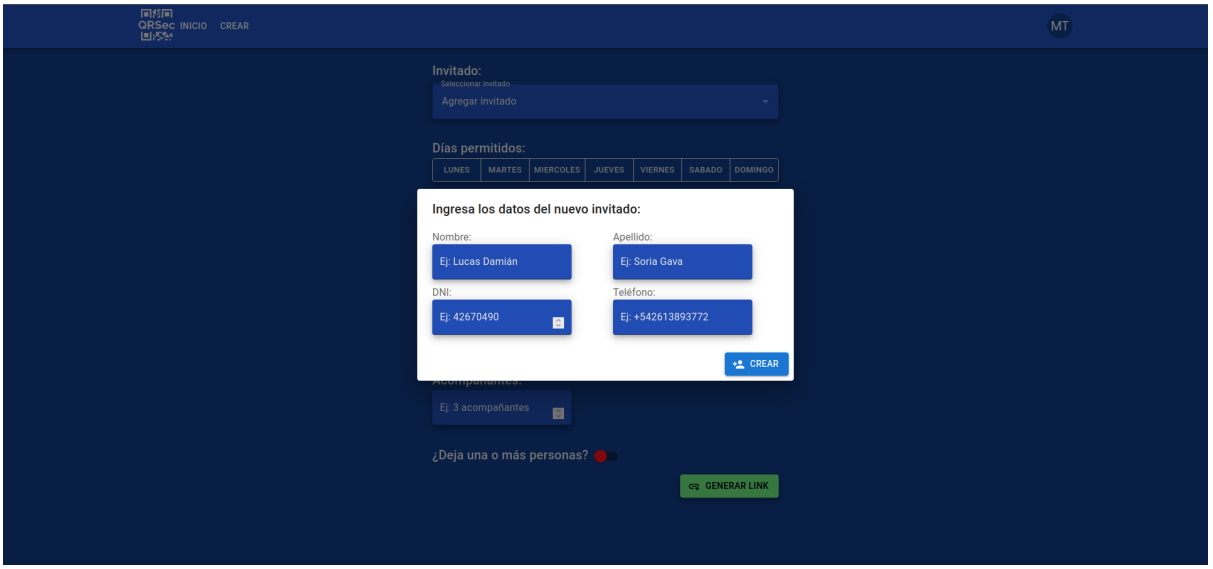
4.2.2.1.2 Desarrollo de un la barra superior:

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.

4.2.2.2 Sprint 2:

4.2.2.2.1 Desarrollo de la pantalla ‘Agregar invitado’:

Durante el segundo Sprint se desarrolló la funcionalidad de ‘Agregar invitado’ dentro del menú de selección de Invitados. Si esta opción es seleccionada se despliega un diálogo que permite completar la información pertinente para generar un nuevo Invitado. Una vez se completan los campos y se presiona el botón de ‘crear’, la página se comunica con el backend a través de la API y este crea el nuevo invitado. Así la próxima vez que se abra el menú desplegable, el nuevo invitado estará disponible.



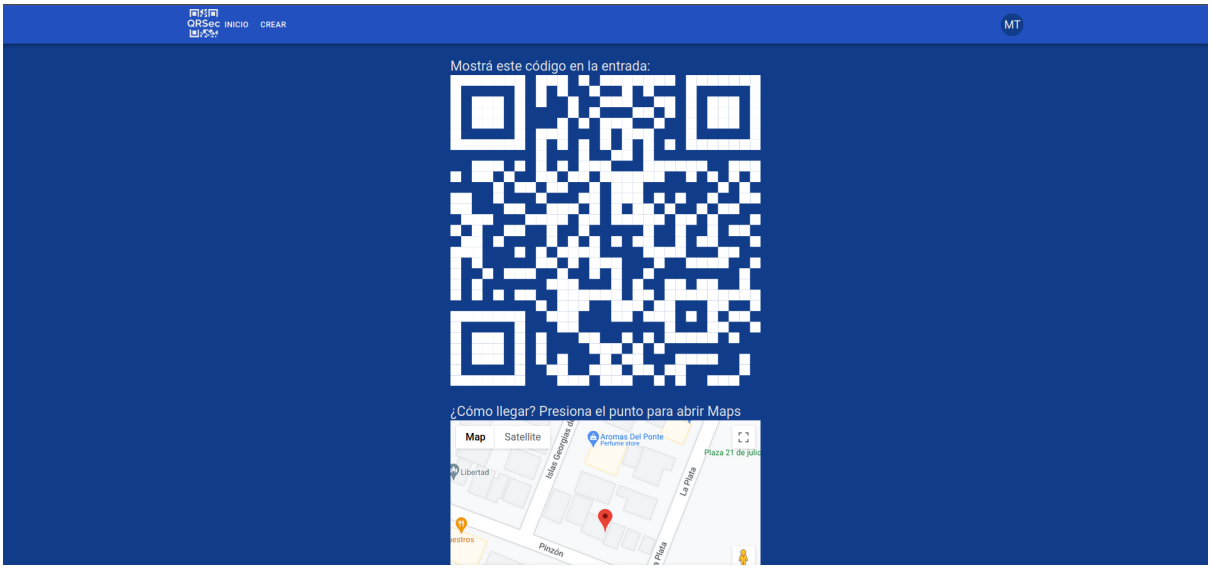
Imágen de elaboración propia

4.2.2.2.2 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 anteriormente. Esta permite que a través de un texto, en este caso el id de la invitación, se genere un código QR que puede ser configurado según la necesidad del usuario. En este caso se necesitó configurar el código para que tenga el 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 le importa a uno perder capacidad de codificación. En el caso de QRSec, el largo de la información a codificar es de 24 caracteres.

Adicionalmente debajo del código QR se localiza un mapa con la dirección de la entrada del barrio marcada, en caso de que el invitado nunca haya asistido al mismo. 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 misma a su

vez debe ser consumida a través de la librería de react-google-maps/api. Cabe notar que después de una cantidad preestablecida de peticiones a la API de Google por mes, este empieza a cobrar por su uso.



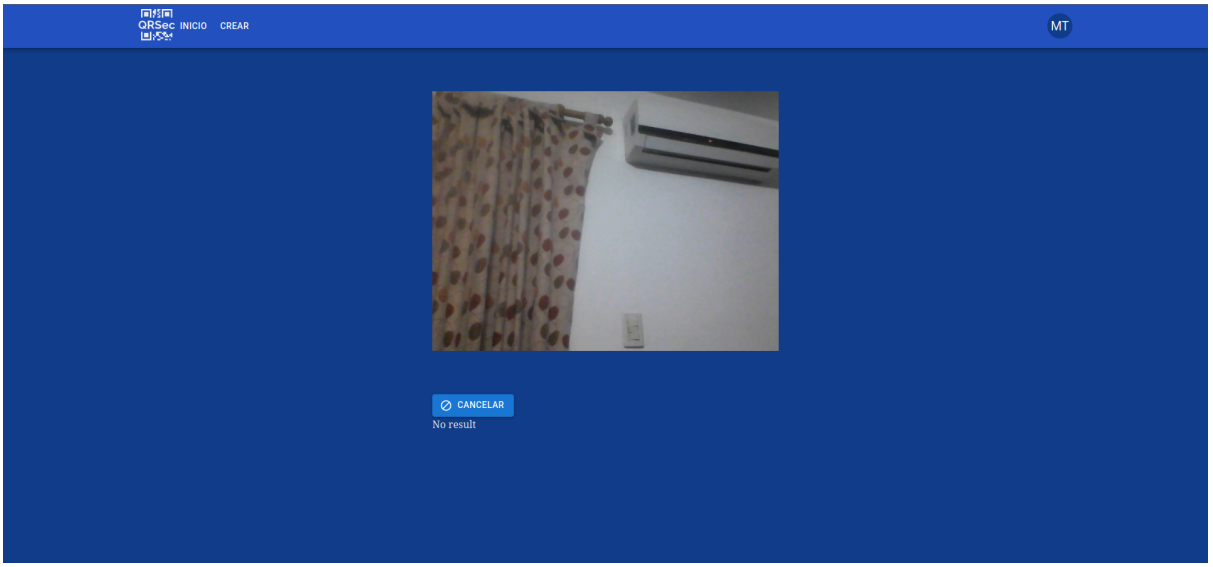
Imágen de elaboración propia

4.2.2.2.3 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.

Luego de realizar un escaneo exitoso del código, se hace una solicitud a la API con el id de la Invitación, en caso de que no exista esa entrada en la base de datos se envía un error 404 y la entrada es inválida. En el caso que la Invitación exista, esta se envía al frontend y se analiza la validez de sus campos, buscando que todos se cumplan, como por ejemplo, que la persona esté intentando ingresar dentro del horario permitido o en alguno de los días que tiene permitido ingresar.

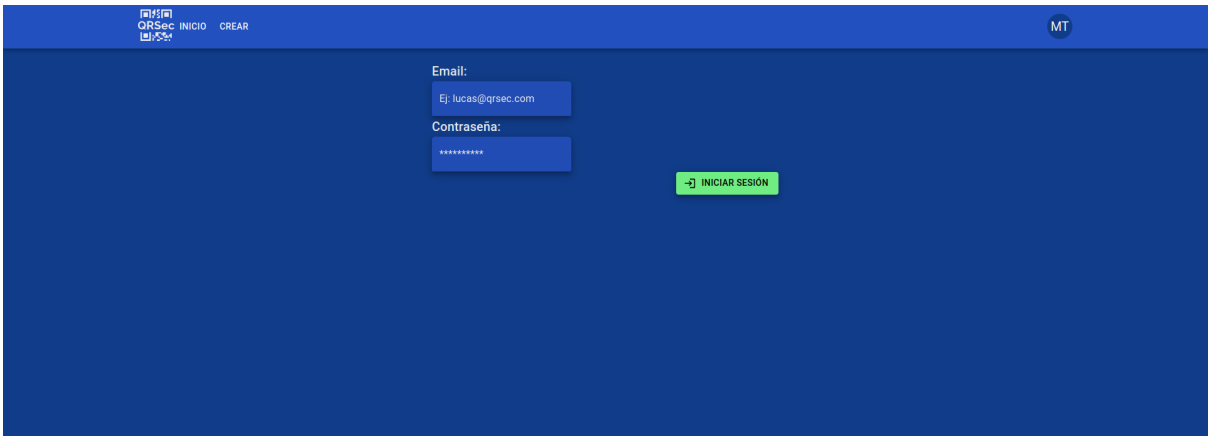
El escaneo del código resultó especialmente desafiante ya que las librerías que se probaron para realizar este proceso contenían todas problemas de compatibilidad ya sea con la versión de React que se utilizó (React 18), con el navegador o el sistema operativo. Por esta razón, se optó por un desarrollo del sistema desde cero, utilizando simplemente Javascript y HTML5.



Imágen de elaboración propia

4.2.2.2.4 Desarrollo de la pantalla de inicio de sesión:

Esta es una pantalla simple que contiene un pequeño formulario como se planteó en la sección 4.2.1.2.3 cuya funcionalidad es la de obtener el JWT (JSON Web Token) del usuario que utilizará la aplicación, con el objetivo de dirigirlo a la pantalla correspondiente (creación de Invitaciones para el residente y escaneo de códigos para el guardia) y poder mantener su sesión.



Imágen de elaboración propia

4.2.3 Desarrollo del Backend:

4.2.3.1 Sprint 1:

4.2.3.1.1 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 los constantes de

entorno que se encuentran en los archivos de application-dev.yaml y application-prod.yaml, descritos en la sección 4.1.4. Dicha configuración es:

```
spring:
  data:
      mongodb:
      authentication-database: admin
      username: root
      password: password
      database: qrsec
      port: 27017
      host: localhost
      auto-index-creation: true
  security:
      user:
      name: root
      password: password
```

Variables que concuerdan con la configuración de la base de datos definidos en la sección 4.1.5. La configuración presentada fue posible de desarrollar gracias a las guías de 'AmigosCode' en su tutorial de YouTube, y a la documentación de MongoDB respecto a Spring.

4.2.3.1.2 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, a continuación se listan mostrando su estereotipo, nombre, atributos y tipo de dato:

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

phone	String

<Enum> Authority
ADMIN
OWNER
GUARD

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

<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

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

<Class> House	
block	String
house	Integer

<Class> Location	
type	String
coordinates	List<Double>

Es importante destacar que cada clase con el estereotipo de Documento posee un atributo adicional *id*.
La última clase especificada (Location) sigue las especificaciones para una petición geoespacial al motor de base de datos MongoDB, según lo especifica su documentación.

4.2.3.1.3 Creación de los primeros endpoint:

Una vez creadas las clases que se relacionarán con la base de datos, es importante definir una clase *Repository* por cada una de ellas, en el caso de QRSec se haría 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/guest* y */api/invite*, para los cuales el primero tiene definido el método GET, para que la pantalla creación de invitación pueda listar los Invitados presentes en la base de datos. Para el segundo endpoint se definió el método POST para que la pantalla de creación de invitaciones pueda efectivamente generar una nueva Invitación. 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 temporalmente el desarrollo de estos ya que no son una parte indispensable del desarrollo de la aplicación.

4.2.3.1.4 Configuración de la política de CORS:

Un problema encontrado durante el desarrollo del frontend fue la política de CORS, la cual impedía poder hacer peticiones a la API desde la página web. Este pudo superarse gracias a la documentación de Spring y el tutorial de Baeldung al respecto, culminando en la creación de la Clase 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");
	}

}
```

4.2.3.2 Sprint 2:

4.2.3.2.1 Modificación de endpoints:

Ya que los endpoints básicos y sus configuraciones fueron mayormente programadas durante el primer Sprint, en este Sprint se modificaron los mismos para admitir nuevos métodos como por ejemplo POST en */api/guest* para poder agregar nuevos Invitados desde la pantalla desarrollada para ello en este Sprint, GET en /api/invite para poder validar los datos de una Invitación tras escanearla desde la página correspondiente y PATCH en la misma para poder modificar las horas de entrada y salida del barrio.

4.2.3.2.2 Inicio de sesión:

Por último, se programó el sistema de generación de JWT para mantener las sesiones de los usuarios y para poder relacionar automáticamente la generación de las Invitaciones o los nuevos Invitados al usuario que está realizando dichas peticiones a la API.

Referencias bibliográficas: