INFORME DE TESTING

DIGITAL BOOKING

INTRODUCCION

Digital Booking es una plataforma web desarrollada con tecnologías como React, HTML y CSS, que permite a los usuarios seleccionar hospedajes temporales. Para garantizar la calidad y funcionalidad de la plataforma, se llevaron a cabo diversas pruebas mediante diferentes herramientas de testing.

Una de las herramientas utilizadas fue Jest que es un framework de testing de JavaScript utilizado para realizar pruebas unitarias y de integración. Uno de los beneficios de utilizarla es que es muy rápido, lo que permite una ejecución rápida de las pruebas. Además, Jest proporciona una amplia variedad de funciones, como la simulación de eventos y la gestión de módulos, lo que facilita la creación de pruebas.

Otra herramienta utilizada fue React Testing Library, que es una biblioteca de pruebas para aplicaciones de React. React Testing Library proporciona funciones para simular interacciones de usuario y para buscar elementos en la interfaz de usuario. Uno de los beneficios de esta biblioteca es que está diseñada para centrarse en las pruebas que importan para los usuarios, lo que significa que las pruebas se basan en la funcionalidad y no en los detalles de implementación.

Se utilizó React Testing Scripts, que es una biblioteca de pruebas para aplicaciones de React y se ejecutan en navegadores. Esta biblioteca proporciona funciones para simular interacciones de usuario en el navegador, como hacer clic en botones y rellenar formularios. Uno de los beneficios de React Testing Scripts es que se integra fácilmente con otras herramientas de testing, como Jest.

Por último, se utilizó Selenium, que es un framework de pruebas de automatización que se utiliza para probar aplicaciones web. Selenium se utiliza para automatizar pruebas de usuario en navegadores web, como hacer clic en botones y navegar por diferentes páginas. Uno de los beneficios es que se puede utilizar con varios navegadores, lo que permite probar la aplicación en diferentes plataformas.

SPRINT 1

En el primer sprint el objetivo principal fue asegurarse de que todos los componentes de la plataforma se renderizaran correctamente y que los campos de entrada de datos del registro y inicio de sesión se validaran correctamente.

Esto es esencial para garantizar que los datos ingresados por los usuarios se procesen de manera adecuada y se almacenen correctamente en la base de datos. Además, la validación de los datos de entrada también ayuda a prevenir errores y a mejorar la experiencia de usuario al proporcionar comentarios instantáneos sobre posibles problemas con los datos ingresados.

En general, el objetivo del primer sprint fue establecer una base sólida para el desarrollo posterior de la plataforma, asegurándose de que los aspectos fundamentales de la funcionalidad y la validación de datos se implementen correctamente.

PRUEBAS REALIZADAS

Manual Testing:

Se llevaron a cabo diversas pruebas para asegurar la calidad del sistema en desarrollo. Se realizaron smoke tests para verificar el correcto funcionamiento de los componentes básicos del sistema. Además, se llevaron a cabo pruebas de integración para comprobar que los diferentes módulos del sistema funcionan adecuadamente y se comunican entre sí sin problemas.

Se prestó especial atención a los nuevos componentes introducidos, como el datepicker y el location input, realizando pruebas de regresión para garantizar que no afecten negativamente el sistema existente. Se verificó que los datos introducidos a través de estos componentes lleguen correctamente a la base de datos.
Si desea visualizar los resultados de estas pruebas, puede hacer click en el siguiente enlace para acceder a los informes correspondientes.

Testing DB Sprint 1

Automation testing:

Se crearon scripts de pruebas mediante el uso de distintas herramientas como Jest y React-testing library ya que en este primer sprint las funcionalidades desarrolladas serán base para pruebas de regresión hasta el final del proyecto, por lo tanto se decidió que sean automatizadas para evitar cualquier error humano. Pudiéndose en próximos sprints agregar más.
Las mismas se encuentran en el proyecto.

Para el back end se utilizo Postman y Swagger UI para verificar el correcto funcionamiento de la API y la base de datos creada.

SPRINT 2

El enfoque del segundo sprint del proyecto Digital Booking fue garantizar que los componentes del placeholder y el datepicker estuvieran funcionando correctamente, y que los datos ingresados se enviaran a la base de datos sin errores. Se espera que, cuando se integre la base de datos, los datos se envíen correctamente gracias a las pruebas realizadas en este sprint. Todo esto contribuyó a un desarrollo exitoso del proyecto y a una experiencia positiva para los usuarios.

Esto es esencial para garantizar que los datos ingresados por los usuarios se procesen de manera adecuada y se almacenen correctamente en la base de datos. Además, la validación de los datos de entrada también ayuda a prevenir errores y a mejorar la experiencia de usuario al proporcionar comentarios instantáneos sobre posibles problemas con los datos ingresados.

En general, el objetivo del primer sprint fue establecer una base sólida para el desarrollo posterior de la plataforma, asegurándose de que los aspectos fundamentales de la funcionalidad y la validación de datos se implementen correctamente.

Se comenzó a realizar pruebas con Selenium IDE.

PRUEBAS REALIZADAS

Manual Testing:

Se llevaron a cabo diversas pruebas para asegurar la calidad del sistema en desarrollo. Se realizaron smoke tests para verificar el correcto funcionamiento de los componentes básicos del sistema. Además, se llevaron a cabo pruebas de integración para comprobar que los diferentes módulos del sistema funcionan adecuadamente y se comunican entre sí sin problemas.

Se prestó especial atención a los nuevos componentes introducidos, como el datepicker y el location input, realizando pruebas de regresión para garantizar que no afecten negativamente el sistema existente. Se verificó que los datos introducidos a través de estos componentes lleguen correctamente a la base de datos.

Automation testing:

Durante el desarrollo del proyecto, se continuó con la implementación de pruebas automatizadas para asegurar la calidad del sistema. Se utilizaron herramientas como Jest, Selenium y Selenium IDE para crear scripts de pruebas que permitieron verificar el correcto funcionamiento de las funcionalidades desarrolladas.

Estas pruebas automatizadas fueron diseñadas para ser reutilizables y escalables, permitiendo su uso en próximos sprints y en futuros proyectos. De esta manera, se aseguró que el sistema cumpla con los requerimientos establecidos y se previnieron posibles errores humanos en la ejecución de pruebas manuales.

Para el back end, se utilizó nuevamente Postman y Swagger UI para realizar pruebas de API y verificar la correcta integración con la base de datos. Estas pruebas se llevaron a cabo de manera automatizada y se incluyeron en el proceso de integración continua del proyecto.

Todos los scripts de pruebas, tanto de front-end como de back-end, se encuentran integrados en el proyecto y se ejecutan de forma automática después de cada cambio en el código fuente. De esta manera, se asegura que cualquier modificación en el sistema no afecte negativamente las funcionalidades ya implementadas.

SPRINT 3

Durante el tercer sprint de desarrollo, se continuó con el enfoque en las pruebas funcionales y unitarias utilizando React Testing Library y Jest para validar las nuevas funcionalidades de la aplicación. El principal énfasis se puso en las reservas y el registro de usuarios para garantizar que estas características estén funcionando correctamente y de manera consistente.

Además, se automatizaron más pruebas utilizando Selenium IDE en diferentes navegadores web para asegurar que la aplicación funcione como se espera en distintos entornos. Esto permitió realizar pruebas más exhaustivas y precisas, y detectar cualquier problema que pueda surgir en diferentes plataformas.
En general, este sprint se centró en garantizar que las nuevas funcionalidades de la aplicación cumplan con los requisitos y expectativas del usuario final.

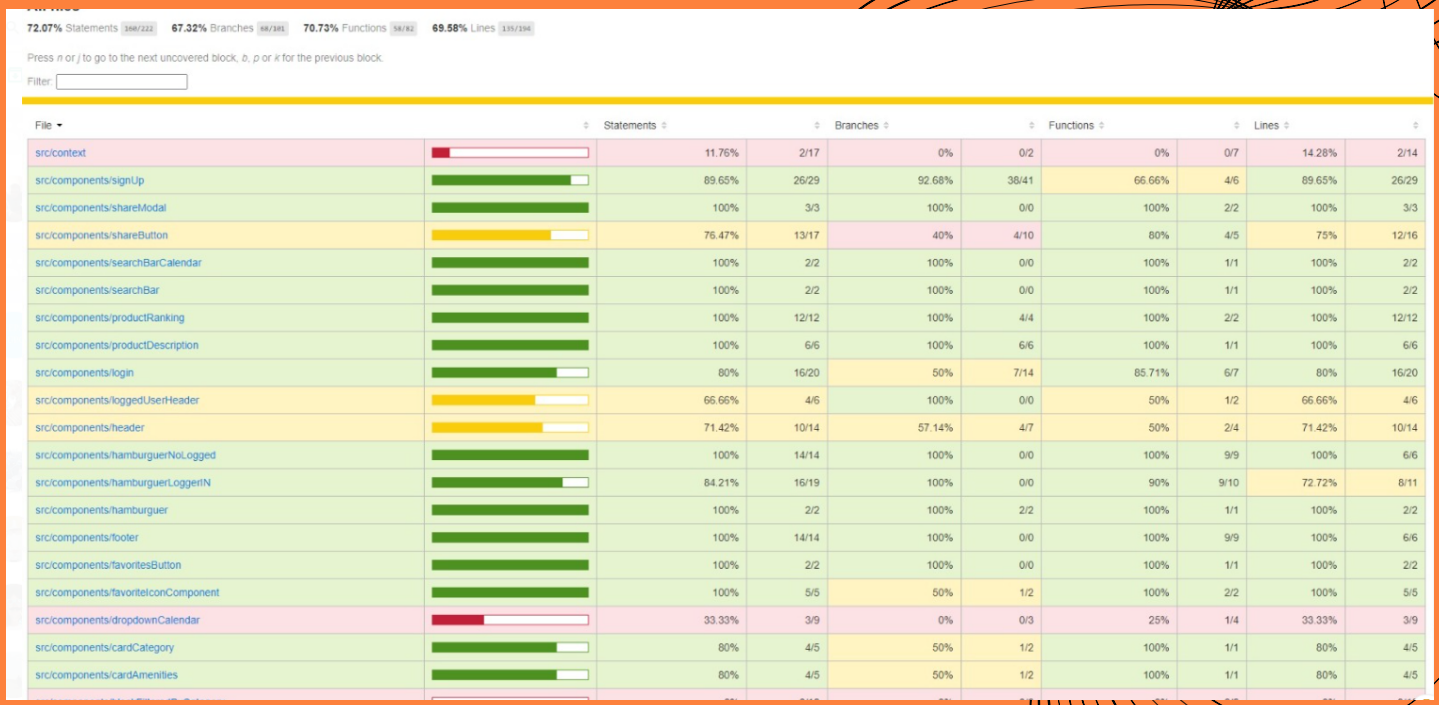
Es importante destacar que en la mayoría de las pruebas unitarias realizadas durante este tercer sprint, se logró alcanzar un coverage superior al 70%. Esto significa que estamos probando una gran cantidad de casos posibles y que la mayoría del código está siendo validado en las pruebas unitarias.

Esto nos da una mayor confianza en la calidad del código, ya que nos aseguramos de que el mismo esté cubierto por las pruebas y de que los resultados sean los esperados. Además, al tener un alto coverage, nos permite detectar rápidamente cualquier problema que pueda surgir en el futuro y corregirlo antes de que afecte al usuario final.

Test front end.

```
Test Suites: 3 failed, 13 passed, 16 total
Tests:       4 failed, 34 passed, 38 total
Snapshots:   0 total
Time:        4.698 s
```

Coverage



72.07% Statements **67.32%** Branches **70.73%** Functions **69.58%** Lines

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
src/context	11.76%	2/17	0%	0/2	0%	0/7	14.28%	2/14
src/components/signUp	89.65%	26/29	92.68%	38/41	66.66%	4/6	89.65%	26/29
src/components/shareModal	100%	3/3	100%	0/0	100%	2/2	100%	3/3
src/components/shareButton	76.47%	13/17	40%	4/10	80%	4/5	75%	12/16
src/components/searchBarCalendar	100%	2/2	100%	0/0	100%	1/1	100%	2/2
src/components/searchBar	100%	2/2	100%	0/0	100%	1/1	100%	2/2
src/components/productRanking	100%	12/12	100%	4/4	100%	2/2	100%	12/12
src/components/productDescription	100%	6/6	100%	6/6	100%	1/1	100%	6/6
src/components/login	80%	16/20	50%	7/14	85.71%	6/7	80%	16/20
src/components/loggedUserHeader	66.66%	4/6	100%	0/0	50%	1/2	66.66%	4/6
src/components/header	71.42%	10/14	57.14%	4/7	50%	2/4	71.42%	10/14
src/components/hamburguerNoLogged	100%	14/14	100%	0/0	100%	9/9	100%	6/6
src/components/hamburguerLoggerIN	84.21%	16/19	100%	0/0	90%	9/10	72.72%	8/11
src/components/hamburguer	100%	2/2	100%	2/2	100%	1/1	100%	2/2
src/components/footer	100%	14/14	100%	0/0	100%	9/9	100%	6/6
src/components/favoritesButton	100%	2/2	100%	0/0	100%	1/1	100%	2/2
src/components/favoriteIconComponent	100%	5/5	50%	1/2	100%	2/2	100%	5/5
src/components/dropdownCalendar	33.33%	3/9	0%	0/3	25%	1/4	33.33%	3/9
src/components/cardCategory	80%	4/5	50%	1/2	100%	1/1	80%	4/5
src/components/cardAmenities	80%	4/5	50%	1/2	100%	1/1	80%	4/5

SPRINT 4

Durante el cuarto sprint de desarrollo, el enfoque principal estuvo en el testeo del back-end utilizando JUnit y Mockito para validar las funcionalidades nuevas y existentes del proyecto. El objetivo era asegurarse de que los diferentes componentes del back-end, como los repositorios, los servicios y los controladores, estuvieran funcionando correctamente. Se logró un coverage superior al 50% del proyecto, con un enfoque particular en los paquetes fundamentales mencionados, los cuales alcanzaron un coverage superior al 70%.

Además, se continuó con la implementación de Selenium IDE para probar el proyecto terminado, lo que permitió detectar y corregir cualquier problema que pudiera afectar a los usuarios finales. Las pruebas de regresión también fueron satisfactorias, lo que significa que los cambios realizados no afectaron negativamente a las funcionalidades existentes del proyecto.

En general, este sprint se centró en garantizar que el back-end del proyecto cumpliera con los requisitos y expectativas del usuario final, y se logró un alto nivel de cobertura en las pruebas. Esto nos brinda mayor confianza en la calidad del código y nos permite detectar y corregir rápidamente cualquier problema que pueda surgir en el futuro.

Services Test

com.dh.digital_booking_back.service in digital_booking_back: 103 total, 103 passed

TipoliticaServiceTest		147 ms
listarTipoliticas_TipoliticaRepositoryReturnsEmptyList_ThrowsResourceNotFoundException	passed	134 ms
buscarTipoliticaXId_TipoliticaDoesNotExist_ThrowsResourceNotFoundException	passed	6 ms
buscarTipoliticaXId_TipoliticaExists_ReturnsTipolitica	passed	1 ms
eliminarTipolitica_TipoliticaDoesNotExist_ThrowsResourceNotFoundException	passed	1 ms
registrarTipolitica_InvalidTipoliticaDTO_ThrowsBadRequestException	passed	1 ms
registrarTipolitica_ValidTipoliticaDTO_ReturnsSavedTipolitica	passed	2 ms
listarTipoliticas_TipoliticaRepositoryReturnsNonEmptyList_ReturnsSameList	passed	1 ms
editarTipolitica_TipoliticaDoesNotExist_ThrowsResourceNotFoundException	passed	1 ms
UsuarioServiceTest		349 ms
buscarUsuarioXId_whenNoExiste	passed	327 ms
buscarUsuarioXId_whenExiste	passed	1 ms
eliminarUsuarioXId	passed	2 ms
listarUsuarios_whenListaNoVacia	passed	2 ms
buscarUsuarioXemail_whenExiste	passed	2 ms
editarUsuarioXId_whenNoExiste	passed	2 ms
loadUserByUsername_whenNoExiste	passed	2 ms
registrarUsuario_whenYaExisteEmail	passed	1 ms
listarUsuarios_whenListaVacia	passed	1 ms
loadUserByUsername_whenExiste	passed	2 ms
registrarUsuario_whenDatosCorrectos	passed	3 ms
buscarUsuarioXemail_whenNoExiste	passed	2 ms
registrarUsuario_whenFaltanDatos	passed	2 ms
LikeServiceTest		43 ms
testEliminarLikeXId	passed	30 ms
testListaNoVacia	passed	
testListaNoVaciaEmpty	passed	
testEliminarLikeXIdNotFound	passed	2 ms
testListarLikesEmpty	passed	1 ms
testCrearLikeProductoNotFound	passed	2 ms
testCrearLikeUsuarioNotFound	passed	2 ms
testCrearLike	passed	2 ms
testBuscarLikeXIdNotFound	passed	1 ms
testListarLikes	passed	1 ms
testEliminarLikeXIdReferenced	passed	1 ms
testBuscarLikeXId	passed	1 ms
PuntuacionServiceTest		34 ms
eliminarPuntuacionTest()	passed	25 ms
registrarPuntuacionTest()	passed	4 ms
listarPuntuacionXProductoIdTest()	passed	1 ms
listarPuntuacionesTest()	passed	2 ms
buscarPuntuacionXIdTest()	passed	2 ms
CategoriaServiceTest		21 ms
Prueba listarCategorias - Lista vac&#65533;a	passed	13 ms
Prueba listarCategorias - Lista con elementos	passed	1 ms
Prueba crearCategoria - Categor&#65533;a creada correctamente	passed	1 ms
Prueba buscarCategoriaXIdCategor&#65533;a no encontrada	passed	1 ms
Prueba actualizarCategoria - Categor&#65533;a no encontrada	passed	1 ms
Prueba eliminarCategoria - Categor&#65533;a no encontrada	passed	1 ms
Prueba eliminarCategoria - NO se puede eliminar categoria por estar asociada a un producto	passed	1 ms
Prueba actualizarCategoria - Categor&#65533;a no encontrada	passed	1 ms
Prueba buscarCategoriaXId - Categor&#65533;a encontrada	passed	1 ms
CaracteristicaServiceTest		21 ms
testListarCaracteristicas()	passed	12 ms
testEliminarCaracteristicaXIdConExcepcion()	passed	2 ms
testEliminarCaracteristicaXId()	passed	1 ms
testListarCaracteristicasConExcepcion()	passed	1 ms
testBuscarCaracteristicaXIdConExcepcion()	passed	1 ms
testBuscarCaracteristicaXId()	passed	2 ms
testCrearCaracteristica()	passed	2 ms

PoliticaServiceTest			32 ms
buscarPoliticaXId_successful()	passed	17 ms
eliminarPoliticaXId_successful()	passed	2 ms
editarPolitica_notFound()	passed	1 ms
buscarPoliticaXId_notFound()	passed	1 ms
crearPolitica_invalidFields()	passed	1 ms
eliminarPoliticaXId_referenced()	passed	2 ms
crearPolitica_badRequest()	passed	2 ms
crearPolitica_tipoliticaNotFound()	passed	2 ms
listarPoliticas_successful()	passed	1 ms
eliminarPoliticaXId_notFound()	passed	2 ms
listarPoliticas_notFound()	passed	1 ms
ReservaServiceTest			22 ms
shouldThrowExceptionWhenNoReservasFoundForProductoId()	passed	2 ms
shouldThrowExceptionWhenNoReservasFoundForProducto()	passed	2 ms
shouldFindReservaById()	passed	1 ms
shouldListReservasByProductoId()	passed	1 ms
shouldThrowExceptionWhenReservaNotFound()	passed	2 ms
shouldThrowExceptionWhenProductoNotFoundWhileCreatingReserva()	passed	4 ms
shouldThrowExceptionWhenNoReservasFound()	passed	2 ms
shouldThrowExceptionWhenProductoNotFoundForReserva()	passed	1 ms
shouldListAllReservas()	passed	1 ms
shouldCreateReserva()	passed	6 ms
RolServiceTest			18 ms
buscarRolXId_RolNoExistente_ExcepcionLanzada()	passed	12 ms
eliminarRolXId_RolExists_RolDeleted()	passed	1 ms
listarRoles_NoRolesExist_ResourceNotFoundExceptionThrown()	passed	1 ms
eliminarRolXId_RolDoesNotExist_ResourceNotFoundExceptionThrown()	passed	1 ms
listarRoles_RolesExist_ReturnsListOfRoles()	passed	1 ms
eliminarRolXId_RolIsReferenced_BadRequestExceptionThrown()	passed	1 ms
buscarRolXId_RolExistente_RolRetornado()	passed	1 ms
UbicacionServiceTest			37 ms
testBuscarUbicacionXId_ResourceNotFoundException()	passed	25 ms
testCrearUbicacion()	passed	1 ms
testEliminarUbicacionXId_ResourceNotFoundException()	passed	1 ms
testListarUbicaciones_ResourceNotFoundException()	passed	1 ms
testListarUbicaciones()	passed	1 ms
testBuscarUbicacionXId()	passed	1 ms
testActualizarUbicacion_BadRequestException()	passed	4 ms
testEliminarUbicacionXId()	passed	2 ms
testActualizarUbicacion_ResourceNotFoundException()	passed	1 ms
ImagenServiceTest			
buscarImagenXId_IdInvalido_ResourceNotFoundException()	passed	
buscarImagenXId_IdValido_ImagenRetornada()	passed	
eliminarImagenXId_IdInvalido_ResourceNotFoundException()	passed	
eliminarImagenXId_IdValido_ImagenEliminada()	passed	
listarImagenes_ListaNoVacia_ListaRetornada()	passed	
listarImagenes_ListaVacia_ResourceNotFoundException()	passed	
ProductoServiceTest			
buscarProductoXId_debeLanzarExcepcionResourceNotFoundException()	passed	
listarProductos_debeRetornarListaDeProductos()	passed	
eliminarProducto_debeLanzarExcepcionResourceNotFoundException()	passed	
buscarProductoXId_debeRetornarProducto()	passed	
crearProducto_debeRetornarProductoCreado()	passed	

Controller Test

com.dh.digital_booking_back.controller in digital_booking_back: 45 total, 45 passed

ProductoControllerTest			134 ms
eliminarProducto_thenReturn200	passed	123 ms	
buscarProductoXId_thenReturnProducto	passed	7 ms	
buscarProductoXCiudad_thenReturnList	passed	1 ms	
listarProductosAleatorio_thenReturnList	passed	1 ms	
buscarProductosXCategoria_thenReturnList	passed	1 ms	
listarProductos_thenReturnList	passed	1 ms	
LikeControllerTest			560 ms
buscarLikeXIdTest()	passed	455 ms	
listarLikesTest()	passed	11 ms	
listarLikesXUsuarioTest()	passed	10 ms	
eliminarLikeTest()	passed	10 ms	
listarLikesXProductoTest()	passed	8 ms	
eliminarLike_idExistente_DebeRetornar200()	passed	66 ms	
CategoriaControllerTest			22 ms
testEliminarCategoria()	passed	22 ms	
PoliticaControllerTest			17 ms
testEliminarPolitica()	passed	14 ms	
testListarPoliticas()	passed	1 ms	
testAgregarPolitica()	passed	1 ms	
ReservaControllerTest			18 ms
listarReservasTest()	passed	14 ms	
listarReservasXProductoIdTest()	passed	1 ms	
eliminarReservaTest()	passed	1 ms	
listarReservasXUsuarioIdTest()	passed	1 ms	
buscarReservaXIdTest()	passed	1 ms	
TipoliticaControllerTest			17 ms
testListarTipoliticas()	passed	13 ms	
testBuscarCategoriaXId()	passed	1 ms	
testEditarTipolitica()	passed	1 ms	
testAgregarTipolitica()	passed	1 ms	
testEliminarTipolitica()	passed	1 ms	
UbicacionControllerTest			18 ms
listarUbicaciones_deberiaRetornarLista()	passed	13 ms	
agregarUbicacionTest()	passed	1 ms	
eliminarUbicacionTest()	passed	1 ms	
listarUbicaciones_deberiaLanzarResourceNotFoundException()	passed	3 ms	
PuntuacionControllerTest			13 ms
eliminarPuntuacionXId_success()	passed	13 ms	
ImagenControllerTest			17 ms
agregarImagenTest()	passed	12 ms	
eliminarImagenTest()	passed	1 ms	
listarImagenesTest()	passed	2 ms	
editarImagenTest()	passed	1 ms	
buscarImagenTest()	passed	1 ms	
RolControllerTest			18 ms
testEditarRol()	passed	13 ms	
testBuscarRolXId()	passed	2 ms	
testAgregarRol()	passed	1 ms	
testEliminarRol()	passed	1 ms	
testListarRoles()	passed	1 ms	
UsuarioControllerTest			16 ms
shouldAddUser()	passed	14 ms	
shouldReturnListOfUsers()	passed	1 ms	
shouldFindUserById()	passed	1 ms	

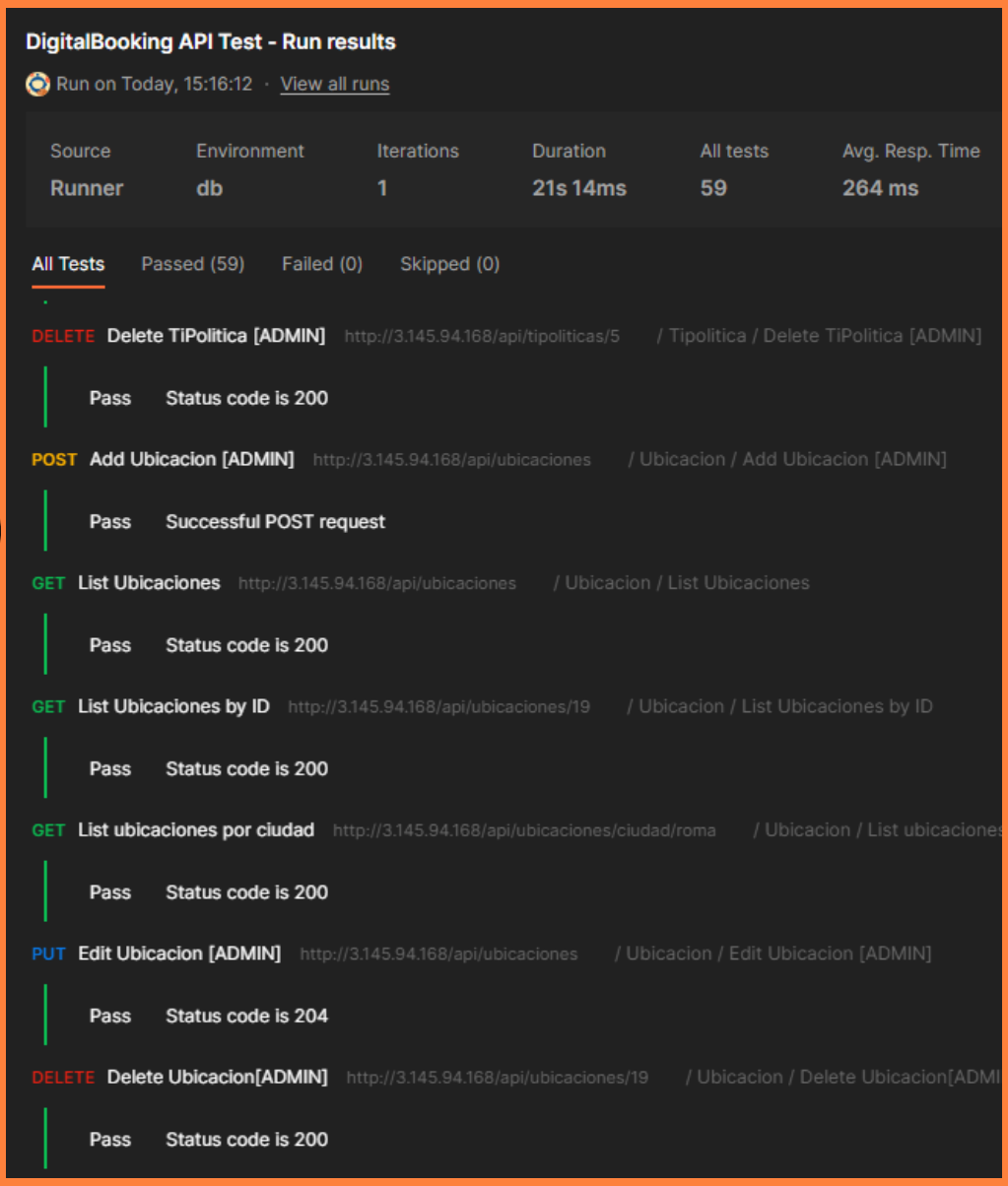
API TESTING

Para el testeo de la API Rest creada para el uso de la applicacion, se utilizó Postman, esto nos permitió probar el 100% de los endpoint con todos sus metodos,verificando que cumplen con lo requerido.

Esto es de suma importancia ,ya que si algo fallara aquí, podria traer tanto problemas de funcionalidad, como de seguridad.

A continuacion encontrará un link con la coleccion exportada en Postman.

POSTMAN COLLECTION



Coverage final.

72.07% Statements `160/222` **67.32%** Branches `68/101` **70.73%** Functions `58/82` **69.58%** Lines `135/194`

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Overall Coverage Summary

Package	Class, %	Method, %	Line, %
all classes	89,3% (67/75)	55,2% (273/495)	55% (618/1124)

Coverage Breakdown

Package	Class, %	Method, %	Line, %
com.dh.digital_booking_back	100% (2/2)	80% (4/5)	80% (4/5)
com.dh.digital_booking_back.controller	100% (12/12)	69,6% (55/79)	60,7% (68/112)
com.dh.digital_booking_back.exception	100% (4/4)	57,1% (4/7)	57,1% (4/7)
com.dh.digital_booking_back.model	93,3% (14/15)	42,5% (45/106)	44,3% (54/122)
com.dh.digital_booking_back.model.DTO	76% (19/25)	46,7% (77/165)	46,7% (77/165)
com.dh.digital_booking_back.security	100% (2/2)	100% (7/7)	100% (87/87)
com.dh.digital_booking_back.security.jwt	100% (2/2)	16,7% (2/12)	7,1% (3/42)
com.dh.digital_booking_back.service	100% (12/12)	71,8% (79/110)	56,1% (321/572)
com.dh.digital_booking_back.util	0% (0/1)	0% (0/4)	0% (0/12)

generated on 2023-04-04

CONCLUSIÓN.

En conclusión, el proyecto digital booking ha sido sometido a rigurosas pruebas desde diferentes aristas y utilizando una variedad de herramientas de testing. Se han realizado pruebas manuales, de regresión, smoke test, de validación y de integración, entre otras, con el fin de garantizar la calidad y la funcionalidad del producto final.

Entre las herramientas utilizadas se encuentran React Testing Library, Jest, Postman, Swagger, JUnit, Mockito y Selenium. Cada una de estas herramientas ha demostrado ser eficaz en su respectiva área de aplicación y ha permitido detectar y corregir problemas en el proyecto.

React Testing Library y Jest se han utilizado para realizar pruebas funcionales y unitarias del front-end, asegurándose de que las nuevas funcionalidades de la aplicación estén funcionando correctamente y de manera consistente. Postman y Swagger se han utilizado para probar los endpoints del back-end y garantizar su correcto funcionamiento.

Por otro lado, JUnit y Mockito han permitido realizar pruebas del back-end de manera automatizada, asegurando la calidad de los componentes fundamentales del proyecto, como los repositorios, los servicios y los controladores. Selenium ha sido utilizado para probar la aplicación en diferentes navegadores y entornos, garantizando su correcto funcionamiento en distintos escenarios.

En general, el proyecto ha alcanzado una alta cobertura del lado del testing, superando el mínimo establecido del 40%. Esto significa que se han probado una gran cantidad de casos posibles y que la mayoría del código ha sido validado en las pruebas. Esto brinda una mayor confianza en la calidad del código y permite detectar rápidamente cualquier problema.

Quedando apto para produccion.