

Universidad ORT Uruguay

Facultad de Ingeniería

Diseño de aplicaciones 2

Descripción del diseño

Obligatorio 1

Emiliano Barboza (147067)
Mauricio Dalgalarondo (189280)

Docentes:
Ignacio Valle
Daniel Acevedo

Entregado como requisito de la materia Diseño de
aplicaciones 2
<https://github.com/ORT-DA2/barboza-dalgalarondo>

15 de octubre de 2020

Declaraciones de autoría

Nosotros, Emiliano Barboza y Mauricio Dalgarrondo, declaramos que el trabajo que se presenta en esa obra es de nuestra propia mano. Podemos asegurar que:

- La obra fue producida en su totalidad mientras realizábamos la materia
- Cuando hemos consultado el trabajo publicado por otros, lo hemos atribuido con claridad;
- Cuando hemos citado obras de otros, hemos indicado las fuentes. Con excepción de estas citas, la obra es enteramente nuestra;
- En la obra, hemos acusado recibo de las ayudas recibidas;
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, hemos explicado claramente qué fue contribuido por otros, y qué fue contribuido por nosotros;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.

Resumen

Este documento del obligatorio 1 plantea poner en práctica los conocimientos vistos hasta el momento en el curso del punto de vista la ejecución de pruebas:

1. Postman

Palabras claves

- Postman, es una aplicación cliente que nos permite crear request y otro tipo de tests.

Índice general

1. Descripción del Proyecto	5
1.1. Introducción y Objetivos	5
1.2. Alcance de la Aplicación	5
2. Postman	7
2.1. Nomenclatura para carpetas y requests	7
2.2. Variables de entorno	7
2.3. Resumen de request por entidad	8
2.3.1. Admins Actions	9
2.3.2. Tourist Actions	11
3. Pruebas de integración para casos marcados con *	13
3.1. (*)Buscar hospedajes para un cierto punto turístico con los parámetros especificados.	13
3.2. (*) Realizar una reserva de un hospedaje.	13
3.3. (*) Dar de alta un nuevo hospedaje o borrar uno existente, para un punto turístico existente.	13
3.4. (*) Modificar la capacidad actual de un hospedaje.	13
3.5. (*) Cambiar el estado de una reserva, indicando una descripción. . . .	13
4. Conclusiones	14
4.1. Mejoras detectadas	14

1. Descripción del Proyecto

1.1. Introducción y Objetivos

El objetivo de este proyecto es crear una api rest para dar soporte a la marca Uruguay Natural.

Las tecnologías involucradas son:

- Web Services (REST API) - Se expondrá las apis en un IIS.
- Microsoft Visual Studio Code (lenguaje C)
- Microsoft SQL Server Express 2017
- Postman
- NET Core SDK 3.1 / ASP.NET Core 3
- Entity Framework Core 3.1.3

1.2. Alcance de la Aplicación

El nuevo sistema, deberá dar acceso a 3 tipos de usuarios posibles en la plataforma (turistas, administradores y super administradores). Para los cuales se listan los requerimientos funcionales para los mismos.

Requerimientos funcionales:

- RF1 - Búsqueda de puntos turísticos por región y por categoría:.
- RF2 - Elegir un punto turístico y realizar una búsqueda de hospedajes.
- RF3 - Dado un hospedaje, realizar una reserva.

Requerimientos no funcionales

- RNF1 - Independencia de librerías.

Se debe diseñar la solución que al modificar el código fuente minimice el impacto del cambio en los componentes físicos de la solución.

Cada paquete lógico debe ser implementado en un assembly independiente.

- RNF2 - Acceso a las funcionalidades mediante HTTP

Acceso mediante web service.

- RNF3 - Persistencia en base de datos.

El diseño debe contemplar el modelado de una solución de persistencia adecuada para el problema utilizando Entity Framework (Code First).

Se espera que como parte de la entrega se incluya dos respaldos de la base de datos: uno vacío y otro con datos de prueba.

Se debe entregar el archivo .bak y también el script .sql para ambas bases de datos.

- RNF4 - Mantenibilidad

Estar en un repositorio Git.

Haber sido escrito utilizando TDD

Se debe utilizar el framework Moq para realizar los Mocking.

- RNF5 - Control de versiones

GitFlow

2. Postman

Es el cliente que se utilizó para realizar las pruebas de integración. A su vez tuvimos la necesidad de respaldarnos en otras herramientas para calcular fechas online dado que nuestras fechas son pasadas con formato Ticks de c.

2.1. Nomenclatura para carpetas y requests

- Las carpetas darán vida a una entidad dentro de nuestro negocio a excepción de la de Versiones, que es utilizada con el fin de mostrar el versionado de requests.
- Para las requests se eligió seguir el mismo criterio usado en los test unitarios, aunque podrán verse nombres desactualizados dado que se estandarizó en una etapa más tarde
Ejemplo: Login - user not exists in DB - Returns Invalid credentials.

2.2. Variables de entorno

Se hizo uso de las variables de entorno para poder configurar distintos entornos. Ejmplo: cada desarrollador tenía un puerto diferente donde levanta la aplicación.

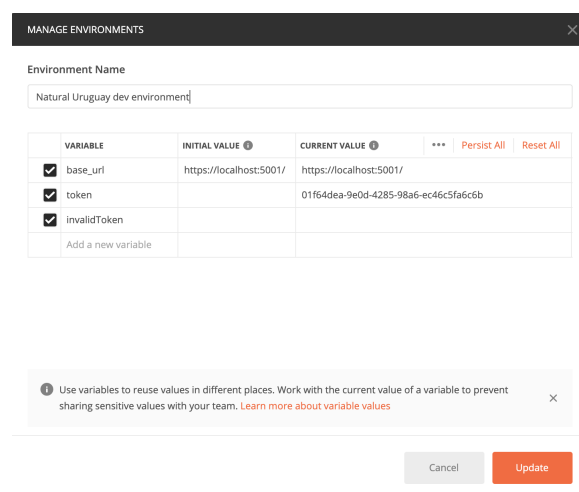


Figura 2.1: Postman, configuración de ambiente

2.3. Resumen de request por entidad

Los requests se separaron en dos grandes grupos, **Admins Actions** y **Tourist Actions**.

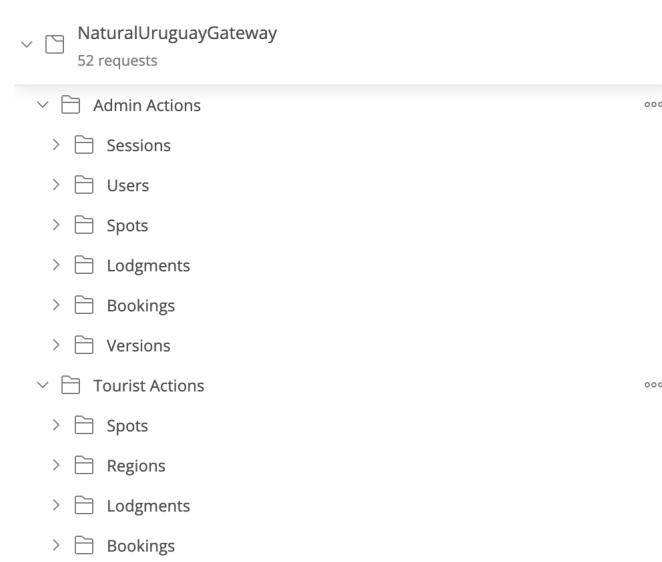


Figura 2.2: Vista general de Postman

Las Admins Actions, son acciones que solamente pueden ser ejecutadas por usuarios que deben loguearse y obtener una sesión en el sistema. A su vez, hay acciones que únicamente las podrá efectuar un Super Administrador.

Cómo mínimo de la solución se provee de un Super Administrador para lograr ese cometido y sus credenciales son:

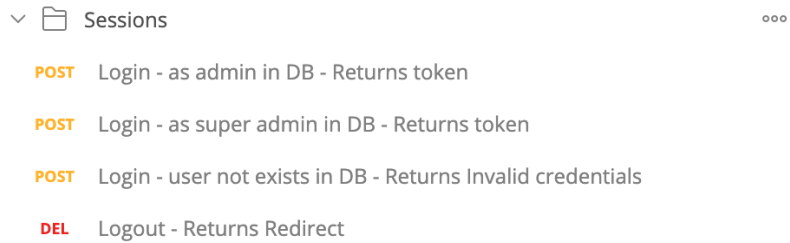
- email: super@admin.com
- password: hakunamatata

A su vez, se deja constancia que la password por defecto será **hakunamatata** para todo usuario creado por este Super Administrador. La misma es configurable dentro del appsettings.json.

Las Tourist Actions, son acciones que un usuario puede realizar sin necesidad de estar logueado en el sistema. Ejmplos de estas serían, obtener las regiones, obtener listado de hospedajes, etc.

2.3.1. Admins Actions

Sessions

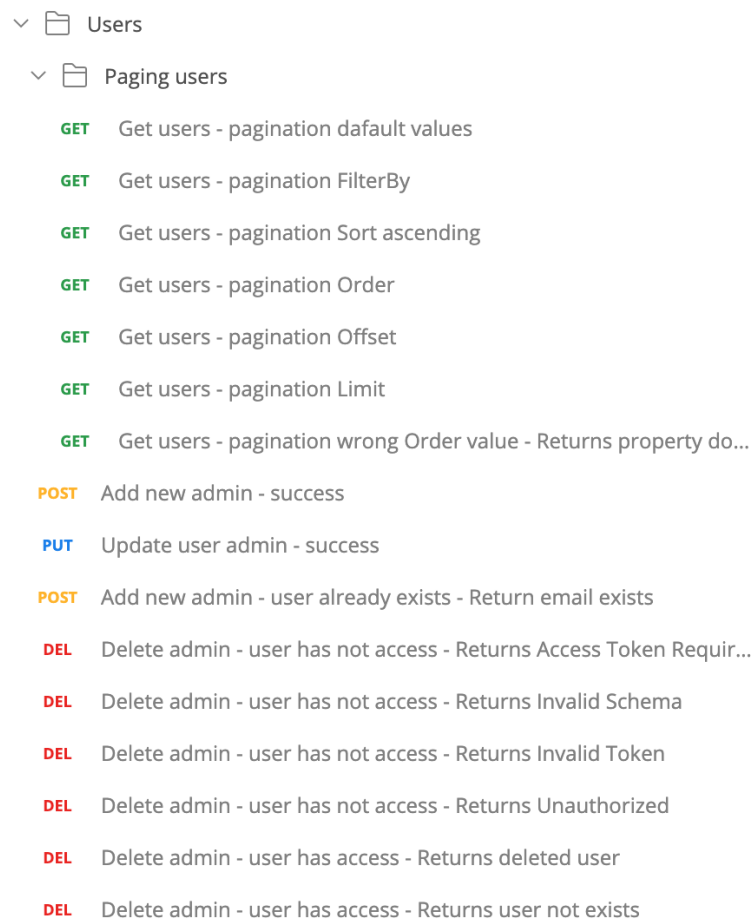


The screenshot shows a REST client interface for the 'Sessions' endpoint. It lists four actions: two POST requests for login (as admin and as super admin) and one POST request for login when the user does not exist, all returning a token. It also shows a DELETE request for logout, returning a redirect.

Method	Action	Response
POST	Login - as admin in DB	Returns token
POST	Login - as super admin in DB	Returns token
POST	Login - user not exists in DB	Returns Invalid credentials
DEL	Logout	Returns Redirect

Figura 2.3: Sessions

Users



The screenshot shows a REST client interface for the 'Users' endpoint. It lists 19 actions, including GET requests for pagination (with various filters and sort orders), POST for adding new admins, PUT for updating users, and DELETE for removing users under different conditions (access, schema, token, authorization, and user existence).

Method	Action	Response
GET	Get users - pagination default values	
GET	Get users - pagination FilterBy	
GET	Get users - pagination Sort ascending	
GET	Get users - pagination Order	
GET	Get users - pagination Offset	
GET	Get users - pagination Limit	
GET	Get users - pagination wrong Order value	Returns property do...
POST	Add new admin - success	
PUT	Update user admin - success	
POST	Add new admin - user already exists	Return email exists
DEL	Delete admin - user has not access	Returns Access Token Requir...
DEL	Delete admin - user has not access	Returns Invalid Schema
DEL	Delete admin - user has not access	Returns Invalid Token
DEL	Delete admin - user has not access	Returns Unauthorized
DEL	Delete admin - user has access	Returns deleted user
DEL	Delete admin - user has access	Returns user not exists

Figura 2.4: Users

Spots

✓	📁	Spots	...
✓	📁	Lodgments	...
POST		Add Lodgment - Success	
DEL		Delete Lodgment - Success	
DEL		Delete Lodgment - Lodgment not exists	
POST		Add Lodgment - Without Tourist spot - Returns Tourist point not...	
POST		Add Lodgment - Lodgment already exists	
POST		Add Lodgment - Data required	
POST		Add spot - Success	
POST		Add spot - Wrong category	
POST		Add spot - Already exists	
POST		Add spot - Region not exists	
POST		Add spot - Data required	
POST		Add spot - Access denied	

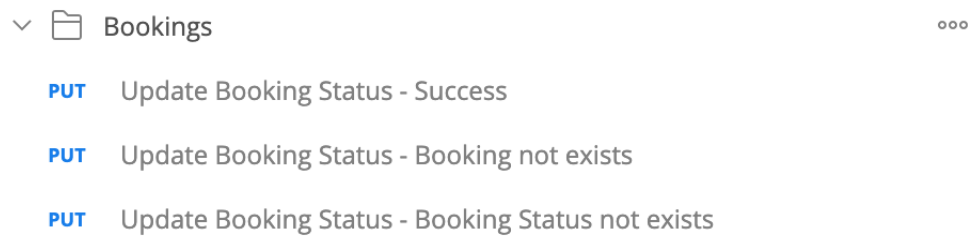
Figura 2.5: Spots

Lodgments

✓	📁	Lodgments	...
POST		Activate lodgment	
DEL		Activate lodgment Copy	

Figura 2.6: Lodgments

Bookings

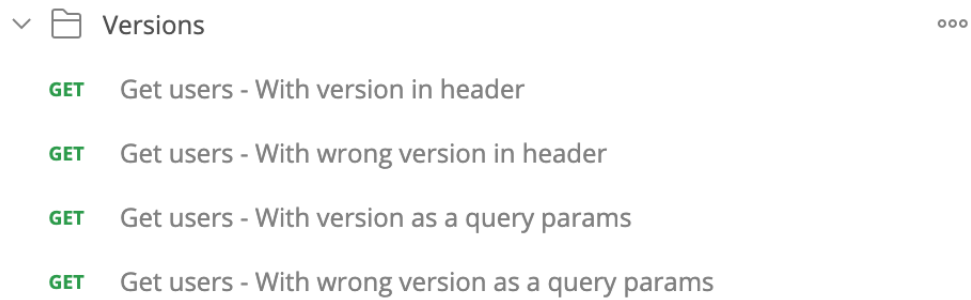


This screenshot shows a list of API endpoints under the 'Bookings' category. The category is expanded, indicated by a downward arrow and a folder icon. There are three endpoints, all using the PUT method, listed in blue text. To the right of the list is a three-dot menu icon.

✓	📁	Bookings	...
PUT		Update Booking Status - Success	
PUT		Update Booking Status - Booking not exists	
PUT		Update Booking Status - Booking Status not exists	

Figura 2.7: Bookings

Versions



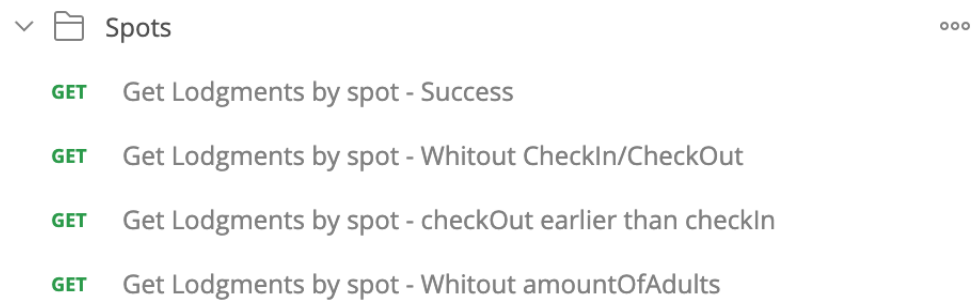
This screenshot shows a list of API endpoints under the 'Versions' category. The category is expanded, indicated by a downward arrow and a folder icon. There are four endpoints, all using the GET method, listed in green text. To the right of the list is a three-dot menu icon.

✓	📁	Versions	...
GET		Get users - With version in header	
GET		Get users - With wrong version in header	
GET		Get users - With version as a query params	
GET		Get users - With wrong version as a query params	

Figura 2.8: Bookings

2.3.2. Tourist Actions

Spots



This screenshot shows a list of API endpoints under the 'Spots' category. The category is expanded, indicated by a downward arrow and a folder icon. There are four endpoints, all using the GET method, listed in green text. To the right of the list is a three-dot menu icon.

✓	📁	Spots	...
GET		Get Lodgments by spot - Success	
GET		Get Lodgments by spot - Whitout CheckIn/CheckOut	
GET		Get Lodgments by spot - checkOut earlier than checkIn	
GET		Get Lodgments by spot - Whitout amountOfAdults	

Figura 2.9: Spots

Regions

▼	📁	Regions	...
GET		Get Regions - Success	
GET		Get Regions - With parameters	
GET		Get Spots by Region - Success	
GET		Get Spots by Region - Using filters	

Figura 2.10: Regions

Lodgments

▼	📁	Lodgments	...
▼	📁	Bookings	...
POST		Add Booking - Success	
POST		Add Booking - Whitout json params	

Figura 2.11: Lodgments

Bookings

▼	📁	Bookings	...
GET		Get Booking Status - Success	

Figura 2.12: Bookings

3. Pruebas de integración para casos marcados con *

Para estas pruebas utilizamos la herramienta Loom la cual nos facilita el grabar videos mediante una extensión de Chrome y publicarlos. Estos quedan públicos así el lector puede acceder a ellos.

3.1. (*) Buscar hospedajes para un cierto punto turístico con los parámetros especificados.

Link para ver en Loom, tiempo 2:33

3.2. (*) Realizar una reserva de un hospedaje.

Link para ver en Loom, tiempo 1:24

3.3. (*) Dar de alta un nuevo hospedaje o borrar uno existente, para un punto turístico existente.

Link para ver en Loom, tiempo 2:13

3.4. (*) Modificar la capacidad actual de un hospedaje.

Link para ver en Loom, tiempo 1:33

3.5. (*) Cambiar el estado de una reserva, indicando una descripción.

Link para ver en Loom, tiempo 1:08 4

4. Conclusiones

La práctica nos dejó un nuevo conocimiento sobre Postman, lo cual nos hizo entender el potencial al momento de crear soluciones tanto para testear apis, como para hacer tests de integración automatizados.

4.1. Mejoras detectadas

- Se deben actualizar nombres de requests para q cumplan con la nueva nomenclatura.
- Automatización de los casos, es posible crear grupos de ejecución. Estos los probamos, pero no nos sentímos cómodos para entregarlos. Igualmente a futuro sería posible crearlos.
- Si bien entregamos un ambiente sólo de configuración, podríamos crear varios para distintos entornos.

Bibliografía

- [1] Universidad ORT Uruguay. (2013) Documento 302 - Facultad de Ingeniería. [Online]. Available: <http://www.ort.edu.uy/fi/pdf/documento302facultaddeingenieria.pdf>
- [2] Anónimo. (2012) Postman Learning Center. [Online]. Available: <https://learning.postman.com/docs/getting-started/introduction/>